# CSC 656
# Homework #2
# By
# Andrew Hutzel
# (915841776)

Question 1)

1. (a) **Assembly language to Machine language:** Write down the machine language representation for the following instruction.

   sb $s0,10($s1)

   [Hint: Assume $s0 = 16, $s1 = 17]

**1a Answer)**

**Opcode $s1 $s0 operandAddress => (opcode is 6 bits, $s1 is 5 bits, $s0 is 5 bits, and opAddr is 16 bits => 32 bits in total)**

**Translated to machine code is…**

**(1010) (0010) (0011) (0000) (0000) (0000) (0001) (1011)**

(b) **Machine language to Assembly language:** Given this MIPS machine language code fragment in memory:

| Address | Contents |
|---|---|
| 0x00400024 | 0x3411000b |
| 0x00400028 | 0x3c101001 |
| 0x0040002c | 0x2a21000b |
| 0x00400030 | 0x14200007 |
| 0x00400034 | 0xae080004 |
| 0x00400038 | 0x2231ffff |
| 0x0040003c | 0x01129020 |
| 0x00400040 | 0x2210fffc |
| 0x00400044 | 0x2a210002 |
| 0x00400048 | 0x1020fffb |

Disassemble and show the MIPS instructions.
For constants in I-format instructions, you may leave them in hexadecimal, or translate them into decimal.
Do not worry about labels. Show clearly target addresses for all branches and jumps.

R-TYPE
**(6 bits (OP), 5 bits (RS), 5 bits (RT), 5 bits (RD), 5 bits (SHAMT, shift amount), 6 bits (funct))**

**I-TYPE ← Using this**
**(6 bits (OP), 5 bits (RS), 5 bits (RT), 16 bits (CONSTANT OR ADDRESS))**

1b ANSWER)

(Sorry for carrying over to the next page, I couldn't prevent it. Microsoft word sucks!)

Now lets translate the contents
0x3411000b => (0011) (0100) (0001) (0001) (0000) (0000) (0000) (1011)

0x3c101001 => (0011) (1100) (0001) (0000) (0001) (0000) (0000) (0001)

0x2a21000b => (0010) (1010) (0010) (0001) (0000) (0000) (0000) (1011)

0x14200007 => (0001) (0100) (0010) (0000) (0000) (0000) (0000) (0111)

0xae080004 => (1010) (1110) (0000) (1000) (0000) (0000) (0000) (0100)

0x2231ffff => (0010) (0010) (0011) (0001) (1111) (1111) (1111) (1111)

0x01129020 => (0000) (0001) (0001) (0010) (1001) (0000) (0010) (0000)

0x2210fffc => (0010) (0010) (0001) (0000) (1111) (1111) (1111) (1100)

0x2a210002 => (0010) (1010) (0010) (0001) (0000) (0000) (0000) (0010)

0x1020fffb => (0001) (0000) (0010) (0000) (1111) (1111) (1111) (1011)

And now that all of the contents have been translated to binary, lets associate them with their instructions….

ORI $s1 $zero 0x000B

LUI $s0 0x1001

SLTI $at $s1 0x000B

BNE $at $zero 0x0007

SW $t0 0x0004 $s0

ADDI $s1 $s1 0xFFFF

ADD $s2 $t0 $s2

ADDI $s0 $s0 0xFFFC

SLTI $at $s1 0x0002

BEQ $at $zero 0xFFFB

Question 2 Exercise 2.14 )

ANSWER)

Assembly instructions for following binary value (0000) (0010) (0001) (0000) (1000) (0000) (0010) (0000)

| Bin value | 000000 | 10000 | 10000 | 100000 | 0000 | 100000 |
|---|---|---|---|---|---|---|
| Assembly | Opcode=0 (adding) | Rs=16 ($s0) | Rt=16 ($s0) | Rd=16 16($s0) | Shamt=0 | Funct=0x20 (add) |

And so we get => add $s0, $s0($s0)

Question 3 Exercise 2.15)

Provide the type and hexadecimal representation of the following instruction

Sw $t1, 32($t2)

Answer)

I-Type Instruction

Opcode = 6 bits, rs 5 bits, rt 5 bits, and address is 16 bits.

As for the hexadecimal representation it is as follows:

Opcode (sw)=101011

$t1= 01001

$t2= 01010

Address= 0000 0000 0010 0000

Which means that this binary address appears as this to hex:

1010 1101 0010 1010 0000 0000 0010 0000 to hex is

0xAD490020

Question 4 Exercise 3.13)

**3.13** [20] <§3.3> Using a table similar to that shown in Figure 3.6, calculate the product of the hexadecimal unsigned 8-bit integers 62 and 12 using the hardware described in Figure 3.5. You should show the contents of each register on each step.

Answer, get ready for pain!)

Calculation for the product of the hexadecimal unsigned 8-bit int 62 and 12 are as follows:

0x64*0x12 in bin it is

0000 0110 0100

x 0000 0001 0010

0110 1110 0100

DEC: 1,764

Hex: 6E4

***For the sake of making it easy to read, the table is on the next page***.

| Iteration | Step | Multiplier | Multiplicand | Product |
|---|---|---|---|---|
| 0 | Beginning… | 0001 0010 | 0000 0000 0110 0010 | 0000 0000 0000 0000 |
| 1 | 0, No operation | 0001 0010 | 0000 0000 0110 0010 | 0000 0000 0000 0000 |
| | Shift Left Multiplicand | 0001 0010 | 0000 0000 1100 0100 | 0000 0000 0000 0000 |
| | Shift Right Multiplier | 0000 1001 | 0000 0000 1100 0100 | 0000 0000 0000 0000 |
| 2 | Prod=Prod+Mcand | 0000 1001 | 0000 0000 1100 0100 | 0000 0000 1100 0100 |
| | Shift Left Multiplicand | 0000 1001 | 0000 0001 1000 1000 | 0000 0000 1100 0100 |
| | Shift right Multiplier | 0000 0100 | 0000 0001 1000 1000 | 0000 0000 1100 0100 |
| 3 | No operation | 0000 0100 | 0000 0001 1000 1000 | 0000 0000 1100 0100 |
| | Shift Left Multiplicand | 0000 0100 | 0000 0011 0001 0000 | 0000 0000 1100 0100 |
| | Shift right multiplier | 0000 0010 | 0000 0011 0001 0000 | 0000 0000 1100 0100 |
| 4 | No operation… | 0000 0010 | 0000 0011 0001 0000 | 0000 0000 1100 0100 |
| | Shift left multiplicand | 0000 0010 | 0000 0110 0010 0000 | 0000 0000 1100 0100 |
| | Shift right multiplier | 0000 0001 | 0000 0110 0010 0000 | 0000 0000 1100 0100 |
| 5 | Prod=Prod+Mcand | 0000 0001 | 0000 0110 0010 0000 | 0000 0110 1110 0100 |
| | Shift left multiplicand | 0000 0001 | 0000 1100 0100 0000 | 0000 0110 1110 0100 |
| | Shift right multiplier | 0000 0000 | 0000 1100 0100 0000 | 0000 0110 1110 0100 |

Question 5 3.14)

**3.14** [10] <§3.3> Calculate the time necessary to perform a multiply using the approach given in Figures 3.3 and 3.4 if an integer is 8 bits wide and each step of the operation takes 4 time units. Assume that in step 1a an addition is always performed—either the multiplicand will be added, or a zero will be. Also assume that the registers have already been initialized (you are just counting how long it takes to do the multiplication loop itself). If this is being done in hardware, the shift s of the multiplicand and multiplier can be done simultaneously. If this is being done in soft ware, they will have to be done one aft er the other. Solve for each case.

Answer)

So far we know, note all binary are assumed to be base 2… :

A = E4 and B=E5

Convert to binary and dec….

A= E4 (HEX) = 1110 0100 (BIN) = 228 (DEC)

B= E5 (HEX) = 1110 0101 (BIN) = 229 (DEC)

Using ALU and the part of the hexadecimal multiplier, the ALU will have 4 units of input.

1110 0100 + 1110 0101 => 1110 0100 1110 0101

E5*E5=cbf4 = 52212 = 1100 1011 1110100

**Binary E5**

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

Question 6 Exercise 3.18)

**3.18** [20] <§3.4> Using a table similar to that shown in Figure 3.10, calculate 74 divided by 21 using the hardware described in Figure 3.8. You should show the contents of each register on each step. Assume both inputs are unsigned 6-bit integers.

Answer)

| Iteration | Operation | Quotient | Divisor | Remainder |
|---|---|---|---|---|
| 0 | First values… | 0000 00 | 010001 000000 | 000000 111100 |
| 1 | Rem = Rem – Div | 0000 00 | 010001 000000 | 101111 111100 |
| | Rem<0 => Div, sll Q, Q0=0 | 0000 00 | 010001 000000 | 000000 111100 |
| | Shift Div Right | 0000 00 | 001000 100000 | 000000 111100 |
| 2 | Rem = Rem – Div | 0000 00 | 001000 100000 | 111000 011100 |
| | Rem<0 => Div, sll Q, Q0=0 | 0000 00 | 001000 100000 | 000000 111100 |
| | Shift Div Right | 0000 00 | 000100 010000 | 000000 111100 |
| 3 | Rem = Rem – Div | 0000 00 | 000100 010000 | 111100 101100 |
| | Rem<0 => Div, sll Q, Q0=0 | 0000 00 | 000100 010000 | 000000 111100 |
| | Shift Div Right | 0000 00 | 00010 001000 | 000000 111100 |
| 4 | Rem = Rem – Div | 0000 00 | 000010 001000 | 111110 110100 |
| | Rem<0 => Div, sll Q, Q0=0 | 0000 00 | 000010 001000 | 000000 111100 |
| | Shift Div Right | 0000 00 | 000001 000100 | 000000 111100 |
| 5 | Rem = Rem – Div | 0000 00 | 000001 000100 | 111111 111000 |
| | Rem<0 => Div, sll Q, Q0=0 | 0000 00 | 000001 000100 | 000000 111100 |
| | Shift Div Right | 0000 00 | 000000 100010 | 000000 111100 |
| 6 | Rem = Rem – Div | 0000 00 | 000000 100010 | 000000 011010 |
| | Rem<0 => Div, sll Q, Q0=0 | 0000 01 | 000000 100010 | 000000 011010 |
| | Shift Div Right | 0000 01 | 000000 010001 | 000000 011010 |
| 7 | Rem = Rem – Div | 0000 01 | 000000 010001 | 000000 001001 |
| | Rem<0 => Div, sll Q, Q0=0 | 0000 11 | 000000 010001 | 000000 001001 |
| | Shift Div Right | 0000 11 | 000000 001000 | 000000 001001 |

And so we know 74/21 = 3.52 approximately and we are left with a remainder of 5… Somewhat verifying my work here.

Question 7 Exercise 3.24)

**3.24** [10] <§3.5> Write down the binary representation of the decimal number 63.25 assuming the IEEE 754 double precision format.

Answer)

Decimal number given is 63.25 so lets begin by converting it to binary

| 64 | 32 | 16 | 8 | 4 | 2 | 1 | 1/2 | 1/4 |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |

So we have (111111.01) base 2 , we next normalize it yielding (1.1111101 x2^5)

Exp = 1023+5 = 1028 , lastly we now do the binary representation using DOUBLE precision formatting yielding… a whole lot of zeros…

**0 10000000100 1111101000000000000000000000000000000000000000000**

Question 8 Exercise 3.27)

**3.27** [20] <§3.5> IEEE 754-2008 contains a half precision that is only 16 bits wide. Th e left most 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. Write down the bit pattern to represent -1.5625x10^-1 assuming a version of this format, which uses an excess-16 format to store the exponent. Comment on how the range and accuracy of this 16-bit fl oating point format compares to the single precision IEEE 754 standard.

Answer 8A)

Decimal to binary

-1.5625 x 10^-1 = -0.15625

= -0.00101 (Bin, now move 3 digits to the right…)

=-1.01 x 2^-3

Exponent=-3 + 15

=12

=01100

Mantissa = 0100000000

(Pos/Neg bit) (Exponent) (Mantissa) for the binary pattern below.

Binary Pattern will be: 1 01100 0100000000

Answer 8 COMMENTS)

Range and accuracy of this 16-bit floating point format can possibly be out of range for the given numbers. If single precision IEEE 754 standard was used here, then the number can be fully represented because of an increase of 32 bits are used in that format.

Question 9 Exercise 3.3)

**3.30** [30] <§3.5> Calculate the product of –8.0546875 _ $10_0$ and _1.79931640625 _ $10_{-1}$ by hand, assuming A and B are stored in the 16-bit half precision format described in Exercise 3.27. Assume 1 guard, 1 round bit, and 1 sticky bit, and round to the nearest even. Show all the steps; however, as is done in the example in the text, you can do the multiplication in human-readable format instead of using the techniques described in Exercises 3.12 through 3.14. Indicate if there is overfl ow or underfl ow. Write your answer in both the 16-bit fl oating point format described in Exercise 3.27 and also as a decimal number. How accurate is your result? How does it compare to the number you get if you do the multiplication on a calculator?

Answer)

Given we know:

(-8.0546875 x 10^-1) X (-1.79931640625 x 10^-1)

Convert to binary

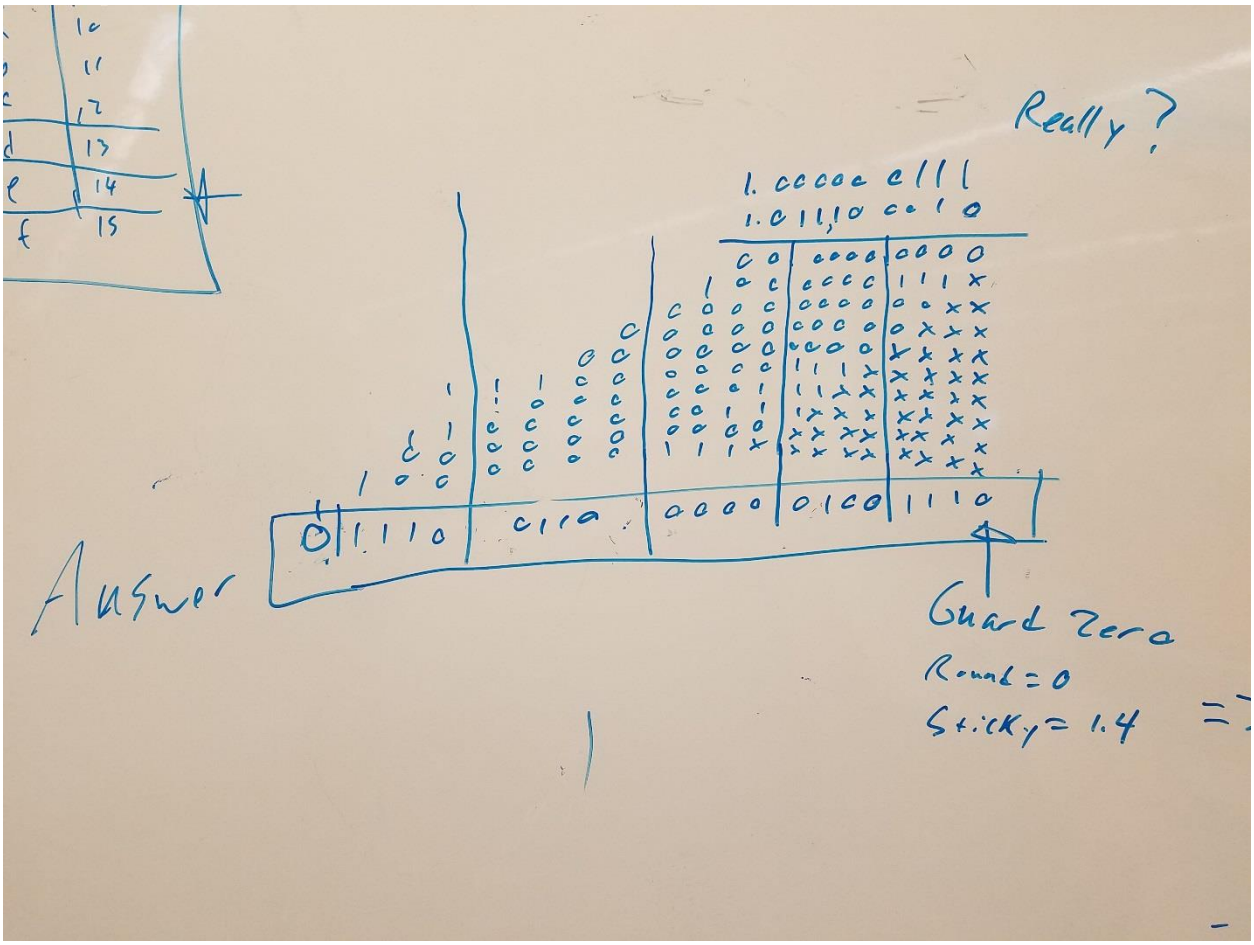
-8.0546875 (Showing work for fractional conversion…)

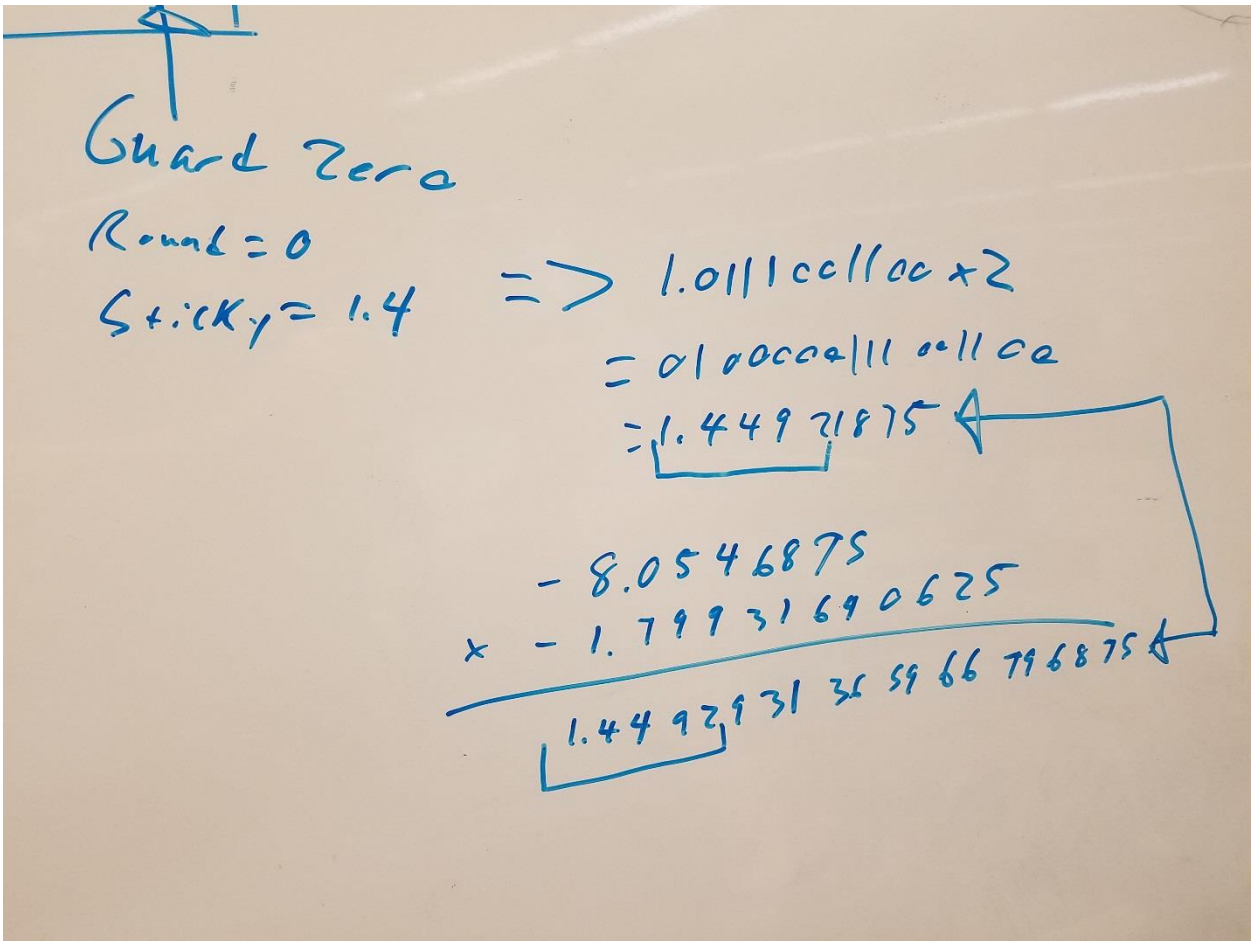
| ½ | ¼ | 1/8 | 1/16 | 1/32 | 1/64 | 1/128 |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |

- 1000 . 0000111 => -1.0000000111 x 2^3

Taking the exponent we have -3 +3 = 0 so 0+16 = 16 => [1 0000, binary]

Forget doing this with tables, next page is all hand written, rest of the problem is there.


Really?
Answer
Guard Zero
Round = 0
Sticky = 1.4

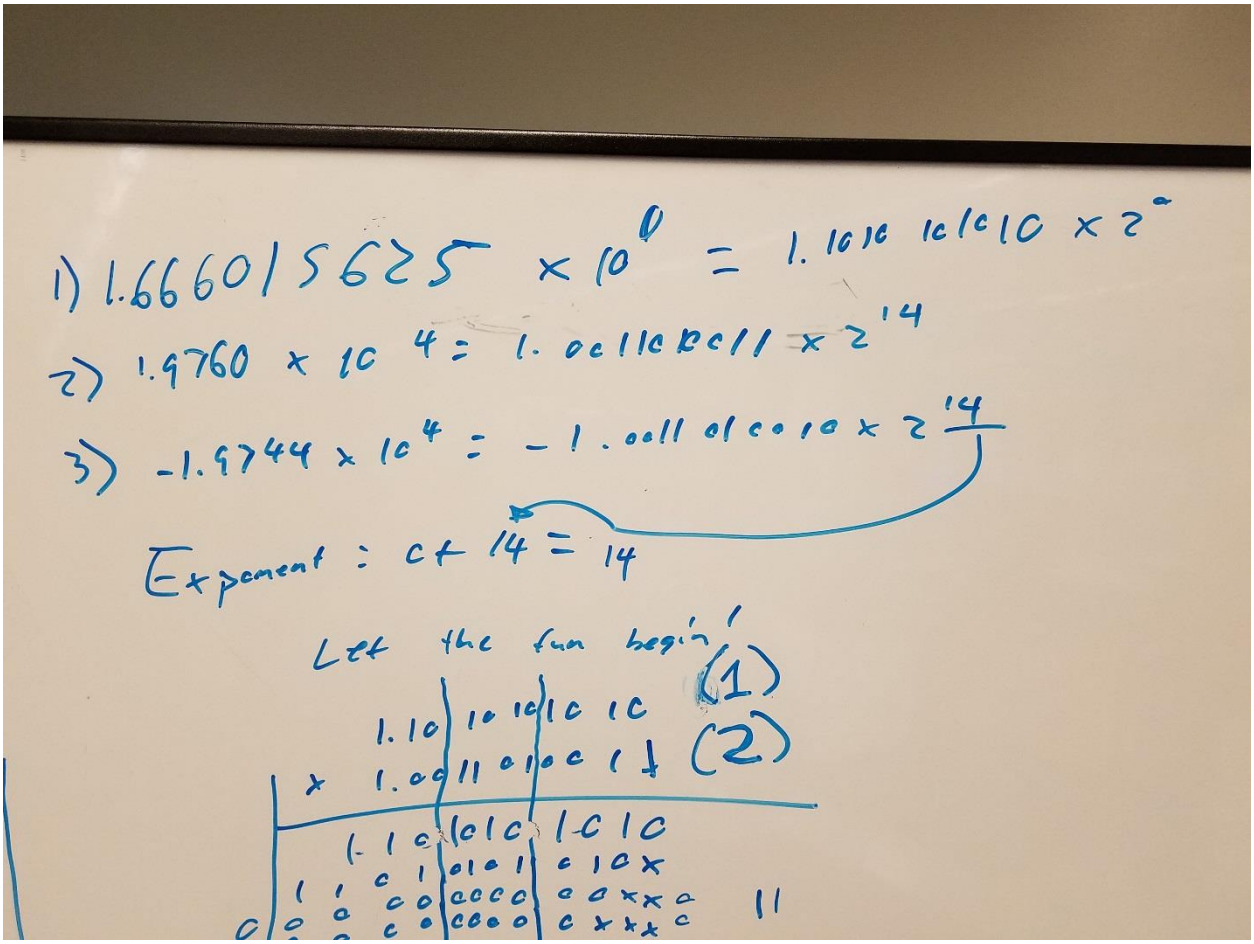


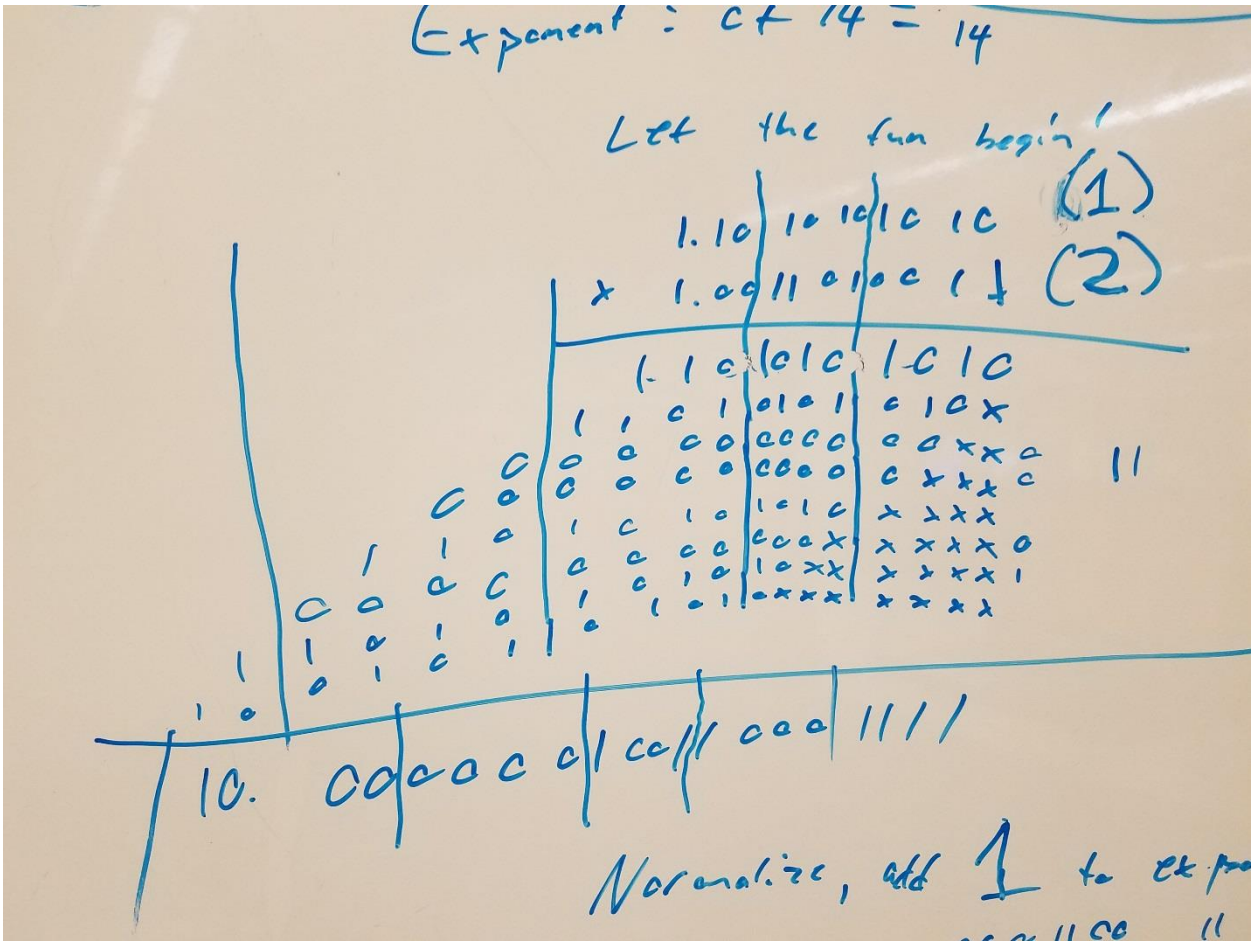

My result is accurate to the 4th decimal place. In contrast between my answer and the calculators it would seem as though the calculator has higher precision but my answer is able to ball park it with a minimal loss of data.

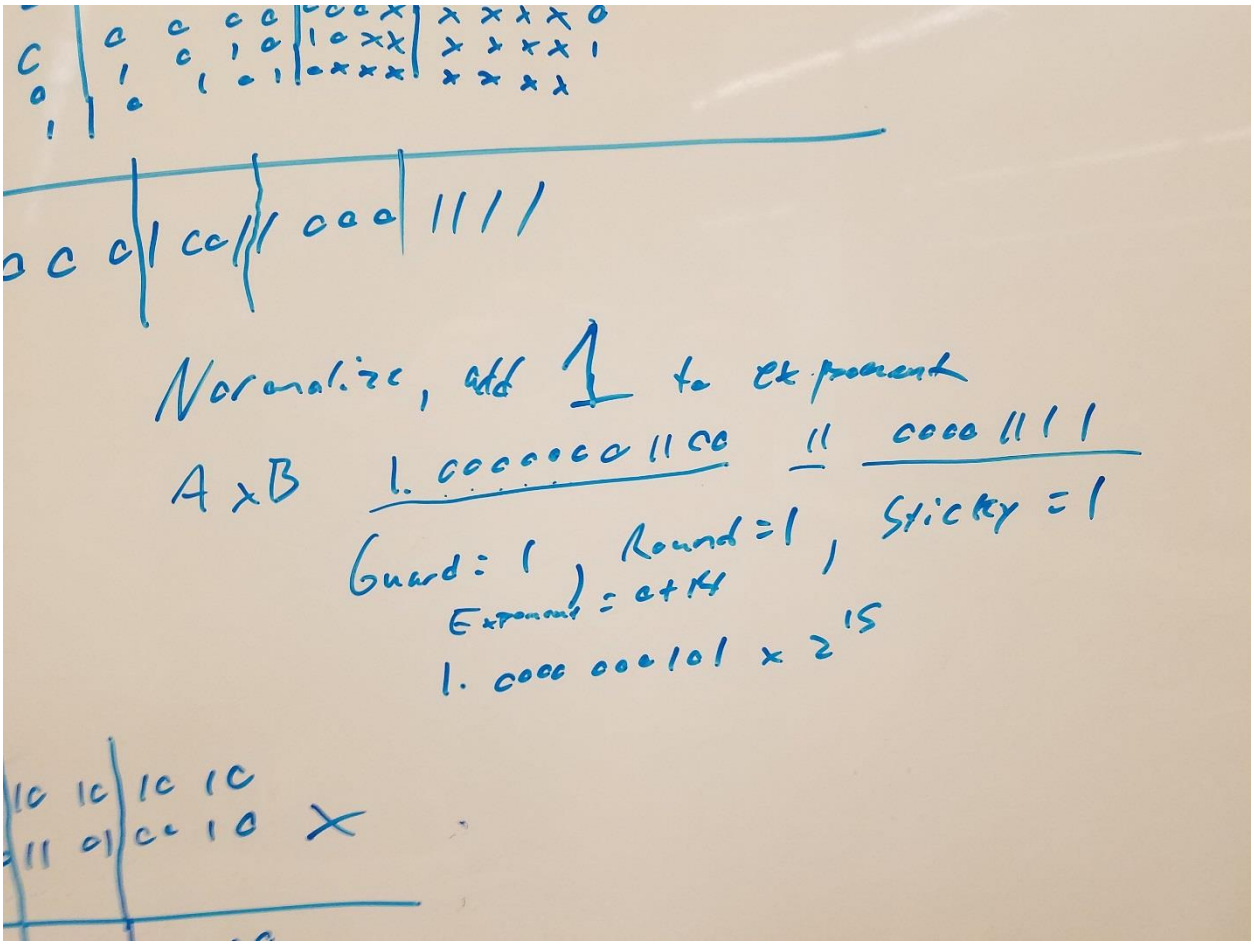
Question 10 Exercise 3.38)

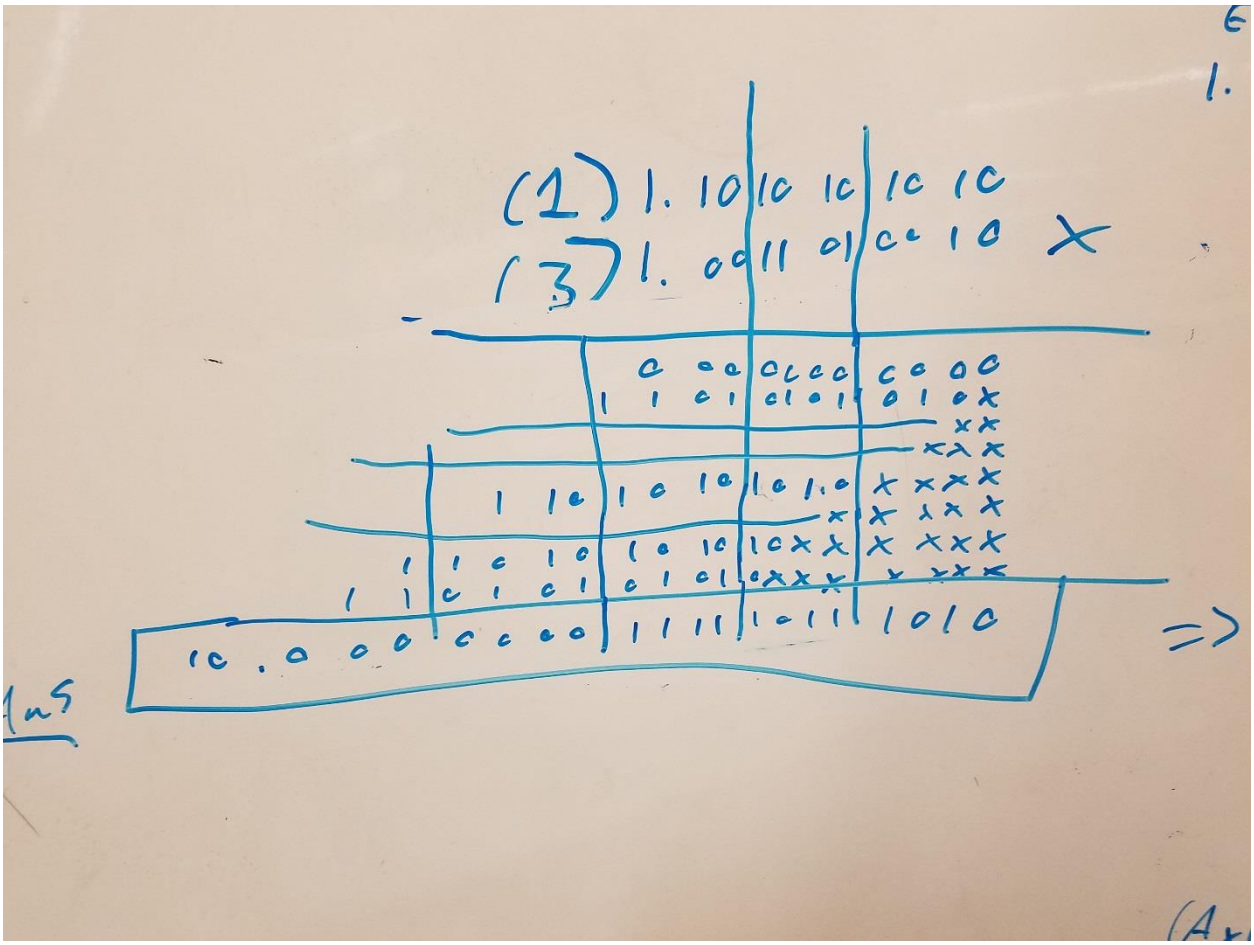
**3.38** [30] <§3.9> Calculate 1.666015625 _ $10_0$_ (1.9760 _ $10_4$ _ _1.9744 _ $10_4$) by hand, assuming each of the values are stored in the 16-bit half precision format described in Exercise 3.27 (and also described in the text). Assume 1 guard, 1 round bit, and 1 sticky bit, and round to the nearest even. Show all the steps, and write your answer in both the 16-bit fl oating point format and in decimal.

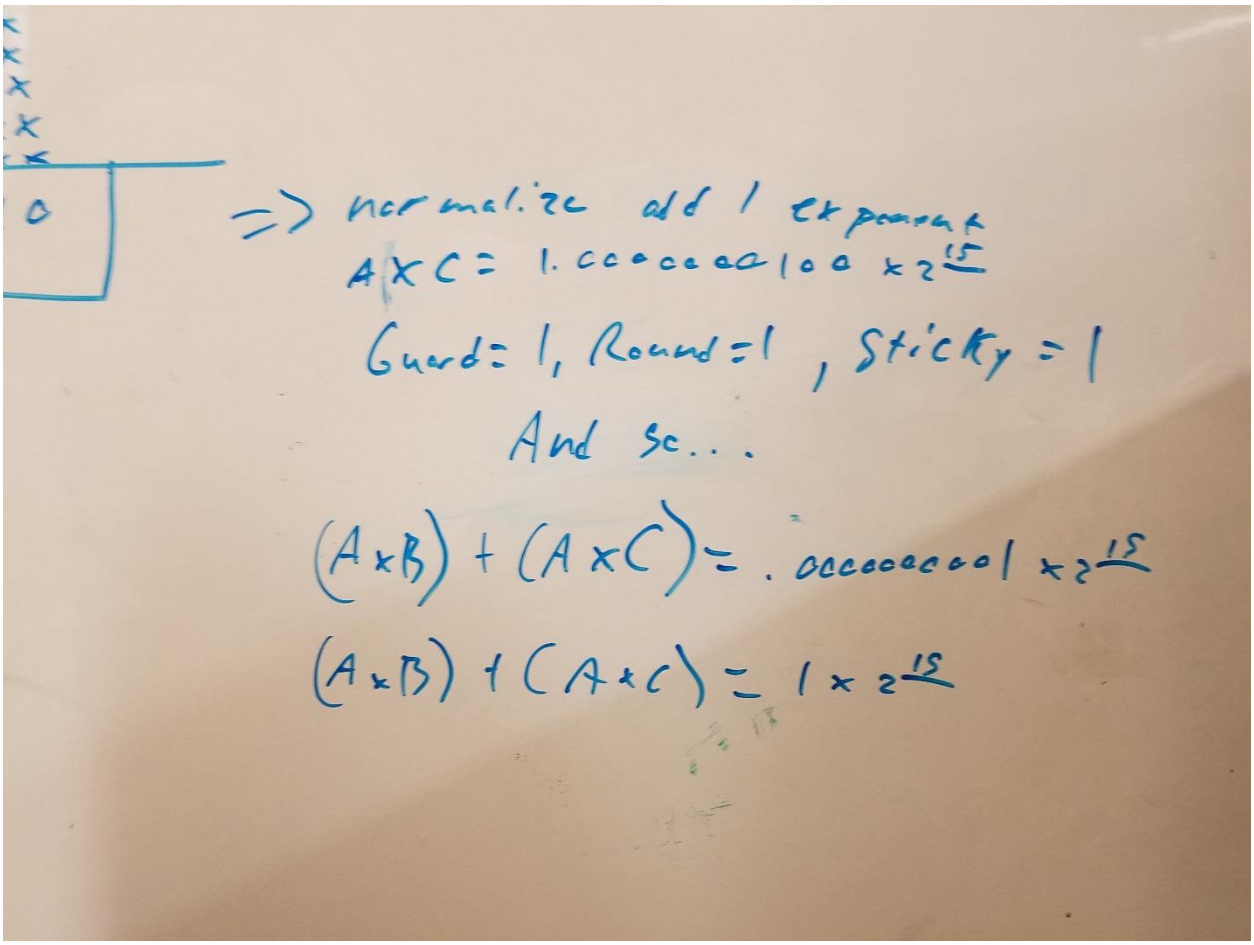
Answer)




Exponent : 14 = 14
Let the fun begin!
(1)
(2)
Normalize, add 1 to


Normalize, add 1 to exponent
A x B
Guard: 1, Round = 1, Sticky = 1
Exponent = e+14
1. 0000 0001 01 x 2^15




0
=> normalize add 1 exponent
A x C = 1.0000000100 x 2^15
Guard: 1, Round: 1, Sticky: 1
And so...
(A x B) + (A x C) = .0000000001 x 2^15
(A x B) + (A x C) = 1 x 2^15