Homework Chapter 3

Name: `Andrew Januszko`

DO NOT CHANGE THE PAGING OR SPACING OF THIS DOCUMENT. IF YOU NEED TO ADD ADDITIOAL PAGES ---- PUT THEM AT THE END!!!! IF YOU CHANGE THE PAGINATION, YOU WILL NOT GET CREDIT.

1. Using base-16, show how to perform:
   a. 3a5b + 01d2

```
     +1
   3  A  5  B
 + 0  1  D  2
 ____________
   3  C  2  D
```

   b. 4cd0 – 1b87

```
        [C] +16
   4  C  D  0
 - 1  B  8  7
 ____________
   3  1  4  9
```

2. Using a table similar to that shown in Figure 3.6, calculate the product of hexadecimal, unsigned 8-bit integers 45 and 24 using the hardware described in Figure 3.5. You should show the contents of each register on each step.

45 hex -> 69 decimal -> 01000101
24 hex -> 36 decimal -> 00100100

`01000101 x 00100100`

	Product	Multipicand
0	0000 0000 0100 0101	0010 0100
//		
	0010 0100 010 00101	0010 0100
1	0001 0010 0010 0010	0010 0100
//		
	0001 0010 0010 0010	0010 0100
2	0000 1001 0001 0001	0010 0100
//		
	0000 1001 0001 0001	0010 0100
	0010 1101 0001 0001	0010 0100
3	0001 0110 1000 1000	0010 0100
//		
	0001 0110 1000 1000	0010 0100
4	0000 1011 0100 0100	0010 0100
//		
	0000 1011 0100 0100	0010 0100
5	0000 0101 1010 0010	0010 0100
//		
	0000 0101 1010 0010	0010 0100
6	0000 0010 1101 0001	0010 0100
//		
	0000 0010 1101 0001	0010 0100
	0010 0110 1101 0001	0010 0100
7	0001 0011 0110 1000	0010 0100
//		
	0001 0011 0110 1000	0010 0100
8	0000 1001 1011 0100	0010 0100

3. Show how to multiply the two binary numbers: 0b1010_0010 x 0b0010_0010

```
        10100010
      x 00100010
      __________
        00000000
       101000100
      0000000000
     00000000000
    000000000000
   1010001000000
  00000000000000
 000000000000000
________________
 001010110000100
```

4. Write down the binary representation of the decimal number: 127.125 assuming IEEE 754 single precision format.

```
127 in binary is 1111111
0.125 in binary is 0.001

1111111.001 → 1.111111001 * 2^6

Since single precision is 127, we add our exponent to it: 127 + 6 = 133 → 10000101

Store the sign bit → 0
Store our exponent value with single precision → 010000101
Store the mantissa → 010000101111111001
Add zeros till 32-bit → 01000010111111100100000000000000

127.125 in binary is 01000010111111100100000000000000
```

5. Write down the binary representation of the decimal number 1.25e10 assuming an IEEE 754 double precision format.

```
1.25e10 in binary is 1011101001000011101101110100000000

1011101001000011101101110100000000 → 1.011101001000011101101110100000000 * 2^33

Since double precision is 1023, we add our exponent to it: 1023 + 33 = 1056 → 10000100000

Store the sign bit → 0
store our exponent value with double precision → 010000100000
Store the mantissa → 0100001000001011101001000011101101110100000000
Add zeros till 64-bit → 0100001000001011101001000011101101110100000000000000000000000000

1.25e10 in binary is 0100001000001011101001000011101101110100000000000000000000000000
```

6. Calculate, by hand, the sum of $3.95 \times 10^8 + 7.13 \times 10^2$, assuming the two numbers are stored as IEEE 754 single precision. Assume 1 guard bit, 1 round, and 1 sticky bit, and round to the nearest even number. Show all the steps.

```
    3.95 * 10^8 → 395000000
    7.13 * 10^2 → 713

    395000000 in binary is → 10111100010110011100011000000
    713 in binary is → 1011001001

   10111100010110011100011000000
 + 00000000000000000001011001001
 ───────────────────────────────
   10111100010110011101110001001 → 1.0111100010110011101110001001 * 2^28

 Sign bit → 0
 Exponent → 127 + 28 → 155
 Mantissa → 01111000101100111011100

 3.95 * 10^8 + 7.13 * 10^2 = 395000713 → 01001101101111000101100111011100
```

7. Using the IEEE 754 float point single precision format, write down the pattern that would represent -0.4. Can this be represented exactly?

```
Sign Bit → 1
0 → 000000


But 0.4
0.8 ⇒ 0.4 * 2 ⇒ 0
1.6 ⇒ 0.8 * 2 ⇒ 1
1.2 ⇒ 0.6 * 2 ⇒ 1
0.4 ⇒ 0.2 * 2 ⇒ 0
0.8 ⇒ 0.4 * 2 ⇒ 0
…

It would get stuck in an infinite loop when trying to compute -0.4 because of
the list of numbers shown above. So at some point it would need to cut off and
the number represented would not be 100% accurate to the actual value.
```

8. The MIPS32 allows "double precision" values, but these require 64-bit registers. How did the engineers implement this on the MIPS?

   Double precision values were stored into pairs which means that when storing double values in MIPS, they would need to be stored in pairs like $f0 / $f2 / $f4 / $f6 or $f1 / $f3 / $f5 / $f7 because it takes up two 32-bit registers to store the number.

9. What is the "unit of least precision"?

   The number of bits in error in the least significant bits of the significand between the actual number and the number that can be represented

10. What does 0x4000 0000 represent if it's an IEEE 754 32-bit number?

    0×4000 000 → 0100 0000 0000 0000 0000 0000 0000 0000

    Read as an IEEE 754 value → 0 10000000 00000000000000000000000

    We can assume that all zeros == 1 since the number has to equal something other than zero.

    Since the maximum value for exponents is 128 for 32-bit, we store this at 2^1.

    This makes our number 1*2^1 or 2.

11. Why is the following C code incorrect:

```
float x = get_value( );
if (x == nan) {
  printf("ERROR\n");
```

    You cannot compare a number to a value that represents "Not a Number"

12. Define "guard digit"

```
Guard digit is the fist of two extra bits kept on the right during intermediate
calculations of floating-point numbers; They are used to improve rounding
accuracy.
```

13. Given the 32-bit hexadecimal value: 0x8e02_0024, show what this same 32-bit value as:

a) unsigned 32-bit integer

```
1000 1110 0000 0010 0000 0000 0010 0100 in decimal is 2382495780
```

b) signed 32-bit integer

```
1000 1110 0000 0010 0000 0000 0010 0100 gets bit flipped to become 0111 0001 1111 1101 1111 1111 1101 1011

  0111 0001 1111 1101 1111 1111 1101 1011
+ 0000 0000 0000 0000 0000 0000 0000 0001
—————————————————————————————————————————
  0111 0001 1111 1101 1111 1111 1101 1100


0111 0001 1111 1101 1111 1111 1101 1100 in decimal is -1912471516
```

c) MIPS instruction

```
Hex → 0×8e020024 becomes 1000 1110 0000 0010 0000 0000 0010 0100 in binary.
100011 → LW
10000 → $s0
00010 → $v0
0000000000100100 → 36

This becomes: LW $v0 36($s0)
```

d) IEEE 754 single precision

```
1000 1110 0000 0010 0000 0000 0010 0100 → 1 00011100 00000100000000000100100

Sign bit → 1
Exponent → 00011100 → 28 - 127 → -99
Mantissa → 00000100000000000100100 → 131108

00000100000000000100100 * 2^-99

2^(-6) + 2^(-18) + 2^(-21) → 0.0156292915

Multiply by 2^-99 → 1.0156292915 * 2^-99 → 1.60238048 * 10^-30

1000 1110 0000 0010 0000 0000 0010 0100 becomes 1.60238048 * 10^-30
```

14. Show the IEEE 754 encoding for:
    a. 0

       `0 00000000 00000000000000000000000`

    b. -0

       `1 00000000 00000000000000000000000`

    c. +inf

       `0 11111111 00000000000000000000000`

    d. -inf

       `1 11111111 00000000000000000000000`

    e. nan

       `1 11111111 11111111111111111111111`