**Q1. Convert the hexadecimal number 973D4 to a number with base 15**. **(2 points)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **a** | **b** | **c** | **d** | **e** | **f** |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |

**Step 1:**

Convert Hex to Dec = (4\*16^0) + (13\*16^1) + (3\*16^2) + (7\*16^3) + (9\*16^4) = 619476

**Step 2:**

Convert to base 15…

619476 / 15 = 41298 R = 6

41298 / 15 = 2753 R = 3

2753 / 15 = 183 R = 8

183 / 15 = 12. R = 3

12 / 15 = C

**Step 3:**

Answer = C3836

**Q2. Floating point numbers. (2 points)**

**You have to show the steps, otherwise you get zero**.

* Convert the following floating-point numbers to hexadecimal number in IEEE single-precision format. Please give the result as eight hexadecimal digits.
  1. -69/32 (-69 divide by 32) = -2.15625

Step 1) determine the sign and exponent

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | | 0 | 0 | A | 0 | 0 | 0 | 0 |
| 1 100 | | 0000 | 0000 | 1010 | 0000 | 0000 | 0000 | 0000 |
| 1 | 100 0000 0 | |  | | | | | |
| sign | exponet | | mantissa | | | | | |

2^1 = 2 \* 1.078125

127 + 1 = 128

Step 2) figure out the mantissa

.078125 \* 2 = .15625. // 0

.15625 \* 2 = .3125 // 0

.3125 \* 2 = .625. // 0

.625 \* 2 = .25 // 1

.25 \* 2 = .5 // 0

.5 \* 2 = 1

***ANSWER: Hex =* C00A 0000**

* 1. 13.625

Step 1) determine the sign and exponent // sign is 0 since not negative

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | | 1 | 5 | A | 0 | 0 | 0 | 0 |
| 0100 | | 0001 | 0101 | 1010 | 0000 | 0000 | 0000 | 0000 |
| 0 | 100 0001 0 | |  | | | | | |
| sign | exponet | | mantissa | | | | | |

2^3 = 8 \* 1.703125 = 13.625

127 + 3 = 130

Step 2) figure out the mantissa

.703125 \* 2 = .40625 // 1

.40625 \* 2 = .8125 // 0

.8125 \* 2 = .625 // 1

.625 \* 2 = .25 // 1

.25 \* 2 = .5 // 0

.5 \* 2 = 1

***ANSWER: Hex =* 415A 0000**

* Convert the following hexadecimal numbers in IEEE single-precision format to floating-point numbers:
  1. **42E48000**

Step 1) Convert to Binary:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | | 2 | E | 4 | 8 | 0 | 0 | 0 |
| 0100 | | 0010 | 1110 | 0100 | 1000 | 0000 | 0000 | 0000 |
| 0 | 100 0010 1 | | 110 0100 1000 0000 0000 0000 | | | | | |
| sign | exponet | | mantissa | | | | | |

Exponent = 128 + 4 + 1 = 133 2^(133-127) = 64

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 502 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|  |  | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

**Step 3)**

1 + (2^-1) + (2^-2) + (2^-5) + (2^-8) =

1 + (.5) + (.25) + (.03125) + (.00390625) = 1.78515625

**Step 4) *Answer***

64 \* 1.78515625 = 114.25

* 1. **C6F00040**

 Step 1) Convert to Binary:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | | 6 | F | 0 | 0 | 0 | 4 | 0 |
| 1 100 | | 0110 | 1111 | 0000 | 0000 | 0000 | 0100 | 0000 |
| 1 | 100 0110 1 | | 111 0000 0000 0000 0100 0000 | | | | | |
| sign | exponent | | mantissa | | | | | |

Exponent = 128 + 8 + 4 + 2+ 1 = 141 2^(141-127) = 2^14 = 16384

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 502 | 256 | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

**Step 3)**

1 + (2^-1) + (2^-2) + (2^-3) + (2^-17) =

1 + (.5) + (.25) + (.125) + (.000007629394531) = 1.875007629394531

**Step 4) *Answer***

-1 \* (16384 \* 1.875007629394531) = -30720.125

**Q3. Error Finding (4 points)**

Each of the following 68K assembly language instructions will cause an assembler error. Examine each instruction and explain why the assembler would flag it as an error.

1) MOVE.B      $A000, A3

**ANSWER:** The error with this code is that your trying to move a byte when the actual value is larger than a byte. By changing the .B -> . L, you get code that compiles (MOVE.L).

2) ADD.B         #$1000, D2

**ANSWER:** The error with this code is that your trying to add a byte when the actual value is larger than a byte. By changing the .B -> . L, you get code that compiles (ADD.L).

3) MOVEA.W    $1234, D0

**ANSWER:** The error with this code is an invalid addressing mode. By using MOVEA, you’re moving to an address, but the address is a direct register. By changing the D0 to A#, you can avoid this error.

4) ANDI.B         #23, #$100

**ANSWER:** The issue with this code is that the second number #$100, needs to be a direct register address, D#. By doing this, you can then be able to do this calculation (ANDI.B #23, D0).

**Q4.** **Create a source file and analyze the results. (4 points)**

**Submit the .X68 and .L68 files and answer the question. For simplicity, name your source file as HW1Q4.X68 and your listing file as HW1Q4.L68.**

**QUESTION**: What is the **WORD VALUE** (not byte, or longword) of the data in memory location $4000, when the program is just about to loop back to the place where "start" is labelled? **Please describe how you got the answer as well**. (For example, you can describe how you analyzed the code segments, or how you traced the code segments with debug tools)

***Answer Below:***

It looks like the data value at $4000 is 45 15. To get to this answer, I typed up the code below, then stepped through the program prior to the ‘jmp start’ statement and then checked the memory in the EASy68K simulator.

**Q5. Two’s complement (7 points)**

Assume that we are using a**16-bit system**. Represent a negative integer with two’s complement format.

1. (2 pts) Convert the decimal numbers -102 and -87 into hexadecimal number.

-102 = FF9A

- 87 = FFA9

1. (1 pt) Add two numbers of the previous question as hexadecimal, and state

1111

FF9A

+FFA9

1FF43

* 1. whether the sign bit of the result is 1, and

Sign is 1 since ‘F’ is 1111 in binary.

* 1. whether an overflow occurred.

Overflow did occur since the answer has more than 16 bits.

1. (4 pts) Write a program in assembly language to add the two numbers (-102 and -87). Inputs should be in decimal format. Store the result as hexadecimal numbers at address $6000. Print out the result in command output window in ***decimal***format. (Hint: use the trap function task #3). If an error (overflow) happens, you should also print out the error (overflow) message as well.

**Negative Case:**A screenshot of a cell phone

Description automatically generated

**Positive Case**

**A screenshot of a cell phone

Description automatically generated**

**Overflow Case:**

**A screenshot of a cell phone

Description automatically generated**

**Memory:**

****

**You are not allowed to hardcode the two input numbers!**

**You should design your program to accept any two numbers as inputs!**

**You may ignore sanity check on corner cases, but assume user inputs are always in an expected format.**

**You may need to design your program to convert ASCII characters into integer numbers! Think about how to convert string "-102" into integer -102 in JAVA or C++!**

**Copy and paste the screen shot of the output(command) window into your submission file. Do not submit an individual file for the image!**

**Copy and paste the screen shot of your memory system at address $6000 into your submission file. Do not submit an individual file for the image!**

**Submit the source file (.X68) and the listing file(.L68) individually. Do not attach the code into your submission file!** **For simplicity, name your source file as HW1Q5.X68 and your listing file as HW1Q5.L68.**