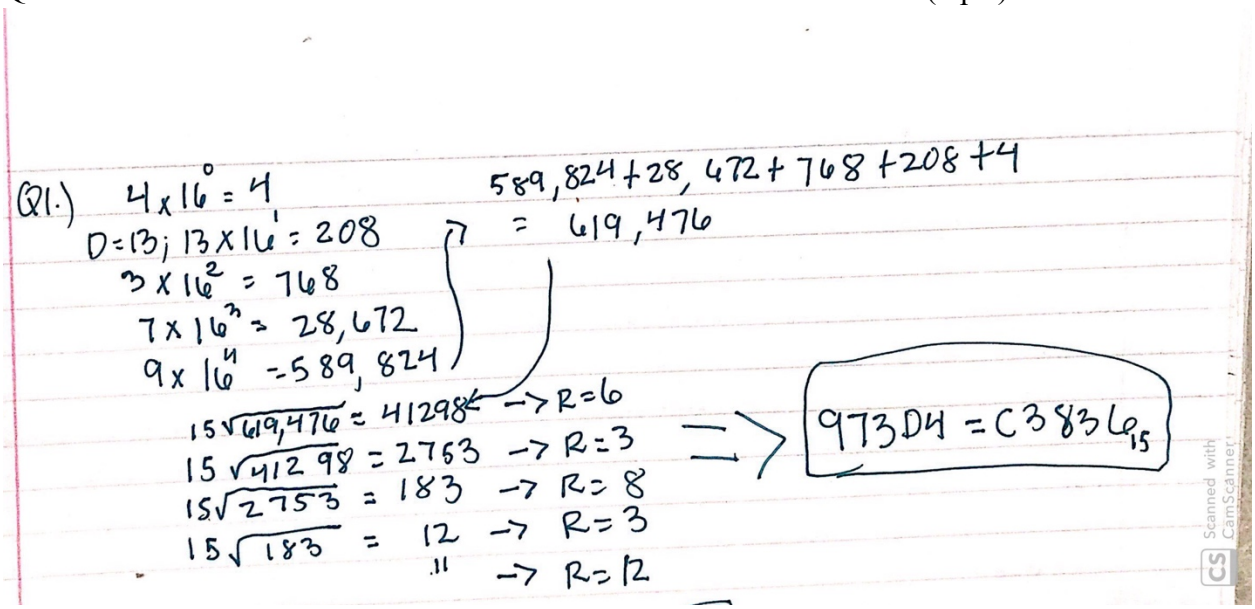
Regie Pineda
Dr. Amed Awad
CSS422
October 16th, 2019

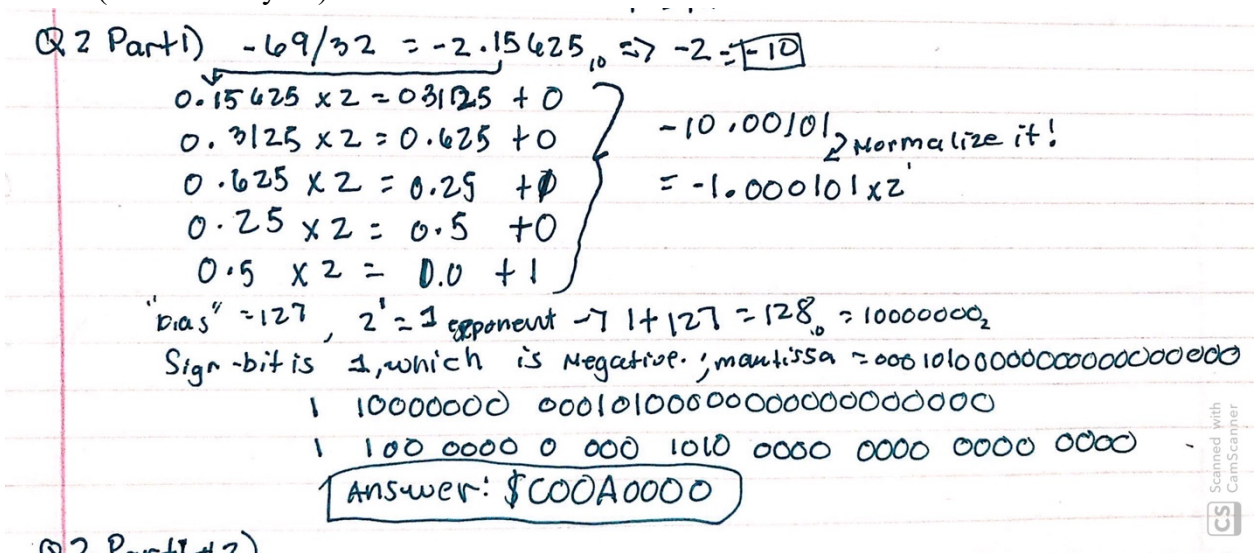
Homework 1

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



Q2. Floating point numbers. (2 points)

- 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. 13.625

Q2 Part I #2)

Convert 13.625 ; 13 = 1101

0.625 x2 = 0.250 + 1 0.5 x2 = 0.0 +1

0.25x2 = 0.5 +0 => 1101.101 = 1.10110 | x 2^3

Sign bit is 0, which is positive ; exponent = 3, so 127 +3 = 130_{10} = 10000010_2

= 0 10000010 10110100000000000000

Answer: $ 415A 0000

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

Q2 Part 2 #1

Convert 42E48000 to floating point number:

0 100 0010 | 110 0100 | 000 0000 0000 0000

exponent mantissa

133-127 = 6 => exponent = 6

1.1100100 | x 2^6 = 1110010.01

$2^1 + 2^4 + 2^5 + 2^6$ = 114

0.01 ; 1/2 = .5

0+0.5 ÷2 = 0.25

Answer: 114.25

  2. C6F00040

Q2Part2#2

Convert $C6F00040 to floating point.

1 100 0110 | 111 0000 0000 0000 0100 0000

exp. mantissa

141 -127 = 14 ; exponent = 14

1.111 0000 0000 0000 0100 x 2^{14} .

11110000 0000 0000.01

$2^{15} + 2^{14} + 2^{13} + 2^{12}$ = -30720_{10}

0.01 ; 1 ÷2 ÷0.5 ; 0.5+0 ÷2=.25

(0.25 to) ÷2 = 0.125

= Answer: -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

- The assembly language instructions will cause an assembler error for two different reasons. First, the size of the operation is wrong. It's better to use MOVEA since that opcode limits the size to Word and Long Integer sizes. The minimum size for this intended operation should have been at least the size of a Word with it affected by sign extension. Also, MOVEA opcode should've been chosen for this because it involves moving or copying data to an address register. It should be noted that Easy68K allowed the use of a normal MOVE opcode.

2) ADD.B #$1000, D2

- The assembly language instructions will cause an assembler error for two different reasons. First, the value that is being added is #$1000 that is represented by 4 hexadecimal digits which is equivalent to 2 bytes. ADD.B will only account for a single byte, not the value of #$1000. This is a size of a word so the operation should be ADDI.W or ADDI.L. Also, ADD opcode itself requires two effective addresses. Since the goal was to add an immediate value, the ADDI opcode should be used instead.

3) MOVEA.W $1234, D0

- The assembly language instructions will cause an assembler error because it is copying data to a data register rather than an address register. The operation that would work is using MOVE.W

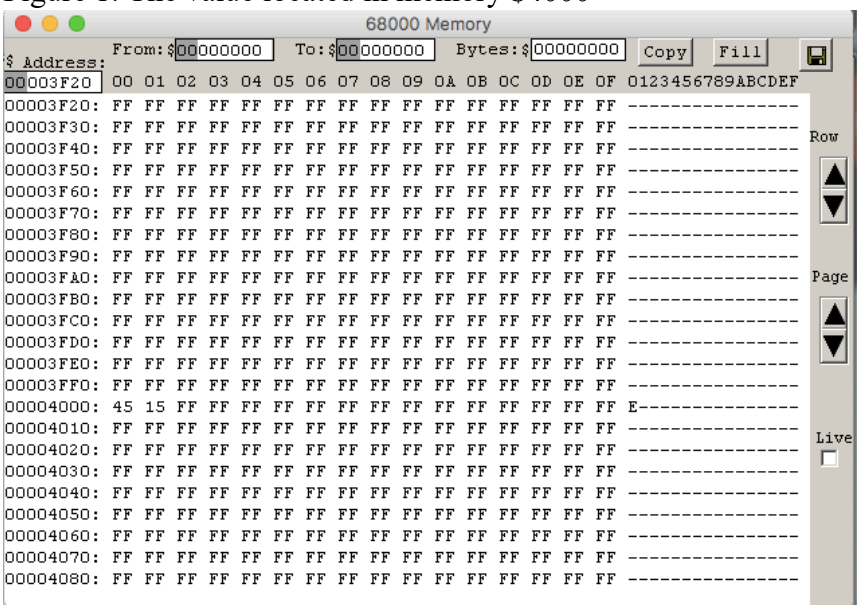
4) ANDI.B #23, #$100

- The assembly language instructions will cause an assembler error because the destination should not be an immediate data. I say this because the destination operand in this operation is a value when proper usage requires it to be either data register, indirect address register, and memory address.

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

- The .X68 and .L68 files are submitted separately.
- *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)*
  - The word value of the data in memory location $4000, when the program is just about to loop back to the place where "start" is labelled is $4515 in hexadecimal. I got this answer by doing a trace as it was executed. I was able to see the value in memory $4000 by clicking on View then viewing the memory. Please see Figure 1 to confirm the word value at memory location $4000.

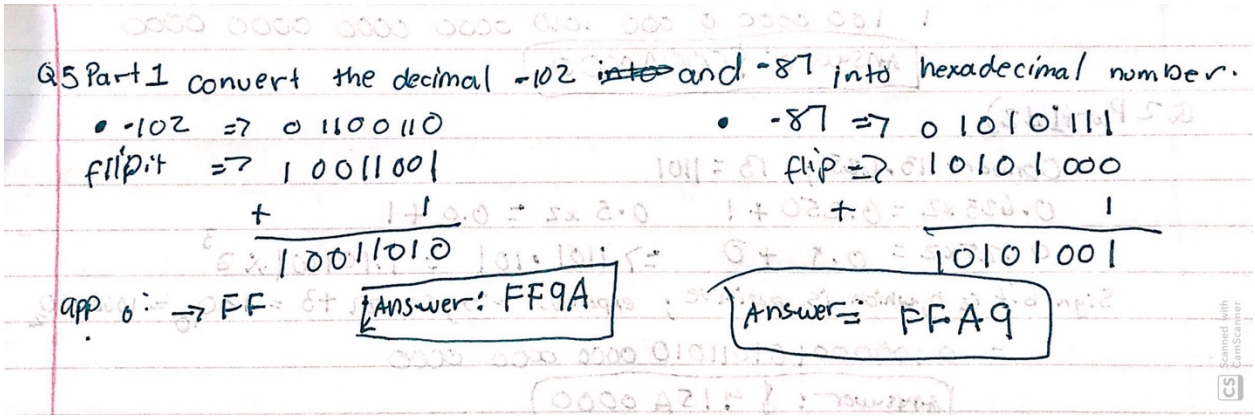
Figure 1: The value located in memory $4000



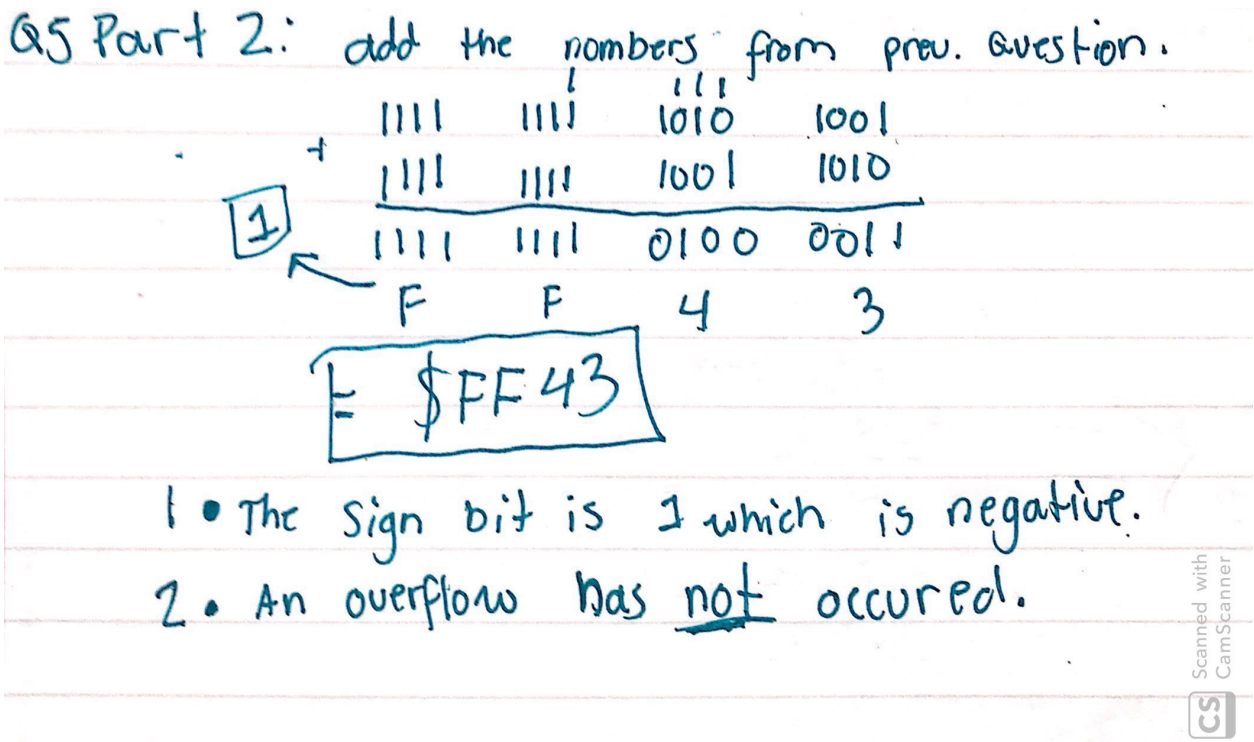
Q5. Two's complement (7 points)

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.



2. (1 pt) Add two numbers of the previous question as hexadecimal, and state
    1. whether the sign bit of the result is 1, and
    2. whether an overflow occurred.



3. (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.

    - The .X68 and .L68 files are submitted separately.

Figure 2 shows the output result and Figure 3 shows the answer that is stored in memory system at address $6000.

Figure 2: Output of the Program

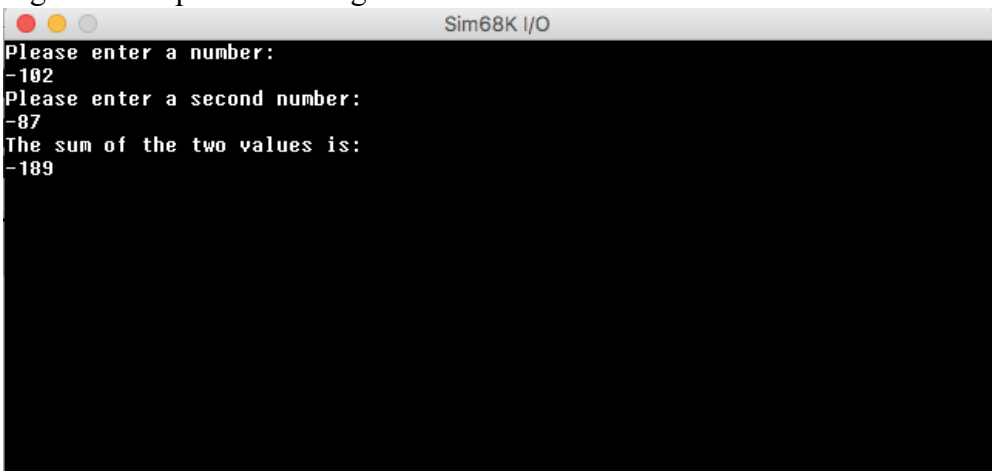


Figure 3: Answer located at memory address $6000