

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)

Q1.) $4 \times 16^0 = 4$
 $D = 13; 13 \times 16^1 = 208$
 $3 \times 16^2 = 768$
 $7 \times 16^3 = 28,672$
 $9 \times 16^4 = 589,824$

$589,824 + 28,672 + 768 + 208 + 4$
 $= 619,476$

$15 \sqrt{619,476} = 41298 \rightarrow R=6$
 $15 \sqrt{41298} = 2763 \rightarrow R=3$
 $15 \sqrt{2763} = 183 \rightarrow R=8$
 $15 \sqrt{183} = 12 \rightarrow R=3$
 $" \rightarrow R=12$

$\Rightarrow 973D4 = C3836_{15}$

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)

Q2 Part 1) $-69/32 = -2.15625_{10} \Rightarrow -2.15625_{10}$

$0.15625 \times 2 = 0.3125 + 0$

$0.3125 \times 2 = 0.625 + 0$

$0.625 \times 2 = 1.25 + 0$

$0.25 \times 2 = 0.5 + 0$

$0.5 \times 2 = 1.0 + 1$

-10.00101_2 Normalize it!

$= -1.000101 \times 2^1$

"bias" = 127, $2^1 = 1$ exponent $\rightarrow 1 + 127 = 128_{10} = 10000000_2$

Sign-bit is 1, which is Negative; mantissa = 000101000000000000000000

1 10000000 000101000000000000000000

1 100 0000 0 000 1010 0000 0000 0000 0000

Answer: $\$C00A0000$

2. 13.625

Q2 Part 1 #2

Convert 13.625; 13 = 1101

$$0.625 \times 2 = 0.250 + 1 \quad 0.5 \times 2 = 0.0 + 1$$

$$0.25 \times 2 = 0.5 + 0 \quad \Rightarrow 1101.101 = 1.101101 \times 2^3$$

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

$$= 0100000101011010000000000000$$

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.

$$0100001011000100000000000000$$

exponent mantissa

$$133 - 127 = 6 \Rightarrow \text{exponent} = 6$$

$$1.11061001 \times 2^6 = 1110610.51$$

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

$$0.01; 1/2 = 0.5$$

$$0 + 0.5 \div 2 = 0.25$$

Answer: 114.25

2. C6F00040

Q2 Part 2 #2

Convert C6F00040 to floating point.

$$11000110110000000000000001000000$$

exp. mantissa

$$141 - 127 = 14, \text{ exponent} = 14$$

$$1.1110000000000001 \times 2^{14}$$

$$111000000000000.01$$

$$2^{11} + 2^{12} + 2^{13} + 2^{14} = -30720_{10}$$

$$0.01; 1 \div 2 = 0.5; 0.5 + 0 \div 2 = 0.25$$

$$(0.25 + 0) \div 2 = 0.125$$

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

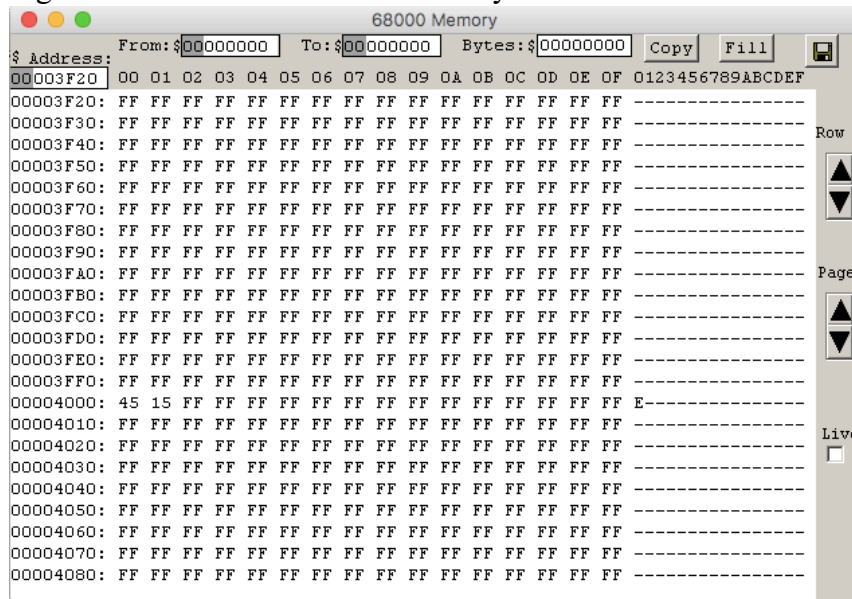
4) ANDI.B #23, #\$100

- 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.

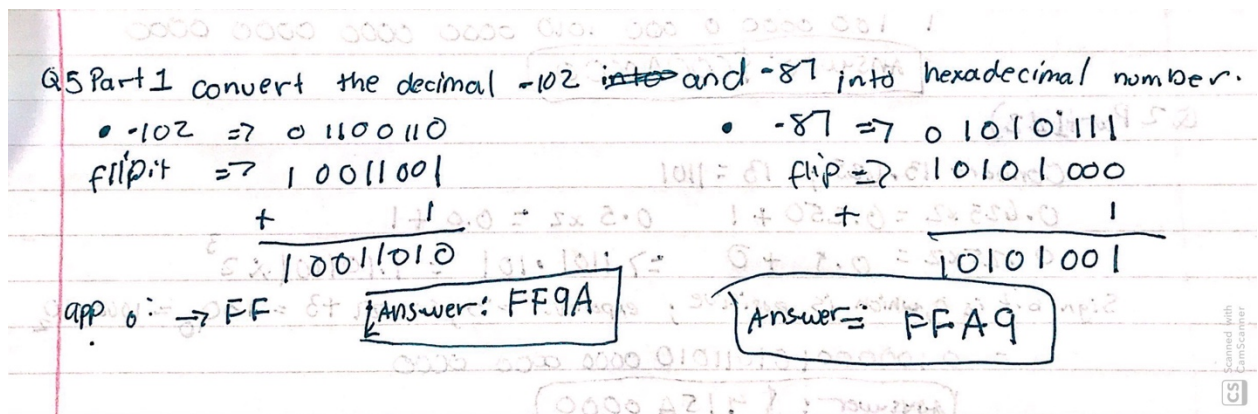
Figure 1: The value located in memory \$4000



Q5. Two's complement (7 points)

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

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



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

Q5 Part 2: add the numbers from prev. question.

$$\begin{array}{cccc}
 & 1111 & 1111 & 1010 & 1001 \\
 + & 1111 & 1111 & 1001 & 1010 \\
 \hline
 1 & 1111 & 1111 & 0100 & 0011 \\
 & F & F & 4 & 3 \\
 \hline
 & \boxed{F} & \boxed{F} & \boxed{4} & \boxed{3}
 \end{array}$$

1. The sign bit is 1 which is negative.
2. An overflow has not 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.

- o 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

```

Sim68K I/O
Please enter a number:
-102
Please enter a second number:
-87
The sum of the two values is:
-189

```

Figure 3: Answer located at memory address \$6000

Registers

D0=00000003	D4=00000000	A0=00000000	A4=00000000	T S INT XNZVC	Cycles
D1=FFFFFF43	D5=00000000	A1=00001089	A5=00000000	SR=00100000000010000	124
D2=0000FF9A	D6=00000000	A2=00000000	A6=00000000	US=00FF0000	Clear Cycles
D3=00000000	D7=00000000	A3=00000000	A7=01000000	SS=01000000	PC=00001048

68000 Memory

Address: From: \$00000000 To: \$00000000 Bytes: \$00000000 Copy Fill

Address	Code	Hex	Assembly	Comment
00001010	4E4F	00005F30	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	0123456789ABCDEF
00001012		00005F30	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
00001012	3401	00005F40	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
00001014		00005F50	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
00001014	43F9 0000106B	00005F60	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
0000101A	303C 000D	00005F70	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
0000101E	4E4F	00005F80	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
00001020		00005F90	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
00001020	303C 0004	00005FA0	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
00001024	4E4F	00005FB0	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
00001026		00005FC0	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
00001026	B242	00005FD0	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
00001028	691E	00005FE0	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
0000102A	D242	00005FF0	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
0000102C	691A	00006000	FF 43 FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-C-----
0000102E		00006010	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
0000102E	31C1 6000	00006020	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
00001032		00006030	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
00001032	43F9 00001089	00006040	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
00001038	303C 000D	00006050	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
0000103C	4E4F	00006060	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
0000103E		00006070	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
0000103E	103C 0003	00006080	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
00001042	4E4F	00006090	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF	-----
00001044				
00001044	FFFF FFFF		43	SIMHALT
00001048			44	
00001048	43F9 000010A7		45	OVERFLOW LEA OVERFLOWMSG,A1 Alerts the user about