Bisrat Asefaw
CSS 422 (Hardware)
Prof. Yang

Homework #1

Q1. Convert the hexadecimal number 973D0 to a number with base 15

Hex→ Dec → base-15

- $9 \times 16^4 + 7 \times 16^3 + 3 \times 16^2 + 14 \times 16^1 + 0 \times 16^0$
  → $(619488)_{10}$

  15→619472→2
  15→41298→3
  15→2753→8
  15→183→3
  15→12→12
  15→0

  Ans. C3832 →$_{\text{base 15}}$

Q2. Floating point numbers.

I. Convert the following floating-point numbers to hexadecimal number in IEEE single-precision format. Please give the result as eight hexadecimal digits.

A. -69/32 = - 2.15625 → -10.00101

Step 1: Change to binary

Integer part: 10
2 → 2 → 0
2→1→1
2→0

Fraction part: 00101
0.15625 x 2 →0
0.3125 x 2 →0
0.625 x 2 →1
0.25 x 2 → 0
0.5 x 2 → 1

Step 2: Normalize -1.000101×2^1
Sign bit: 1
Exponent: 127 + 1 = 128 = 10000000
Mantissa: 010 1000 0000 0000 0000 0000
Therefore, real number in floating point representation is:
1 100 0000 0 000 1010 0000 0000 0000 0000

Sign(1)	Exponent(8)	Mantissa (23)
1	100 0000 0	000 1010 0000 0000 0000 0000

The Hex representation is by grouping, binary digits into 4 bits
Ans. C00A0000

B. 13.625

- Change to binary
  - **Integer**
  - 2→13→1
  - 2→6→0
  - 2→3→1
  - 2→1 →1
  - 2→0

- For fraction
  - 0.625 x 2 = 1
  - 0.25x2 = 0
  - 0.5 x 2 = 1

1101.101
1.101101 x 2^3 => mantissa => 101101
127+3=130 => binary representation = > 10000010
2→130→0
2→65→ 1
2→32→0
2→16→0
2→8→0
2→4→0
2→2→0
2→1→1
2→0

Sign(1)	Exponent(8)	Mantissa (23)
0	100 0001 0	101 1010 0000 0000 0000 0000

floating point representation:
0100 0001 0101 1010 0000 0000 0000 0000 → using grouping to hex → $ 415A0000

Ans. 0x 415A0000

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

A. 42E48000

- Hex → Binary → change from hex to binary by directly representing numbers in their binary representation
  - 0100 0010 1110 0100 1000 0000 0000 0000

Floating point representation

Sign (1)	Exponent (8)	Mantissa (23)
0	100 0010 1	110 0100 1000 0000 0000 0000

- 0 100 0010 1 110 0100 1000 0000 0000 0000

10000101 => 133-127= 6 ➔ 2^6 ➔ 1110010.01000000000000000 ➔ $(114.25)_{10}$

Ans. 114.25

B. C6F00040

- Hex → Binary → change from hex to binary by directly representing numbers in their binary representation
  - 1100 0110 1111 0000 0000 0000 0100 0000

Floating point representation

Sign(1)	Exponent(8)	Mantissa (23)
1	10001101	1110000 00000000 01000000

Exponent ➔ 10001101= Decimal representation = 141 – 127 = 14 ➔ 2^{14}

Number after shifting 14 digits ➔ 1. 1110000 00000000 01000000
Number before shifting 14 digits ➔ 111000000000000.001000000
Sign ➔ negative (- ve)
Binary representation of the number ➔ - 111100000000000.001000000
Decimal representation ➔ $1 \times 2^{14}+1 \times 2^{13}+1 \times 2^{12}+ 1 \times 2^{11} = 30720$
Fraction representation ➔$1 \times 2^{-3} = 0.125$

Floating point representation: -30720.125

Q3. Error Finding

1) MOVE.B $A000, A3

From manual MOVE.B is size limited to 8-bit

- The MOVE.B is size limited to 8-bit, which the hexadecimal value $A000(16-bit) is more than 8-bit. Also, MOVEA or MOVE.L opcode should've been chosen for this because it involves moving or copying data to an address register. This flags out an error because the immediate data excides size limit.

2) ADD.B   #$1000, D2

From manual ADD.B is size limited to 8-bit

Syntax: **ADD < ea > ,Dn**
ADD Dn, < ea >

- From manual the data size limit for ADD.B is an 8-bit, adding the hex value #$1000 (13 bit) to data register D2 cause to flag an error because the size limit excides 8-bit, as it excides 8-bit size. ADD adds the source operand to the destination operand using binary addition and stores the result in the destination location, in this instruction destination is D2.

3) MOVEA.W $1234, D0

From the manual the syntax should be **MOVEA.W <ea>, An**

- Clearly There is an error in the syntax for coping an hexadecimal address value from source to destination ($1234 to D0) . MOVEA moves the contents of the source to the destination address register. This is error and can be fixed by changing D0 to address mode (An).

4) ANDI.B   #23, #$100

  - From the easy68K manual   **ANDI # < data >, < ea >**

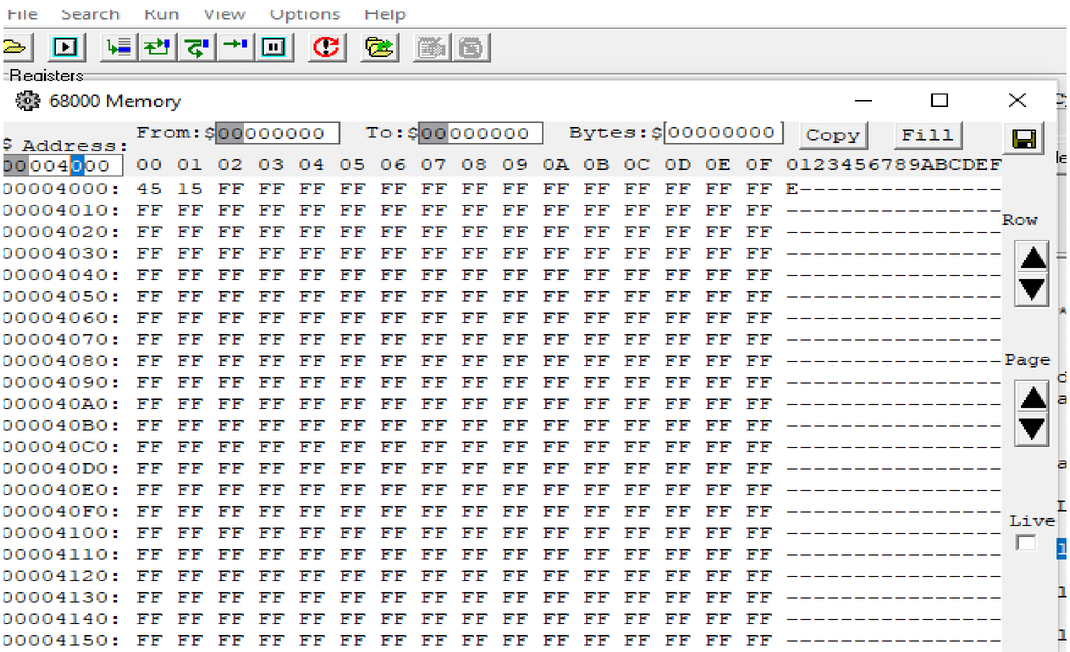
- ANDI (AND immediate) → Adds a value to an effective addressing mode, instead in this operation the value #23 is adding to hexadecimal value #$100. Therefore, this syntax is an error. To correct this error the #$100 can be replace by effective addressing (Dn).

Q4. Create a source file and analyze the results.

- **HW1Q4.X68 and HW1Q4.L68 files submitted**

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 $45 15 in hexadecimal. I got this answer by doing a trace as it was executed. I use both “auto trace” and “step into” for tracing, and I analyze the memory value change at address $4000 by looking into the memory view window.

Fig. 1 value saved at memory $6000



Q5. Two's complement

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

1. Convert the decimal numbers -102 and -87 into hexadecimal number.

   Step 1: Convert to Bin.

   2→102→0
   2→51→1
   2→25→1
   2→12→0
   2→6→0
   2→3→1
   2→1→1
   2→0

102→0000000001100110
Step 2: flip 0's to 1's and vice versa 1111111110011001
Step 3, add 1: 1111111110011001 + 1 = 1111111110011010
Step 4 convert to hex: **<u>FF9A</u>**

For -87
Step 1: change to bin → 0000000001010111
Step 2: flip numbers → 1111111110101000
Step 3: add 1→1111111110101000 +1 = 1111111110101001
Step 4: change to hex: **<u>FF A9</u>**

2. Add the two numbers from above

FF9A + FFA9 = 1| **FF43** **N=1, Z=V=0, C=1**

<u>Answer = FF43</u>

1111111110011010
<u>1111111110101001</u>
1|1111111101000011

Ans: FF43

- **The sign bit of the result is 1, which is the result is a negative number.**
- **Adding these two negative number doesn't result in a positive number, therefore no overflow.**

3.

- **HW1Q5.X68 and HW1Q5.L68 files submitted**

Fig.2 the output result of two value (-102 and -87)

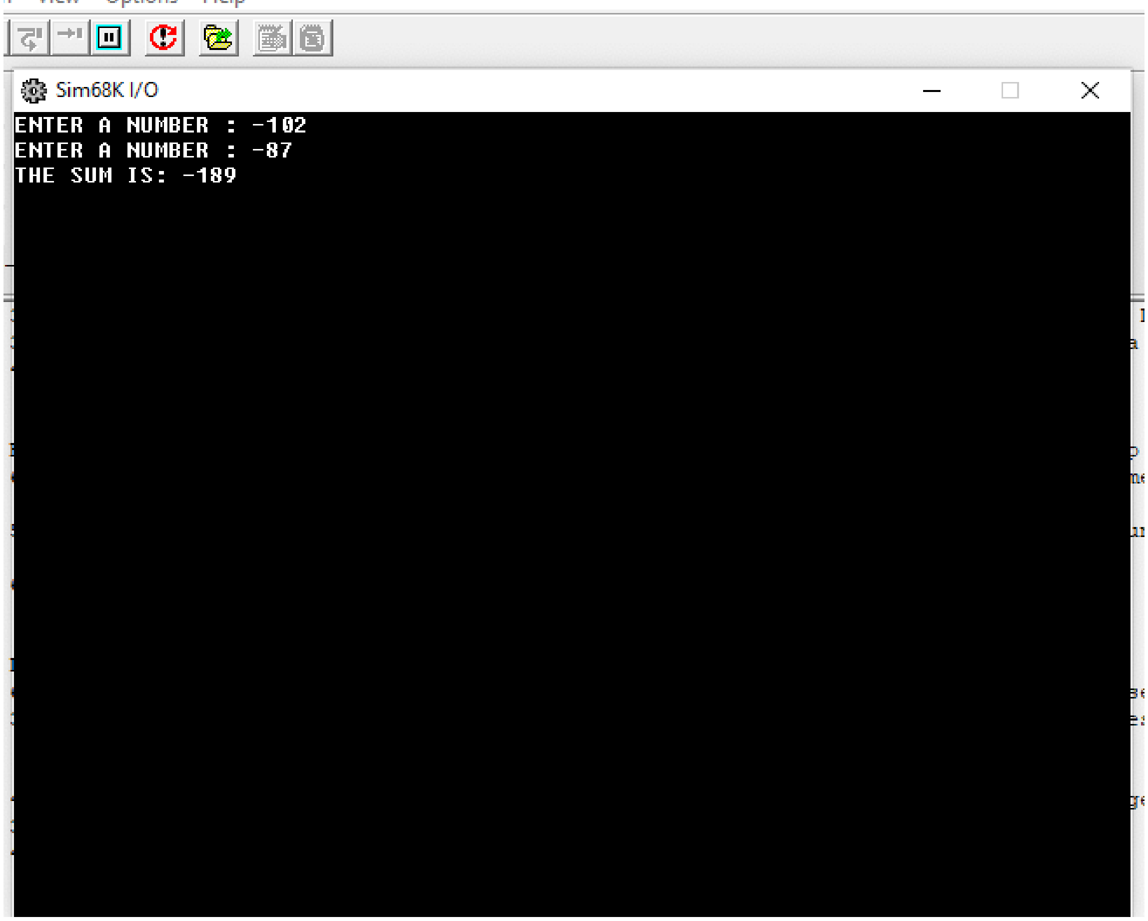
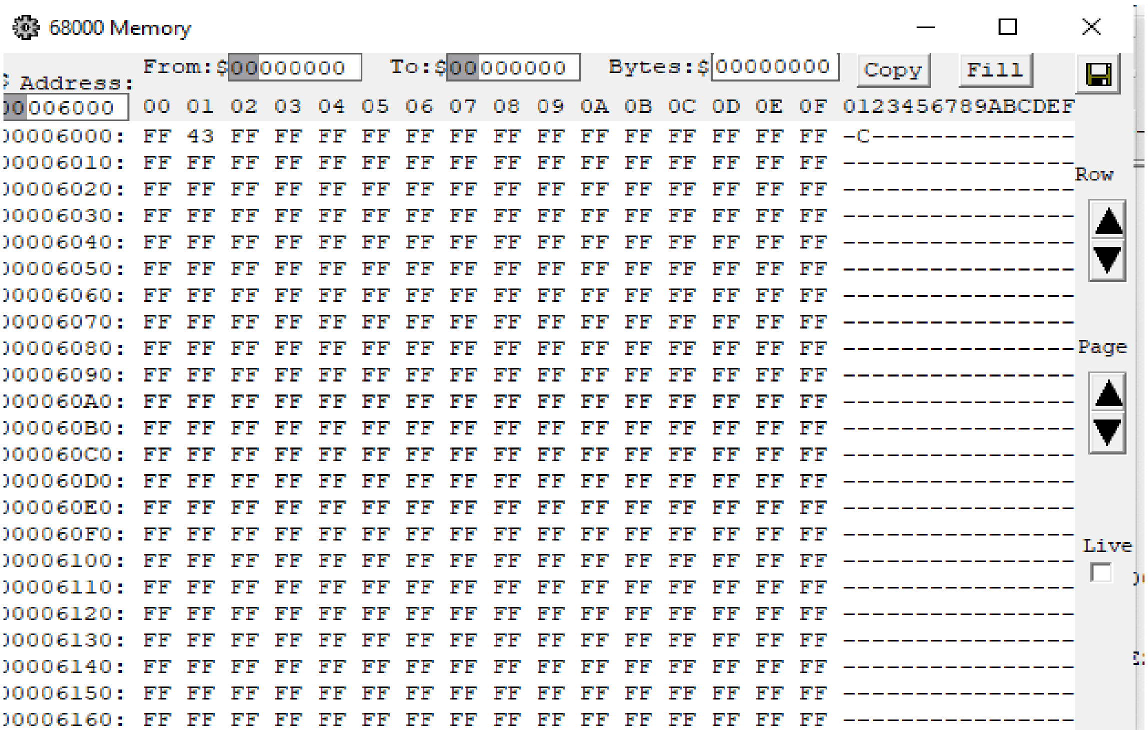


Fig 3. Value stored at memory address $6000


68000 Memory
Address:
From:$00000000
To:$00000000
Bytes:$00000000
Copy
Fill
00006000
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF
00006000: FF 43 FF FF FF FF FF FF FF FF FF FF FF FF FF FF -C--------------
00006010: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00006020: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00006030: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00006040: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00006050: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00006060: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00006070: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00006080: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00006090: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000060A0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000060B0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000060C0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000060D0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000060E0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
000060F0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00006100: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00006110: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00006120: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00006130: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00006140: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00006150: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00006160: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
Row
Page
Live