Q1. Assemble the codes. (2 points)

Convert the following 68K assembly language instructions to the machine codes.

1) MOVE.W D1, $0000A000 **ANSWER = 33C1 0000A000** **[PAGE 220]**

00		SIZE		DES. REGISTER			DES.MODE			SOU.MODE			SOU.REGISTER		
0	0	1	1	0	0	1	1	1	1	0	0	0	0	0	1
3				**3**				**C**				**1**			

2) MOVE.B $42A7, (A1)+ **ANSWER = 12F8 42A7** **[PAGE 220]**

00		SIZE		DES. REGISTER			DES.MODE			SOU.MODE			SOU.REGISTER		
0	0	0	1	0	0	1	0	1	1	1	1	1	0	0	0
1				**2**				**F**				**8**			

3) ADD.L D7, D0 **ANSWER = D087** **[PAGE 108]**

11		01		REGISTER			OPMODE			EA.MODE			EA.REGISTER		
1	1	0	1	0	0	0	0	1	0	0	0	0	1	1	1
D				**0**				**8**				**7**			

4) MOVEA.L D3, A0 **ANSWER = 2043** **[PAGE 223]**

00		SIZE		DES. REGISTER			001			SOU.MODE			SOU.REGISTER		
0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	1
2				**0**				**4**				**3**			

Q2. Floating point numbers. (2 points)

WORK SHOWN ON NEXT PAGE. PRETTY BAD HANDWRITING

A.) Convert the following decimal numbers in IEEE single-precision format. Give the result as eight hexadecimal digits.

1.) -69/32 (-69 divide by 32) - > **C20A0000**

2.) 13.625 **-> 415A0000**

B.) Convert the following floating IEEE single-precision floating-point numbers from hex to decimal:

1.) 42E48000 -> **114.0**

2.) C6F00040 - > **-30,720.125**

422 | HW2 | Question 2 | Tyler Quayle

2)

A) Decimal → Single Precision → Answer in hex

1) -69/32

- 34.5 → 0010 0010 (34) $.5 \cdot 2 = 1$

0010 0010.1 → $1.000101 \cdot 2^5$

5+127 = 132 → 1000 0100

1 1 0 0 0 0 1 0 0 . 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

-69/32 → C20A0000

2) 13.625

13 = 1101 $.625 \cdot 2 = .25 + 1$ = 101
$.25 \cdot 2 = .5 + 0$
$.5 \cdot 2 = .0 + 1$

1101.101 → $1.101101 \cdot 2^3$

3+127 = 130 → 1000 0010

0 1 0 0 0 0 0 1 0 . 1 0 1 1 0 1 0 ... 17 0's ... 0

13.625 = 415A0000

B)

1) 42E40000

0 1 0 0 0 0 1 0 1 . 1 1 0 0 1 0 0 ... 17 0's ... 0

+ 133
- 127
exp 6 → 1110010.0 → 114.0

2) C6F00040

1 1 0 0 0 1 1 0 1 . 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0

- 141
- 127
exp 14 → 111100000000000.001

-30,720.125

Q3. Bit Shift (2 points)

Assume that D0 contains the value $C1A8E392. For each of the operations below, evaluate the value in D0 and the state of the CCR. Assume that initially XNZVC=00000. (2 points)

$C1A8E392

XNZVC 0 0 0 0 0

1	1	0	0	0	0	0	1	1	0	1	0	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	1	0	0	1	0
C				1				A				8				E				3				9				2			

1.) ASL.B #2, D0

1	0	0	1	0	0	1	0
9				2			

0	1	0	0	1	0	0	0
4				8			

$C1A8E348

XNZVC 1 0 0 0 1

1	1	0	0	0	0	0	1	1	0	1	0	1	0	0	0	1	1	1	0	0	0	1	1	0	1	0	0	1	0	0	0
C				1				A				8				E				3				4				8			

2.) ASL.L #5, D0. Longword, shifts all bits. ASL adds 0's into LSB. $C1A8E345 >351C6900

1	1	0	0	0	0	0	1	1	0	1	0	1	0	0	0	1	1	1	0	0	0	1	1	0	1	0	0	1	0	0	0
C				1				A				8				E				3				4				8			

$351C6900

XNZVC 1 0 0 0 1

0	0	1	1	0	1	0	1	0	0	0	1	1	1	0	0	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0
3				5				1				C				6				9				0				0			

3.) LSR.B #4, D0. No change, D0.B is 00, LSR adds 0's into MSB slot. $351C6900->$351C6900

0	0	0	0	0	0	0	0
0				0			

0	0	0	0	0	0	0	0
0				0			

$351C6900

XNZVC 0 0 0 0 0

0	0	1	1	0	1	0	1	0	0	0	1	1	1	0	0	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0
3				5				1				C				6				9				0				0			

4.) ROR.W #2, D0. 351C6900 -> 351C1A40. No 1's were rotated out. No change to XNZVC

0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0
6				9				0				0			

0	0	0	1	1	0	1	0	0	1	0	0	0	0	0	0
1				A				4				0			

$351C1A40

XNZVC 0 0 0 0 0

0	0	1	1	0	1	0	1	0	0	0	1	1	1	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0	0	0	0
3				5				1				C				1				A				4				0			

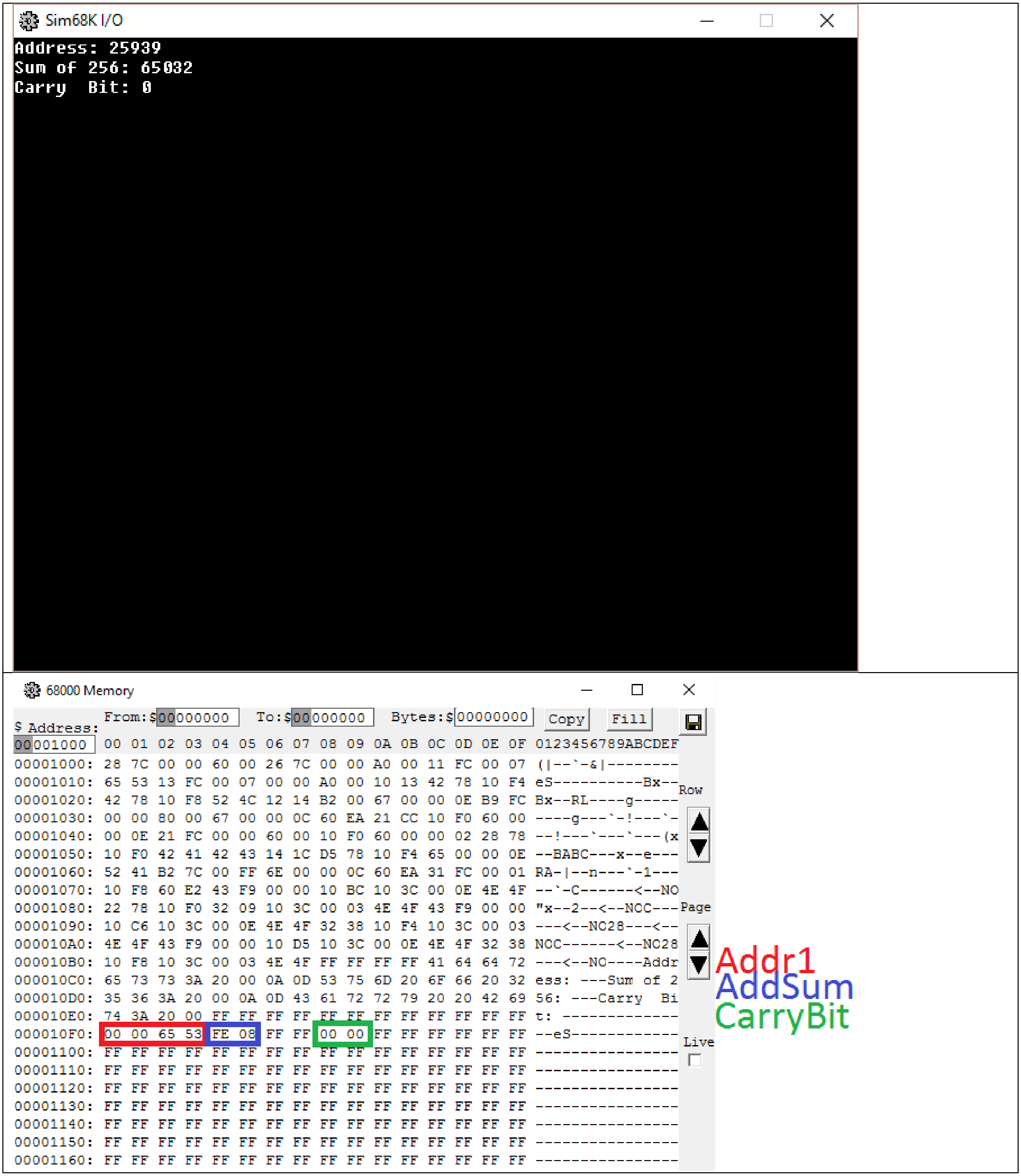
Q4. Pattern Finding and Cumulative program. (5 points)

Write a program in 68K assembly code that satisfy the following specifications:

1. Your program should start at the memory location $1000. Read each **byte data** stored in memory between the addresses $6000 and $8000 and compare it to the (byte) data at address $A000.
2. If it finds the data matching data at address $A000, it stores the **longword address of the data** in memory. Please define a variable called address **Addr1** to save the data. If it fails to find the data within the specified memory range ($6000 and $8000), then put Addr1 = $6000, which is similar to the idea of "error code" or "invalid index" in your C++ code.
3. Then, add a series of **bytes** stored in **256** consecutive memory locations beginning at address Addr1, no matter you find the target data or not.
4. Store the **sum** into memory as a **WORD** variable called **Addsum**. There is a chance that the sum might exceed $FFFF, (exceeding the range of word value), so you will also need to store the carry bit if an overflow occurs. Store the **carry bit as a BYTE** variable called **CarryBit**.
5. Print the **Addr1**, **Addsum** and **CarryBit** in the output window

```
*-----------------------------------------------------------
*-----------------------------------------------------------
* Title      : Pattern Finding and Cumulative Program
* Written by : Tyler Quayle
* Date       : April 21, 2016
* Description: Question 4, in HW2. Find byte in memory, then find SUM of 256 bytes
*-----------------------------------------------------------
* FOLLOWING TWO VARIABLES ARE USED TO HARD CODE THE BYTE TO FIND,
* OTHERWISE WOULD 'SUCCEED', ON FIRST COMPARE SINCE ALL MEMORY DEFAULTS
* TO 'FF'
NumTo   EQU $07      * Number hardcoded to find
NumLoc  EQU $6553    * Address between $6k-$8k, where variable is 'Hidden'
************************************************************
CompA       EQU $A000    * The memory location that is being compared against
StarA       EQU $6000    * Starting Address to search From
EndA        EQU $8000    * End Address to search thru

Addr1       EQU $10F0    * Defined in HW2, used to track the 'successful' find of matching data
Addsum      EQU $10F4    * Defined in HW2, Used to sum 256 consecutive memory addresses
CarryBit    EQU $10F8      * Defined in HW2, Stores carrybit from ADD Addsum

    ORG    $1000
START:
        MOVEA.L     #$6000, A4      * Assign address $6000 (start) to A4
        MOVEA.L     #CompA, A3      * Assign Address $A300 (compare) to A3
        MOVE.B      #NumTo, NumLoc  * Hardcode number for loop to find
        MOVE.B      #NumTo, CompA   * Hardcode number that loop to find to A300 (A4/CompA)
        MOVE.B      (A3),D0         * Give D0 the byte data in A3
        CLR.W       Addsum          * Set 4 Bytes to Zero at Addsum address. Otherwise starts summing
at FFFF
        CLR.W       CarryBit        * Set 4 Bytes to Zero, So carry bit is represent correctly.
SEARCHLOOP
        ADDA.W      #$1,A4          * Increment the Address of A4 by $1
        MOVE.B      (A4),D1         * Give D1 the current Byte A4 is on in the loop
        CMP.B       D0, D1          * Compare the byte of data in D0 ($A300) and D1 (A4 Current Byte)
        BEQ         FOUND           * If D1,D0 Equal, go to find
        CMPA.L      #EndA, A4       * Compare current A4 Address to end address
        BEQ         ERROR           * If EndA hit, jump to error
        BRA         SEARCHLOOP      * Restart Loop if A) Not found B) not the end
```

```
FOUND
        MOVE.L      A4,Addr1         * Update Variable Addr1 to Address that was successfully found
        BRA         CONTINUE         * Jump to CONTINUE
ERROR
        MOVE.L      #$6000, Addr1    * 'Error Code' to insert $6000 into Addr1
        BRA         CONTINUE         * Jump to CONTINUE
CONTINUE
        MOVEA.L     Addr1, A4        * Update A4 with the Address of Addr1
        CLR.W       D1               * Clear out D1 to Zero, so it can be used to count down properly
        CLR.W       D3               * Clear out D3 word, need to Add BYTE but store in WORD.
ADDLOOP
        MOVE.B      (A4)+, D2        * Move the Byte value in A4 to D2, than increment A4
        ADD.W       D2, Addsum       * Add the current Byte in D2 to Addsum
        BCS         CARRY            * If Addsum exceeds 4 Bytes (Word). BCS
        ADD.W       #1, D1           * Add 1 to D1, used to count the 'loops'
        CMP.W       #255, D1         * Compare 255 to D1, used to see if D1 is greater than 255
        BGT         FINISH           * If D1 is greater than 255, finish the program
        BRA         ADDLOOP          * D1 is still less than 255, loop again
CARRY
        MOVE.W      #1, CarryBit     * Assign 1 to Carrybit, to represent CCR carrybit
        BRA         ADDLOOP          * Jump back to ADDLOOP
FINISH
*-------Display Address---------------------------------------------
        LEA         AddOut, A1       * Move AddOut into A1 for display
        MOVE.B      #14, D0          * Move (Task)14 into D0 for Trap 15
        TRAP        #15              * Display contents of A1
        MOVEA.L     Addr1, A1        * Move Address of Addr1 ($6553) into A1
        MOVE.W      A1, D1           * Move Address of A1 into D1 for display
        MOVE.B      #3, D0           * Move (Task)3[display D1 in decimal] into D0 for Trap 15
        TRAP        #15              * Display contents of D1
*-------Display Sum of Memory---------------------------------------
        LEA         SumOut, A1       * Move SumOut into A1 for display
        MOVE.B      #14, D0          * Move (Task)14 into D0 for Trap 15
        TRAP        #15              * Display contents of A1
        MOVE.W      Addsum, D1       * Move Addsum into D1
        MOVE.B      #3, D0           * Move (Task)3[Display D1 in decimal] into D0 for Trap 15
        TRAP        #15              * Display contents of D1, in decimal (Task 3)
*-------Display Carry Bit-------------------------------------------
        LEA         CarOut, A1       * Move CarOut into A1 for display
        MOVE.B      #14, D0          * Move (Task)14 into D0 for Trap 15
        TRAP        #15              * Display contents of A1
        MOVE.W      CarryBit, D1     * Move CarryBit into D1
        MOVE.B      #3, D0           * Move (Task)3[Display D1 in decimal] into D0 for Trap 15
        TRAP        #15              * Display contents of D1, in decimal (Task 3)

    SIMHALT             ; halt simulator
CR      EQU $0D             ;ASCII code for Carriage Return
LF      EQU $0A             ;ASCII code for Line Feed
AddOut  DC.B 'Address: ',0
SumOut  DC.B LF, CR,'Sum of 256: ',0
CarOut  DC.B LF, CR,'Carry  Bit: ',0

    END    START        ; last line of source
```


Sim68K I/O
Address: 25939
Sum of 256: 65032
Carry Bit: 0
68000 Memory
$ Address: 00001000
From:$00000000 To:$00000000 Bytes:$00000000 Copy Fill
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF
00001000: 28 7C 00 00 60 00 26 7C 00 00 A0 00 11 FC 00 07
00001010: 65 53 13 FC 00 07 00 00 A0 00 10 13 42 78 10 F4
00001020: 42 78 10 F8 52 4C 12 14 B2 00 67 00 00 0E B9 FC
00001030: 00 00 80 00 67 00 00 0C 60 EA 21 CC 10 F0 60 00
00001040: 00 0E 21 FC 00 00 60 00 10 F0 60 00 00 02 28 78
00001050: 10 F0 42 41 42 43 14 1C D5 78 10 F4 65 00 00 0E
00001060: 52 41 B2 7C 00 FF 6E 00 00 0C 60 EA 31 FC 00 01
00001070: 10 F8 60 E2 43 F9 00 00 10 BC 10 3C 00 0E 4E 4F
00001080: 22 78 10 F0 32 09 10 3C 00 03 4E 4F 43 F9 00 00
00001090: 10 C6 10 3C 00 0E 4E 4F 32 38 10 F4 10 3C 00 03
000010A0: 4E 4F 43 F9 00 00 10 D5 10 3C 00 0E 4E 4F 32 38
000010B0: 10 F8 10 3C 00 03 4E 4F FF FF FF FF 41 64 64 72
000010C0: 65 73 73 3A 20 00 0A 0D 53 75 6D 20 6F 66 20 32
000010D0: 35 36 3A 20 00 0A 0D 43 61 72 72 79 20 20 42 69
000010E0: 74 3A 20 00 FF FF FF FF FF FF FF FF FF FF FF FF
000010F0: 00 00 65 53 FE 08 FF FF 00 00 FF FF FF FF FF FF
00001100: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00001110: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00001120: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00001130: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00001140: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00001150: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00001160: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
Row
Page
Live
Addr1
AddSum
CarryBit

Q5. Decode a floating point number. (5 points)

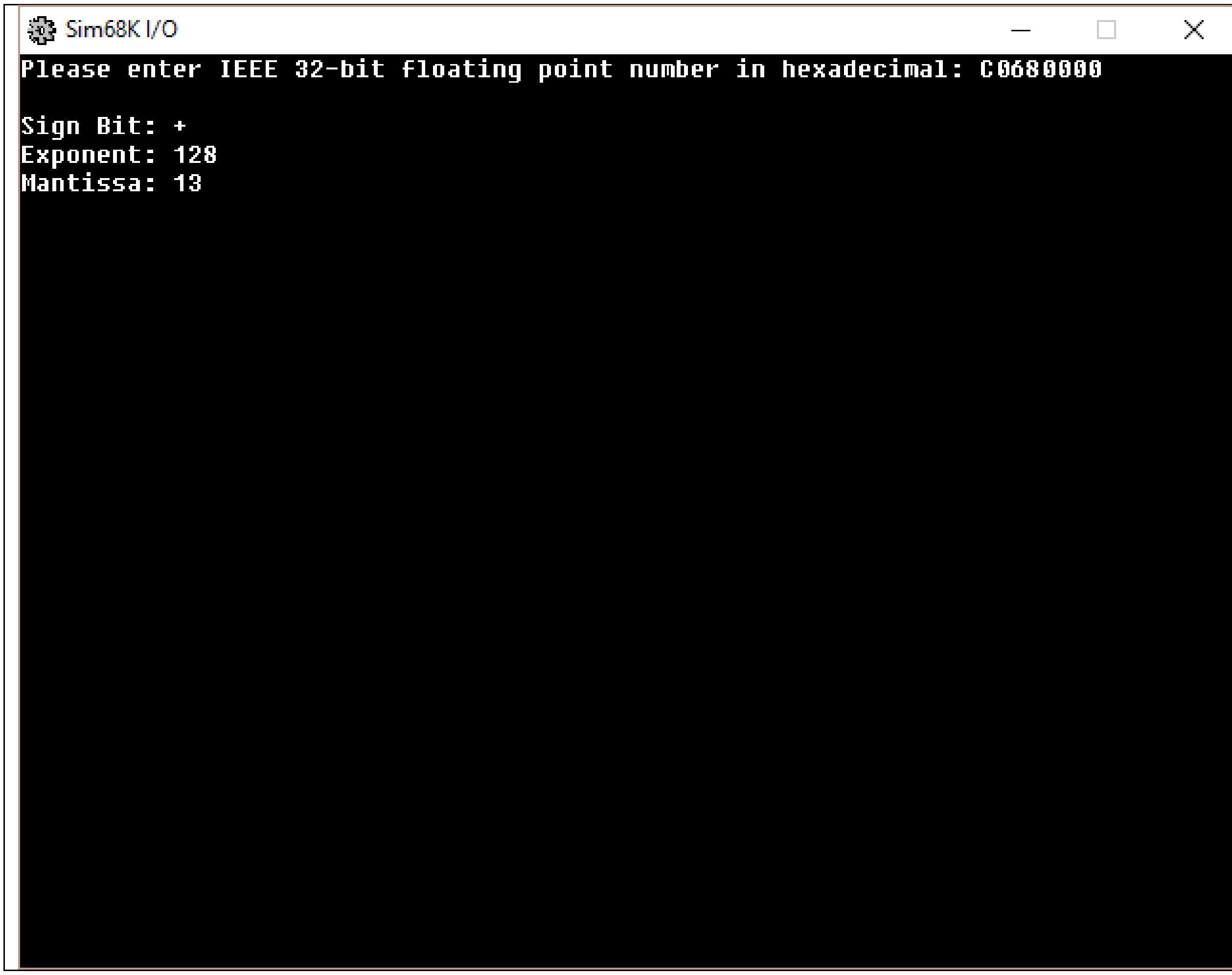
Write a program in 68K assembly code to decipher IEEE 32-bit floating point hexadecimal representation to sign bit, exponent bit, and mantissa bits. Program specifications follow:

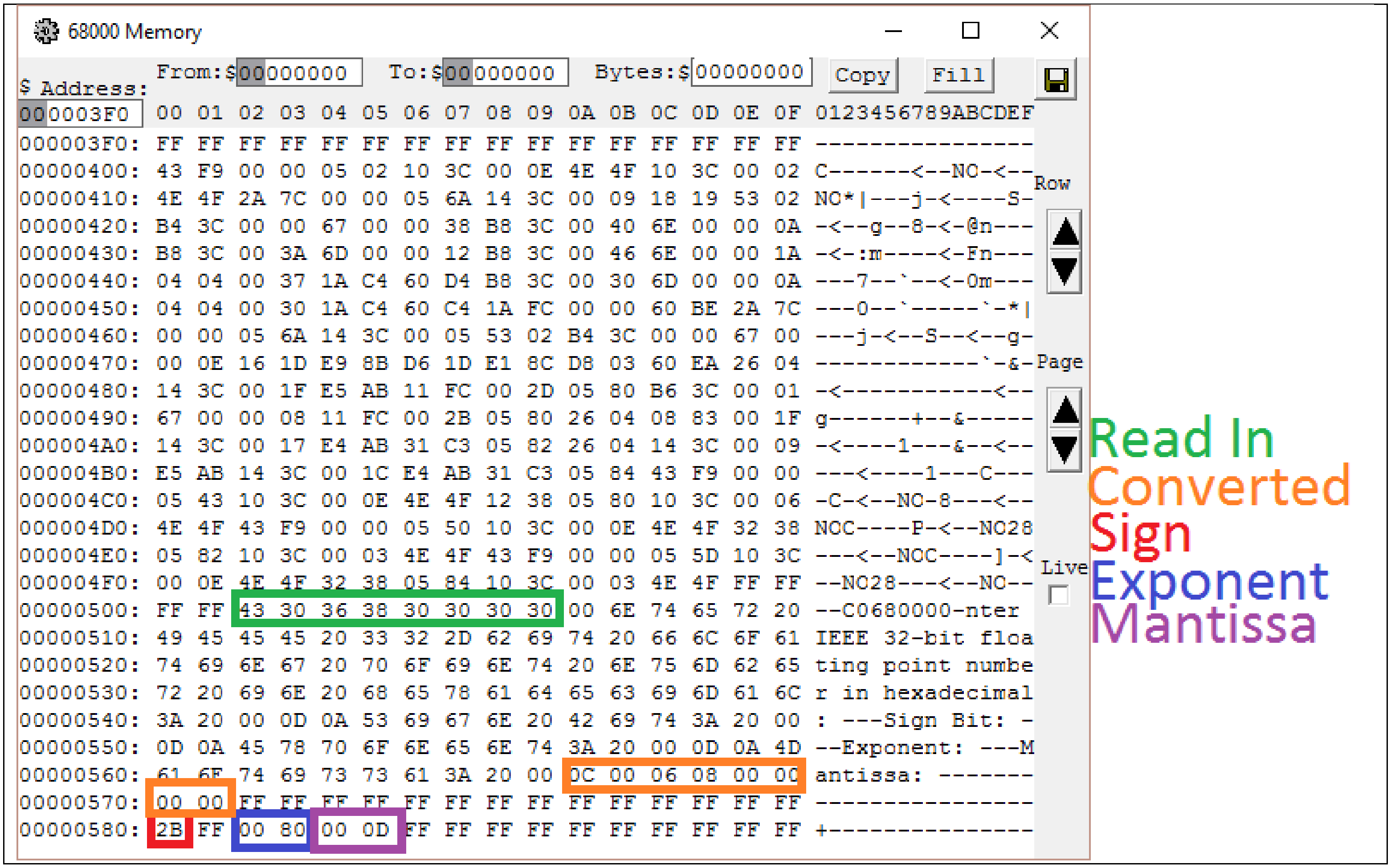
1. Your program should start at the memory location $400.
2. The program should print the instruction in the output window to get a user input: "Please enter IEEE 32-bit floating point number in hexadecimal":
3. User Input should be in IEEE 32-bit floating point number Hexadecimal format.
4. After getting the user input, save the number in the memory address right after your program code.
5. Then print out the following information to the output window: Sign bit: ("+" or "-"), Exponent: (in decimal) and Mantissa: (in decimal). For example, if the user input is :C0680000, then the memory should have the data: C0680000 in the address right after your program cod, and then the output window shows:
6. **Sign bit: -**
   Exponent:128
   Mantissa: 13

```
*-----------------------------------------------------------
* Title      : Decode a Floating Point Number
* Written by : Tyler Quayle
* Date       : April 22, 2016
* Description: Given 32-Bit Floating point Hexadecimal, give sign, exponent, mantissa
*-----------------------------------------------------------
SignBit     EQU $580
Exponent    EQU $582
Mantissa    EQU $584
    ORG    $400
START:                  ; first instruction of program

* Put program code here
    LEA     InMsg, A1    * Insert Msg to prompt user into A1 for display
    MOVE.B  #14, D0      * Task 14, Trap 15 - Display NULL terminate Sting, no CR/LF
    TRAP    #15          * Display Prompt
    MOVE.B  #2, D0       * Task 2 for Trap 15, Read in String
    TRAP    #15          * Get User input

    MOVEA.L #$56A, A5    * Give A5 Starting Address
    MOVE.B  #9, D2       * D2 used for Looping 8 times

CONVERT * For(D2 = 8, D2 > 0, D2--)
    MOVE.B  (A1)+,D4     * Give D4 first Byte of user input, then increment

    SUB.B   #1, D2       * For Loop decrement
    CMP.B   #0, D2       * Check for end of For-Loop
    BEQ     MOVEBITS     * Conversion Done, Go To Movebits

    CMP.B   #$40, D4     * Compare if D4 contains greater than $40 (A = $41, B = $42)
    BGT     CONV_LETTER * Letter Possibly Found, go to Conv_Letter to Confirm

    CMP.B   #$3A, D4     * Compare D4 if less than $3A (Hex for :) (9 = $39, 8 = $38)
    BLT     CONV_NUMBER * Number possibly Found, go to Conv_Number to Confirm
```

```
CONV_LETTER
    CMP.B   #$46, D4     * Check to see if D4 falls within $41-$46 (A-F)
    BGT     MISSING      * Not $41-$46 (HEX A-F), Bad/No Data, Jump to Missing
    SUB.B   #$37, D4     * Subtract $37, to convert (ASCII->HEX)
    MOVE.B  D4, (A5)+    * Give current A5 the D4 hex value and Increment
    BRA CONVERT          * Successfully Convert (ASCII->HEX), Branch back to Convert Loop

CONV_NUMBER
    CMP.B   #$30, D4     * Check to see if D4 falls within $30-$39 (0-9)
    BLT     MISSING      * Not $30-$39 (HEX 0-9), Bad/No Data, Jump to Missing
    SUB.B   #$30, D4     * Subtract $30, to convert (ASCII->HEX)
    MOVE.B  D4, (A5)+    * Give current A5 the D4 hex value and Increment
    BRA CONVERT          * Successfully Convert (ASCII->HEX), Branch back to Convert Loop

MISSING * Here in case user did not enter 8 numbers, 'fills' rest with 0's
    MOVE.B  #$00, (A5)+ * Give Current address Byte value $00, increment A5
    BRA CONVERT          * Branch back to Convert Loop

MOVEBITS
    MOVEA.L #$56A, A5    * Reset A5 to starting address
    MOVE.B  #5, D2       * Give D2 $5, used in MOVELOOP logic

MOVELOOP * For(D2 = 5, D2 > 0, D2--)
    SUB.B   #1, D2       * For Loop decrement
    CMP.B   #0, D2       * Check if conditions have been met
    BEQ     FINDSIGN     * Bit moving done, D4 Now contains correct 32-Bit Hex
    MOVE.B (A5)+, D3     * Move Current Byte into D3
    LSL.L   #4, D3       * Logical Shift Bits 1 Nibble Left (0X->X0)
    ADD.B  (A5)+,D3      * Add next Byte into Bit Shifted D3(XX)
    LSL.L   #8, D4       * Logical Shift D4 1 Byte (00XX->XX00)
    ADD.B   D3, D4       * Add D3 Byte to D4 (XXXX)
    BRA MOVELOOP         * Repeat Loop

FINDSIGN
    MOVE.L  D4, D3           * Move D4(User Hex Value) into D3 for manipulation
    MOVE.B  #31, D2          * Move 31 into D2, used for Following Logic Shift
    LSL.L   D2, D3           * Logic-Shift-Left 31, leaving only MSB
    MOVE.B  #$2D, SignBit    * Default to $2D (Hex for '-')
    CMP.B   #$1, D3          * Compare D3 to 1 (check for negative input)
    BEQ     FINDEXP          * If MSB was negative, continue
    MOVE.B  #$2B, SignBit    * MSB was 0, change Sign to $2B (Hex for '+')
FINDEXP
    MOVE.L  D4, D3           * Move D4(User Hex Value) into D3 for manipulation
    BCLR.L  #31, D3          * Clear MSB
    MOVE.B  #23, D2          * Move 23 into D2,
    LSR.L   D2, D3           * Logic-Shift-Right 23 || Get rid of Mantissa
    MOVE.W  D3, Exponent     * Give variable Exponent the value in D3
FINDMAN
    MOVE.L  D4, D3           * Move D4(User Hex Value) into D3 for manipulation
    MOVE.B  #9, D2           * Move 9 into D2, Used for Following Logic Shift
    LSL.L   D2, D3           * Logic-Shift-Left 9,
    MOVE.B  #28, D2          * Move 28 into D2, Used for following logic shift
    LSR.L   D2, D3           * Logic-Shift-Right 28, moving first 4 bits of decimals into D3.B
    MOVE.W  D3, Mantissa     * Give variable Mantissa value in D3

PRINT
```

```
    LEA     OutSig, A1       * Load OutSig message into A1 for display
    MOVE.B  #14, D0          * Task 14, Trap 15 - Display NULL terminate Sting, no CR/LF
    TRAP    #15              * Display OutSig (Sign) message
    MOVE.B  SignBit, D1      * Move variable SignBit into D1 for display
    MOVE.B  #6, D0           * Task 6, Trap 15 - Display Single Char in D1 (Hex code for +/-)
    TRAP    #15              * Display char in D1, Hex value for either '+' or '-'

    LEA     OutExp, A1       * Load OutExp message into A1 for display
    MOVE.B  #14, D0          * Task 14, Trap 15 - Display NULL terminate Sting, no CR/LF
    TRAP    #15              * Display OutExp (Exponent) message
    MOVE.W  Exponent, D1     * Move variable Exponent into D1 for display
    MOVE.B  #3, D0           * Task 3, Trap 15 - Display Value in D1 as Decimal
    TRAP    #15              * Display variable Exponent

    LEA     OutMan, A1       * Load OutMan message into A1 for display
    MOVE.B  #14, D0          * Task 14, Trap 15 - Display NULL terminate Sting, no CR/LF
    TRAP    #15              * Display OutMan
    MOVE.W  Mantissa, D1     * Move variable Mantissa into D1 for display
    MOVE.B  #3, D0           * Task 3, Trap 15 - Display Value in D1 as Decimal
    TRAP    #15              * Display variable mantissa

    SIMHALT              ; halt simulator

*---------OutPut Values-------------------------------------------------------
CR      EQU $0D              ;ASCII code for Carriage Return
LF      EQU $0A              ;ASCII code for Line Feed
InMsg   DC.B CR,'Please enter IEEE 32-bit floating point number in hexadecimal: ' ,0
OutSig  DC.B CR, LF,'Sign Bit: ',0
OutExp  DC.B CR, LF,'Exponent: ',0
OutMan  DC.B CR, LF,'Mantissa: ',0

* Put variables and constants here

    END    START         ; last line of source
```


Sim68K I/O
Please enter IEEE 32-bit floating point number in hexadecimal: C0680000
Sign Bit: +
Exponent: 128
Mantissa: 13


68000 Memory
$ Address: From:$00000000 To:$00000000 Bytes:$00000000 Copy Fill
000003F0 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF
000003F0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00000400: 43 F9 00 00 05 02 10 3C 00 0E 4E 4F 10 3C 00 02 C------<--NO-<--
00000410: 4E 4F 2A 7C 00 00 05 6A 14 3C 00 09 18 19 53 02 NO*|---j-<----S-
00000420: B4 3C 00 00 67 00 00 38 B8 3C 00 40 6E 00 00 0A -<--g--8-<-@n---
00000430: B8 3C 00 3A 6D 00 00 12 B8 3C 00 46 6E 00 00 1A -<-:m----<-Fn---
00000440: 04 04 00 37 1A C4 60 D4 B8 3C 00 30 6D 00 00 0A ---7--`--<-0m---
00000450: 04 04 00 30 1A C4 60 C4 1A FC 00 00 60 BE 2A 7C ---0--`-----`-*|
00000460: 00 00 05 6A 14 3C 00 05 53 02 B4 3C 00 00 67 00 ---j-<--S--<--g-
00000470: 00 0E 16 1D E9 8B D6 1D E1 8C D8 03 60 EA 26 04 ------------`-&-
00000480: 14 3C 00 1F E5 AB 11 FC 00 2D 05 80 B6 3C 00 01 -<-----------<--
00000490: 67 00 00 08 11 FC 00 2B 05 80 26 04 08 83 00 1F g------+--&-----
000004A0: 14 3C 00 17 E4 AB 31 C3 05 82 26 04 14 3C 00 09 -<----1---&--<--
000004B0: E5 AB 14 3C 00 1C E4 AB 31 C3 05 84 43 F9 00 00 ---<----1---C---
000004C0: 05 43 10 3C 00 0E 4E 4F 12 38 05 80 10 3C 00 06 -C-<--NO-8---<--
000004D0: 4E 4F 43 F9 00 00 05 50 10 3C 00 0E 4E 4F 32 38 NOC----P-<--NO28
000004E0: 05 82 10 3C 00 03 4E 4F 43 F9 00 00 05 5D 10 3C ---<--NOC----]-<
000004F0: 00 0E 4E 4F 32 38 05 84 10 3C 00 03 4E 4F FF FF --NO28---<--NO--
00000500: FF FF 43 30 36 38 30 30 30 30 00 6E 74 65 72 20 --C0680000-nter
00000510: 49 45 45 45 20 33 32 2D 62 69 74 20 66 6C 6F 61 IEEE 32-bit floa
00000520: 74 69 6E 67 20 70 6F 69 6E 74 20 6E 75 6D 62 65 ting point numbe
00000530: 72 20 69 6E 20 68 65 78 61 64 65 63 69 6D 61 6C r in hexadecimal
00000540: 3A 20 00 0D 0A 53 69 67 6E 20 42 69 74 3A 20 00 : ---Sign Bit: -
00000550: 0D 0A 45 78 70 6F 6E 65 6E 74 3A 20 00 0D 0A 4D --Exponent: ---M
00000560: 61 6E 74 69 73 73 61 3A 20 00 0C 00 06 08 00 00 antissa: -------
00000570: 00 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
00000580: 2B FF 00 80 00 0D FF FF FF FF FF FF FF FF FF FF +---------------
Row
Page
Live
Read In
Converted
Sign
Exponent
Mantissa