Regie Pineda

Dr. Amed Ahwad

CSS 422

October 28th, 2019

Homework 2

**Q1. Convert the following 68K assembly language instructions to the machine codes. (2 points)**

**You have to show your work, otherwise you will get zero**.

Note: I used the manual provided to answer these questions. Therefore, my work is based on the manual.

1) MOVE.W        D1,  $0000A000

**Answer:** 33C1 0000A000

By the manual, the 15th and 14th bits are always marked as 0 for MOVE operations. Also, the 13th and 12th bits indicate the size and since in this case the operation is of size “WORD” the 13th bit will be 1 and the 12th bit will also be of value 1. The 9th through 11th bits are for the Destination Register, whereas the 6th through 9th bits are for the Destination Mode. I set the Destination Mode bits as 111 and Destination Register as 001 to go with the (xxx).L instruction. I set the Source Mode as 000 and the Source Register as 001 because I was using the Dn addressing mode and the source register is D1.

Then, I found that the binary number was 0011001111000001 which gives a value 33C1 in hexadecimal. Therefore, I got my answer by adding the absolute address in long format, so that’s how I got my answer of 33C1 0000A000

2) MOVE.B        $42A7, (A1)+

**Answer:** 12F8 42A7

The 13th and 12th bits indicate the size and since in this case the operation is of size “BYTE” the 13th bit will be 0 and the 12th bit will be set to value 1. The 9th through 11th bits are for the Destination Register, whereas the 6th through 9th bits are for the Destination Mode. I set the Destination Mode bits as 011 and Destination Register as 001 to comply with the (A1)+ instruction. I set the Source Mode as 111 and the Source Register as 000 because I was using the (xxx).W addressing mode and the source register is $42A7.

Then, I found that the binary number was 0001001011111000 which give a value of 12F8 in hexadecimal. Therefore, I got my answer by adding the absolute address in word format, so that’s how I got my answer of 12F8 42A7.

3) ADD.L            D7, D0

**Answer:** D087

The 12th through 15th bits are marked as 1101 for ADD operations. Also, the 9th through 11th bits are for the Register, whereas the 6th through 8th bits are for the OpMode, the 3rd through 5th bits are for Mode, and the 0th through 2nd bits are for the general Register. Since this operation involved the addition of two data registers. I treated the D0 as the destination and the D7 as the source. That way, I was enabled to use the format where 9th through 11th bits indicate the destination data register, and the 6th through 8th bits indicate the size for the destination. As a result, the 9th through 11th bit values were formatted as 000 and the 6th through 8th bits were formatted as 010. The 3rd through 5th bits were marked as 000 to indicate the source is a data register whereas the 0th through 2nd bits were together marked as 111 to indicate the register number D7.

Then, I found that the binary number was 1101000010000111 which gives a value of D087 in hexadecimal.

4) MOVEA.L       D3, A0

**Answer:** 2043

The 15th and 14th bits are always marked as 0 for MOVEA operations. The 13th and 12th bits indicate the size and since in this case the operation is of size “LONG” the 13th bit will be 1 and the 12th bit will be 0. The 9th through 11th bits are for the Destination Register, whereas the 6th through 8th bits are marked as 001. Since this is a MOVEA operation I set the Destination Register as 000 to indicate that the address register is A0 and the 6th through 8th bits were marked as 001. I set the Source Mode as 000 and the Source Register as 011 because I was using the Dn addressing mode and the source register is D3.

Then, I found that the binary number was 0010000001000011 which gives a value of 2043 in hexadecimal.

**Q2. For each of the operations below, evaluate the value in D0 and the state of the CCR after completing the operation. XNZVC=00000 and D0=$C1A8E392 at the beginning of each operation. (4 points)**

1. ASL.B #2, D0

* 9 2 will be affected in this operation because it is a size of BYTE. I set these 2 bits in binary form and then did a arithmetic shift left. Therefore, 1001 0010 originally will become 0100 1000 which gives new values of 4 8. The ‘V’ flag is set to 1 by this operation because the most significant bit was changed by this operation.

**ANSWER: XNZVC** = 00010 **D0** = C1A8E348

1. ASL.L #5, D0

* All of the bits in D0 will be affect because it is a size of LONG. We will be arithmetically shifting to the left by 5 bits. I set the bits in binary and did the arithmetic shift left operation. So, 1100 0001 1010 1000 1110 0011 1001 0010 will become 0011 0101 0001 1100 0111 0010 0100 0000 which gives new value of 351C7240. The ‘V’ flag is set to 1 by this operation because the most significant bit was changed by this operation.

**ANSWER: XNZVC** = 00010 **D0** = 351C7240

1. LSR.B #4, D0

* 9 2 will be affected from the original D0 since this is a size of BYTE. This is a logical shift right which will shift right by 4 bits. I set 9 2 to binary and shifted 4 bits to the right. So, 1001 0010 will become 0000 1001 after the operation which gives new values of 0 9. None of the flags were set because the last bit shifted out of the operand was of value 0, since X is equal to C and it is dependent on this last bit shifted out X is set to 0 and C is cleared according to the manual.

**ANSWER: XNZVC** = 00000 **D0** = C1A8E309

1. ROR.W #2, D0

* E 3 9 2 will be affected from the original D0 since this is a size of WORD. This will rotate to the right by 2 bits. Originally, I set E 3 9 2 to binary and then shifted 2 bits to the right. So, 1110 0011 1001 0010 will become 1011 1000 1110 0100 after the operation which would give a new value of B8E4. The N flag is set because the result is negative and the C flag is set because the last bit rotated out of the operand is 1.

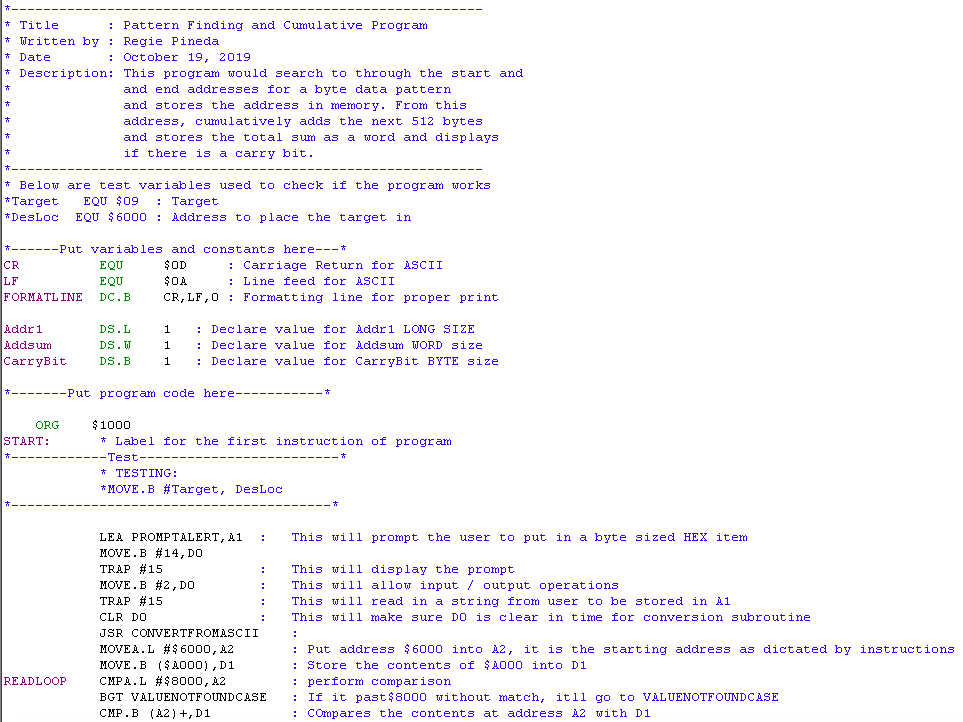
**ANSWER: XNZVC** = 01001 **D0** = C1A8B8E4

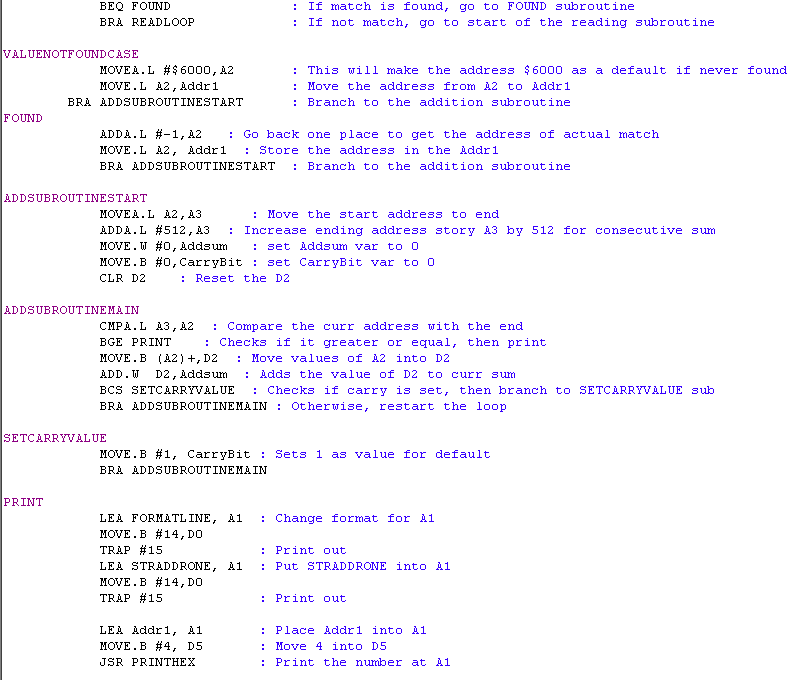
**Q3. Pattern Finding and Cumulative program. (5 points)**

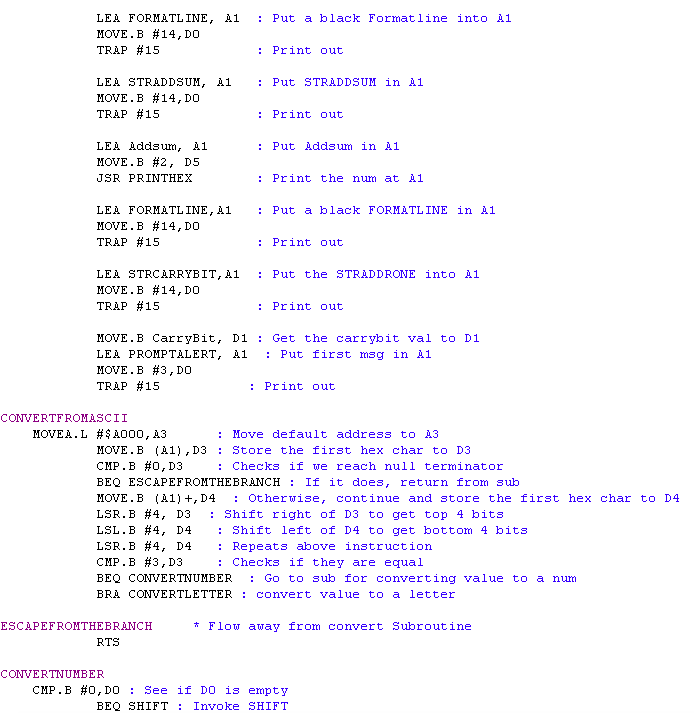
Write a program in 68K assembly code that satisfies the following specifications:

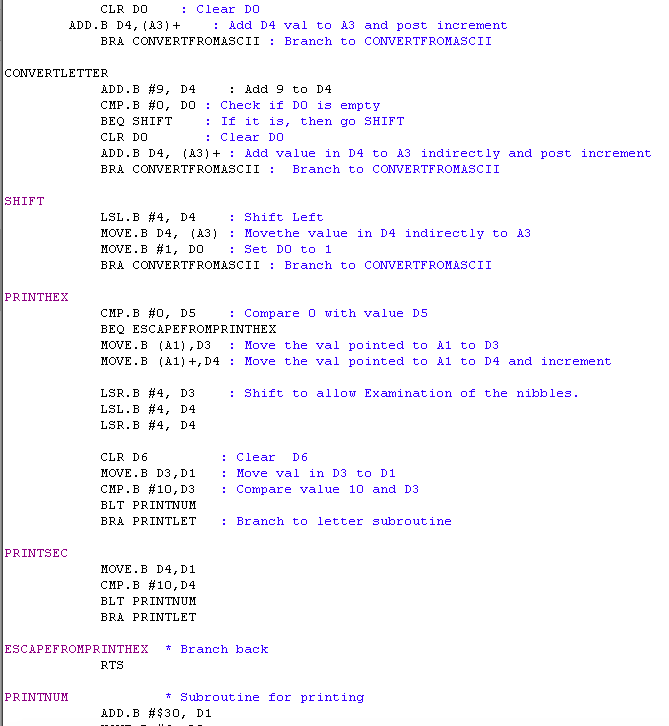
1. Your program should start at the memory location $1000 (hint: use the ORG directive).
2. **Take a one-byte hexadecimal value from user input** and save the value at address $A000 (hint: you can consider this input a target value). You should not hardcode the inputs. You may ignore sanity check on corner cases but assume user inputs are always in an expected format.
3. Read each **byte data** stored in memory between the addresses $6000 and $8000 and compare it to the (byte) data at address $A000 (hint: you can consider the memory between $6000 and $8000 an array).
4. Please define a **LONGWORD**variable called address **Addr1**. In the specified memory range, if the data at address $A000 is found, the **longword address of the data**in memory is saved into variable **Addr1**. 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.
5. Please define a **WORD** variable called address **Addsum**. Add a series of **bytes** stored in the **512**consecutive memory locations beginning at address Addr1, no matter you find the target data or not. Save the sum value into **Addsum**.
6. 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**.
7. Print the **Addr1**, **Addsum** and **CarryBit** in the output window.
8. Submit the followings:
   * **Copy and paste your source code** in your submission file!
   * **Additionally**, please submit the source file (.X68) **(If no source file, then you got zero point for this question)!**
   * Test your program and report the result with your user input. **Capture the images of output window and memory view** to support your results, **copy and paste the images**into your submission file.

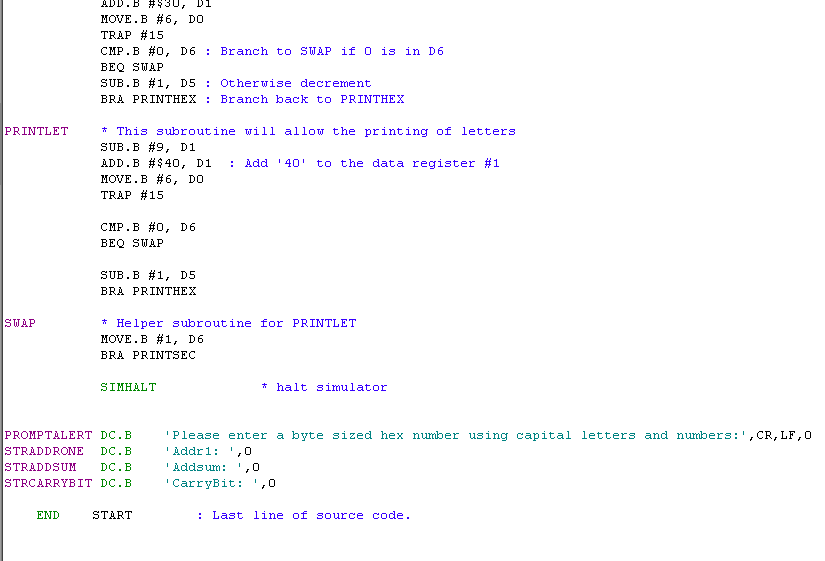
I tested my program by setting a target of ‘09’ and placing that target at location of $6000. As I inputted ‘09’ at location $6000, I know that the Addr1 will be 6000. Therefore, I can say that my program works. I commented out what I hardcoded to test my program but it sure works. Below are my source code screenshots and my results.

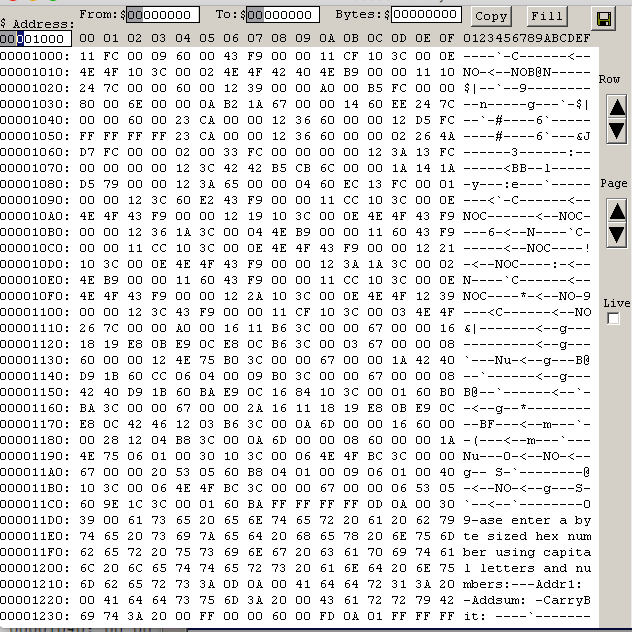


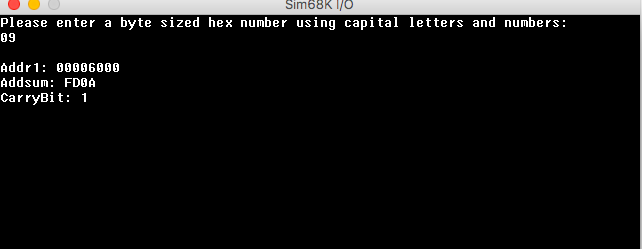












**Q4. Decode a floating-point number. (5 points)**

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

1. Your program should start at the memory location $4000.
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. You should not hardcode the inputs. You may ignore sanity check on corner cases, but assume user inputs are always in an expected format.
4. After getting the user input, save the input number into a variable that you define, but not into a register.
5. 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 output window shows:

**Sign bit: -**

**Exponent:128**

**Mantissa: 13**

1. Submit the followings
   * **Copy and paste your source code** in your submission file!
   * **Additionally**, please submit the source file (.X68) **(If no source file, then you got zero point for this question)!**
   * Test your program and report the result with your user input. Capture the images of **output window and memory view** to support your results, **copy and paste the images**into your submission file.

