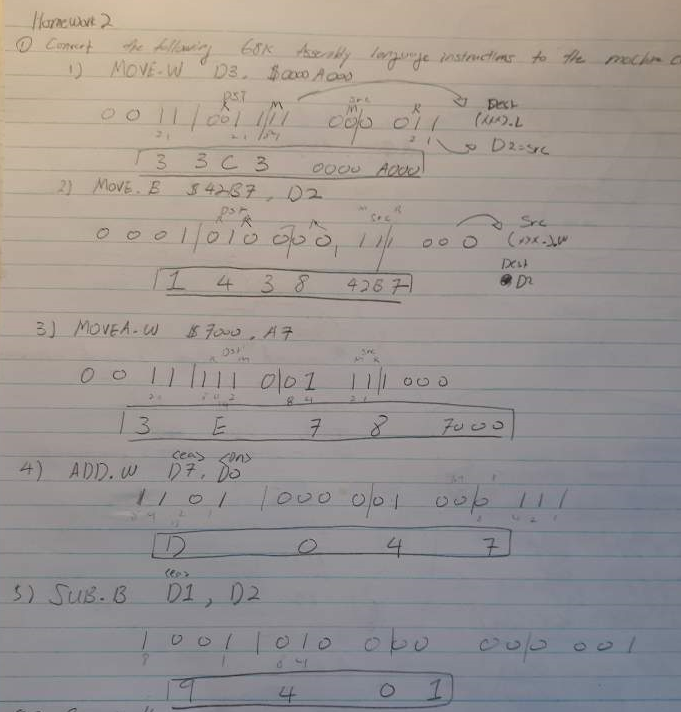
**Jun Hyung Park**

**CSS 422**

**Homework 2**

**Q1. (5pts) Assemble the codes**

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



**Q2. (10pts) Disassemble the codes**

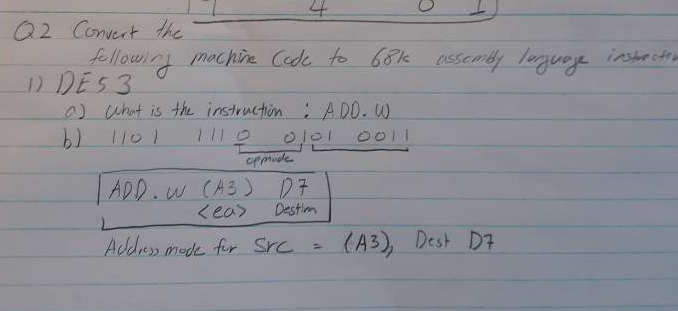
Convert the following machine codes to 68K assembly language instructions. Refer 68K manual, and only refer the MOVE, MOVEA, ADD, SUB instructions. Note that some immediate data format can be various.

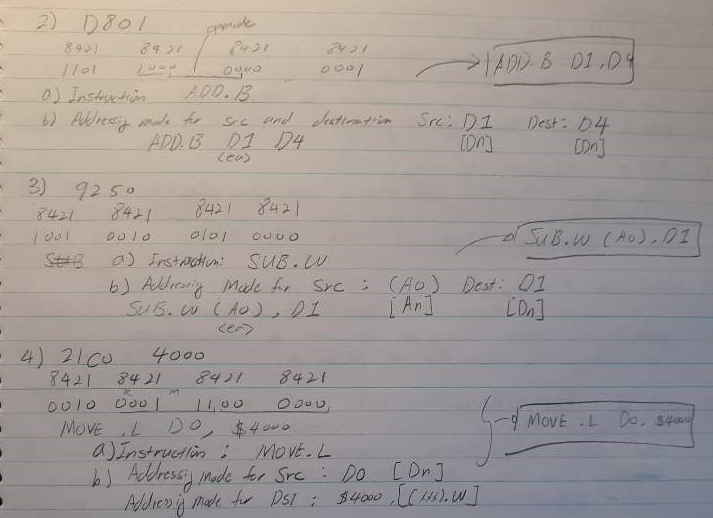
 For each question, you have to answer the followings and show your work. Without these steps, you will get zero

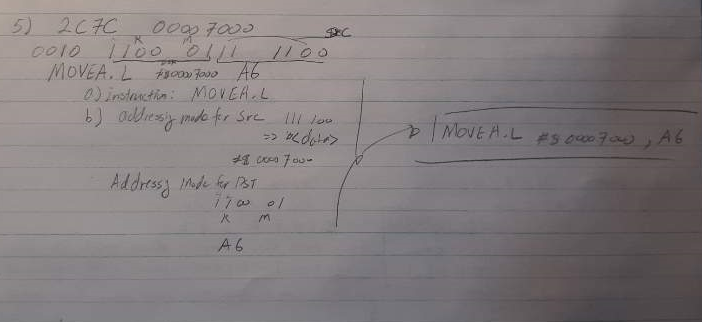
a) What is the instruction?

b) What is the addressing mode for source and destination?

Hint: Convert it to binary. Determine the instruction with opcode, then find the bits for source and destination. From the mode table in the manual, determine its addressing mode.







**Q3. (5 pts) Error Finding**

 Each of the following 68K assembly language instructions will cause an assembler error. Examine each instruction and explain how to fix it.

1) MOVE.B      $A000, A3

The Destination cannot be address register, if the instruction is MOVE. It has to be either change Destination Register to Data register or Change the instruction to MOVEA and size to either word or long word.

2) ADD.B         #$1000, D2

The source immediate data size is not byte size. So, either change the immediate data size of change the size of instruction like ADD.W

3) MOVEA.W    $1234, D0

Since instruction is MOVEA, the destination register cannot be data register. Either change the destination to address register or change the instruction to MOVE.W

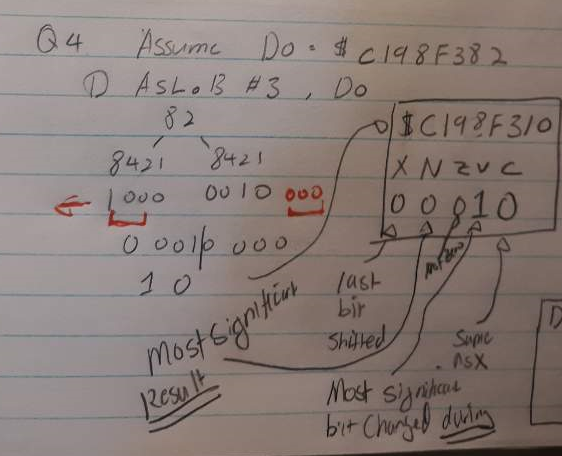
4) ANDI.B         #23,    #$100

The instruction ANDI.B requires #<data>, <ea> and effective address cannot be immediate data. Therefore either use Dn, (An), (An)+, etc… except An and #<data>.

5) SUBI.B            D3,    %1000

The source D3 is the problem. Because the syntax of instruction SUBI is SUBI #<data>, <ea> and D3 is not immediate data, it is data register. So, to fix it, simply change D3 to immediate data whatever you want but size of Byte.

**Q4. (5 pts) For each of the operations below, assume that D0 contains the value $C198F382 and initially XNZVC=00000. Then, evaluate the value in D0 and the state of the CCR.**

****

Therefore, D0: C198F310 and 0 0 0 1 0 for XNZVC bits.

I think it is kind of hard to see why XNZVC is like that, so I will write it in here.

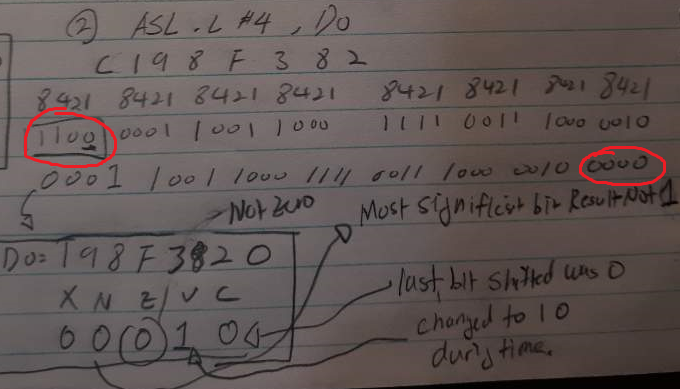
X: 0 because the last bit shifted out was 0

N: 0 because most sig fig on the result is 0

Z: 0 because its not 0

V: 1 because most significant bit changed over time during shift

C: 0 because last bit shifted out was 0



D0: 198F3820 and 0 0 0 1 0 for XNZVC

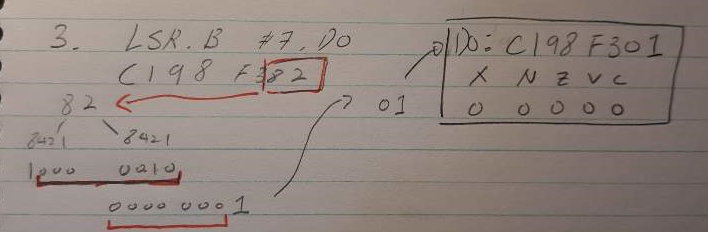
X: 0 because the last bit shifts out was 0

N: 0 because most significant bit is 0

Z: 0 because not 0

V: 1 because most significant bits has changed during shift.

C: 0 because last bit shifts out was 0



D0: C198F301 and 0 0 0 0 0 for XNZVC

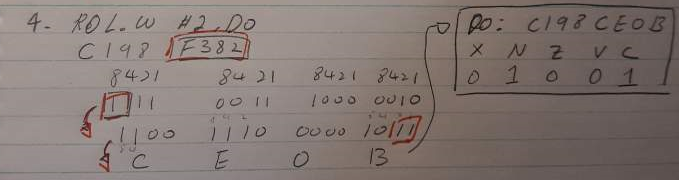
X: 0 because last bit shifted out is 0

N: 0 because the result is not negative

Z: 0 because it is not 0

V: Always clear so 0 because we started as 0

C: 0 because the last bit shifted out is 0



D0: C198CE0B and 0 1 0 0 1 for XNZVC

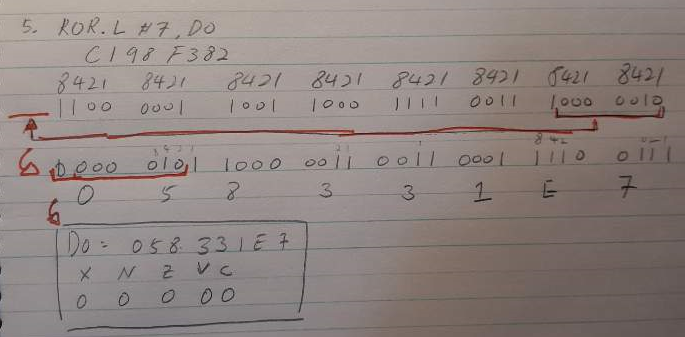
X: Not affected so just 0 (started as 0)

N: 1 because the most significant bit of the result is 1

Z: 0 because it is not zero

V: Always clear so 0 (started as 0)

C: 1 because the last bit rotated out was 1



D0: 058331E7 and 0 0 0 0 0 for XNZVC

X: Not affected so just 0

N: 0 because the most significant bit of the result is 0

Z: 0 because it is not zero

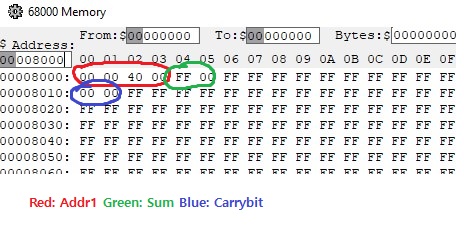
V: always clear (0 because we started as 0).

C: 0 because the last bit rotated out was 0.

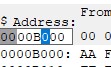
**Q5. (5 pts) Pattern Finding and Cumulative program.**

**Test 1: Nothing Match**

**Memory of Addr1, Addsum, CarryBit**

****

**Target (B000)**

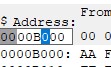
****

**Output**

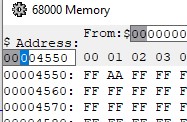
****

**Test 2: Found Target At $4551**

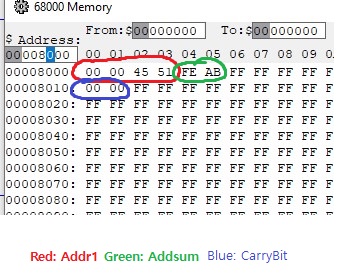
**Target**

****

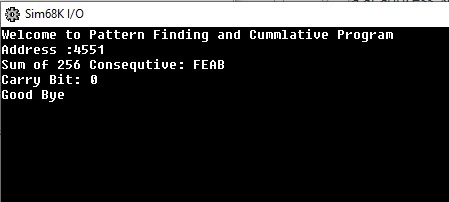
**AA had been in memory location 4551**

****

**Memory of Addr1, Addsum, CarryBit**

****

**Result**

****