Example 1: Find the 16 bit 2's complementary binary representation for the decimal number 1987

A: First convert the decimal 1987 to binary…

Power of 2:	1024	512	256	128	64	32	16	8	4	2	1
	1	1	1	1	1	0	0	0	0	1	1

1024+512+256+128+64+2+1 = 1987

Since the number is positive, there is no need to do any complement in this case.

Pack 0's to the left to make this 16 bit:

0000 0111 1100 0011

Example 2: Find the 16 bit 2's complementary binary representation for the decimal number -1987

A: First step is same as the conversion to binary in previous example:

11111000011

Since we are dealing with a negative number, take a 2's complement. (Remember, 2's complement is actually 1's complement + 1). Simply flip the bits and add 1.

```
00000111100
        +1
-----------
00000111101
```

Since the question requires the representation to be in 16 bit binary format, pack the left of the obtained binary number with 1's (we are packing with 1 because this is a negative number)

1111 1111 1111 1000 0011 1101

Example 3: What is the sign and magnitude of the following 4's complement number? 33333210_4

Since it is on base 4, and the complement number is starting with 3, which is more than the midpoint value of 1, (0—1, 2—3) it certainly represents a negative number. We need to take a 4's complement (3's complement + 1) to find the sign and magnitude of this number. Since the question does not specify any specific digit representation (like 6-digit representation etc) we use the same number of digits as the question for this process.

33333333
33333210

00000123
+1

00000130_4

Answer: -130_4

Example 4: Compute the following by using <u>6-digit 10's complementary</u> method where needed. The numbers are in Decimal.

37968
-24109

To compute this, first take a 6-digit 10's complement of the negative number.

999999
24109
_____
975890 + 1 = 975891

Now we can proceed with the addition to compute the final answer:

37968
975891

1013859

The question states 6 digits, and our answer has 7 digits therefore there is a carry, which we can throw out, and we get 013859, or, 13859

<u>Note:</u> If the given number was (-) (-) 24109, it is actually a positive number, therefore you can obtain the result by performing regular decimal addition. You can obtain the same result by taking complement for each negative sign and then go on with the addition with other number. By taking two separate complement, you will see you are ending up with the same number, except the negative sign ☺

Example 5: Convert the number 19557 to floating point using S E E M M M M format with Excess-40 notation. The implied decimal point is to the left of the first digit of the mantissa. 1 is used for positive, 7 is used for a negative number.

First, normalize the number: $19557 = 19557 \times 10^0 = .19557 \times 10^5$

Now Convert: Since the number given is positive, using the format, we get 1 45 1955 or by rounding, 1 45 1956 (either is correct). Answer: 1451955

Example 6: Following Example 5, What is the range of numbers that can be stored in this format?

Smallest mantissa: .0001 and largest mantissa: .9999; Smallest exponent -40 (remember, this example is using excess 40 notation) and largest exponent 99-40 = 59 (in this example, 2 digits are used to represent the exponent, 99 is the largest number that can be represented using 2 digits).

Therefore, the answer is: $.0001 \times 10^{-40}$ to $.9999 \times 10^{59}$

Example 7: What would be the floating point representation of -19557?

Everything will remain same except the sign bit, which is expressed using 7 as described in example 5 questions. So the answer will be: 7451955

Example 8: What is the floating point representation of .0000019557? All conditions remain same as example 5.

Normalize: .0000019557 = .19557 x 10^{-5} ; Sign: 1, Exponent: 40 – 5 = 35, Mantissa: 1955; Answer: 1 35 1955 = 1351955

Example 9: The following decimal numbers are stored in Excess-50 notation with the decimal point to the left of the first mantissa digit. 9 is used as negative sign. Add them.

05012500 = .12500 x 10^{0}
95325750 = - .25750 x 10^{3}

Since the second number is a negative number, simply perform a subtraction.

95325750
05012500
---

but the Exponents must align before subtraction. Bump up the EE bits to 53 from 50, and pack 3 0 bits after the EE bits then continue with the mantissa. Therefore:

95325750
05300012

95325738 = (-).25738x10^{3} = -257.38

Example 10: Multiply the following numbers and show results in standard decimal format:

94650000
94450000

To do a multiplication operation, first multiply the mantissas.
.5 x .5 = .25

To calculate the exponent, add the exponents of both numbers and subtract the exces-n, in this example we are following excess-50 notation. [Exponent 1 + Exponent 2 – Excess-N] or, 46+44-50 = 40: This is our EE bits

We are multiplying two negative numbers; therefore the resulting sign must be positive.

The result is: 1 40 25000 using a S EE MMMMM format with Excess-50 notation using 1 as positive and 9 as negative sign.

Chapter 6 Examples:

Example 1: A little man program that accepts three values as input and produces the largest of the three as output.

```
00 INP        901      ;input a number and save
01 STO  99    399      ; Store in Loc 99
02 INP        901      ; input another number and save
03 STO  98    398      ; store in Loc 98
04 INP        901      ; input another number and save
05 STO  97    397      ; store in Loc 97
06 SUB  98    298      ;subtract number in 98 from that in 97
07 BRP  10    811      ;number in 97 larger [BR only if SUB result is positive]
08 LDA 98 598          ;number in 98 larger-restore 98
09 BR 11      611
10 LDA 97 597          ;restore 97
11 STO  96    396      ;store larger of (97, 98) in 96
12 SUB  99    299      ;subtract number in 99 from larger
13 BRP  16    816      ;number in 96 larger
14 LDA 99 599          ;number in 99 larger--restore 99
15 BR 17      617
16 LDA 96 596          ;restore 96
17 OUT        902
18 COB        000
```

Example 2: A Little man program that adds a column of input values and produces the sum as output. The first input contains the total number of values that follow as input to be added.

```
00 LDA 90     590
01 STO 99     399     ;initialize sum to zero
02 INP        901     ;input the count..
03 STO 98     399     ;..and save it in 98
04 INP        901     ;input a number
05 ADD 99     199
06 STO 99     399     ;add to sum and store sum
07 LDA 98     598
08 SUB 91     291     ;decrement count by 1
09 BRZ 11     711     ;done--output result
10 BR 04      704     ;not done--ready for next input
11 LDA 99     599
12 OUT        902     ;output result
13 COB
90 DAT 00     000     ;constant value 0
91 DAT 01     001     ;constant value 1
```