Assignment 1: Converting "Number in base 4" to "Hexadecimal". C calling convention.

Submissions which deviate from these instructions will not be graded!

Please make your output exactly as in the examples below.
Otherwise automatic scripts would fail on your assignment and you will lose some credit for this.

Assignment Description

This assignment has two parts:

Part 1 - Converting "Number in base 4" to "Hexadecimal"

We provide you a simple program in C that inputs a "string" (as a line) from the user, and then calls a function my_func that receives a pointer to the beginnig of a null terminated string.
You need to implement my_func in assembly language. Assume that the input string contains digit characters of an unsigned number in base 4, and has up to 32 characters, not including the null terminator.
Your function should convert the given number from the input string into a hexadecimal unsigned number, represented as a string, and print it, followed by a line feed character, and then exit. You may **not** use a C function that does the conversion, you need to compute the output digits yourself.

Note that we are providing a skeleton to use in writing the assembly language code below.
You are not allowed make any changes to the .c file.

Examples:

- given "213022013" as the input string (the quotes are not part of the string), your program should output the string: "27287"
```
> task1.bin
213022013
27287
```
- given "221001" as the input string (the quotes are not part of the string), your program should output the string: "A41"
```
> task1.bin
221001
A41
```

Note that letters in the output string should be upper-case (A - F).

Part 2 - C calling convention - self descriptive numbers

In this part you need to implement entirely on your own. You are to create a C file and an assembly language file as specified in the submission instructions below.

You are to write a C program that:

1. Reads one long long (64 bits) unsigned number x (in hexadecimal) from the user.
   Assume that <u>not</u> all the digits in this number have same value.
2. Reads one (32 bits) integer numOfRounds (in decimal) from the user.
3. Call a function ‘calc_func(long long *x, int numOfRounds)’ written in ASSEMBLY language with the above pointer to long long, and number of rounds as arguments.

The assembly language function ‘calc_func(long long *x, int numOfRounds)' performs the following steps (in a loop) at most numOfRounds times:

1. Calculates a "descriptive number y" for the input number x.
   A descriptive number y is calculated as follows: For every hexadecimal digit d, count how many times d appears in x. Then, put this counter value at the d'th digit in y (assuming that the digits in y are indexed from left to right). For example, if 0 digit appears 8 times in x, then the most significant nibble of y would be 8.

2. Call the C function 'compare (long long * x, long long * y)' that you have to implement, which gets pointers to two numbers and returns true if they are equal, and false otherwise.

3. If 'compare' returns "true" , print x using the C standard library function "printf", and break the from the loop.

4. Otherwise, if the loop was executed less than numOfRounds times, set x <– y and continue, else break.

If the resulting number y was not printed yet, print y.

Example: a single round with x = 1AF2345FF23B0001

-
  ```
  > task2.bin
  1AF2345FF23B0001
  1
  3222110000110003
  ```

  A single round execution explanation:
  x = 1AF2345FF23B0001

  <u>digit - counter of appearances of the digit in x:</u>
  0 - 3
  1 - 2
  2 - 2
  3 - 2
  4 - 1
  5 - 1
  6 - 0
  7 - 0
  8 - 0

9 - 0
A - 1
B - 1
C - 0
D - 0
E - 0
F - 3

Now we have y = 3222110000110003
Since y does not equal x, if the required number of rounds is greater than 1,
execute the next round with x <– 3222110000110003

What We Provide

The attached files:

- Makefile
- main1.c
- task1.s

For Part 1, you only need to edit task1.s in the location indicated in the file, as in Assignment 0. For Part 2, you are required to create 2 new files: main2.c and task2.s. These files will contain the code for Part 2 as described above. You are also required to modify the supplied Makefile so that main2.c and task2.s will be compiled in a manner similar to Part 1.

In order to compile the files for the first part, you only need to use the 'make' utility. In order to compile the files for the second part, you will have to modify the 'Makefile'. You can take example from the way Part 1 is compiled. Make sure the resulting files are in the appropriate directories as described for the first part. Also, make sure the name of the executable for the second part is 'task2.bin'.

Submission Instructions

You are to submit a single zip file, **ID1_ID2.zip** , includes 4 files: main1.c, task1.s, main2.c and task2.s . Do not add new directory structure to the zip file! Make sure you follow the coding and submission instructions correctly (print exactly as requested).

Good Luck!