题目描述：

实现一种整数编码方法，使得待编码的数字越小，编码后所占用的字节数越小。

编码规则如下：

1、编码时 7 位一组，每个字节的低 7 位用于存储待编码数字的补码。

2、字节的最高位表示后续是否还有字节，置 1 表示后面还有更多的字节，置 0 表示当前字节为最后一个字节。

3、采用小端序编码，低位和低字节放在低地址上。

3、编码结果按 16 进制数的字符格式输出，小写字母需转换为大写字母。

输入描述：

输入的为一个字符串表示的非负整数

输出描述：

输出一个字符串，表示整数编码的 16 进制码流

补充说明：

待编码的数字取值范围为[0, 1<<64 - 1]

示例1
输入：0
输出：00
说明：输出的16进制字符，不足两位的前面补0，如00、01、02。

示例2
输入：100
输出：64
说明：100的二进制表示为0110 0100，只需要一个字节进行编码；
字节的最高位置0，剩余7位存储数字100的低7位（110 0100），所以编码后的输出为64。

示例3
输入：1000
输出：E807
说明：1000的二进制表示为0011 1110 1000，至少需要两个字节进行编码；
第一个字节最高位置1，剩余的7位存储数字1000的第一个低7位（110 1000），所以第一个字节的二进制为1110 1000，即E8；
第二个字节最高位置0，剩余的7位存储数字1000的第二个低7位（000 0111），所以第一个字节的二进制为0000 0111，即07；
采用小端序编码，所以低字节E8输出在前，高字节07输出在后。

```
#include <iostream>
#include <cstdio>
#include<stdint.h>
```

```
using namespace std;

uint64_t trans(uint64_t b)
{
	uint64_t ret=0;
	while (b > 0)
	{
		int temp = b & 0x7f;
		ret = (ret << 8) | temp;
		b = b >> 7;
		if (b > 0)
		{
			ret = (ret | 0x80);
		}
	}
	return ret;
}
int main() {
	uint64_t a;
	cin >> a;
	//uint64_t mask = uint64_t(1 << 56);
	uint64_t mask = 1;
	mask = mask << 56;
	if (a < (mask))
	{
		uint64_t ret= trans(a);
		if (ret < (1 << 16))
		{
			printf("%02X", ret);
		}
		else
		{
			printf("%llX", ret);
		}
	}
	else
	{
		uint64_t b = (a&(mask - 1));
		uint64_t ret = trans(b);
		ret = ret | 0x80;
		printf("%llX", ret);
		uint64_t c = a >> 56;
		ret = trans(c);
		printf("%llX", ret);
```

```
    }
}
```