#----------------------------------------------------------------------------------

#Rutvik Parmar

#H-41

#Date – 15th March 2016

#Assignement No:2 – S-DES Implementation

#----------------------------------------------------------------------------------

**import** java.util.\*;  
  
*/\*\* This class Generated two 8-bit subkeys from 10-bit input key \*\*/***class** KeyGeneration  
{  
 **private int**[] **key** = **new int**[10];  
 **private int**[] **k1** = **new int**[8];  
 **private int**[] **k2** = **new int**[8];  
 **private boolean flag** = **false**;  
  
 KeyGeneration()  
 {  
  
 }  
  
 **void** GenerateKeys(String inputkey )  
 {  
 **int**[] key = **new int**[10];  
 */\*  
 int temp;  
 for(int i=9;i>=0;i--)  
 {  
 temp = inputkey % 10;  
 key[i] = temp;  
 if(temp != 0 && temp != 1)  
 {  
 System.out.println("-- Error Occured : Invalid Key ");  
 System.exit(0);  
 return;  
 }   
 inputkey = inputkey/10;  
 }  
 \*/* **char** c1;  
 String ts ;  
  
 **try** {  
 **for**(**int** i=0;i<10;i++)  
 {  
 c1 = inputkey.charAt(i);  
 ts = Character.*toString*(c1);  
 key[i] = Integer.*parseInt*(ts);  
  
 **if**(key[i] !=0 && key[i]!=1)  
 {  
 Print.*msg*(**"\n .. Invalid Key .."**);  
 System.*exit*(0);  
 **return** ;  
 }  
 }  
 }  
 **catch**(Exception e)  
 {  
 Print.*msg*(**"\n .. Invalid Key .. "**);  
 System.*exit*(0);  
 **return** ;  
  
 }  
 **this**.**key** = key;  
  
 Print.*msg*(**"Input Key : "**);  
 Print.*array*(**this**.**key**,10);  
 Print.*msg*(**"\n"**);  
  
 permutationP10();  
  
 Print.*msg*(**"After Permutation(P10) Key : "**);  
 Print.*array*(**this**.**key**,10);  
 Print.*msg*(**"\n"**);  
  
 leftshiftLS1();  
  
 Print.*msg*(**"After LeftShift LS-1 Key : "**);  
 Print.*array*(**this**.**key**,10);  
 Print.*msg*(**"\n"**);  
  
  
 **this**.**k1** = permutationP8();  
  
 Print.*msg*(**"Subkey K1 Generated : "**);  
 Print.*array*(**this**.**k1**,8);  
 Print.*msg*(**"\n"**);  
  
 leftshiftLS2();  
  
 Print.*msg*(**"After LeftShift LS-2 Key : "**);  
 Print.*array*(**this**.**key**,10);  
 Print.*msg*(**"\n"**);  
  
 **this**.**k2** = permutationP8();  
 Print.*msg*(**"Subkey K2 Generated : "**);  
 Print.*array*(**this**.**k2**,8);  
 Print.*msg*(**"\n"**);  
  
 **flag** = **true**;  
  
 }  
  
 */\*\* Perform permutation P10 on 10-bit key  
 P10(k1, k2, k3, k4, k5, k6, k7, k8, k9, k10) = (k3, k5, k2, k7, k4, k10, k1, k9, k8, k6)  
 \*\*/* **private void** permutationP10()  
 {  
 **int**[] temp = **new int**[10];  
  
 temp[0] = **key**[2];  
 temp[1] = **key**[4];  
 temp[2] = **key**[1];  
 temp[3] = **key**[6];  
 temp[4] = **key**[3];  
 temp[5] = **key**[9];  
 temp[6] = **key**[0];  
 temp[7] = **key**[8];  
 temp[8] = **key**[7];  
 temp[9] = **key**[5];  
  
  
 **key** = temp;  
  
 }  
  
 */\*\* Performs a circular left shift (LS-1), or rotation, separately on the first  
 five bits and the second five bits. \*\*/* **private void** leftshiftLS1()  
 {  
 **int**[] temp = **new int**[10];  
  
 temp[0] = **key**[1];  
 temp[1] = **key**[2];  
 temp[2] = **key**[3];  
 temp[3] = **key**[4];  
 temp[4] = **key**[0];  
  
 temp[5] = **key**[6];  
 temp[6] = **key**[7];  
 temp[7] = **key**[8];  
 temp[8] = **key**[9];  
 temp[9] = **key**[5];  
  
 **key** = temp;  
  
 }  
  
 */\*\* apply Permutaion P8, which picks out and permutes 8 of the 10 bits according to the following  
 rule: P8[ 6 3 7 4 8 5 10 9 ] , 8-bit subkey is returned \*\*/* **private int**[] permutationP8()  
 {  
 **int**[] temp = **new int**[8];  
  
 temp[0] = **key**[5];  
 temp[1] = **key**[2];  
 temp[2] = **key**[6];  
 temp[3] = **key**[3];  
 temp[4] = **key**[7];  
 temp[5] = **key**[4];  
 temp[6] = **key**[9];  
 temp[7] = **key**[8];  
  
 **return** temp;  
  
 }  
  
  
 **private void** leftshiftLS2()  
 {  
 **int**[] temp = **new int**[10];  
  
 temp[0] = **key**[2];  
 temp[1] = **key**[3];  
 temp[2] = **key**[4];  
 temp[3] = **key**[0];  
 temp[4] = **key**[1];  
  
 temp[5] = **key**[7];  
 temp[6] = **key**[8];  
 temp[7] = **key**[9];  
 temp[8] = **key**[5];  
 temp[9] = **key**[6];  
  
 **key** = temp;  
  
 }  
  
  
 **public int**[] getK1()  
 {  
 **if**(!**flag**)  
 {  
 Print.*msg*(**"\nError Occured: Keys are not generated yet "**);  
 **return null**;  
 }  
 **return k1**;  
 }  
  
 **public int**[] getK2()  
 {  
 **if**(!**flag**)  
 {  
 Print.*msg*(**"\nError Occured: Keys are not generated yet "**);  
 **return null**;  
 }  
 **return k2**;  
 }  
  
}  
  
  
**class** Encryption  
{  
 **private int**[] **K1** = **new int**[8];  
 **private int**[] **K2** = **new int**[8];  
 **private int**[] **pt** = **new int**[8];  
  
 **void** SaveParameters(String plaintext , **int**[] k1, **int**[] k2)  
 {  
 **int**[] pt = **new int**[8];  
   
 */\*  
 int temp;  
 for(int i=7;i>=0;i--)   
 {  
 temp = plaintext % 10;  
 pt[i] = temp;  
 if(temp != 0 && temp != 1)  
 {  
 System.out.println("-- Error Occured : please enter valid 8-bit plaintext ");  
 System.exit(0);  
 return;  
 }   
 plaintext = plaintext/10;  
 }  
 \*/* **char** c1;  
 String ts ;  
  
 **try** {  
 **for**(**int** i=0;i<8;i++)  
 {  
 c1 = plaintext.charAt(i);  
 ts = Character.*toString*(c1);  
 pt[i] = Integer.*parseInt*(ts);  
  
 **if**(pt[i] !=0 && pt[i]!=1)  
 {  
 Print.*msg*(**"\n .. Invalid Plaintext .."**);  
 System.*exit*(0);  
 **return** ;  
 }  
 }  
 }  
 **catch**(Exception e)  
 {  
 Print.*msg*(**"\n .. Invalid Plaintext .. "**);  
 System.*exit*(0);  
 **return** ;  
  
 }  
  
 **this**.**pt** = pt;  
  
 Print.*msg*(**"Plaintext array : "**);  
 Print.*array*(**this**.**pt**,8);  
 Print.*msg*(**"\n"**);  
  
 **this**.**K1** = k1;  
 **this**.**K2** = k2;  
  
 *//Print.array(K1,8);  
 //Print.msg("\n");  
 //Print.array(K2,8);* }  
  
 */\*\* perform Initial Permutation in following manner [2 6 3 1 4 8 5 7] \*\*/* **void** InitialPermutation()  
 {  
 **int**[] temp = **new int**[8];  
  
 temp[0] = **pt**[1];  
 temp[1] = **pt**[5];  
 temp[2] = **pt**[2];  
 temp[3] = **pt**[0];  
 temp[4] = **pt**[3];  
 temp[5] = **pt**[7];  
 temp[6] = **pt**[4];  
 temp[7] = **pt**[6];  
  
 **pt** = temp;  
  
 Print.*msg*(**"Initial Permutaion(IP) : "**);  
 Print.*array*(**this**.**pt**,8);  
 Print.*msg*(**"\n"**);  
  
 }  
 **void** InverseInitialPermutation()  
 {  
 **int**[] temp = **new int**[8];  
  
 temp[0] = **pt**[3];  
 temp[1] = **pt**[0];  
 temp[2] = **pt**[2];  
 temp[3] = **pt**[4];  
 temp[4] = **pt**[6];  
 temp[5] = **pt**[1];  
 temp[6] = **pt**[7];  
 temp[7] = **pt**[5];  
  
 **pt** = temp;  
  
  
 }  
  
 */\*\* mappingF . arguments 4-bit right-half of plaintext & 8-bit subkey \*\*/* **int**[] mappingF(**int**[] R, **int**[] SK)  
 {  
 **int**[] temp = **new int**[8];  
  
 *// EXPANSION/PERMUTATION [4 1 2 3 2 3 4 1]* temp[0] = R[3];  
 temp[1] = R[0];  
 temp[2] = R[1];  
 temp[3] = R[2];  
 temp[4] = R[1];  
 temp[5] = R[2];  
 temp[6] = R[3];  
 temp[7] = R[0];  
  
 Print.*msg*(**"EXPANSION/PERMUTATION on RH : "**);  
 Print.*array*(temp,8);  
 Print.*msg*(**"\n"**);  
  
 *// Bit by bit XOR with sub-key* temp[0] = temp[0] ^ SK[0];  
 temp[1] = temp[1] ^ SK[1];  
 temp[2] = temp[2] ^ SK[2];  
 temp[3] = temp[3] ^ SK[3];  
 temp[4] = temp[4] ^ SK[4];  
 temp[5] = temp[5] ^ SK[5];  
 temp[6] = temp[6] ^ SK[6];  
 temp[7] = temp[7] ^ SK[7];  
  
 Print.*msg*(**"XOR With Key : "**);  
 Print.*array*(temp,8);  
 Print.*msg*(**"\n"**);  
  
 *// S-Boxes* **final int**[][] S0 = { {1,0,3,2} , {3,2,1,0} , {0,2,1,3} , {3,1,3,2} } ;  
 **final int**[][] S1 = { {0,1,2,3}, {2,0,1,3}, {3,0,1,0}, {2,1,0,3}} ;  
  
  
 **int** d11 = temp[0]; *// first bit of first half* **int** d14 = temp[3]; *// fourth bit of first half* **int** row1 = BinaryOp.*BinToDec*(d11,d14); *// for input in s-box S0* **int** d12 = temp[1]; *// second bit of first half* **int** d13 = temp[2]; *// third bit of first half* **int** col1 = BinaryOp.*BinToDec*(d12,d13); *// for input in s-box S0* **int** o1 = S0[row1][col1];  
  
 **int**[] out1 = BinaryOp.*DecToBinArr*(o1);  
  
 Print.*msg*(**"S-Box S0: "**);  
 Print.*array*(out1,2);  
 Print.*msg*(**"\n"**);  
  
 **int** d21 = temp[4]; *// first bit of second half* **int** d24 = temp[7]; *// fourth bit of second half* **int** row2 = BinaryOp.*BinToDec*(d21,d24);  
  
 **int** d22 = temp[5]; *// second bit of second half* **int** d23 = temp[6]; *// third bit of second half* **int** col2 = BinaryOp.*BinToDec*(d22,d23);  
  
 **int** o2 = S1[row2][col2];  
  
 **int**[] out2 = BinaryOp.*DecToBinArr*(o2);  
  
 Print.*msg*(**"S-Box S1: "**);  
 Print.*array*(out2,2);  
 Print.*msg*(**"\n"**);  
  
 *//4 output bits from 2 s-boxes* **int**[] out = **new int**[4];  
 out[0] = out1[0];  
 out[1] = out1[1];  
 out[2] = out2[0];  
 out[3] = out2[1];  
  
 *//permutation P4 [2 4 3 1]* **int** [] O\_Per = **new int**[4];  
 O\_Per[0] = out[1];  
 O\_Per[1] = out[3];  
 O\_Per[2] = out[2];  
 O\_Per[3] = out[0];  
  
 Print.*msg*(**"Output of mappingF : "**);  
 Print.*array*(O\_Per,4);  
 Print.*msg*(**"\n"**);  
  
 **return** O\_Per;  
 }  
  
 */\*\* fK(L, R, SK) = (L (XOR) mappingF(R, SK), R) .. returns 8-bit output\*\*/* **int**[] functionFk(**int**[] L, **int**[] R,**int**[] SK)  
 {  
 **int**[] temp = **new int**[4];  
 **int**[] out = **new int**[8];  
  
  
 temp = mappingF(R,SK);  
  
  
 *//XOR left half with output of mappingF* out[0] = L[0] ^ temp[0];  
 out[1] = L[1] ^ temp[1];  
 out[2] = L[2] ^ temp[2];  
 out[3] = L[3] ^ temp[3];  
  
 out[4] = R[0];  
 out[5] = R[1];  
 out[6] = R[2];  
 out[7] = R[3];  
  
  
 **return** out;  
  
  
 }  
  
 */\*\* switch function (SW) interchanges the left and right 4 bits \*\*/* **int**[] switchSW(**int**[] in)  
 {  
  
 **int**[] temp = **new int**[8];  
  
 temp[0] = in[4];  
 temp[1] = in[5];  
 temp[2] = in[6];  
 temp[3] = in[7];  
  
 temp[4] = in[0];  
 temp[5] = in[1];  
 temp[6] = in[2];  
 temp[7] = in[3];  
  
 **return** temp;  
 }  
  
 **int**[] encrypt(String plaintext , **int**[] LK, **int**[] RK)  
 {  
  
  
 SaveParameters(plaintext,LK,RK);  
  
 Print.*msg*(**"\n---------------------------------------\n"**);  
 InitialPermutation();  
 Print.*msg*(**"\n---------------------------------------\n"**);  
 *//saperate left half & right half from 8-bit pt* **int**[] LH = **new int**[4];  
 **int**[] RH = **new int**[4];  
 LH[0] = **pt**[0];  
 LH[1] = **pt**[1];  
 LH[2] = **pt**[2];  
 LH[3] = **pt**[3];  
  
  
 RH[0] = **pt**[4];  
 RH[1] = **pt**[5];  
 RH[2] = **pt**[6];  
 RH[3] = **pt**[7];  
  
  
 Print.*msg*(**"First Round LH : "**);  
 Print.*array*(LH,4);  
 Print.*msg*(**"\n"**);  
  
 Print.*msg*(**"First Round RH: "**);  
 Print.*array*(RH,4);  
 Print.*msg*(**"\n"**);  
  
 *//first round with sub-key K1* **int**[] r1 = **new int**[8];  
 r1 = functionFk(LH,RH,**K1**);  
  
 Print.*msg*(**"After First Round : "**);  
 Print.*array*(r1,8);  
 Print.*msg*(**"\n"**);  
 Print.*msg*(**"\n---------------------------------------\n"**);  
 *//Switch the left half & right half of about output* **int**[] temp = **new int**[8];  
 temp = switchSW(r1);  
  
 Print.*msg*(**"After Switch Function : "**);  
 Print.*array*(temp,8);  
 Print.*msg*(**"\n"**);  
 Print.*msg*(**"\n---------------------------------------\n"**);  
 *// again saperate left half & right half for second round* LH[0] = temp[0];  
 LH[1] = temp[1];  
 LH[2] = temp[2];  
 LH[3] = temp[3];  
  
 RH[0] = temp[4];  
 RH[1] = temp[5];  
 RH[2] = temp[6];  
 RH[3] = temp[7];  
  
  
 Print.*msg*(**"Second Round LH : "**);  
 Print.*array*(LH,4);  
 Print.*msg*(**"\n"**);  
  
 Print.*msg*(**"Second Round RH: "**);  
 Print.*array*(RH,4);  
 Print.*msg*(**"\n"**);  
  
  
 *//second round with sub-key K2* **int**[] r2 = **new int**[8];  
 r2 = functionFk(LH,RH,**K2**);  
  
 **pt** = r2;  
  
 Print.*msg*(**"After Second Round : "**);  
 Print.*array*(**this**.**pt**,8);  
 Print.*msg*(**"\n"**);  
 Print.*msg*(**"\n---------------------------------------\n"**);  
  
 InverseInitialPermutation();  
  
 Print.*msg*(**"After Inverse IP (Result) : "**);  
 Print.*array*(**this**.**pt**,8);  
 Print.*msg*(**"\n"**);  
  
 *//Encryption done... return 8-bit output .* **return pt**;  
  
  
  
  
 }  
  
}  
  
  
**public class** SDES  
{  
 **public static void** main(String[] args)  
 {  
  
 KeyGeneration KG = **new** KeyGeneration();  
 Encryption enc = **new** Encryption();  
 Scanner sc = **new** Scanner(System.***in***);  
  
 String pt ;  
 String key;  
 **int**[] ct = **new int**[8];  
  
 **try** {  
  
  
 *//Ex Input : 10101010* System.***out***.print(**"Enter 8-bit Plaintext : "**);  
 pt = sc.next();  
  
  
 System.***out***.println(**" \n "**);  
  
 *//Ex Input : 1010000010* System.***out***.print(**"Enter 10-bit Key : "**);  
 key = sc.next();  
  
  
 System.***out***.println(**" \n "**);  
  
 Print.*msg*(**"\n Key Generation ...\n"**);  
 Print.*msg*(**"\n---------------------------------------\n"**);  
 KG.GenerateKeys(key);  
 Print.*msg*(**"\n---------------------------------------\n"**);  
 ct = enc.encrypt( pt ,KG.getK1(),KG.getK2());  
  
 Print.*msg*(**"\n---------------------------------------\n"**);  
 System.***out***.println(**" \n Decryption "**);  
  
 *//Ex Input : 10001101* System.***out***.print(**"Enter 8-bit Ciphertext : "**);  
 pt = sc.next();  
  
  
 System.***out***.println(**" \n "**);  
  
 *//Ex Input : 1010000010* System.***out***.print(**"Enter 10-bit Key : "**);  
 key = sc.next();  
  
  
 System.***out***.println(**" \n "**);  
  
 Print.*msg*(**"\n Key Generation ...\n"**);  
 Print.*msg*(**"\n---------------------------------------\n"**);  
 Print.*msg*(**"\n For decryption Two Sub-keys will be used in reverse order \n"**);  
 Print.*msg*(**"\n---------------------------------------\n\n"**);  
 KG.GenerateKeys(key);  
 Print.*msg*(**"\n---------------------------------------\n"**);  
  
 ct = enc.encrypt( pt ,KG.getK2(),KG.getK1());  
  
 Print.*msg*(**"\n---------------------------------------\n"**);  
  
  
  
  
  
 }  
 **catch**(InputMismatchException e)  
 {  
 System.***out***.println(**"-- Error Occured : Invalid Input "**);  
 }  
 **catch**(Exception e)  
 {  
 System.***out***.println(**"-- Error Occured : "**+e);  
 }  
  
 }  
  
}  
  
  
*/\*\* Class to print Strings & arrays shortly \*\*/***class** Print  
{  
 */\*\* Prints array to console \*\*/* **static void** array(**int**[] arr,**int** len)  
 {  
 System.***out***.print(**" - "**);  
  
 **for**(**int** i=0;i<len;i++)  
 {  
 System.***out***.print(arr[i] + **" "**);  
 }  
 }  
  
 **static void** msg(String msg)  
 {  
 System.***out***.print(msg);  
 }  
}  
  
**class** BinaryOp  
{  
 */\*\* Gets binary digits as arguments & returns decimal number  
 for example input args [1,0,0] will return 4 \*\*/* **static int** BinToDec(**int**...bits)  
 {  
  
  
 **int** temp=0;  
 **int** base = 1;  
 **for**(**int** i=bits.**length**-1 ; i>=0;i--)  
 {  
 temp = temp + (bits[i]\*base);  
 base = base \* 2 ;  
 }  
  
 **return** temp;  
 }  
  
 */\*\* gets decimal number as argument and returns array of binary bits  
 for example input arg [10] will return [1,0,1,0]\*\*/* **static int**[] DecToBinArr(**int** no)  
 {  
 *// 13 1  
 // 6 0  
 // 3 1  
 // 1 1  
 // 0* **if**(no==0)  
 {  
 **int**[] zero = **new int**[2];  
 zero[0] = 0;  
 zero[1] = 0;  
 **return** zero;  
 }  
 **int**[] temp = **new int**[10] ;  
  
  
 **int** count = 0 ;  
 **for**(**int** i= 0 ; no!= 0 ; i++)  
 {  
 temp[i] = no % 2;  
 no = no/2;  
 count++;  
 }  
  
  
 **int**[] temp2 = **new int**[count];  
  
  
 **for**(**int** i=count-1, j=0;i>=0 && j<count;i--,j++)  
 {  
 temp2[j] = temp[i];  
 }  
  
 *//because we requires 2-bits as output .. so for adding leading 0* **if**(count<2)  
 {  
 temp = **new int**[2];  
 temp[0] = 0;  
 temp[1] = temp2[0];  
 **return** temp;  
 }  
  
 **return** temp2;  
 }  
}

Output –

Enter 8-bit Plaintext : 10101010

Enter 10-bit Key : 1010000010

Key Generation ...

---------------------------------------

Input Key : - 1 0 1 0 0 0 0 0 1 0

After Permutation(P10) Key : - 1 0 0 0 0 0 1 1 0 0

After LeftShift LS-1 Key : - 0 0 0 0 1 1 1 0 0 0

Subkey K1 Generated : - 1 0 1 0 0 1 0 0

After LeftShift LS-2 Key : - 0 0 1 0 0 0 0 0 1 1

Subkey K2 Generated : - 0 1 0 0 0 0 1 1

---------------------------------------

Plaintext array : - 1 0 1 0 1 0 1 0

---------------------------------------

Initial Permutaion(IP) : - 0 0 1 1 0 0 1 1

---------------------------------------

First Round LH : - 0 0 1 1

First Round RH: - 0 0 1 1

EXPANSION/PERMUTATION on RH : - 1 0 0 1 0 1 1 0

XOR With Key : - 0 0 1 1 0 0 1 0

S-Box S0: - 1 0

S-Box S1: - 0 1

Output of mappingF : - 0 1 0 1

After First Round : - 0 1 1 0 0 0 1 1

---------------------------------------

After Switch Function : - 0 0 1 1 0 1 1 0

---------------------------------------

Second Round LH : - 0 0 1 1

Second Round RH: - 0 1 1 0

EXPANSION/PERMUTATION on RH : - 0 0 1 1 1 1 0 0

XOR With Key : - 0 1 1 1 1 1 1 1

S-Box S0: - 0 0

S-Box S1: - 1 1

Output of mappingF : - 0 1 1 0

After Second Round : - 0 1 0 1 0 1 1 0

---------------------------------------

After Inverse IP (Result) : - 1 0 0 0 1 1 0 1

---------------------------------------

Decryption

Enter 8-bit Ciphertext : 10001101

Enter 10-bit Key : 1010000010

Key Generation ...

---------------------------------------

For decryption Two Sub-keys will be used in reverse order

---------------------------------------

Input Key : - 1 0 1 0 0 0 0 0 1 0

After Permutation(P10) Key : - 1 0 0 0 0 0 1 1 0 0

After LeftShift LS-1 Key : - 0 0 0 0 1 1 1 0 0 0

Subkey K1 Generated : - 1 0 1 0 0 1 0 0

After LeftShift LS-2 Key : - 0 0 1 0 0 0 0 0 1 1

Subkey K2 Generated : - 0 1 0 0 0 0 1 1

---------------------------------------

Plaintext array : - 1 0 0 0 1 1 0 1

---------------------------------------

Initial Permutaion(IP) : - 0 1 0 1 0 1 1 0

---------------------------------------

First Round LH : - 0 1 0 1

First Round RH: - 0 1 1 0

EXPANSION/PERMUTATION on RH : - 0 0 1 1 1 1 0 0

XOR With Key : - 0 1 1 1 1 1 1 1

S-Box S0: - 0 0

S-Box S1: - 1 1

Output of mappingF : - 0 1 1 0

After First Round : - 0 0 1 1 0 1 1 0

---------------------------------------

After Switch Function : - 0 1 1 0 0 0 1 1

---------------------------------------

Second Round LH : - 0 1 1 0

Second Round RH: - 0 0 1 1

EXPANSION/PERMUTATION on RH : - 1 0 0 1 0 1 1 0

XOR With Key : - 0 0 1 1 0 0 1 0

S-Box S0: - 1 0

S-Box S1: - 0 1

Output of mappingF : - 0 1 0 1

After Second Round : - 0 0 1 1 0 0 1 1

---------------------------------------

After Inverse IP (Result) : - 1 0 1 0 1 0 1 0

---------------------------------------