1	**LDR (Load)**	**LDR R0 R1**	**000 Reg Reg**
	R0 receives value that is stored at memory address held in R1		
2	**STR (Store)**	**STR R0 R1**	**001 Reg Reg**
	R0 has its value stored in memory at the address held in R1		
3	**Bitwise AND**	**AND R0 R1**	**010 Reg Reg**
	Result is put in first input register (in this case, R0)		
4	**Bitwise XOR (also acts as CMP)**	**XOR R0 R1 R2** **XOR R0 R1**	**011 Reg Reg**
	Result is put in first input register (in this case, R0)		
5	**ADD**	**ADD R0 R0 R1** **ADD R0 R1 r0=r0+r1**	**100 Reg Reg**
	Result is put in first input register (in this case, R0)		
6	**LSL (Logical shift left)**	**LSL R0**	**101 000 Reg**
	Result is put into the input register (in this case, R0)		
7	**BEQ (Branch if equal)**	**BEQ R0 R1**	**110 Reg Reg**
	If previous ALU operation resulted in 0, branch by the value of R1		
8	**IMD /or MOV (Load immediate into Reg0)**	**MOV R0 R6** **MOV R5 R1** **IMD #12** **IMD LUT #8**	**111 SS* *****
	Four Cases: 111 00R Reg Move value from Reg into the register #R (rather 1 or 0) 111 01R Reg Move value from register #R (rather 1 or 0) into Reg 111 10* *** Load register R0 with the immediate * *** 111 11* *** Load register R0 with an immediate from a LUT at address * ***		

Program 1
Message in [0:51]

Preamble length [61]
LFSR tap seq [62]
Starting LFSR State [63]
Put encryptions starting in [64]

Setup:
Registers
- Start R1 with value 61
- Start R2 with value 0
- Start R4 with value 64
- Start R5 with value 54
- Start R6 with value 62
- Start R7 with value 63

Look-up table: [0-5]
- 128 //start of free space
- 15 //end_loop_line offset
- -18 //loop_line offset
- 16
- -21
- 0 //space ascii value
- 61

ARM Code:
1. LDR R7 R7
2. LDR R6 R6
3. IMD LUT #6
4. LDR R1 R0
5. IMD LUT #0 //#loop_line going to 128
6. STR R2 R0 //save val of R2 before branch check
7. IMD LUT #3 //load end_pre_loop value 16
8. XOR R2 R1 //counter for the message
9. BEQ R2 R0
10. IMD LUT #0 //#loop_line going to 128
11. LDR R2 R0 //load val of R2 after branch check
12. IMD LUT #5
13. MOV R3 R0 //load char
14. XOR R3 R7 //encrypt char
15. STR R3 R4 //store char
16. IMD #1
17. ADD R2 R0 //increment preamble number
18. ADD R4 R0 //increment store index
19. MOV R0 R7 //R0 = R7
20. AND R0 R6 //this line and next line will generate next lsfr
21. LSL R7 //NOTE: we need to shift in the parity bit flag in hardware
22. IMD LUT #4 //load pre_loop value

23. XOR R1 R1
24. BEQ R1 R0
25. IMD #0
26. MOV R2 R0 //R2 = 0
27. IMD LUT #0 //#loop_line going to 128
28. STR R2 R0 //save val of R2 before branch check
29. IMD LUT #1 //load end_loop_line value
30. XOR R2 R5 //counter for the message
31. BEQ R2 R0 // if we reached to the end
32. IMD LUT #0 //#loop_line going to 128
33. LDR R2 R0 //load val of R2 after branch check
34. LDR R3 R2 //load char
35. XOR R3 R7 //encrypt char
36. STR R3 R4 //store char
37. IMD #1
38. ADD R2 R0 //increment load index
39. ADD R4 R0 //increment store index
40. MOV R0 R7 //R0 = R7
41. AND R0 R6 //this line and next line will generate next lsfr
42. LSL R7 //NOTE: we need to shift in the parity bit flag in hardware
43. IMD LUT #2
44. XOR R1 R1
45. BEQ R1 R0
46. IMD #0 //#end_loop_line

Program 2

Setup:
Registers

- R1 = 64
- R4 = 9
- R7 = 73

Look-up table

0	-14
1	128
2	129
3	200
4	130

5	38
6	-41
7	64
8	0 //space
9	-11
10	118
11	74
12	-16
13	
14	
15	

Memory

- Mem[200] = 0x60 //0110 0000 //96
- Mem[201] = 0x48 //0100 1000 //72
- Mem[202] = 0x78 //0111 1000 //120
- Mem[203] = 0x72 //0111 0010 //114
- Mem[204] = 0x6A //0110 1010 //106
- Mem[205] = 0x69 //0110 1001 //105
- Mem[206] = 0x5C //0101 1100 //92
- Mem[207] = 0x7E //0111 1110 //126
- Mem[208] = 0x7B //0111 1011 //123

ARM Code:

1. LDR R7 R7 //load the address of 73
2. LDR R5 R1   //R5 = lsfr starting state
3. IMD LUT #7
4. MOV R1 R0
5. IMD LUT #1   //put the the addres 128
6. STR R2 R0   //Mem[128] = R2
7. IMD LUT #5   //for_loop1_end offset
8. XOR R2 R4   //checking if R2 == 9
9. BEQ R2 R0 //
10. IMD LUT #1 // r0 = 128
11. LDR R2 R0   //R2 = Mem[128]
12. IMD LUT #3 //ro =200

13. ADD R0 R2 //R0 = 200 + R2
14. LDR R6 R0 //load tap pattern number [R2] (loads patterns 0-8)
15. IMD LUT #7
16. MOV R1 R0
17. IMD LUT #2 //r0 = 129
18. STR R3 R0 // r3= mem[129]
19. IMD #11 //for_loop2_end offset
20. XOR R3 R4 // if r3 = =r4 (mem129 == 9)
21. BEQ R3 R0
22. IMD LUT #2 //ro = 129
23. LDR R3 R0 // r3 = mem(129)
24. IMD #1
25. ADD R3 R0
26. MOV R0 R5 //move lfsr states to r0
27. AND R0 R6 // and and shift
28. LSL R5 // add the lfsr back to address
29. IMD LUT #12
30. XOR R1 R1
31. BEQ R1 R0
32. IMD LUT #7
33. MOV R1 R0
34. IMD LUT #4
35. STR R5 R0
36. IMD #11 //for_loop1_end offset
37. XOR R5 R7
38. BEQ R5 R0
39. IMD LUT #4
40. LDR R5 R0
41. LDR R5 R1 //lsfr starting state
42. IMD #1
43. ADD R2 R0
44. IMD LUT #6
45. XOR R1 R1
46. BEQ R1 R0
47. IMD LUT #7
48. MOV R1 R0
49. IMD LUT #4
50. LDR R5 R0
51. MOV R0 R5 //R0 = R5
52. AND R0 R6 //this line and next line will generate next lsfr
53. LSL R5
54. IMD #0 //load 0
55. MOV R1 R0 //R1 = 0
56. MOV R0 R4 //R0 = 9

```
57. MOV R3 R0  //R3 = 9
58. IMD #8
59. XOR R3 R1  //r1 com to r3 r1 ==9?
60. BEQ R3 R0
61. IMD LUT  #8   //space
62. STR R0 R1   //put space in address 0
63. IMD #1    //to add
64. ADD R1 R0   //add address
65. IMD LUT #9  //branch value
66. XOR R3 R3 //unconditional branch address
67. BEQ R3 R0
68. IMD #1
69. ADD R4 R0
70. IMD LUT #11      // LOAD ADDRESS TO STOR R2  74 r2
71. MOV R2 R0    //save val of R2 before branch check
72. IMD #13      // BRANCH TOO
73. MOV R1 R0 // OPTATING THE BRANCH VALUE
74. IMD LUT  #10  // END OF MESAGE SHOULD BE #118
75. XOR R0 R2   //counter for the message
76. BEQ R0 R1 //IF EQUAL DONE  ////
77. LDR R3 R2    //load char
78. XOR R3 R5    //DECRYPT char
79. STR R3 R4    //store char
80. IMD #1
81. ADD R2 R0    //increment load index
82. ADD R4 R0    //increment store index
83. MOV R0 R5    //R0 = R5
84. AND R0 R6    //this line and next line will generate next lsfr
85. LSL R5    //NOTE: we need to shift in the parity bit flag in hardware
86. IMD LUT #12    //load pre_loop value
87. XOR R1 R1
88. BEQ R1 R0
89. IMD #0    //#end_loop_line
```

```
R5 = lsfr starting state = Mem[64]  //00000001
for(R2=0;R2<9;R2++){
        R6 = tap pattern [i]
                for(R4=0;R4<9;R4++){
                        MOV R0 R5
                        AND R0 R6
```

```
                    LSL R5
                }
                if(R5 = Mem[73]){
                    Break;
                }
        R5 = lsfr starting state
}
R5 = lsfr starting state
…decoding…
…decoding…
for (R2=0;R2 < 9 ; R2++){   //we know first 10 whould be space
mem[0] = " " //hex'20       //so putting space
}
RX =74    //the actual coded info
for(R2=9;R2 < 54 ;R2++ ){  // loop throug all
RX =   //decoding and advacing the lfsr
{
MOV R0 R5
AND R0 RX
LSL R5
}
}
```

```
x=200;
y=210;
r=” “;
w=64;
For (int i = 0 ; i <10 ; i++){

        For (int j = 0 ;j<10 ; j++){

For (int i = 0 ; i <10 ; i++){
        if( FSR_state[i+200] == {data_out^'h5f})
{LFSR tap seq = mem [62]
Starting LFSR State[i+200] = [63]

200-208 tap patterns
210-218 lsfr registers to modify
Mem[64-73] the first 10 lsfr states in our encrypted message
Starting lsfr state = Mem[64]
```

Program 3

Setup:
Registers

- R1 = 64
- R4 = 9
- R7 = 73

Look-up table

0	-14
1	128
2	129
3	200

4	130
5	38
6	-41
7	64
8	0 //space
9	-11
10	118
11	74
12	-16
13	-7
14	-1
15	-12

Memory

- Mem[200] = 0x60 //0110 0000 //96
- Mem[201] = 0x48 //0100 1000 //72
- Mem[202] = 0x78 //0111 1000 //120
- Mem[203] = 0x72 //0111 0010 //114
- Mem[204] = 0x6A //0110 1010 //106
- Mem[205] = 0x69 //0110 1001 //105
- Mem[206] = 0x5C //0101 1100 //92
- Mem[207] = 0x7E //0111 1110 //126
- Mem[208] = 0x7B //0111 1011 //123

ARM Code:

1. LDR R7 R7 //load the address of 73
2. LDR R5 R1   //R5 = lsfr starting state
3. IMD LUT #7
4. MOV R1 R0
5. IMD LUT #1   //put the the addres 128
6. STR R2 R0   //Mem[128] = R2
7. IMD LUT #5   //for_loop1_end offset
8. XOR R2 R4   //checking if R2 == 9
9. BEQ R2 R0 //
10. IMD LUT #1 // r0 = 128

11. LDR R2 R0   //R2 = Mem[128]
12. IMD LUT #3 //ro =200
13. ADD R0 R2   //R0 = 200 + R2
14. LDR R6 R0   //load tap pattern number [R2] (loads patterns 0-8)
15. IMD LUT #7
16. MOV R1 R0
17. IMD LUT #2 //r0 = 129
18. STR R3 R0 // r3= mem[129]
19. IMD #11   //for_loop2_end offset
20. XOR R3 R4   // if r3 = =r4   (mem129 == 9)
21. BEQ R3 R0
22. IMD LUT #2 //ro = 129
23. LDR R3 R0   // r3 = mem(129)
24. IMD #1
25. ADD R3 R0
26. MOV R0 R5 //move lfsr states to r0
27. AND R0 R6 // and and shift
28. LSL R5   // add the lfsr back to address
29. IMD LUT #12
30. XOR R1 R1
31. BEQ R1 R0
32. IMD LUT #7
33. MOV R1 R0
34. IMD LUT #4
35. STR R5 R0
36. IMD #11   //for_loop1_end offset
37. XOR R5 R7
38. BEQ R5 R0
39. IMD LUT #4
40. LDR R5 R0
41. LDR R5 R1   //lsfr starting state
42. IMD #1
43. ADD R2 R0
44. IMD LUT #6
45. XOR R1 R1
46. BEQ R1 R0
47. IMD LUT #7
48. MOV R1 R0
49. IMD LUT #4
50. LDR R5 R0
51. MOV R0 R5   //R0 = R5
52. AND R0 R6   //this line and next line will generate next lsfr
53. LSL R5
54. IMD #0   //load 0

```
55. MOV R1 R0   //R1 = 0
56. MOV R0 R4    //R0 = 9
57. MOV R3 R0  //R3 = 9
58. IMD #8
59. XOR R3 R1  //r1 com to r3 r1 ==9?
60. BEQ R3 R0
61. IMD LUT  #8   //space
62. STR R0 R1   //put space in address 0
63. IMD #1    //to add
64. ADD R1 R0   //add address
65. IMD LUT #9  //branch value
66. XOR R3 R3 //unconditional branch address
67. BEQ R3 R0
68. IMD #1
69. ADD R4 R0
70. IMD LUT #11      // LOAD ADDRESS TO STOR R2  74 r2
71. MOV R2 R0    //save val of R2 before branch check
72. IMD #13      // BRANCH TOO
73. MOV R1 R0 // OPTATING THE BRANCH VALUE
74. IMD LUT  #10  // END OF MESAGE SHOULD BE #118
75. XOR R0 R2   //counter for the message
76. BEQ R0 R1 //IF EQUAL DONE  ////
77. LDR R3 R2    //load char
78. XOR R3 R5    //DECRYPT char
79. STR R3 R4    //store char
80. IMD #1
81. ADD R2 R0    //increment load index
82. ADD R4 R0    //increment store index
83. MOV R0 R5    //R0 = R5
84. AND R0 R6    //this line and next line will generate next lsfr
85. LSL R5    //NOTE: we need to shift in the parity bit flag in hardware
86. IMD LUT #12    //load pre_loop value
87. XOR R1 R1
88. BEQ R1 R0
89.
90. IMD #14    //#end_loop_line
91. MOV R1 R0 //R1 =-1
92. IMD #1
93. ADD R1 R0 //R1=0
94. LDR R2 R1 //R2=[MEM[R0]
95. IMD LUT #8
96. MOV R3 R0
97. IMD LUT #13
98. XOR R2  R3
```

```
99. BEQ R2 R0
100.    IMD LUT #14
101.    ADD R1 R0
102.    MOV R2 R0 //R2 =-1
103.    IMD LUT #7
104.    MOV R4 R0
105.    IMD #9
106.    XOR R4 R1
107.    BEQ  R4 R0
108.    IMD #1
109.    ADD R2 R0 //R2=0
110.    ADD R1 R0 //R1=ORIGINAL
111.    LDR R3 R1
112.    STR R3 R2
113.    IMD LUT #15
114.    XOR R5 R5
115.    BEQ  R5 R0
116.    IMD #0
```