CSE141L

LAB1

Hyunwoo Choi

A91407351

0. Team.

Hyunwoo Choi A91407351

1. Introduction.

Name of the Architecture: NICE (Nine Is Certainly Enough)

This architecture is using a hybrid of reg-reg and accumulator architecture.

Most of the instructions executes with the form of reg-reg. However, Using a 6-bit register is very limited for the operation, some opcodes support accumulator architecture.

Registers

Registers	BITS	Type	Description
R0	000	Zero	Unwriteable reg Initialized as 0
R1	001	General	General reg
R2	010	General	General reg
R3	011	General	General reg
R4	100	General	General reg
R5	101	General	General reg
R6	110	General	General reg
R7	111	General & Accumulator	Accumulator for BNE, RSH, SET, MOV

Operation

COMMAND	OPCODE	DESCRIPTION	Machine Action
ADD	000	1. **Add** 2. **Increment value** (RaddrA == RaddrB) **3. MOV** (RaddrB == 1111)	1. Reg1 = Reg1+ Reg2 2. Reg1 = Reg1+1 3. Reg1 = acc
SUB	001	**Subtract**	Reg1 = Reg1 - Reg2
XOR	010	**1. XOR** 2. **Right Shift** (RaddrA == 000)	1. Reg1= Reg1^Reg2 2. Acc = Acc << Imm
LDR	011	Load value from memory	Reg1 = memory[Reg2]
STR	100	Store value in memory	Memory[Reg2] = Reg1
BGE	101	1. BNE (RaddrA == 000) 2. BGE	1. if(R7 != 0) branch enabled 2. if(R7 >= Reg1) branch enabled
AND	110	Store the value of accumulator to register	Mem[addr] = Acc
SET	111	Set immediate value in Acc	Acc = imm

REG-REG instruction

Opcode 3-bit	Reg 1	Reg 2
000 to 111	000 - 111	000_111

Opcode 3-bit	Reg 1	Reg 2	Description
XOR	110	001	R6 = R6 ^ R1

Accumulator instruction

Opcode 3-bit	Immediate value	Description
SET	0 to 63 decimal value	R7 = decimal value

Opcode 3-bit	REG 1 (3 bits)	LUT selector (3 bits)	Description
BGE	001 to 111	000 to 111	if(R7 >= R1) Branch enabled (BGE)

Opcode 3-bit	Offset	LUT selector (3 bits)	Description
BGE	000	000 to 111	if(R7 != 0) Branch enabled (BNE)

Opcode 3-bit	Offset	Immediate	Description
XOR	000	000 to 111	R7 << imm (RHL)

Opcode 3-bit	Immediate value	Description
SET	0 to 63 decimal value	R7 = decimal value

Control Flow

The architecture implemented an absolute addressing for the program counter (when the branch is enabled) rather than adding the offset because using absolute address could be more efficient for the LUT tables than those of another. Moreover, this method was very intuitive to translate assembly code to the machine code.
For example, the assembly codes for program 2 has branches like following:

```
L1: #1
    LDR r3, mem[r1]     // r3 = lfsr_pttrn[n]
    LDR r6, mem[62]     // r6 = lfsr
L2: #2
// inner loop
    SET R7, #64         // R7 = 64
    ADD R7,R2           // R7 = 64 + n
    LDR R7,R7           // R7 = mem[64+n]
    AND R7,R5           // R7 = mem[64+n] & 0x7f
    XOR R7,R6           // R7 = mem[64+n] ^ lfsr
    BGE R0,L2_END       // if(r7 !=0)

    SET R7, #9          // R7 = 0
    SUB R7, R2          // R7 = R7 - 9
    BGE R0, L2_2        // if(r2 != 9) jump

    SET R7, #62         // found lfsr_pttrn
    STR R3, R7          // mem[62] = lfsr_pttrn

L2_2: #3
    SET R7, #0          // R7 = 0
    ADD R7, R6          // R7 = lfsr
    XOR R7, #1          // R7 = lfsr << 1
    AND R7, R5          // R7 = (lfsr << 1) & 0x7F
    MOV R4, R7          // R4 = R7
```

```
module LUT(
  input       [ 2:0] Addr,
  output logic[ 9:0] Target
  );

always_comb begin
  Target = 10'h001;    // default to 1
  case(Addr)
  3'b000:  Target = 10'd40;
  3'b001:  Target = 10'd11; // L1
  3'b010:  Target = 10'd14; // L2
  3'b011:  Target = 10'd29; // L2_2
  3'b100:  Target = 10'd45; // L2 end
  3'b101:  Target = 10'd57; // L3
  3'b110:  Target = 10'h6;
  3'b111:  Target = 10'h6;
  endcase
end

endmodule
```

```
module InstFetch(
  input             Reset, // if Start
                    Clk,
                    Branch,
            Start,
            ALU_flag,
            Halt,
  input       [9:0] Target, Start_addr
  output logic [9:0] PC
  );

  always_ff @(posedge Clk)
    if(Reset)
      PC <= 0;
    else if(Start)
      PC <= PC; // hold PC
        else if(Halt)
      PC <= PC;
    else if(Branch && ALU_flag)
      PC <= Target;
```

In the Instruction Fetch Module, at the posedge of branch_enable, the address is assigned to the Program counter.

Program 1 Output:

```
run encryption program; original message =
 Knowledge comes, but wisdom lingers.
LFSR_ptrn = 0x60, LFSR_init = 0x49
i=         0, msg_pad=0x00, lfsr=1001001 msg_crypt w/ parity = 0xc9
i=         1, msg_pad=0x00, lfsr=0010011 msg_crypt w/ parity = 0x93
i=         2, msg_pad=0x00, lfsr=0100110 msg_crypt w/ parity = 0xa6
i=         3, msg_pad=0x00, lfsr=1001101 msg_crypt w/ parity = 0x4d
i=         4, msg_pad=0x00, lfsr=0011011 msg_crypt w/ parity = 0x1b
i=         5, msg_pad=0x00, lfsr=0110110 msg_crypt w/ parity = 0x36
i=         6, msg_pad=0x00, lfsr=1101101 msg_crypt w/ parity = 0xed
i=         7, msg_pad=0x00, lfsr=1011010 msg_crypt w/ parity = 0x5a
i=         8, msg_pad=0x00, lfsr=0110101 msg_crypt w/ parity = 0x35
i=         9, msg_pad=0x00, lfsr=1101011 msg_crypt w/ parity = 0xeb
i=        10, msg_pad=0x00, lfsr=1010110 msg_crypt w/ parity = 0x56
i=        11, msg_pad=0x2b, lfsr=0101101 msg_crypt w/ parity = 0x06
i=        12, msg_pad=0x4e, lfsr=1011011 msg_crypt w/ parity = 0x95
i=        13, msg_pad=0x4f, lfsr=0110111 msg_crypt w/ parity = 0x78
i=        14, msg_pad=0x57, lfsr=1101111 msg_crypt w/ parity = 0xb8
i=        15, msg_pad=0x4c, lfsr=1011110 msg_crypt w/ parity = 0x12
i=        16, msg_pad=0x45, lfsr=0111101 msg_crypt w/ parity = 0x78
i=        17, msg_pad=0x44, lfsr=1111011 msg_crypt w/ parity = 0x3f
i=        18, msg_pad=0x47, lfsr=1110110 msg_crypt w/ parity = 0xb1
i=        19, msg_pad=0x45, lfsr=1101100 msg_crypt w/ parity = 0xa9
i=        20, msg_pad=0x00, lfsr=1011000 msg_crypt w/ parity = 0xd8
i=        21, msg_pad=0x43, lfsr=0110001 msg_crypt w/ parity = 0x72
i=        22, msg_pad=0x4f, lfsr=1100011 msg_crypt w/ parity = 0xac
i=        23, msg_pad=0x4d, lfsr=1000110 msg_crypt w/ parity = 0x8b
i=        24, msg_pad=0x45, lfsr=0001101 msg_crypt w/ parity = 0x48
i=        25, msg_pad=0x53, lfsr=0011010 msg_crypt w/ parity = 0xc9
i=        26, msg_pad=0x0c, lfsr=0110100 msg_crypt w/ parity = 0xb8
i=        27, msg_pad=0x00, lfsr=1101001 msg_crypt w/ parity = 0x69
i=        28, msg_pad=0x42, lfsr=1010010 msg_crypt w/ parity = 0x90
i=        29, msg_pad=0x55, lfsr=0100101 msg_crypt w/ parity = 0xf0
i=        30, msg_pad=0x54, lfsr=1001011 msg_crypt w/ parity = 0x9f
i=        31, msg_pad=0x00, lfsr=0010111 msg_crypt w/ parity = 0x17
i=        32, msg_pad=0x57, lfsr=0101110 msg_crypt w/ parity = 0xf9
i=        33, msg_pad=0x49, lfsr=1011101 msg_crypt w/ parity = 0x14
i=        34, msg_pad=0x53, lfsr=0111011 msg_crypt w/ parity = 0xe8
```

i= 35, msg_pad=0x44, lfsr=1110111 msg_crypt w/ parity = 0x33
i= 36, msg_pad=0x4f, lfsr=1101110 msg_crypt w/ parity = 0x21
i= 37, msg_pad=0x4d, lfsr=1011100 msg_crypt w/ parity = 0x11
i= 38, msg_pad=0x00, lfsr=0111001 msg_crypt w/ parity = 0x39
i= 39, msg_pad=0x4c, lfsr=1110011 msg_crypt w/ parity = 0x3f
i= 40, msg_pad=0x49, lfsr=1100110 msg_crypt w/ parity = 0xaf
i= 41, msg_pad=0x4e, lfsr=1001100 msg_crypt w/ parity = 0x82
i= 42, msg_pad=0x47, lfsr=0011001 msg_crypt w/ parity = 0xde
i= 43, msg_pad=0x45, lfsr=0110010 msg_crypt w/ parity = 0x77
i= 44, msg_pad=0x52, lfsr=1100101 msg_crypt w/ parity = 0xb7
i= 45, msg_pad=0x53, lfsr=1001010 msg_crypt w/ parity = 0x99
i= 46, msg_pad=0x0e, lfsr=0010101 msg_crypt w/ parity = 0x1b
i= 47, msg_pad=0x00, lfsr=0101010 msg_crypt w/ parity = 0xaa
i= 48, msg_pad=0x00, lfsr=1010101 msg_crypt w/ parity = 0x55
i= 49, msg_pad=0x00, lfsr=0101011 msg_crypt w/ parity = 0x2b
i= 50, msg_pad=0x00, lfsr=1010111 msg_crypt w/ parity = 0xd7
i= 51, msg_pad=0x00, lfsr=0101111 msg_crypt w/ parity = 0xaf
i= 52, msg_pad=0x00, lfsr=1011111 msg_crypt w/ parity = 0x5f
i= 53, msg_pad=0x00, lfsr=0111111 msg_crypt w/ parity = 0x3f
i= 54, msg_pad=0x00, lfsr=1111111 msg_crypt w/ parity = 0xff
i= 55, msg_pad=0x00, lfsr=1111110 msg_crypt w/ parity = 0x7e
i= 56, msg_pad=0x00, lfsr=1111100 msg_crypt w/ parity = 0xfc
i= 57, msg_pad=0x00, lfsr=1111000 msg_crypt w/ parity = 0x78
i= 58, msg_pad=0x00, lfsr=1110000 msg_crypt w/ parity = 0xf0
i= 59, msg_pad=0x00, lfsr=1100000 msg_crypt w/ parity = 0x60
i= 60, msg_pad=0x00, lfsr=1000000 msg_crypt w/ parity = 0xc0
i= 61, msg_pad=0x00, lfsr=0000001 msg_crypt w/ parity = 0x81
i= 62, msg_pad=0x00, lfsr=0000010 msg_crypt w/ parity = 0x82
i= 63, msg_pad=0x00, lfsr=0000100 msg_crypt w/ parity = 0x84
encrypted string =
i3Fm;V뛺U땊&5쁙2?QIx뮩+hiX??7?늕A1Y_O"~쁺9;JuKwO_윛쐶?`!"$

program 1:
  0 bench msg: i c9 dut msg: c9
  1 bench msg: 3 93 dut msg: 93
  2 bench msg: F a6 dut msg: a6
  3 bench msg: m 4d dut msg: 4d
  4 bench msg: ; 1b dut msg: 1b

5 bench msg: V 36 dut msg: 36
6 bench msg: ?ed dut msg: ed
7 bench msg: z 5a dut msg: 5a
8 bench msg: U 35 dut msg: 35
9 bench msg: ?eb dut msg: eb
10 bench msg: v 56 dut msg: 56
11 bench msg: & 06 dut msg: 06
12 bench msg: 5 95 dut msg: 95
13 bench msg: ?78 dut msg: 78
14 bench msg: X b8 dut msg: b8
15 bench msg: 2 12 dut msg: 12
16 bench msg: ?78 dut msg: 78
17 bench msg: _ 3f dut msg: 3f
18 bench msg: Q b1 dut msg: b1
19 bench msg: I a9 dut msg: a9
20 bench msg: x d8 dut msg: d8
21 bench msg: ?72 dut msg: 72
22 bench msg: L ac dut msg: ac
23 bench msg: + 8b dut msg: 8b
24 bench msg: h 48 dut msg: 48
25 bench msg: i c9 dut msg: c9
26 bench msg: X b8 dut msg: b8
27 bench msg: ?69 dut msg: 69
28 bench msg: 0 90 dut msg: 90
29 bench msg: ?f0 dut msg: f0
30 bench msg: ? 9f dut msg: 9f
31 bench msg: 7 17 dut msg: 17
32 bench msg: ?f9 dut msg: f9
33 bench msg: 4 14 dut msg: 14
34 bench msg: ?e8 dut msg: e8
35 bench msg: S 33 dut msg: 33
36 bench msg: A 21 dut msg: 21
37 bench msg: 1 11 dut msg: 11
38 bench msg: Y 39 dut msg: 39
39 bench msg: _ 3f dut msg: 3f
40 bench msg: O af dut msg: af
41 bench msg: " 82 dut msg: 82
42 bench msg: ~ de dut msg: de
43 bench msg: ?77 dut msg: 77

```
        44 bench msg: W b7 dut msg: b7
        45 bench msg: 9 99 dut msg: 99
        46 bench msg: ; 1b dut msg: 1b
        47 bench msg: J aa dut msg: aa
        48 bench msg: u 55 dut msg: 55
        49 bench msg: K 2b dut msg: 2b
        50 bench msg: w d7 dut msg: d7
        51 bench msg: O af dut msg: af
        52 bench msg:  5f dut msg: 5f
        53 bench msg: _ 3f dut msg: 3f
        54 bench msg: ?ff dut msg: ff
        55 bench msg: ?7e dut msg: 7e
        56 bench msg: ?fc dut msg: fc
        57 bench msg: ?78 dut msg: 78
        58 bench msg: ?f0 dut msg: f0
        59 bench msg:  60 dut msg: 60
        60 bench msg: ` c0 dut msg: c0
        61 bench msg: ! 81 dut msg: 81
        62 bench msg: " 82 dut msg: 82
        63 bench msg: $ 84 dut msg: 84
score =         64/64
```

Program 2 Output:

```
run encryption program; original message =
Mr. Watson, come here. I want to see you.
LFSR_ptrn = 0x60, LFSR_init = 0x20
i=        0, msg_pad=0x00, lfsr=0100000 msg_crypt w/ parity = 0xa0
i=        1, msg_pad=0x00, lfsr=1000001 msg_crypt w/ parity = 0x41
i=        2, msg_pad=0x00, lfsr=0000011 msg_crypt w/ parity = 0x03
i=        3, msg_pad=0x00, lfsr=0000110 msg_crypt w/ parity = 0x06
i=        4, msg_pad=0x00, lfsr=0001100 msg_crypt w/ parity = 0x0c
i=        5, msg_pad=0x00, lfsr=0011000 msg_crypt w/ parity = 0x18
i=        6, msg_pad=0x00, lfsr=0110000 msg_crypt w/ parity = 0x30
i=        7, msg_pad=0x00, lfsr=1100001 msg_crypt w/ parity = 0xe1
i=        8, msg_pad=0x00, lfsr=1000010 msg_crypt w/ parity = 0x42
i=        9, msg_pad=0x00, lfsr=0000101 msg_crypt w/ parity = 0x05
i=       10, msg_pad=0x00, lfsr=0001010 msg_crypt w/ parity = 0x0a
i=       11, msg_pad=0x00, lfsr=0010100 msg_crypt w/ parity = 0x14
i=       12, msg_pad=0x00, lfsr=0101000 msg_crypt w/ parity = 0x28
i=       13, msg_pad=0x2d, lfsr=1010001 msg_crypt w/ parity = 0xfc
i=       14, msg_pad=0x52, lfsr=0100011 msg_crypt w/ parity = 0x71
i=       15, msg_pad=0x0e, lfsr=1000111 msg_crypt w/ parity = 0xc9
i=       16, msg_pad=0x00, lfsr=0001111 msg_crypt w/ parity = 0x0f
i=       17, msg_pad=0x37, lfsr=0011110 msg_crypt w/ parity = 0xa9
i=       18, msg_pad=0x41, lfsr=0111100 msg_crypt w/ parity = 0x7d
i=       19, msg_pad=0x54, lfsr=1111001 msg_crypt w/ parity = 0x2d
i=       20, msg_pad=0x53, lfsr=1110010 msg_crypt w/ parity = 0x21
i=       21, msg_pad=0x4f, lfsr=1100100 msg_crypt w/ parity = 0x2b
i=       22, msg_pad=0x4e, lfsr=1001000 msg_crypt w/ parity = 0x06
i=       23, msg_pad=0x0c, lfsr=0010001 msg_crypt w/ parity = 0x1d
i=       24, msg_pad=0x00, lfsr=0100010 msg_crypt w/ parity = 0x22
i=       25, msg_pad=0x43, lfsr=1000101 msg_crypt w/ parity = 0x06
i=       26, msg_pad=0x4f, lfsr=0001011 msg_crypt w/ parity = 0x44
i=       27, msg_pad=0x4d, lfsr=0010110 msg_crypt w/ parity = 0xdb
i=       28, msg_pad=0x45, lfsr=0101100 msg_crypt w/ parity = 0x69
i=       29, msg_pad=0x00, lfsr=1011001 msg_crypt w/ parity = 0x59
i=       30, msg_pad=0x48, lfsr=0110011 msg_crypt w/ parity = 0x7b
i=       31, msg_pad=0x45, lfsr=1100111 msg_crypt w/ parity = 0x22
i=       32, msg_pad=0x52, lfsr=1001110 msg_crypt w/ parity = 0x9c
i=       33, msg_pad=0x45, lfsr=0011101 msg_crypt w/ parity = 0xd8
i=       34, msg_pad=0x0e, lfsr=0111010 msg_crypt w/ parity = 0xb4
```

```
i=        35, msg_pad=0x00, lfsr=1110101 msg_crypt w/ parity = 0xf5
i=        36, msg_pad=0x29, lfsr=1101010 msg_crypt w/ parity = 0xc3
i=        37, msg_pad=0x00, lfsr=1010100 msg_crypt w/ parity = 0xd4
i=        38, msg_pad=0x57, lfsr=0101001 msg_crypt w/ parity = 0x7e
i=        39, msg_pad=0x41, lfsr=1010011 msg_crypt w/ parity = 0x12
i=        40, msg_pad=0x4e, lfsr=0100111 msg_crypt w/ parity = 0x69
i=        41, msg_pad=0x54, lfsr=1001111 msg_crypt w/ parity = 0x1b
i=        42, msg_pad=0x00, lfsr=0011111 msg_crypt w/ parity = 0x9f
i=        43, msg_pad=0x54, lfsr=0111110 msg_crypt w/ parity = 0x6a
i=        44, msg_pad=0x4f, lfsr=1111101 msg_crypt w/ parity = 0xb2
i=        45, msg_pad=0x00, lfsr=1111010 msg_crypt w/ parity = 0xfa
i=        46, msg_pad=0x53, lfsr=1110100 msg_crypt w/ parity = 0x27
i=        47, msg_pad=0x45, lfsr=1101000 msg_crypt w/ parity = 0x2d
i=        48, msg_pad=0x45, lfsr=1010000 msg_crypt w/ parity = 0x95
i=        49, msg_pad=0x00, lfsr=0100001 msg_crypt w/ parity = 0x21
i=        50, msg_pad=0x59, lfsr=1000011 msg_crypt w/ parity = 0x9a
i=        51, msg_pad=0x4f, lfsr=0000111 msg_crypt w/ parity = 0x48
i=        52, msg_pad=0x55, lfsr=0001110 msg_crypt w/ parity = 0xdb
i=        53, msg_pad=0x0e, lfsr=0011100 msg_crypt w/ parity = 0x12
i=        54, msg_pad=0x00, lfsr=0111000 msg_crypt w/ parity = 0xb8
i=        55, msg_pad=0x00, lfsr=1110001 msg_crypt w/ parity = 0x71
i=        56, msg_pad=0x00, lfsr=1100010 msg_crypt w/ parity = 0xe2
i=        57, msg_pad=0x00, lfsr=1000100 msg_crypt w/ parity = 0x44
i=        58, msg_pad=0x00, lfsr=0001001 msg_crypt w/ parity = 0x09
i=        59, msg_pad=0x00, lfsr=0010010 msg_crypt w/ parity = 0x12
i=        60, msg_pad=0x00, lfsr=0100100 msg_crypt w/ parity = 0x24
i=        61, msg_pad=0x00, lfsr=1001001 msg_crypt w/ parity = 0xc9
i=        62, msg_pad=0x00, lfsr=0010011 msg_crypt w/ parity = 0x93
i=        63, msg_pad=0x00, lfsr=0100110 msg_crypt w/ parity = 0xa6
encrypted string =
 A0aB
(|qI)}-!+"D[iY{"X4uCT~ij2z'-!H[8qbD         $I&


program 2:
        0 bench msg:   20 dut msg: 20
        1 bench msg:   20 dut msg: 20
        2 bench msg:   20 dut msg: 20
        3 bench msg:   20 dut msg: 20
```

4 bench msg: 20 dut msg: 20
5 bench msg: 20 dut msg: 20
6 bench msg: 20 dut msg: 20
7 bench msg: 20 dut msg: 20
8 bench msg: 20 dut msg: 20
9 bench msg: 20 dut msg: 20
10 bench msg: 20 dut msg: 20
11 bench msg: 20 dut msg: 20
12 bench msg: 20 dut msg: 20
13 bench msg: M 4d dut msg: 4d
14 bench msg: r 72 dut msg: 72
15 bench msg: . 2e dut msg: 2e
16 bench msg: 20 dut msg: 20
17 bench msg: W 57 dut msg: 57
18 bench msg: a 61 dut msg: 61
19 bench msg: t 74 dut msg: 74
20 bench msg: s 73 dut msg: 73
21 bench msg: o 6f dut msg: 6f
22 bench msg: n 6e dut msg: 6e
23 bench msg: , 2c dut msg: 2c
24 bench msg: 20 dut msg: 20
25 bench msg: c 63 dut msg: 63
26 bench msg: o 6f dut msg: 6f
27 bench msg: m 6d dut msg: 6d
28 bench msg: e 65 dut msg: 65
29 bench msg: 20 dut msg: 20
30 bench msg: h 68 dut msg: 68
31 bench msg: e 65 dut msg: 65
32 bench msg: r 72 dut msg: 72
33 bench msg: e 65 dut msg: 65
34 bench msg: . 2e dut msg: 2e
35 bench msg: 20 dut msg: 20
36 bench msg: I 49 dut msg: 49
37 bench msg: 20 dut msg: 20
38 bench msg: w 77 dut msg: 77
39 bench msg: a 61 dut msg: 61
40 bench msg: n 6e dut msg: 6e
41 bench msg: t 74 dut msg: 74
42 bench msg: 20 dut msg: 20

```
        43 bench msg: t 74 dut msg: 74
        44 bench msg: o 6f dut msg: 6f
        45 bench msg:   20 dut msg: 20
        46 bench msg: s 73 dut msg: 73
        47 bench msg: e 65 dut msg: 65
        48 bench msg: e 65 dut msg: 65
        49 bench msg:   20 dut msg: 20
        50 bench msg: y 79 dut msg: 79
        51 bench msg: o 6f dut msg: 6f
        52 bench msg: u 75 dut msg: 75
        53 bench msg: . 2e dut msg: 2e
        54 bench msg:   20 dut msg: 20
        55 bench msg:   20 dut msg: 20
        56 bench msg:   20 dut msg: 20
        57 bench msg:   20 dut msg: 20
        58 bench msg:   20 dut msg: 20
        59 bench msg:   20 dut msg: 20
        60 bench msg:   20 dut msg: 20
        61 bench msg:   20 dut msg: 20
        62 bench msg:   20 dut msg: 20
        63 bench msg:   20 dut msg: 20
score =         64/64
```

Program1.c:

```c
#include <stdio.h>

int main(void)
{
   char c[52] = "Mr. Watson, come here. I want to see you.";
   char output[64];
   int length = 41;
   int i, r1;
   unsigned char lfsr_init = 0x49;
   unsigned char lfsr_ptrn = 0x60;
   unsigned char lfsr;
   lfsr = lfsr_init;

   for (i = 0; i < 64; i++)
   {
       if (i < 10 || i > 50)
           output[i] = ' ';
       else
           output[i] = c[i - 10];
   }
   for (r1 = 0; r1 < 64; r1++)
   {
       unsigned char a; // original char
       unsigned char b; // encoded char
       unsigned char f; // XOR reduction
       unsigned char p;

       a = output[r1];
       a = a - 0x20;

       // reduction xor
       f = lfsr & lfsr_ptrn;
       f = ((f >> 4) ^ f) & 0x7f;
       f = ((f >> 2) ^ f) & 0x7f;
       f = ((f >> 1) ^ f) & 0x7f;
       f = f & 0x01;
```

```
      p = (a ^ lfsr);
      p = ((p >> 4) ^ p);
      p = ((p >> 2) ^ p);
      p = ((p >> 1) ^ p);
      p = p & 0x01;

      b = (a ^ lfsr) + (p << 7);
      printf("b: %x", b);
      printf("index: %d lfsr: 0x%x original: %c Encoded: %x\n", r1, lfsr, a +
0x20, b);
      lfsr = ((lfsr << 1) + f) & 0x7f;
  }
  return 0;
```

Program1 assembly:

Program1 assembly:

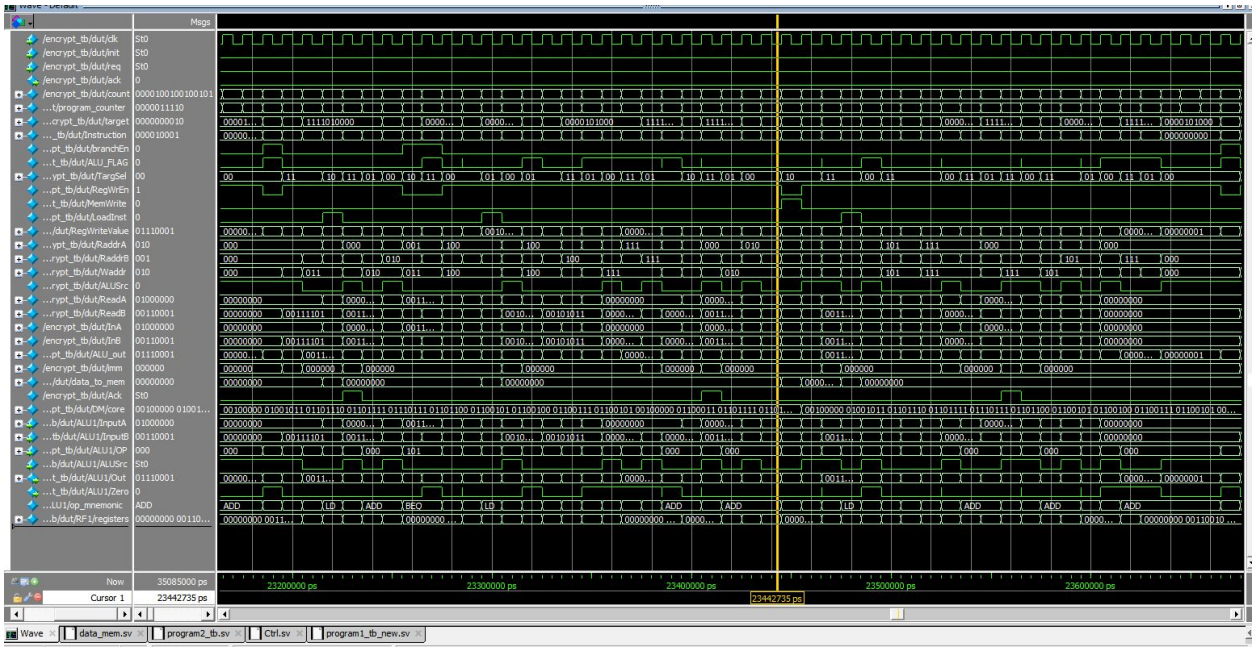
```
        SET    r1, 0                    // R1 = i
        SET            r3, mem[61]              // R3 = pre_length
        LDR    R6, mem[63]              // r6 = lfsr

L1: #1
        SET   r7, #0
        SUB  r7, r3                     // R7 = pre_length
        SUB  r7,r7                      // R7 = pre_length - 1
        BGE  r1, b1                     // LUT #b1   if(pre_length-1 >= R1)

        SUB R4,R1,R2                            // R4 = i - pre_length
        SET R7, 0                               // R4 = i - pre_length
        ADD R7, R1                      // R4 = i - pre_length
        SUB R7,R2                       // R4 = i - pre_length
        MOV R4,R7                       // R4 = i - pre_length
        LDR r4, mem[r4]                         // load message
        SET R7, 0x20;
        SUB R4,R7;                      // R4 = R4-0x20

b1:#3
        XOR R4,R6                       // b = original ^ lfsr
        ADD R5,R4                       // R5 = b
        AND R4, R4                      // Reduction xor R4
        SHR R4, #7                      // Parity bit <= MSB
        ADD R5,R4,R5                            // Encoded message
        STR R4, mem[64+i]                       // mem[64+i] = encoded message

        LDR R7, mem[63]                         // r5 = lfsr_pttrn
        rXOR R6,R7                      // Reduction XOR lfsr & lfsr_pttrn
        SHR R6,1
        ADD R6,R5

        ADD R1,R1                       // R1++
        SET            r7, #63                          // R3 = 63
        BGE   r1, L1                    // LUT #b0  if (63 >= r1)
        Halt
```

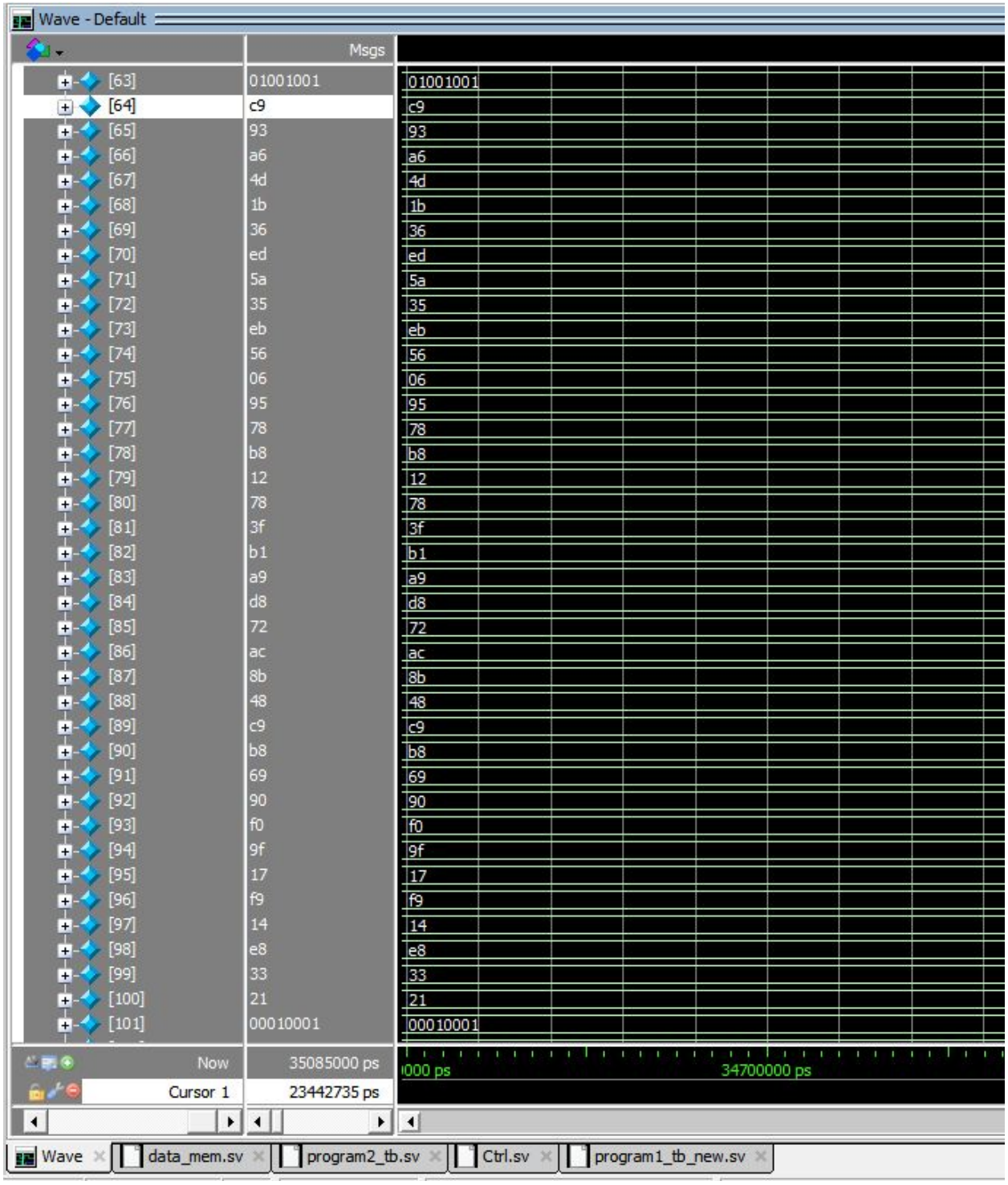
Program1 Machine Code:

```
111_000_000   // R7 = 0
000_001_111   // MOV R1 = 0
111_111_111   // R7 = 63
011_110_111   // LD R6, mem[63]
111_111_101   // R7 = 61
011_011_111   // R3 = mem[61], pre_length
111_000_000   // R7 = 0 L0: 7
000_111_011   // R7 = pre_length
001_111_111   // R7 = pre_length - 1
101_001_011   // if(pre_length - 1 >= R1)  branch to B1
000_100_001   // R4 = index
001_100_011   // R4 = index - pre_length
011_100_100   // R4 = mem[index-pre_length]
111_100_000   // R7 = 0X20
001_100_111   // R4 = mem[index-pre_length] - 0x20
010_100_110   // b = mssg ^ lfsr B1:16
000_101_100   // R5 = b
110_100_100   // R4 = parity
111_000_000   // R7 = 0
000_111_100   // R7 = parity
010_000_111   // R7 << 7
000_010_111   // R2 = 7
000_101_010   // R5 = b + (f << 7)
111_111_111   // R7 = 63
000_010_111   // R2 = 63
000_010_010   // R2 = 64
000_010_001   // R2 = 64+i
100_101_010   // mem[64+i] = R5
111_111_110   // R7 = 62
011_101_111   // R5 = mem[62]
111_000_000   // R7 = 0
110_101_110   // AND R6, R5, R5 = feedback
110_101_101   // R5 = xor feedback
000_111_110   // R7 = lfsr
010_000_001   // R7 = lfsr << 1
000_111_101   // R7 = lfsr + feedback
000_110_111   // R6 = r7
```

111_111_111 // r7 = 63
010_000_001 // R7 << 1
000_101_111 // r5 = R7
000_101_101 // R5 = 0x7f
110_110_101 // R6 & 0x7f
111_000_000 // R7 = 0
000_101_111 // R5 = 0
000_100_111 // R4 = 0
000_001_001 // index++
111_111_111 // R7 = 63
101_001_000 // If(63 >= R1) Branch to L0
001_000_000 // Halt

Waveforms:




Wave - Default
Msgs
[63] 01001001
[64] c9
[65] 93
[66] a6
[67] 4d
[68] 1b
[69] 36
[70] ed
[71] 5a
[72] 35
[73] eb
[74] 56
[75] 06
[76] 95
[77] 78
[78] b8
[79] 12
[80] 78
[81] 3f
[82] b1
[83] a9
[84] d8
[85] 72
[86] ac
[87] 8b
[88] 48
[89] c9
[90] b8
[91] 69
[92] 90
[93] f0
[94] 9f
[95] 17
[96] f9
[97] 14
[98] e8
[99] 33
[100] 21
[101] 00010001
Now 35085000 ps
Cursor 1 23442735 ps
000 ps
34700000 ps
Wave
data_mem.sv
program2_tb.sv
Ctrl.sv
program1_tb_new.sv

Program2.c:

```c
#include <stdio.h>

int main(void)
{
   char c[52] = "Mr. Watson, come here. I want to see you.";
   char output[64];
   int length = 41;
   int r1, r2;
   unsigned char r6;
   unsigned char lfsr_init = 0x20; // 010 0000
   unsigned char lfsr_ptrn = 0x60; // 110 0000 xor-> 100 0000  parity -> 1
   unsigned char lfsr;
   unsigned char input[10] = {0xa0, 0x41, 0x03, 0x06, 0x0c, 0x18, 0x30, 0xe1,
0x42, 0x05};
   unsigned char taps[9] = {0x60, 0x48, 0x78, 0x72, 0x6a, 0x69, 0x5c, 0x7e,
0x7b};

   lfsr = input[0];    // r3
   lfsr = lfsr & 0x7f; //r3
   printf("0x%x\n", lfsr);
   int a = 0x0C;
   a = a >> 8 | a << 24;
   printf("dsfsf 0x%x\n", a);
   for (r1 = 0; r1 < 9; r1++) // #1 , if r1 >= 9, exit the loop - 1
   {
       unsigned char pt = taps[r1]; // r3
       unsigned char l = lfsr;      //r6
       for (r2 = 0; r2 < 10; r2++)  // #2 if r2 >= 10 go to #1 - 2
       {
           unsigned char f;
           unsigned char res;
           unsigned char l1;
           l1 = l;
           res = (input[r2] & 0x7f);
           res = res ^ l;
```

```
            if (res != 0x00) // go to #1
                break;

            if (r2 == 9)
                printf("%x\n", pt);

            f = l & pt; //

            f = ((f >> 4) ^ f) & 0x7f;
            f = ((f >> 2) ^ f) & 0x7f;
            f = ((f >> 1) ^ f) & 0x7f;
            f = f & 0x01;

            l = ((l << 1) + f) & 0x7f;
            printf("origianl lfsr: %x,R1: %d, R2: %d R6: 0x%x feedback: %d PTTRN:
0x%x\n", l1, r1, r2, l, f, pt);
            //jump to #2
        }
        // jump to #1
    }
    return 0;
}
```

Program2 Assembly:

```
SET r7, #64            // R7 = 63
LDR R6, mem[64]        // R6 = lfsr
LDR R5, #0x7f          // R5 = 0x7f
SET r7, #62            // R7 = 62
SET R5, 0x7f           // R5 = 0x7f
AND R6, R5             // R6 = lfsr_init
STR R6, mem[R7]        // mem[62] = lfsr_init

// index: r1 r2
L1: #1
   LDR r3, mem[r1]     // r3 = lfsr_pttrn[n]
   LDR r6, mem[62]     // r6 = lfsr
L2: #2
// inner loop
   SET R7, #64         // R7 = 64
   ADD R7,R2           // R7 = 64 + n
   LDR R7,R7           // R7 = mem[64+n]
   AND R7,R5           // R7 = mem[64+n] & 0x7f
   XOR R7,R6           // R7 = mem[64+n] ^ lfsr
   BGE R0,L2_END       // if(r7 !=0)

   SET R7, #9          // R7 = 0
   SUB R7, R2          // R7 = R7 - 9
   BGE R0, L2_2        // if(r2 != 9) jump

   SET R7, #62         // found lfsr_pttrn
   STR R3, R7          // mem[62] = lfsr_pttrn

L2_2: #3
   SET R7, #0          // R7 = 0
   ADD R7, R6          // R7 = lfsr
   XOR R7, #1          // R7 = lfsr << 1
   AND R7, R5          // R7 = (lfsr << 1) & 0x7F
   MOV R4, R7          // R4 = R7

   SET R7, #0          // R7 = 0
   ADD R7, R6          // R7 = lfsr
```

```
    AND R7, R3          // R7 = lfsr & lfsr_pttrn
    XOR R7, R7          // R7 = feedback

    ADD R7,R4           // R7 = (lfsr << 1) & 0x7F + feedback
    MOV R6,R7           // lfsr = new lfsr;

    SET R7, #9          // R7 = 9
    ADD R2,R2           // R2++
    BGE R2,L2_END       // if(9 >= R2) jump to l2
    SET R7, #0
    BGE R0,L2           // jump to L2

L2_END: #4
    SET R7,#0           // R7 = 0
    MOV R2,R7           // R1 = 0
    SET R7, #8          // R7 = 8
    ADD R1,R1           // R1++
    BGE R7,R1, L1       // if(8 >= R1) jump to L1
// found the pattern

    SET R7, #62         // R7 = 62
    LDR R6, R7          // R6 = lfsr_pttrn
    ADD R7,R7           // R7 = 63
    LDR R4, R7          // R4 = lfsr_init

    SET R7, #0          // R7 = 0
    MOV R1, #0          // R1 = 0

L3: #5
    SET R7, #64         // R7 = 64
    ADD R7, R1          // R7 = 64 + i;
    LDR R2, R7          // R2 = mem[64+i]
    AND R2, R5          // R2 = mem[64+i] ^ 0x7F
    XOR R2, R4          // R2 = (mem[64+i] ^ 0x7F) ^ lfsr decoded
    STR R2, R1          // mem[i] = R2

    SET R7, #0          // R7 = 0
    ADD R7, R4          // R7 = lfsr
    XOR R7, #1          // R7 = lfsr << 1
```

```
AND R7, R5          // R7 = (lfsr << 1) & 0x7F
MOV R3, R7          // R3 = R7

SET R7, #0          // R7 = 0
ADD R7, R4          // R7 = lfsr
AND R7, R6          // R7 = lfsr & lfsr_pttrn
XOR R7, R7          // R7 = feedback

ADD R7,R3           // R7 = (lfsr << 1) & 0x7F + feedback
MOV R4,R7           // lfsr = new lfsr;

SET R7, #63         // R7 = 63
ADD R1,R1           // R1++
BGE R7,R1,#5        // if(63 >= R1) jump to L3
```

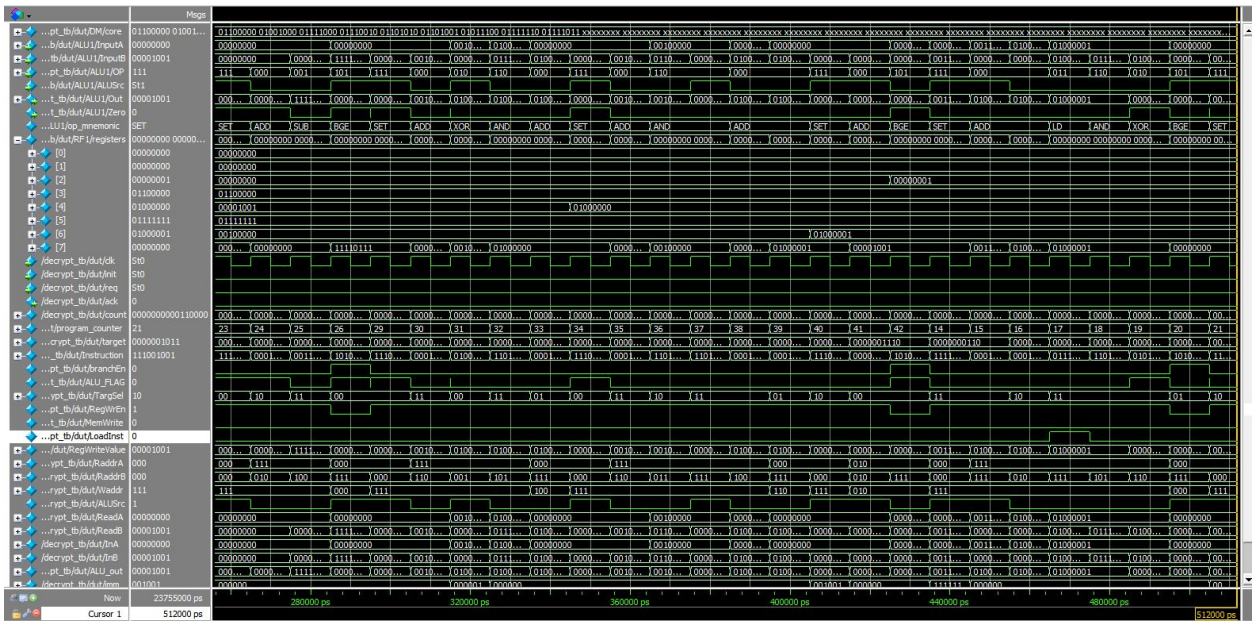
Program 2 Machine Code:

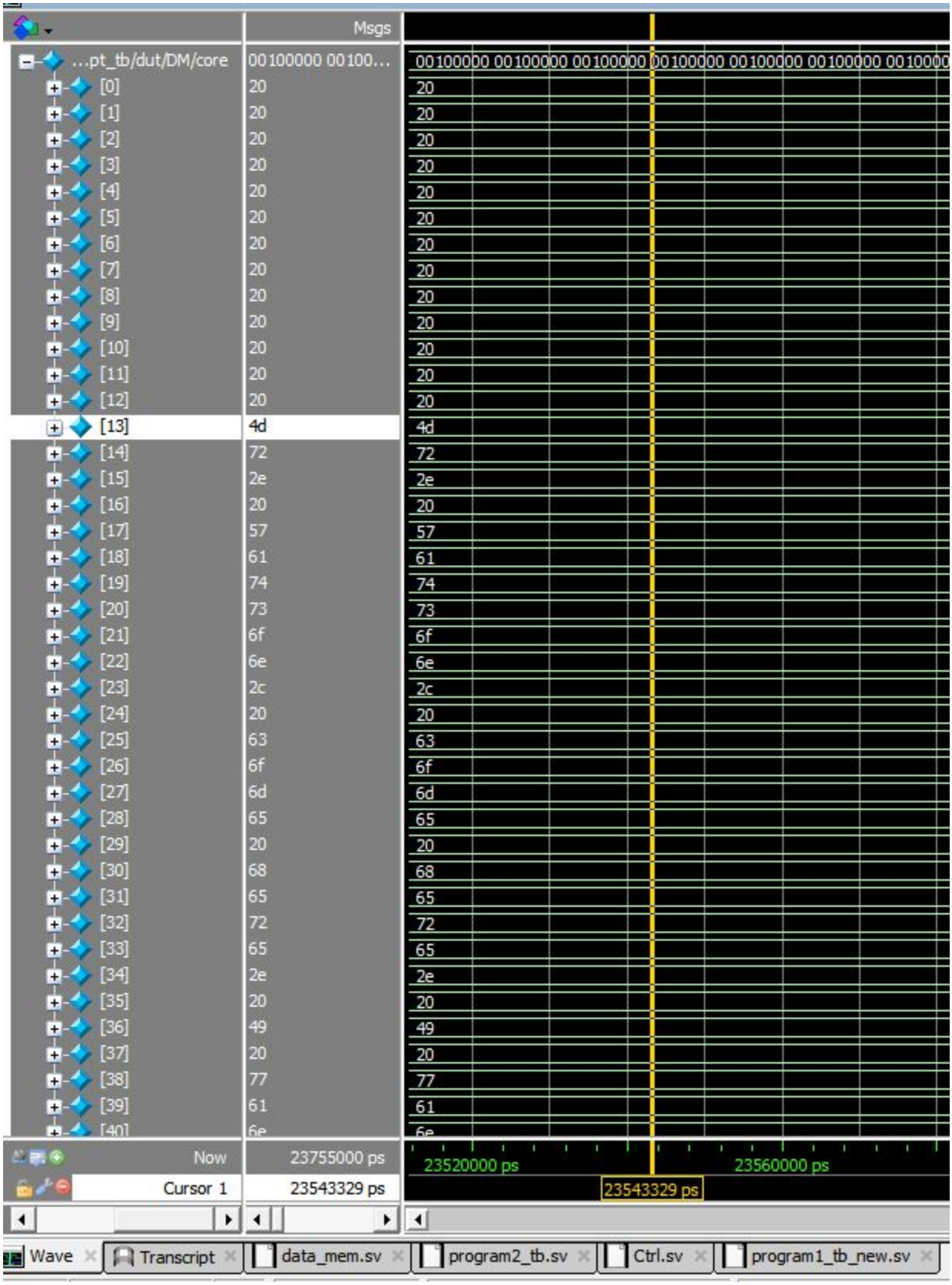
```
000_000_000
111_111_111   // SET R7 = #63
000_110_111   // R6 = #63
000_110_110   // R6 = #64
011_110_110   // R6 = mem[64]
010_000_001   // R7  << 1
000_101_111   // r5 = R7
000_101_101   // R5 = 0x7f
110_110_101   // R6 = lfsr & 0x7f
111_111_111   // R7 = 63
100_110_111   // mem[63] = lfsr_init
011_011_001     // R3 = mem[i] , pttrn  // L1 start here
111_111_111     // R7 = 63
011_110_111     // R6 = mem[63]
111_111_111     // R7 = 63      // L2
000_111_111     // R7 = 64
000_111_010     // R7 = 64 + j
011_111_111     // R7 = mem[64+j]
110_111_101     // R7 = mem[64+n] & 0x7f
010_111_110     // R7 = mem[64+n] ^ lfsr
101_000_100     // if( r7 != 0) Jump to L2_end
111_001_001     // R7 = 9
000_100_111     // R4 = 9
111_000_000     // R7 = 0
000_111_010     // R7 = R2
001_111_100     // R7 = R2-9
101_000_011     // if (r2 != 9) jump to L2_2
111_111_110     // R7 = 62
100_011_111     // mem[62] = pttrn
111_000_000     // R7 = 0       // // L2_2
000_111_110     // R7 = lfsr
010_000_001     // R7 = lfsr << 1
110_111_101     // R7 = lfsr << 1 & 0x7F
000_100_111     // R4 = lfsr << 1 & 0x7F
111_000_000     // R7 = 0
000_111_110     // R7 = lfsr
110_111_011     // R7 = lfsr & lfsr_pttrn
110_111_111     // R7 = feedback
```

```
000_111_100    // R7 = lfsr << 1 & 0x7F + feedback
000_110_111    // R6 = R7
111_001_001    // R7 = 9
000_010_010    // R2++
101_010_010    // if(9 >= r2) jump to L2
111_000_000    // R7 = 0
101_000_011    // Jump to L2
111_000_000    // R7 = 0  // // L2_end
000_010_111    // R2 = 0
111_001_000    // R7 = 8
000_001_001    // R1++
101_001_001    // if( 8 >= R1) jump to L1 // pttrn found
111_111_110    // R7 = 62
011_110_111    // R6 = lfsr_pttrn
000_111_111    // R7 = 63
011_100_111    // R4 = lfsr_init
111_000_000    // R7 = 0
000_001_111    // R1 = 0
111_111_111    // R7 = 63
000_111_111    // R7 = 64   // L3
000_111_001    // R7 = 64+i
011_010_111    // R2 = mem[64+i]
110_010_101    // R2 = mem[64+i] & 0x7F
010_010_100    // R2 = (mem[64+i] ^ 0x7F) ^ lfsr decoded
111_100_000    // R7 = 0x20
000_111_010    // R7 = decoded + 0x20
100_111_001    // mem[i] = R7
111_000_000    // R7 = 0
000_111_100    // R7 = lfsr
010_000_001    // R7 = lfsr << 1
110_111_101    // R7 = lfsr << 1 ^ 0x7F
000_011_111    // R3 = R7
111_000_000    // R7 = 0
000_111_100    // R7 = lfsr
110_111_110    // R7 = lfsr & lfsr_pttrn
110_111_111    // R7 = feedback
000_111_011    // R7 = (lfsr << 1) & 0x7F + feedback
000_100_111    // R4 = new lfsr
111_111_111    // R7 = 63
```

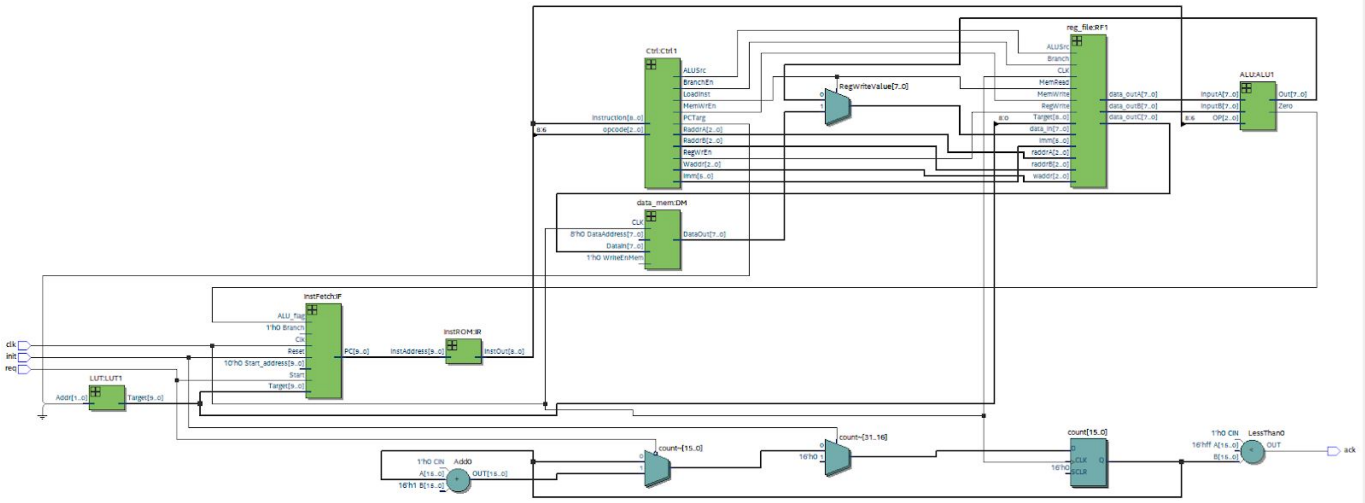
000_001_001 // R1++
101_001_101 // if(63 >= R1) jump to L3
001_000_000 // Halt

Waveform:




Msgs
...pt_tb/dut/DM/core
00100000 00100...
[0] 20
[1] 20
[2] 20
[3] 20
[4] 20
[5] 20
[6] 20
[7] 20
[8] 20
[9] 20
[10] 20
[11] 20
[12] 20
[13] 4d
[14] 72
[15] 2e
[16] 20
[17] 57
[18] 61
[19] 74
[20] 73
[21] 6f
[22] 6e
[23] 2c
[24] 20
[25] 63
[26] 6f
[27] 6d
[28] 65
[29] 20
[30] 68
[31] 65
[32] 72
[33] 65
[34] 2e
[35] 20
[36] 49
[37] 20
[38] 77
[39] 61
[40] 6e
Now
23755000 ps
Cursor 1
23543329 ps
23520000 ps
23560000 ps
23543329 ps
Wave
Transcript
data_mem.sv
program2_tb.sv
Ctrl.sv
program1_tb_new.sv

Quartus:


reg_file:RF1
Ctrl:Ctrl1
ALU:ALU1
data_mem:DM
RegWriteValue[7..0]
instFetch:IF
instROM:IR
LUT:LUT1
count[15..0]
Add0
LessThan0
clk
init
req
ack