```
0) ADDI R23, RO, 8
1)  ADDI R24, R0, 4
2)  ADDI R25, R0, 2
3)  ADDI R26, R0, 1

4)  LOOP: LB R29, R0, 136                    // R29 = mode
5)  LB R28, R0, 140                          // R28 = di_vld
6)  LB R27, R0, 144                          // R27 = key_vld

7)  BEQ R29, R23 <BRANCH ADDR: HALT>              //HALT
8)  BEQ R29, R24 <BRANCH ADDR: DECRYPTION>              //DEC
9)  BEQ R29, R25 <BRANCH ADDR: ENCRYPTION>              //ENC
10) BEQ R29, R26 <BRANCH ADDR: KEY_EXPANSION>          //KEY_EXP

11) ADDI R22, R0, 1
12) JMP <LOOP>

13) BNE R26, R27 <LOOP>

---KEY_EXPANSION CODE

14) KEY_EXPANSION: ADDI R22, R0, 0
15) ADDI R30, R0, 0
16) ADDI R31, R0, 0

//Key Expansion: Initialization of S Array with magic constants.
//R19 = 1, R4 = 112

17) LB R19, R0, 148
18) SB R19, R0, 112
19) ADDI R19, R0, 1
20) ADDI R4, R0, 112
21) ADDI R3, R0, 12                  //R3 IS THE COUNTER FOR MEMORY LOCATION

//LOADING P AND Q IN R1 & R2
22) LB R1, R0, 128          // R1 = MEM[128] = P
23) LB R2, R0, 132                   // R2 = MEM[132] = Q

//S[0] = P; MEM[8] = P
24) SB R1, R0, 8
25) ADD R5, R0, R1                   //INITIALIZING R5 TO P
```

```
// S[1] = P + Q; S[2] = P + 2Q; ..... S[25] = P + 25Q
    26) INIT LOOP: ADD R5, R5, R2
    27) SB R5, R3, 0
    28) ADDI R3, R3, 4
    29) BNE R3, R4, <BRANCH ADDRESS: INIT LOOP>
    30) SB R0, R0, 116
    31) SB R0, R0, 120
    32) SB R0, R0, 124

//R19 =1, R4 = 78, R5 = 112, R6 = 128
//Data memory is byte addressable
//S[0] starts from mem[8] to mem[11],....,S[25] starts from mem[108] to mem[111]
//L[3] starts from mem[112] to mem[115],....,L[0] is from mem[124] to mem[127]

//Intitalization of counter
    33) ADDI R19, R0, 1
    34) ADDI R6, R0, 128
    35) ADDI R4, R0, 78
    36) ADDI R5, R0, 112

    37) ADDI R1,R0,0  //R1 = K_CNT = 0
    38) ADDI R2,R0,8  //R2 = I_CNT = 8
    39) ADDI R3,R0,112  //R3 = J_CNT = 112

    40) ADD R7,R0,R0   //R7 = a_tmp1
    41) ADD R8,R0,R0   //R8 = S[i]
    42) ADD R9,R0,R0   //R9 = a_reg
    43) ADD R10,R0,R0   //R10 = b_reg
    44) ADD R11,R0,R0   //R11 = a_reg + b_reg
    45) ADD R12,R0,R0   //R12 = ab_tmp
    46) ADD R13,R0,R0   //R13 = L[j]
    47) ADD R14,R0,R0   //R14 = b_temp1
    48) ADDI R15, R0, 3 //R15 = 3, DATA INDEPENDENT LEFT ROTATE

//A = S[i] = ROTL(S[i] + (A+B), 3)

//R11 = R9 + R10 = a_reg + b_reg
    49) 78_loops: ADD R11, R9, R10

//a_tmp1 = R7 = R8 + R11 = S[i] + a_reg + b_reg
    50) LB R8, R2, 0    //R8 = S[i]
    51) ADD R7, R8, R11
```

```
//a_reg = R9 = R7 << R15 = a_tmp1 << 3
    52) ADD R9, R7, R0                                  //R9 INITAILIZED TO R7
    53) ADD R16, R15, R0                                        // R16 = R15  // COUNTER

    54) START:BLT R9, R0, <Branch address:MSB1>                         // JUMP TO MSB1 WHEN R9 < 0
    55) SHL R9, R9, 1                                   // SHIFT R9 BY 1
    56) SUB R16, R16, R19                                       // DECREMENT COUNTER
    57) BNE R16, R0, <Branch address:START>
    58) BEQ R16, R0, <Branch address:AFTER_ROTATE>

    59) MSB1: SHL R9, R9, 1                                     // SHIFT R9 BY 1
    60) ADD R9, R9, R19
    61) SUB R16, R16, R19                                               // DECREMENT
        COUNTER

    62) BNE R16, R0, <Branch address:START>

//S[i]= a_reg = R9
    63) AFTER_ROTATE: SB R9,R2,0            //Mem[R2 + 0] = R9


//B = L[j] = ROTL(L[j] + (A+B), (A+B))

//ab_tmp = R12 = R9 + R10 = a_reg + b_reg
    64) ADD R12, R9, R10

//b_tmp1 = R14 = R13 + R12 = L[j] + ab_tmp
    65) LB R13, R3, 0                   // R13 = L[j]
    66) ADD R14, R13, R12
    67) ANDI R12, R12, 1F

//b_reg = R10 = R14 << R12 = b_tmp1 << ab_tmp
    68) ADD R10, R14, R0            //R10 INITAILIZED TO R14
    69) ADD R16, R12, R0            // R16 = R12  // COUNTER
    70) BEQ R16, R0, <Branch address: AFTER_ROTATE >
    71) START:BLT R10, R0, <Branch address:MSB1>        // JUMP TO MSB1 WHEN R10 < 0
    72) SHL R10, R10, 1                 // SHIFT R10 BY 1
    73) SUB R16, R16, R19                                       // DECREMENT COUNTER
    74) BNE R16, R0, <Branch address:START>
    75) BEQ R16, R0, <Branch address:AFTER_ROTATE>
```

76) MSB1: SHL R10, R10, 1 // SHIFT R10 BY 1
77) ADD R10, R10, R19
78) SUB R16, R16, R19 // DECREMENT COUNTER
79) BNE R16, R0, <Branch address:START>

//L[j]= b_reg = R10
80) AFTER_ROTATE: ADD R10, R10, R0
81) SB R10,R3,0 //Mem[R3 + 0] = R10

//Incrementing counter
82) ADDI R1, R1, 1 // R1 = k, k = k+1
83) ADDI R2, R2, 4 // R2 = i, i = i+4
84) ADDI R3, R3, 4 // R3 = j, j = j+4

85) BNE R2, R5, <Branch address:j_cnt>

//When i_cnt reaches max value, it resets to 8
86) ADDI R2, R0, 8

87) j_cnt: BNE R3, R6, <Branch Address:k_cnt>

//When j_cnt reaches max value, it resets to 112
88) ADDI R3, R0, 112

89) k_cnt: BNE R1, R4, <Branch Address: 78_loops>
90) ADD R31, R1, R0

91) SB R0, R0, 144
92) JMP <LOOP>

93) BNE R26, R28 <LOOP>

---ENCRYPTION CODE
94) ENCRYPTION: ADDI R22, R0, 0
95) ADDI R30, R0, 0
96) ADDI R31, R0, 0

97) ADDI R16,0X0C(R0) //R16 = 12, Memory counter
98) ADDI R18,0X0C(R0) //R18 = 12, counter max value 12
99) ADDI R19,1(R0) //R19 <- 1

```
    100)        LB R1,0(R0)                     //R1 = A  //R1 <- MEM[0+0], R0 = 0
    101)        LB R2,4(R0)                     //R2 = B  //R2<- MEM[0+4]

//Preround state
//A + S[0]
    102)        LB R3,8(R0)                     //R3 <- S[0]
    103)        ADD R1,R1,R3            //R5 = A + S[0]

//B + S[1]
    104)        LB R4,12(R0)            //R4 <- S[1]
    105)        ADD R2,R2,R4            //R6 <- B + S[1]

    106)        ADD R21, R0, R0         //R21 = 0, counter for 12 iterations 0 TO 11 WHEN = 12 BREAK

//12 iterations begining

//((A XOR B)<<<B) + S[2 * i]

//A XOR B
    107)        Enc_Loop: NOR R10,R1,R2
    108)        NOR R11,R10,R1
    109)        NOR R12,R10,R2
    110)        NOR R10,R11,R12
    111)        NOR R4,R10,R10      //R4 <- A XOR B

//((A XOR B)<<<B)
//left rotate
    112)        ANDI R10, 0X1F(R2)          //R10 = R2 AND 0X001F

// R15 = R4 <<< R10
// R0 = 0, R19 = 1, R1 = A, R2 = B, R4 = LEFT ROTATED VALUE

    113)        ADD R15, R4, R0         //R15 INITAILIZED TO R4
    114)        ADD R20, R10, R0         // R20 = R10  // I COUNTER = R10

    115)        BEQ R20, R0, <Branch address:shift_zero> // JUMP TO shift_zero when true //BRANCH
        +9

    116)        START:BLT R15, R0, <Branch address:MSB1> // JUMP TO MSB1 WHEN R4 < 0 //BRANCH
        +4
    117)        SHL R15, R15, 1                                         // SHIFT A BY 1
```

```
118)    SUB R20, R20, R19                                   // DECREMENT
    COUNTER
119)    BNE R20, R0, <Branch address:START>  // JUMP TO START WHEN R20 IS NOT R0 //
120)    BEQ R20, R0, <Branch address:HALT>   // JUMP TO HALT WHEN R20 IF 0 // BRANCH +4

121)    MSB1: SHL R15, R15, 1                               // SHIFT A BY 1
122)    ADD R15, R15, R19
123)    SUB R20, R20, R19                                   // DECREMENT
    COUNTER
124)    BNE R20, R0, <Branch address:START> //JUMP TO START WHEN R20 IS NOT 0 //
    BRANCH -9
125)    shift_zero: ADD R15, R15, R0     // COPY R15 IN R15

//((A XOR B)<<<B) + S[2 * i]
126)    ADDI R16, 0X04(R16)                 //R16 = 16
127)    LB R17,0(R16)                       //R17 = MEM[0+R16], R17 = MEM[16] =
    S[2], S[4], S[6], .....S[24]
128)    ADD R1,R15,R17                      //R1 = ((A XOR B)<<<B) + S[2 * i]

//((B XOR A)<<<A) + S[(2 * i) +1]

//B XOR A
129)    NOR R10,R1,R2
130)    NOR R11,R10,R1
131)    NOR R12,R10,R2
132)    NOR R10,R11,R12
133)    NOR R4,R10,R10      //R4 <- B XOR A

//((B XOR A)<<<A)
//left shift
134)    ANDI R10, 0X1F(R1)          //R10 = R1 AND 0X001F

135)    ADD R15, R4, R0           //R15 INITAILIZED TO R4
136)    ADD R20, R10, R0          // R20 = R10  // I COUNTER = R10

137)    BEQ R20, R0, <Branch address:shift_zero> // JUMP TO shift_zero when true //BRANCH
    +9

138)    START:BLT R15, R0, <Branch address:MSB1> // JUMP TO MSB1 WHEN R4 < 0 //BRANCH
    +4
139)    SHL R15, R15, 1                                     // SHIFT A BY 1
```

```
140)    SUB R20, R20, R19                                   // DECREMENT
        COUNTER
141)    BNE R20, R0, <Branch address:START>  // JUMP TO START WHEN R20 IS NOT R0 //
142)    BEQ R20, R0, <Branch address:HALT>   // JUMP TO HALT WHEN R20 IF 0 // BRANCH +4

143)    MSB1: SHL R15, R15, 1                               // SHIFT A BY 1
144)    ADD R15, R15, R19
145)    SUB R20, R20, R19                                   // DECREMENT
        COUNTER
146)    BNE R20, R0, <Branch address:START> //JUMP TO START WHEN R20 IS NOT 0 //
        BRANCH -9
147)    shift_zero: ADD R15, R15, R0     // COPY R15 IN R15

//((B XOR A)<<<A) + S[2 * i + 1]
148)    ADDI R16, 0X04(R16)                 //R16 = 20
149)    LB R17,0(R16)                       //R17 = MEM[0+R16], R17 = MEM[20] =
        S[3], S[5], S[7],....S[25]
150)    ADD R2,R15,R17                      //R2 = ((B XOR A)<<<A) + S[2 * i
        + 1]
151)    ADD R21, R21, R19                   //R21 = 1,2,3,..12
152)    BNE R21,R18,<<BRANCH ADDR = Enc_Loop: > // IF BRANCH IS NOT EQUAL THEN GO TO
        Enc_Loop:
153)    ADDI R30, R0, 1
154)    SB R0, R0, 140

155)    JMP <LOOP>

156)    BNE R26, R28 <LOOP>

        ---DECRYPTION CODE
157)    DECRYPTION: ADDI R22, R0, 0
158)    ADDI R30, R0, 0
159)    ADDI R31, R0, 0
//R1 = A, R2 = B, R19 =1, R0 = 0, R16 = 112, R5 = 12, R10 =4, R12 = 32

160)    ADDI R16, Ox70, R0          //R16 = 112, MEM COUNTER
161)    ADDI R19, 0x01, R0          //R19 = 1
162)    ADDI R5, 0X0C, R0           //R5 = 12, RC5 DECRYPTION COUNTER FROM 12 TO 1
163)    ADDI R10, 0x04, R0          //R10 = 4
164)    ADDI R12, 0x20, R0          //R12 = 32

165)    LB R1,0(R0)                 //R1 = A, FROM MEM[0]
```

```
    166)        LB R2,4(R0)                 //R2 = B, FROM MEM[4]

//START OF 12 ITERATIONS

//B = (B - S[2*i + 1]) >> A) XOR A

//R6 = B - S[2*i + 1]
    167)        Dec_Loop: SUB R16, R16, R10             //R16 = R16 - 4 = 108, 100,...
    168)        LB R17, 0(R16)              //R17 = MEM[R16] = S[25], S[23], S[21],....S[3]
    169)        SUB R6, R2, R17             //R6 = B - S[2*i + 1]

//R7 = (B - S[2*i + 1]) >> A) = R6 >> A = R6 >> R1

    170)        ANDI R11, 0x1F, R1          //R11 = R1(4 DOWNTO 0)

//RIGHT ROTATE BY X = LEFT ROTATE BY (32 - X)
    171)        SUB R11, R12, R11           //R11 = 32 - R11

//R7 = R6 << R11, LEFT ROTATE BY R11, RIGHT ROTATED BY R1

    172)        ADD R7, R6, R0          //R7 INITAILIZED TO R6
    173)        ADD R20, R11, R0        // R20 = R11  // I COUNTER = R11

    174)        BEQ R20, R0, <Branch address:shift_zero> // JUMP TO shift_zero when true //BRANCH
        +9

    175)        START:BLT R7, R0, <Branch address:MSB1> // JUMP TO MSB1 WHEN R7 < 0 //BRANCH
        +4
    176)        SHL R7, R7, 1                                       // SHIFT R7 BY 1
    177)        SUB R20, R20, R19                                   // DECREMENT
        COUNTER
    178)        BNE R20, R0, <Branch address:START>  // JUMP TO START WHEN R20 IS NOT R0 //
    179)        BEQ R20, R0, <Branch address:shift_zero>   // JUMP TO HALT WHEN R20 IF 0 // BRANCH
        +4

    180)        MSB1: SHL R7, R7, 1                                 // SHIFT R7 BY
        1
    181)        ADD R7, R7, R19
    182)        SUB R20, R20, R19                                   // DECREMENT
        COUNTER
    183)        BNE R20, R0, <Branch address:START> //JUMP TO START WHEN R20 IS NOT 0 //
        BRANCH -9
```

```
    184)        shift_zero: ADD R7, R7, R0     // COPY R7 IN R7

//B = (B - S[2*i + 1]) >> A) XOR A
//R2 = R7 XOR R1
    185)        NOR R13,R1,R7
    186)        NOR R11,R13,R1
    187)        NOR R18,R13,R7
    188)        NOR R13,R11,R18
    189)        NOR R2,R13,R13      //R2 <- R7 XOR R1


//A = (A - S[2*i]) >> B) XOR B
//R8 = A - S[2*i]
    190)        SUB R16, R16, R10          //R16 = R16 - 4 = 104, 96,...
    191)        LB R17, 0(R16)            //R17 = MEM[R16] = S[24], S[22], S[20],....S[2]
    192)        SUB R8, R1, R17            //R8 = A - S[2*i]

//R9 = (A - S[2*i]) >> B) = R8 >> B = R8 >> R2

    193)        ANDI R11, 0x1F, R2         //R11 = R2(4 DOWNTO 0)

//RIGHT ROTATE BY X = LEFT ROTATE BY (32 - X)
    194)        SUB R11, R12, R11          //R11 = 32 - R11

//R9 = R8 << R11, LEFT ROTATE BY R11, RIGHT ROTATED BY R2

    195)        ADD R9, R8, R0          //R9 INITAILIZED TO R8
    196)        ADD R20, R11, R0        // R20 = R11  // I COUNTER = R11

    197)        BEQ R20, R0, <Branch address:shift_zero> // JUMP TO shift_zero when true //BRANCH
        +9

    198)        START:BLT R9, R0, <Branch address:MSB1> // JUMP TO MSB1 WHEN R9 < 0 //BRANCH
        +4
    199)        SHL R9, R9, 1                                          // SHIFT R9 BY 1
    200)        SUB R20, R20, R19                                          // DECREMENT
        COUNTER
    201)        BNE R20, R0, <Branch address:START>  // JUMP TO START WHEN R20 IS NOT R0 //
    202)        BEQ R20, R0, <Branch address:HALT>   // JUMP TO HALT WHEN R20 IF 0 // BRANCH +4

    203)        MSB1: SHL R9, R9, 1                                        // SHIFT R9 BY
        1
```

```
    204)        ADD R9, R9, R19
    205)        SUB R20, R20, R19                                   // DECREMENT
        COUNTER
    206)        BNE R20, R0, <Branch address:START> //JUMP TO START WHEN R20 IS NOT 0 //
        BRANCH -9
    207)        shift_zero: ADD R9, R9, R0     // COPY R9 IN R9

//A = (A - S[2*i]) >> B) XOR B
//R1 = R9 XOR R2
    208)        NOR R13,R2,R9
    209)        NOR R11,R13,R2
    210)        NOR R12,R13,R9
    211)        NOR R13,R11,R12
    212)        NOR R1,R13,R13      //R1 <- R9 XOR R2


//Decrementing Decryption Counter
    213)        SUB R5,R5,R19               //R5 = 11 TO 0
    214)        BNE R5,R0,<<BRANCH ADDR = Dec_Loop: > // IF BRANCH IS NOT EQUAL THEN GO TO
        Dec_Loop:


//B = B - S[1]
    215)        LB R4,12(R0)            //R4 <- S[1]
    216)        SUB R2,R2,R4            //R6 <- B - S[1]

//A = A - S[0]
    217)        LB R3,8(R0)                     //R3 <- S[0]
    218)        SUB R1,R1,R3            //R5 = A - S[0]
    219)        ADDI R30, R0, 1
    220)        SB R0, R0, 140
    221)        JMP <LOOP>

    222)        HALT: HALT
```