CS 401

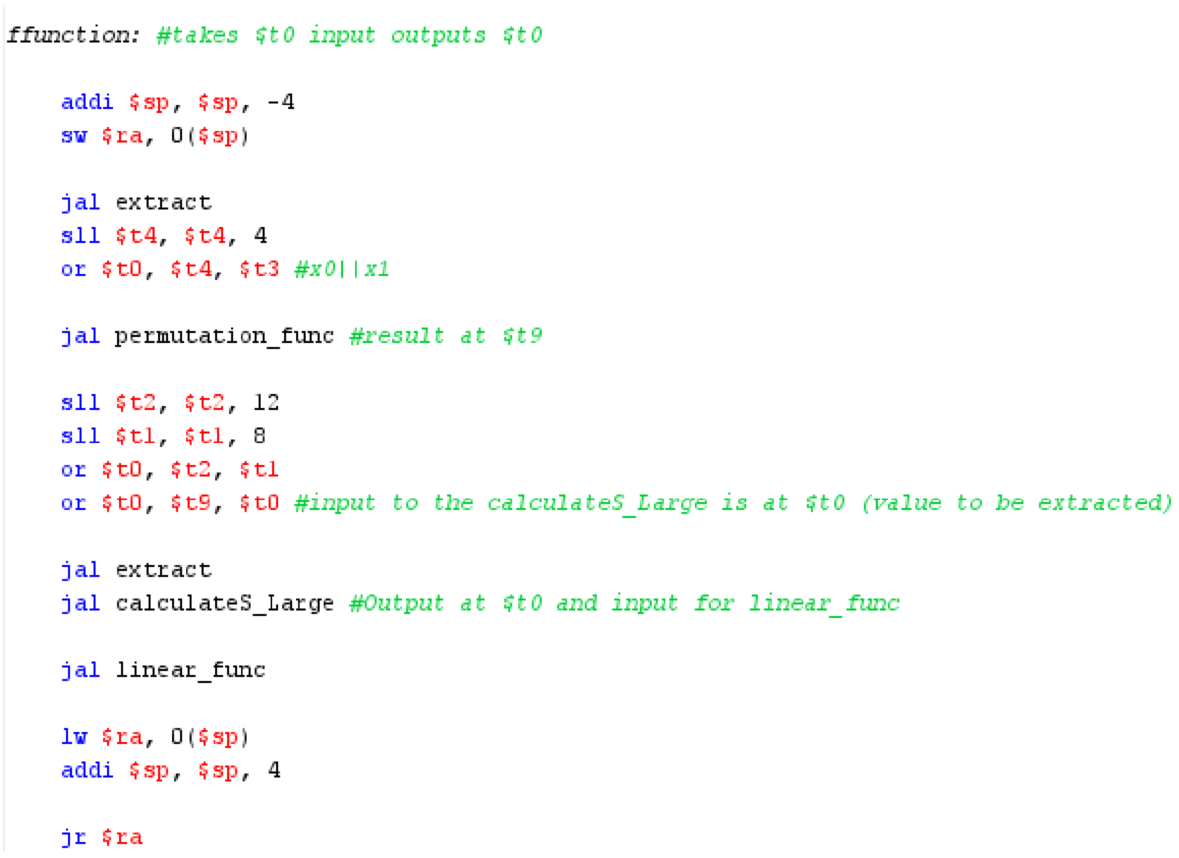
Term Project Phase II

Mustafa Onuralp Devres 29310

Ege Kaan Özalp 28989

3.1 F Function

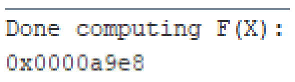
The implemented F Function takes it input from the $t0 register then outputs back at the $t0 register.



```
ffunction: #takes $t0 input outputs $t0

    addi $sp, $sp, -4
    sw $ra, 0($sp)

    jal extract
    sll $t4, $t4, 4
    or $t0, $t4, $t3 #x0||x1

    jal permutation_func #result at $t9

    sll $t2, $t2, 12
    sll $t1, $t1, 8
    or $t0, $t2, $t1
    or $t0, $t9, $t0 #input to the calculateS_Large is at $t0 (value to be extracted)

    jal extract
    jal calculateS_Large #Output at $t0 and input for linear_func

    jal linear_func

    lw $ra, 0($sp)
    addi $sp, $sp, 4

    jr $ra
```

Figure 1. F Function

First the value inside the $t0 register is extracted and is separated to x0, x1, x2 and x3. The output from the permutation function with the x0 and x1 inputs and the x2 and x3 values are concatenated with each other and given as input to the S(X). Lastly, the output of the linear function from the output of the S(X) is calculated. When the value for X = 0xbbaa is given as input the following correct output is seen.

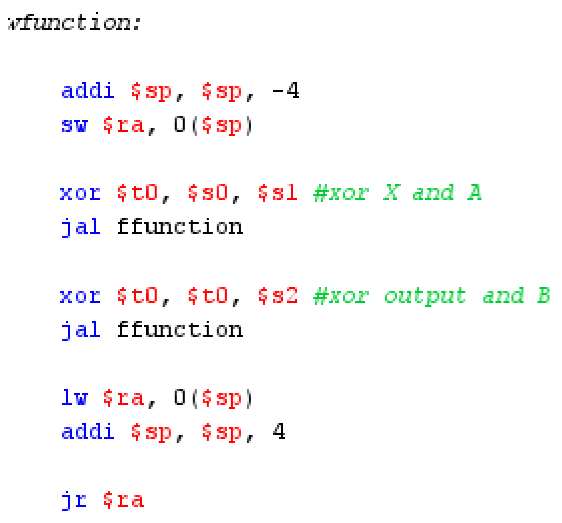


```
Done computing F(X):
0x0000a9e8
```

Figure 2. Output to F Function

3.2 W Function

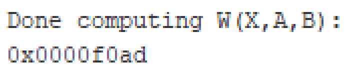
The implemented W Function takes X, A and B values as inputs from the $s0, $s1 and $s2 registers respectively. The output is stored in $t0 register.



```
wfunction:

    addi $sp, $sp, -4
    sw $ra, 0($sp)

    xor $t0, $s0, $s1 #xor X and A
    jal ffunction

    xor $t0, $t0, $s2 #xor output and B
    jal ffunction

    lw $ra, 0($sp)
    addi $sp, $sp, 4

    jr $ra
```

Figure 3. W Function

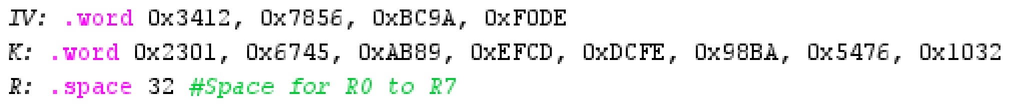
X and A values are XOR'ed with each other and the output is given to the F Function. Later the previous output and B values are XOR'ed and it is given as input to yet another F Function. When the X = 0xBBAA, A=0xCCCC, B=0xDDDD are given as input the following correct output is observed.



```
Done computing W(X,A,B):
0x0000f0ad
```

Figure 4. Output to W Function

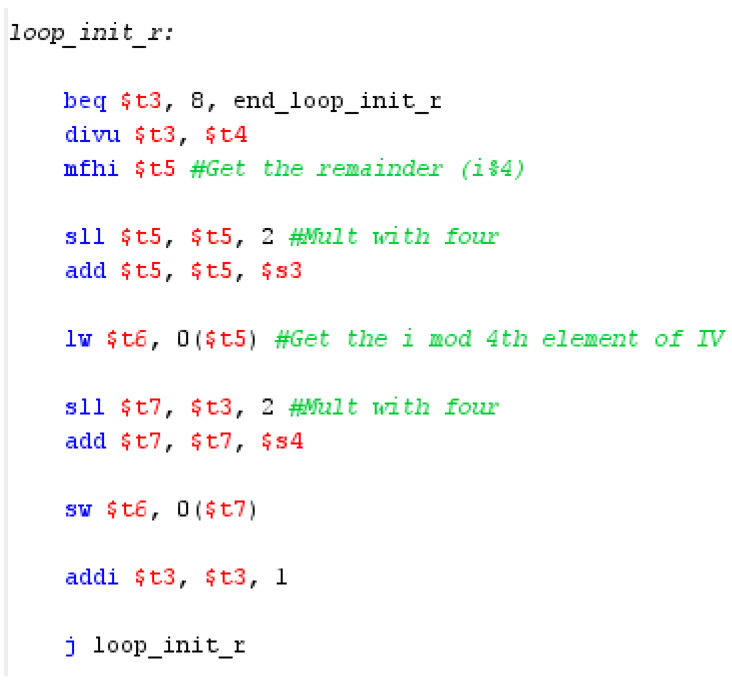
3.3 Initialization of State Vector



```
IV: .word 0x3412, 0x7856, 0xBC9A, 0xF0DE
K: .word 0x2301, 0x6745, 0xAB89, 0xEFCD, 0xDCFE, 0x98BA, 0x5476, 0x1032
R: .space 32 #Space for R0 to R7
```

Figure 5. Vectors in Memory

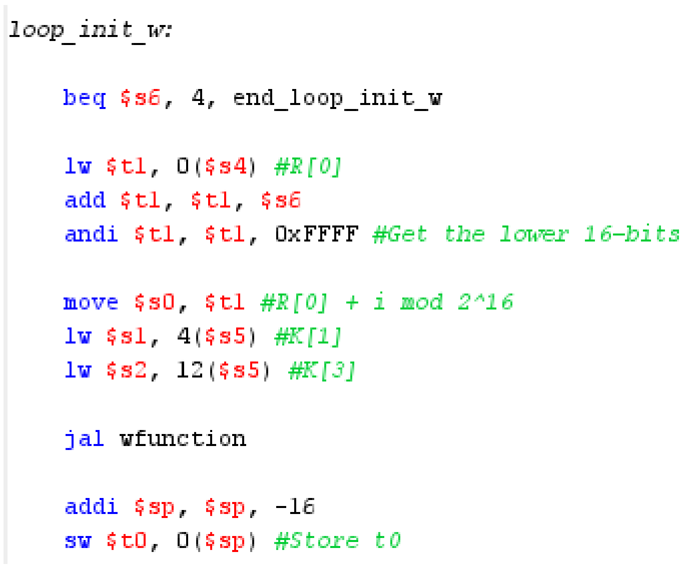
The initial vector, keys and the state vector are all stored in the memory. The first loop saves the arguments in the i mod 4th index of the initial vector to ith index of the state vector R.



```
loop_init_r:

    beq $t3, 8, end_loop_init_r
    divu $t3, $t4
    mfhi $t5 #Get the remainder (i%4)

    sll $t5, $t5, 2 #Mult with four
    add $t5, $t5, $s3

    lw $t6, 0($t5) #Get the i mod 4th element of IV

    sll $t7, $t3, 2 #Mult with four
    add $t7, $t7, $s4

    sw $t6, 0($t7)

    addi $t3, $t3, 1

    j loop_init_r
```

Figure 6. Initial Loop

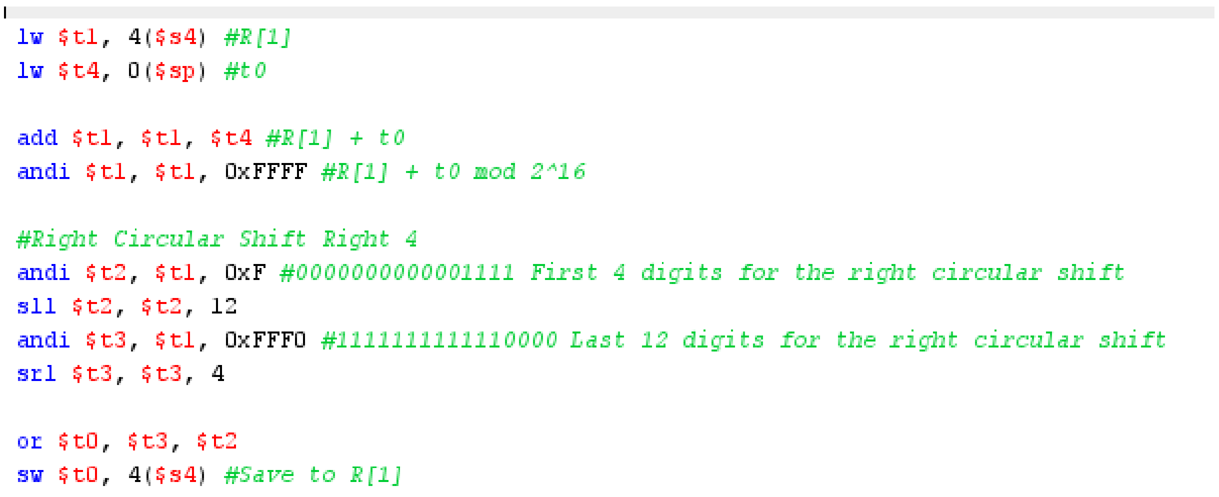
In the second loop the 0th index of the state vector is retrieved and summed with the value of i. The least significant 16 bits are retrieved with a mask. Along with the retrieved 16 bits, the respective keys are given as input to the W Function and the output is stored in the stack for later use. Later, until the line 8 of the algorithm 1 initialization similar procedure is carried out with different values.



```
loop_init_w:

    beq $s6, 4, end_loop_init_w

    lw $t1, 0($s4) #R[0]
    add $t1, $t1, $s6
    andi $t1, $t1, 0xFFFF #Get the lower 16-bits

    move $s0, $t1 #R[0] + i mod 2^16
    lw $s1, 4($s5) #K[1]
    lw $s2, 12($s5) #K[3]

    jal wfunction

    addi $sp, $sp, -16
    sw $t0, 0($sp) #Store t0
```

Figure 7. Second Loop First Step

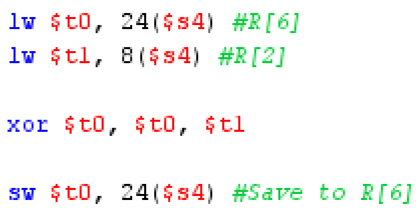
Consequently, values from the state vector are summed with previously calculated values at the first step which were retrieved from the stack. The respective circular shift is carried out with the help of masks and the computed value is stored back into the state vector. Below is an example to be seen.



```
lw $t1, 4($s4) #R[1]
lw $t4, 0($sp) #t0

add $t1, $t1, $t4 #R[1] + t0
andi $t1, $t1, 0xFFFF #R[1] + t0 mod 2^16

#Right Circular Shift Right 4
andi $t2, $t1, 0xF #0000000000001111 First 4 digits for the right circular shift
sll $t2, $t2, 12
andi $t3, $t1, 0xFFF0 #1111111111110000 Last 12 digits for the right circular shift
srl $t3, $t3, 4

or $t0, $t3, $t2
sw $t0, 4($s4) #Save to R[1]
```

Figure 8. Right Circular Shift Example

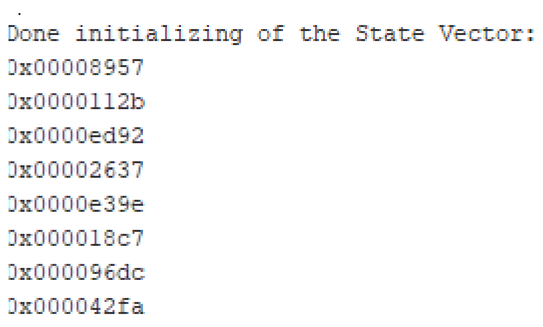
Lastly, values from different indices of the stack vector are XOR'ed with each other and the output is stored into the respective position.



```
lw $t0, 24($s4) #R[6]
lw $t1, 8($s4) #R[2]

xor $t0, $t0, $t1

sw $t0, 24($s4) #Save to R[6]
```

Figure 9. XOR Example

When the whole initialization process is run with the test vector the state vector shown in Figure 5 the state vector is initialized with the following values.



```
Done initializing of the State Vector:
0x00008957
0x0000112b
0x0000ed92
0x00002637
0x0000e39e
0x000018c7
0x000096dc
0x000042fa
```

Figure 10. Initialized State Vector