MIPS Assembly Implementation of an Encryption Algorithm

CS 401 - Computer Architectures
Term Project for Spring 2023-2024

A. U. Ay
C. Yıldırım
Computer Science & Engineering
Sabancı University
İstanbul

Abstract

You are required to implement a symmetric encryption algorithm in the MIPS Assembly Language. The description of the algorithm will be given in phases. See Appendix I for the schedule, the grading policy, and other details.

1 Introduction

The project will be completed in four phases. All phases are described in the following sections.

2 Phase I

2.1 Non-linear Function

The most important part of a symmetric encryption algorithm is its s-boxes (i.e., substitutions boxes), which implement a highly non-linear transformation. An s-box takes a certain number of bits as input and produces a certain number of bits as output. Since the s-box transformation is very complicated and requires bit-wise manipulation, a common technique is to use lookup tables.

The lookup tables of the cryptographic algorithm you will implement are given in Table 1, in which inputs and outputs are *nibbles* (i.e., 4-bit numbers) given in hexadecimal representation.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_0(x)$	2	F	C	1	5	6	A	D	E	8	3	4	0	B	9	7
$S_1(x)$	F	4	5	8	9	7	2	1	A	3	0	E	6	C	D	B
$S_2(x)$	4	A	1	6	8	F	7	C	3	0	E	D	5	9	B	2
$S_3(x)$	7	C	E	9	2	1	5	F	B	6	D	0	4	8	A	3

Table 1: S-boxes

You will store these four s-boxes in the memory and implement the following function:

$$S(X) = S_0(x_0)||S_1(x_1)||S_2(x_2)||S_3(x_3),$$

where $X = (x_0||x_1||x_2||x_3)$ is a 16-bit number while x_0, x_1, x_2 and x_3 are nibbles and $||$ stands for the concatenation operation. You will use two methods to implement the function $S(X)$ and compare the cache performance of these methods:

1. **Small tables method:** You will store the four separate tables in the memory and each s-box computation will result in one memory(/cache) access.
2. **Large tables method:** You implement more than one s-boxes in a single table. This way you can reduce the number of memory(/cache) accesses.

In your implementation, you will use the Cache Simulator of MARS and count the number of memory accesses, and calculate the cache miss rates in both cases. Assume that the cache size is at most 8 KB and you can adjust other cache features (block size, associativity, etc.) that will minimize the number of memory/cache accesses and maximize the cache performance in the computation of $S(X)$. To measure the memory/cache performance, you are required to generate randomly chosen 100 X values (i.e., 100 16-bit integers) and compute the corresponding $S(X)$ values[1]. Once you do so, you are expected to present your findings in the report.

The implementation with the best memory and cache performances (and the earliest submission) will receive 20% bonus points from this phase of the term project.

2.2 Linear Function

As the second step in the first phase, you will implement the linear function:

$$L(X) = X \oplus (X \lll 6) \oplus (X \ggg 6),$$

where X is a 16-bit number, $\oplus$ is bitwise XOR operation, $\lll$ and $\ggg$ are left and right circular shift (rotation), respectively. The output is also a 16-bit number. The following are two test vectors for the circular shift operation and the $L(X)$ function:

```
X = 0xbbaa -> X<<<6 = 0xeaae
X = 0x1111 -> X<<<6 = 0x4444
X = 0xbbaa -> L(X) = 0xfbea
X = 0x1111 -> L(X) = 0x1111
```

2.3 Permutation Function

For the third step of the first phase, you will implement a permutation function $P(X)$. This function gets an 8-bit number as its input and rearranges its bits according to a fixed permutation, which is specified in the permutation box (p-box) provided in Table 2. You can store this p-box in the memory. The function's output is also an 8-bit number, which is the permuted version of the input. In this table, x represents the index of each bit in the output, starting from the leftmost bit (i.e., 0 denotes the output's leftmost bit, 7 denotes its rightmost bit), and $p(x)$ refers to the

[1]You are not required to implement random number generation in MIPS.

index of the input bit that is mapped to the xth index of the output. For example, the bit in the 7th index of the input will be mapped to the 0th index of the output. Similarly, the bit in the 4th index of the input will be mapped to the 2nd index of the output, and so on.

x	0	1	2	3	4	5	6	7
$p(x)$	7	6	4	2	3	0	5	1

Table 2: P-box

The following are two test vectors for the $P(X)$ function:

```
X = 0b11010110 -> P(X) = 0b01001111
X = 0b10110001 -> P(X) = 0b10011100
```

3 Phase II

3.1 F Function

The first step is to compute the function $F(X) = L(S(x_2||x_3||P(x_0||x_1)))$. Here, $S(X)$ and $L(X)$ are the s-box and linear functions, which were implemented in the first phase of the project. In $F(X)$, the input $X = (x_0||x_1||x_2||x_3)$ is a 16-bit number while x_0, x_1, x_2 and x_3 are nibbles. The output is also a 16-bit number.
The following are two test vectors for the $F(X)$ function:

```
X = 0xbbaa -> F(X) = 0xa9e8
X = 0x1111 -> F(X) = 0x1516
```

3.2 W Function

The second step is to compute the function:

$$W(X, A, B) = F(F(X \oplus A) \oplus B),$$

where the inputs X, A, and B and the output are all 16-bit numbers. The following is a test vector for the $W(X)$ function:

```
X = 0xbbaa
A = 0xcccc
B = 0xdddd
W(X, A, B) = 0xf0ad
```

3.3 Initialization of State Vector

In the final step of the second phase, you are required to implement the initialization step of the encryption algorithm given in Algorithm 1.

Algorithm 1 Initialization

```
Require: Initial Vector:$\{IV_0, IV_1, IV_2, IV_3\}$ and Key:$\{K_0, K_1, \ldots, K_7\}$
Ensure: State Vector:$\{R_0, R_1, \ldots, R_7\}$
 1: for $i = 0$ to 7 do
 2:     $R_i^{(0)} \leftarrow IV_{i \bmod 4}$
 3: end for
 4: for $i = 0$ to 3 do
 5:     $t_0 \leftarrow W((R_0^{(i)} + i) \bmod 2^{16}, K_1, K_3)$
 6:     $t_1 \leftarrow W((R_1^{(i)} + t_0) \bmod 2^{16}, K_5, K_7)$
 7:     $t_2 \leftarrow W((R_2^{(i)} + t_1) \bmod 2^{16}, K_0, K_2)$
 8:     $t_3 \leftarrow W((R_3^{(i)} + t_2) \bmod 2^{16}, K_4, K_6)$
 9:     $R_0^{(i+1)} \leftarrow ((R_0^{(i)} + t_3) \bmod 2^{16}) \lll 7$
10:     $R_1^{(i+1)} \leftarrow ((R_1^{(i)} + t_0) \bmod 2^{16}) \ggg 4$
11:     $R_2^{(i+1)} \leftarrow ((R_2^{(i)} + t_1) \bmod 2^{16}) \lll 2$
12:     $R_3^{(i+1)} \leftarrow ((R_3^{(i)} + t_2) \bmod 2^{16}) \ggg 9$
13:     $R_4^{(i+1)} \leftarrow R_4^{(i)} \oplus R_3^{(i+1)}$
14:     $R_5^{(i+1)} \leftarrow R_5^{(i)} \oplus R_1^{(i+1)}$
15:     $R_6^{(i+1)} \leftarrow R_6^{(i)} \oplus R_2^{(i+1)}$
16:     $R_7^{(i+1)} \leftarrow R_7^{(i)} \oplus R_0^{(i+1)}$
17: end for
18: for $i = 0$ to 7 do
19:     $R_i = R_i^{(4)}$
20: end for
```

In the algorithm, the operation $x + y \bmod 2^{16}$ is the addition of two 16-bit numbers returning only the least significant 16-bit of the result (i.e., ignoring the carry). The following is a test vector for the initialization step:

```
K[8] = {0x2301, 0x6745, 0xAB89, 0xEFCD, 0xDCFE, 0x98BA, 0x5476, 0x1032}
IV[4] = {0x3412, 0x7856, 0xBC9A, 0xF0DE}
R[8] = {0x8957, 0x112b, 0xed92, 0x2637, 0xe39e, 0x18c7, 0x96dc, 0x42fa}
```

The earliest (and correctly functioning) submission will receive a 10% bonus from this phase of the project.

4 Phase III

In this phase of the project, you will encrypt a message which consists of 16-bit integers. The encryption of a single 16-bit message word, P is performed as described in Algorithm 2 and a 16-bit ciphertext C is generated; i.e., $C = E(P)$. For encryption of longer messages, the message is partitioned into 16-bit binary numbers, and each 16-bit is encrypted individually.

Algorithm 2 Encryption

```
Require: Plaintext P, State Vector:$\{R_0, R_1, \ldots, R_7\}$ and Key:$\{K_0, K_1, \ldots, K_7\}$
Ensure: Ciphertext C
 1: $t_0 \leftarrow W((R_0 + P) \bmod 2^{16}, L(K_0 \oplus R_0), L(K_1 \oplus R_1))$
 2: $t_1 \leftarrow W((R_1 + t_0) \bmod 2^{16}, L(K_2 \oplus R_2), L(K_3 \oplus R_3))$
 3: $t_2 \leftarrow W((R_2 + t_1) \bmod 2^{16}, L(K_4 \oplus R_4), L(K_5 \oplus R_5))$
 4: $C \leftarrow W((R_3 + t_2) \bmod 2^{16}, L(K_6 \oplus R_6), L(K_7 \oplus R_7)) + R_0 \bmod 2^{16}$
 5: $T_0 \leftarrow (R_0 + t_2) \bmod 2^{16}$
 6: $T_1 \leftarrow (R_1 + t_0) \bmod 2^{16}$
 7: $T_2 \leftarrow (R_2 + t_1) \bmod 2^{16}$
 8: $T_3 \leftarrow (R_3 + R_0 + t_2 + t_0) \bmod 2^{16}$
 9: $T_4 \leftarrow R_4 \oplus ((R_3 + R_0 + t_2 + t_0) \bmod 2^{16})$
10: $T_5 \leftarrow R_5 \oplus ((R_1 + t_0) \bmod 2^{16})$
11: $T_6 \leftarrow R_6 \oplus ((R_2 + t_1) \bmod 2^{16})$
12: $T_7 \leftarrow R_7 \oplus ((R_0 + t_2) \bmod 2^{16})$
13: for $i = 0$ to 7 do
14:     $R_i = T_i$
15: end for
```

The following are test vectors for the encryption operation:

```
K[8] = {0x2301, 0x6745, 0xAB89, 0xEFCD, 0xDCFE, 0x98BA, 0x5476, 0x1032}
IV[4] = {0x3412, 0x7856, 0xBC9A, 0xF0DE}
R[8] = {0x8957, 0x112b, 0xed92, 0x2637, 0xe39e, 0x18c7, 0x96dc, 0x42fa}
P[8] = {0x1100, 0x3322, 0x5544, 0x7766, 0x9988, 0xBBAA, 0xDDCC, 0xFFEE}
C[8] = {0x926a, 0xf9f8, 0x5bc5, 0xb575, 0x9707, 0x06a0, 0x3407, 0x33f2}
```

5 Phase IV

5.1 Inverse Functions

In this phase of the project, you will implement the decryption operation. For this, you need to implement the inverse of every operation used in the encryption algorithm and they have to be applied in the reverse order. The initialization step is identical to that of the encryption step.

5.1.1 Inverse of S Function

To compute the inverse of S Function, you need the inverses of s-boxes used in the encryption operation, given in Table 3. The inputs and outputs of the inverse s-boxes are nibbles given in hexadecimal representation. consequently, the inverse of S Function can be given as follows:

$$S^{-1}(X) = S_0^{-1}(x_0)||S_1^{-1}(x_1)||S_2^{-1}(x_2)||S_3^{-1}(x_3),$$

where $X = (x_0||x_1||x_2||x_3)$ is a 16-bit number while x_0, x_1, x_2 and x_3 are nibbles and $||$ stands for the concatenation operation.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_0^{-1}(x)$	C	3	0	A	B	4	5	F	9	E	6	D	2	7	8	1
$S_1^{-1}(x)$	A	7	6	9	1	2	C	5	3	4	8	F	D	E	B	0
$S_2^{-1}(x)$	9	2	F	8	0	C	3	6	4	D	1	E	7	B	A	5
$S_3^{-1}(x)$	B	5	4	F	C	6	9	0	D	3	E	8	1	A	2	7

Table 3: Inverse S-boxes

5.1.2 Inverse of L Function

The inverse of the L Function can be computed as follows:

$$\begin{aligned} L(X) &= Y \\ Z &= Y \oplus (Y \lll 10) \oplus (Y \ggg 10) \\ L^{-1}(Y) &= Z \oplus (Z \lll 4) \oplus (Z \ggg 4) \end{aligned}$$

Note that $L^{-1}(Y)$ denotes the inverse function of $L(X)$. In other words, $L^{-1}(Y) = X$, where $L(X) = Y$.

5.1.3 Inverse of P Function

To obtain the inverse of the P Function, you will require the inverse of the p-box that was used in the P Function. You can find the inverse p-box in Table 4. Similar to the P Function, the inverse of the P Function also takes an 8-bit number as input and generates an 8-bit number as output. The input bits are rearranged based on the inverse permutation provided in Table 4.

x	0	1	2	3	4	5	6	7
$p^{-1}(x)$	5	7	3	4	2	6	1	0

Table 4: Inverse P-box

5.1.4 Inverse of F Function

The inverse of the F Function can be calculated as follows:

$$\begin{aligned} F(X) &= Y \\ Z &= S^{-1}(L^{-1}(Y)) \\ F^{-1}(Y) &= P^{-1}(z_2||z_3)||z_0||z_1 \end{aligned}$$

Note that $F^{-1}(Y)$ denotes the inverse function of $F(X)$. In other words, $F^{-1}(Y) = X$, where $F(X) = Y$.

5.1.5 Inverse of W Function

The inverse of the W Function is given as follows:

$$W^{-1}(X, A, B) = F^{-1}(F^{-1}(X) \oplus B) \oplus A.$$

5.2 Decryption Algorithm

Before the decryption operation, the initialization operation in Algorithm 1 is executed to set the state vector R to its value before the encryption operation.

The decryption operation takes a 16-bit ciphertext word and decrypts it, updates the state vector, and move to the decryption of the next ciphertext word. These steps are repeated until all ciphertext words are decrypted. The decryption operation is simply the inverse of the encryption operation. The decryption steps are described in Algorithm 3.

Algorithm 3 Decryption

```
Require: Ciphertext C, State Vector:$\{R_0, R_1, \ldots, R_7\}$ and Key:$\{K_0, K_1, \ldots, K_7\}$
Ensure: Plaintext P
 1: $t_2 \leftarrow (W^{-1}((C - R_0) \bmod 2^{16}, L(K_6 \oplus R_6), L(K_7 \oplus R_7)) - R_3) \bmod 2^{16}$
 2: $t_1 \leftarrow (W^{-1}(t_2, L(K_4 \oplus R_4), L(K_5 \oplus R_5)) - R_2) \bmod 2^{16}$
 3: $t_0 \leftarrow (W^{-1}(t_1, L(K_2 \oplus R_2), L(K_3 \oplus R_3)) - R_1) \bmod 2^{16}$
 4: $P \leftarrow (W^{-1}(t_0, L(K_0 \oplus R_0), L(K_1 \oplus R_1)) - R_0) \bmod 2^{16}$
 5: $T_0 \leftarrow (R_0 + t_2) \bmod 2^{16}$
 6: $T_1 \leftarrow (R_1 + t_0) \bmod 2^{16}$
 7: $T_2 \leftarrow (R_2 + t_1) \bmod 2^{16}$
 8: $T_3 \leftarrow (R_3 + R_0 + t_2 + t_0) \bmod 2^{16}$
 9: $T_4 \leftarrow R_4 \oplus ((R_3 + R_0 + t_2 + t_0) \bmod 2^{16})$
10: $T_5 \leftarrow R_5 \oplus ((R_1 + t_0) \bmod 2^{16})$
11: $T_6 \leftarrow R_6 \oplus ((R_2 + t_1) \bmod 2^{16})$
12: $T_7 \leftarrow R_7 \oplus ((R_0 + t_2) \bmod 2^{16})$
13: for $i = 0$ to 7 do
14:     $R_i = T_i$
15: end for
```

Note that the steps for updating the state vector R (i.e., Steps 5-15) are identical to those in the encryption operation.

In the first step of this final phase, you are required to use the following test vectors:

```
K[8] = {0x2301, 0x6745, 0xAB89, 0xEFCD, 0xDCFE, 0x98BA, 0x5476, 0x1032}
IV[4] = {0x3412, 0x7856, 0xBC9A, 0xF0DE}
R[8] = {0x8957, 0x112b, 0xed92, 0x2637, 0xe39e, 0x18c7, 0x96dc, 0x42fa}
P[8] = {0x1100, 0x3322, 0x5544, 0x7766, 0x9988, 0xBBAA, 0xDDCC, 0xFFEE}
C[8] = {0x926a, 0xf9f8, 0x5bc5, 0xb575, 0x9707, 0x06a0, 0x3407, 0x33f2}
```

Your decryption function should return the plaintext `P[8]`, given the ciphertext `C[8]`.

In the second step of this phase, your program will take the plaintext input from the keyboard, and performs the following steps:

1. Print out the plaintext
2. Encrypt the plaintext and obtain the ciphertext
3. Decrypt the ciphertext and print out the resulting text

where the printouts in Step 1 and Step 3 should be identical.

6 Appendix I: Timeline & Deliverables & Weight & Policies etc.

Project Phases	**Deliverables**	**Due Date**	**Weight**
Project announcement		09/05/2024	
First Phase	Source Codes & Report	19/05/2024	0% [2]
Second Phase	Source Codes & Report	26/05/2024	30%
Third Phase	Source Codes & Report	02/06/2024	30%
Fourth Phase	Source Codes & Final Report	08/06/2024	40%

Table 5: Timetable

6.1 Policies

- You are expected to implement Phase I individually.
- You may work in groups of two for Phases II, III, and IV.
- Submit all deliverables in the zip file "cs401_TPphaseX_SUusername1_SUusername2.zip".
- Include a short report in the submission of each phase, along with the source code. This report should contain any analyses (if requested) in that phase, as well as a brief description regarding the key components of your code (e.g., where the results are stored after your code is executed, how you implement the methods such as small/large tables, etc.).
- You may be asked to demonstrate a project phase to the TA or instructor.
- Your codes will be checked for their similarity to other students' codes; and if the similarity score exceeds a certain threshold you will not get any credit for your work.

[2] will be considered as Lab 3