MIPS Assembly Implementation of an Encryption Algorithm

CS 401 - Computer Architectures
Term Project for Spring 2022-2023

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

Abstract

You are required to implement a symmetric encryption algorithm in MIPS Assembly Language. The description of the algorithm will be given in phases. Therefore, you will access the subsequent phases only after having finished the current phase (and all previous phases). See Appendix I for the schedule, the grading policy and the other details.

1 Introduction

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

2 Phase I

The most important part of a symmetric encryption algorithm is its s-boxes (i.e., substitutions boxes), which implements 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.

In this phase your program will read four lookup tables from a text file that contains table entries separated by a space character. The lookup tables contain 256 entries and each entry is a 32 bit number. The file is given in the attachment and named `tables.dat`. As the file contains character strings, the primary task of your program is to convert the table entries to binary values and store them on the heap memory. And the starting addresses of the tables on the heap should be kept in words in static memory (i.e., you are essentially implementing the pointer concept).

```
.data
T0:  .space 4
T1:  .space 4
T2:  .space 4
T3:  .space 4
...
```

See the template Assembly program (`phase1_template.asm`) given in the attachment. See also Appendix II for the explanations for opening, reading, and closing file and allocating space on the heap memory.

3 Phase II

In this phase of the term project, you will implement two different operations which are round operation and round key generation. The details of operations are described below.

3.1 Round Operation

Round operation is one of the primitive operations of the encryption algorithm which you are implementing in the term project. This operation takes two imputs, which are the round key `r`, and the state `s`. Each of the inputs is a 128-bit integer. Moreover, four look up tables (namely `T0`, `T1`, `T2`, and `T3`) are used. You will use the lookup tables which you read them from `tables.dat` in Phase I. The output of the operation is the next state, `t`, which is also a 128-bit integer. The round operation is implemented mainly by table look ups as follows:

```
t[0] = T3[s[0]>>24]^T1[(s[1]>>16)&0xff]^T2[(s[2]>>8)&0xff]^T0[s[3]&0xff]^rkey[0]
t[1] = T3[s[1]>>24]^T1[(s[2]>>16)&0xff]^T2[(s[3]>>8)&0xff]^T0[s[0]&0xff]^rkey[1]
t[2] = T3[s[2]>>24]^T1[(s[3]>>16)&0xff]^T2[(s[0]>>8)&0xff]^T0[s[1]&0xff]^rkey[2]
t[3] = T3[s[3]>>24]^T1[(s[0]>>16)&0xff]^T2[(s[1]>>8)&0xff]^T0[s[2]&0xff]^rkey[3]
```

where the symbols `^`, `&`, and `>>` stand for logical XOR, logical AND, and logical right shift operations, respectively. Also, the round key `rkey`, the state `s`, and the next state `t` are given as word arrays of four elements (i.e., word = 32-bit unsigned integer).

As test vectors to check the correctness of your program you can use the round key `rkey`, the state `s` given in the data section of your program:

```
...
s:  .word 0xd82c07cd, 0xc2094cbd, 0x6baa9441, 0x42485e3f
rkey:  .word 0x82e2e670, 0x67a9c37d, 0xc8a7063b, 0x4da5e71f
...
```

Assuming the convention `s[0] = 0xd82c07cd`, the output of your program must be

```
t = [0x2892750e, 0x949a0d1f, 0x70523edc, 0xc6933381]
```

for these test vectors.

3.1.1 Cache Performance

In the second part of this section, you will design a cache to increase the look up table access performance by increasing the cache hit rate. Table 1 contains different cache configurations you will try. Enable `Tools->Data Cache Simulator` in MARS simulator, run your program with the cache configurations in Table 1 and answer the following questions:

1. Find the hit rates for each cache configurations.
2. Which cache configuration yields the best performance?
3. Does the largest cache always give the best result? Why or why not?

Block Size	No. of Blocks	Cache Size	Hit Rate
4 B	512	2048 B	?
8 B	128	1024 B	?
16 B	32	512 B	?
32 B	16	512 B	?
64 B	8	512 B	?

Table 1: Cache Configuration

3.2 Key Schedule

The encryption algorithm executes the round operation eight times using 8 different round keys, each of which is a 128-bit number. Round keys are derived from the secret key, which is also a 128-bit number. Before the rounds are executed, we perform the following initialization operation

```
rkey[i] = key[i] for i = 0,1,2,3,
```

where `key` and `rkey` stand for the secret key and round key, respectively. Both are implemented as word arrays. Before every round, the round key is updated as follows:

```
a = (rkey[2] >> 24) & 0xFF
b = (rkey[2] >> 16) & 0xFF
c = (rkey[2] >> 8) & 0xFF
d = rkey[2] & 0xFF

e = (T2[b]&0xFF)^ rcon[i]
f = T2[c]&0xFF
g = T2[d]&0xFF
h = T2[a]&0xFF

tmp = (e << 24) ^ (f << 16) ^ (g << 8) ^ h
rkey[0] = tmp ^ rkey[0]
rkey[1] = rkey[0] ^ rkey[1]
rkey[2] = rkey[1] ^ rkey[2]
rkey[3] = rkey[2] ^ rkey[3].
```

Here the variable `rcon` is defined as follows:

```
rcon = [0x80, 0x40, 0x20, 0x10, 0x08, 0x04, 0x02, 0x01]
```

The following are the test vectors for the key schedule operation:

```
key         = [0x6920e299, 0xa5202a6d, 0x656e6368, 0x69746f2a]
round key 1 = [0x76dba7d4, 0xd3fb8db9, 0xb695eed1, 0xdfe181fb]
round key 2 = [0x1cf3999a, 0xcf081423, 0x799dfaf2, 0xa67c7b09]
round key 3 = [0x62de102c, 0xadd6040f, 0xd44bfefd, 0x723785f4]
round key 4 = [0xc1654464, 0x6cb3406b, 0xb8f8be96, 0xcacf3b62]
round key 5 = [0x88cbd408, 0xe4789463, 0x5c802af5, 0x964f1197]
round key 6 = [0x412e3242, 0xa556a621, 0xf9d68cd4, 0x6f999d43]
round key 7 = [0xb54a7adb, 0x101cdcfa, 0xe9ca502e, 0x8653cd6d]
round key 8 = [0xc0194bc5, 0xd005973f, 0x39cfc711, 0xbf9c0a7c]
```

4 Phase III

In this phase of the project, you are required to implement the encryption algorithm and take input from the user via keyboard.

4.1 Encryption

The encryption operation is simply executing the round operation eight times using the corresponding round key, except for *key whitening* operation which is performed before the round operations. The key whitening operation is simply the bitwise XOR of the secret key and the message, namely

```
s[i] = m[i] ^ key[i] for i = 0, 1, 2, 3
```

where `m` is the message and assumed to be also a 128-bit integer. After the key whitening operation is performed, the round operations applied to update the state `s`; namely `s` is updated with `t` after the round operation as given in Phase 2. The following are the test vectors for the encryption operation:

```
key        = [0x2b7e1516, 0x28aed2a6, 0xabf71588, 0x09cf4f3c]
message    = [0x6bc1bee2, 0x2e409f96, 0xe93d7e11, 0x7393172a]
ciphertext = [0x5c6e212b, 0x5b04b3a0, 0xc9e939d1, 0xa1680daf]
```

4.2 Keyboard Input

Your program will take input from the user via keyboard. As the inputs can be character strings (e.g., your name), you need to convert it to binary. If the input is not a multiple of 128-bit (or shorter) you need to pad the input with 0s to make it so. Then you encrypt and obtain the ciphertext. Note that you are required to display both the input and output on the display in hexadecimal format. For example, when you enter `utku.ay@sabanciuniv.edu`, your program should display

```
0x75746b752e617940736162616e636975
```

`0x6e69762e6564750000000000000000000`

as the input and

`0x919cbe4393809f0d455a1ec41b44a650`
`0xcd5c0409cefa5e73d308c7cc6adec2b2`

as the ciphertext output when the key is

`0x2b7e151628aed2a6abf7158809cf4f3c`.

5 Bonus

In the bonus part, you will develop a MIPS Assembly code that encrypts the plaintext entered from the keyboard, encrypt it and display it in the monitor, which is very similar to what you have done in Phase III.

However, with the keyboard and display here we mean the "Keyboard and Display Simulator" in the MARS simulator tool. You will use "Tools → Keyboard and Display Simulator" in this stage. Do not forget to reset the Keyboard and Display Simulator before running the programs.

To communicate with an I/O device you have two options:

1. **I/O with polling:** The assembly program given in the attachment "`echo_wpolling.s`" reads the characters entered and shows them on the Display using polling approach. Your job is to modify the given program so that its execution stops when a control character is entered; for example, the ASCII character of "ESC". This step will give you extra 5 bonus points.

2. **I/O with interrupts:** The assembly program given in the attachment "`echo_winterrupts.s`" reads the characters entered and shows them on the Display using interrupt IO approach. Your job is to modify the given program so that it stops displaying ASCII character entered when a control character is entered. This step will give you extra 10 bonus points.

Note that "`echo_wpolling.s`" and "`echo_winterrupts.s`" are just examples to help you understand the process. You can develop your own routines.

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

Project Phases	**Deliverables**	**Due Date**	**Weight**
Project announcement		17/05/2023	
First Phase	Source Codes	24/05/2023	30%
Second Phase	Source Codes	31/05/2023	40%
Third Phase	Source Codes	12/06/2023	30%

Table 2: Timetable

6.1 Policies

- You may work in groups of two.
- Submit all deliverables in the zip file "cs401_TPphaseX_SUusername1_SUusername2.zip".
- 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.

7 Appendix II: File Handling and Dynamic Memory Allocation on Heap

To open a file for reading, `syscall` instruction is executed after the following registers are set to proper values:

`$a0` = address of null-terminated string containing file name
`$a1` = flags
`$a2` = mode

After the call, the register `$v0` contains the file descriptor or becomes negative if an error occurs. To read from file, `syscall` instruction is executed after the following registers are set to proper values:

`$a0` = file descriptor
`$a1` = address of the input buffer
`$a2` = maximum number of characters to read

After the call, the register `$v0` contains number of characters read (0 if end-of-file, negative if error). To close a file, syscall instruction is executed after `$a0` is set to the file descriptor. To dynamically allocate space on the heap, `syscall` instruction is executed after register `$a0` is set to the number of bytes to allocate. After the call, register `$v0` contains the address of allocated memory. Note that you are asked to save that addresses in the memory referred by `T0`, `T1`, `T2` and `T3` in the template file.