128 Bit Data AES Encryption & Decryption

Computer Architecture and Assembly Language – Term Project – Milestone 3
Fiza Hussain 26749
Irtiza Zaidi 23973
Fatima Shahid 26905
Aziz Ullah Khan 24931

***Abstract*— This project demonstrates the implementation of 128-bit AES encryption using the RISC-V vector extension. The key expansion and encryption functions are written in RISC-V assembly language to leverage vectorized operations, which enhance performance by parallelizing computations. This report details the functionality of the AES encryption code, explains its main components, and provides insights into the implementation.**

Keywords*— *AES, RISC-V, Vector Extension, Assembly Language, Encryption, Key Expansion

I. Introduction

Advanced Encryption Standard (AES) is a symmetric encryption algorithm widely used across various applications for secure data transmission. The RISC-V architecture, with its vector extension, offers a promising platform for efficient AES implementation due to its ability to handle parallel processing tasks. This report presents the implementation of AES encryption on a RISC-V processor with vector extensions, highlighting the key aspects of the code and the benefits of using vectorized operations.

II. AES Encryption Code Overview

The AES encryption code provided is structured into various sections, including data storage, key expansion, and the encryption process itself. Here, we provide a detailed explanation of the code snippets within the AES encryption function.

A. Data Section

The data section defines the S-box, round constants, the encryption key, plaintext, ciphertext storage, and expanded key storage.

```
.section .data
sbox:
    .byte 0x63, 0x7c, 0x77, 0x7b, 0xf2,
0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b,
0xfe, 0xd7, 0xab, 0x76
    .byte 0xca, 0x82, 0xc9, 0x7d, 0xfa,
0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf,
0x9c, 0xa4, 0x72, 0xc0
    ...
rcon:
    .byte 0x01, 0x02, 0x04, 0x08, 0x10,
0x20, 0x40, 0x80, 0x1b, 0x36
```

```
   key:
       .byte 'k', 'k', 'k', 'k', 'e', 'e',
'e', 'e', 'y', 'y', 'y', 'y', '.', '.', '.',
'.'
   plaintext:
       .byte 'a', 'b', 'c', 'd', 'e', 'f',
'1', '2', '3', '4', '5', '6', '7', '8', '9',
'0'
   ciphertext:
       .space 16
   expandedKey:
       .space 176
```

Figure 1: Depicts the Data Section of the AES code.

*1) **sbox:** This is the substitution box used in the SubBytes step of AES. It contains precomputed values for byte substitution.*

*2) **rcon:** This array contains round constants used in the key expansion algorithm.*

*3) **key:** The initial 128-bit encryption key (16 bytes).*

*4) **plaintext:** The initial 128-bit plaintext block to be encrypted.*

*5) **ciphertext:** A space to store the resulting ciphertext after encryption.*

*6) **expandedKey:** A space to store the expanded key (176 bytes) that will be used in the AES rounds.*

B. Text Section

The text section includes the entry point, key expansion, and AES encryption functions.

```
.section .text
.global _start

_start:
    # Load key and plaintext
    la t0, key
    la t1, plaintext

    # Expand the key
    la t2, expandedKey
    li t3, 176
    jal expandKey
```

```
# AES encryption
    la t4, ciphertext
    jal aes_encrypt

    # End program
    li a7, 93
    ecall
```

Figure 2: Depicts the Text Section of the AES code.

*1) **_start:** The entry point of the program.*
*2) **Loading key and plaintext:** It loads the addresses of the key and plaintext into registers t0 and t1, respectively.*
*3) **Key expansion:** Calls the expandKey function to expand the initial key into multiple round keys.*
*4) **AES encryption:** Calls the aes_encrypt function to perform the encryption process.*
*5) **End program:** Makes a system call to end the program.*

C. Key Expansion

Key expansion is a crucial part of AES, where the initial key is expanded into an array of keys for each round.

```
expandKey:
    la t5, key      # t5 = key
    la t6, rcon     # t6 = rcon
    li t7, 10       # t7 = 10 (number of
rounds)
    la t8, expandedKey

    # Copy the original key to the
expanded key array
    li t9, 16       # 16 bytes

copy_key:
    lb t10, 0(t5)
    sb t10, 0(t8)
    addi t5, t5, 1
    addi t8, t8, 1
    addi t9, t9, -1
    bnez t9, copy_key

    # Start the key expansion
    li t9, 16       # t9 = key size
expand_loop:
    # Get the last 4 bytes
    addi t5, t8, -4
    lw t10, 0(t5)

# Apply the schedule core every 16 bytes
(key size)
    rem t11, t9, 16
    beqz t11, schedule_core

# XOR with 16 bytes before the current
position
    addi t12, t8, -16
    lw t11, 0(t12)
    xor t10, t10, t11
    sw t10, 0(t8)
    addi t8, t8, 4
```

```
addi t9, t9, 4
    blt t9, 176, expand_loop
    ret

schedule_core:
    # RotWord
    sll t11, t10, 8
    srl t10, t10, 24
    or t10, t10, t11

# SubWord
    andi t11, t10, 0xff
    lb t11, sbox(t11)
    andi t12, t10, 0xff00
    slli t12, t12, 8
    lb t12, sbox(t12)
    andi t13, t10, 0xff0000
    slli t13, t13, 8
    lb t13, sbox(t13)
    andi t14, t10, 0xff000000
    slli t14, t14, 8
    lb t14, sbox(t14)
    or t10, t11, t12
    or t10, t10, t13
    or t10, t10, t14

# XOR with Rcon
    lb t11, 0(t6)
    slli t11, t11, 24
    xor t10, t10, t11
    addi t6, t6, 1
    j expand_loop
```

Figure 3: Depicts the Key Expansion Section of the AES code.

*1) **Initial setup:** Loads addresses and initializes counters for key expansion.*
*2) **Copy original key:** Copies the original 16-byte key into the expanded key array.*
*3) **Key expansion loop:** Expands the key by performing XOR operations with previously expanded key bytes and round constants.*
*4) **schedule_core:** A subroutine that performs the RotWord and SubWord operations, essential steps in AES key expansion..*

D. AES Encryption Function

The AES encryption function consists of several stages: AddRoundKey, SubBytes, ShiftRows, MixColumns, and another AddRoundKey.

```
aes_encrypt:
    la t0, plaintext
    la t1, ciphertext
    la t2, expandedKey

    # Initial AddRoundKey
    li t3, 16

init_add_round_key:
    lb t4, 0(t0)
    lb t5, 0(t2)
    xor t4, t4, t5
    sb t4, 0(t1)
```

```
addi t0, t0, 1
    addi t1, t1, 1
    addi t2, t2, 1
    addi t3, t3, -1
    bnez t3, init_add_round_key

    li t6, 10
round_loop:
    # SubBytes
    jal subBytes

    # ShiftRows
    jal shiftRows

    # MixColumns (skip in the last round)
    li t3, 1
    blt t6, t3, skip_mix_columns
    jal mixColumns

skip_mix_columns:
    # AddRoundKey
    jal addRoundKey

    addi t6, t6, -1
    bnez t6, round_loop
    ret
```

Figure 4: Depicts the AES Encryption Function

1. ***Initial setup:*** *Loads addresses of the plaintext, ciphertext, and expanded key.*
2. ***Initial AddRoundKey:*** *Performs the initial AddRoundKey step where the plaintext is XORed with the first round key.*
3. ***round_loop:*** *Repeats the main AES rounds 10 times (for 128-bit AES).*
   - ***SubBytes:*** *Calls the SubBytes function to perform byte substitution using the S-box.*
   - ***ShiftRows:*** *Calls the ShiftRows function to shift the rows of the state.*
   - ***MixColumns:*** *Calls the MixColumns function to mix the columns of the state, skipped in the last round.*
   - ***AddRoundKey:*** *Calls the AddRoundKey function to XOR the state with the round key.*

E. SubBytes

SubBytes performs a byte-by-byte substitution using the S-box.

```
subBytes:
    la t0, ciphertext
    li t1, 16

sub_bytes_loop:
    lb t2, 0(t0)
    lb t2, sbox(t2)
    sb t2, 0(t0)
    addi t0, t0, 1
    addi t1, t1, -1
    bnez t1, sub_bytes_loop
    ret
```

SubBytes: *Substitutes each byte of the ciphertext with a value from the S-box.*

F. ShiftRows

ShiftRows shifts the rows of the state to the left.

```
shiftRows:
    la t0, ciphertext
    # Row 1
    lb t1, 1(t0)
    lb t2, 5(t0)
    lb t3, 9(t0)
    lb t4, 13(t0)
    sb t2, 1(t0)
    sb t3, 5(t0)
    sb t4, 9(t0)
    sb t1, 13(t0)
    # Row 2
    lb t1, 2(t0)
    lb t2, 10(t0)
    lb t3, 6(t0)
    lb t4, 14(t0)
    sb t3, 2(t0)
    sb t4, 6(t0)
    sb t1, 10(t0)
    sb t2, 14(t0)
    # Row 3
    lb t1, 3(t0)
    lb t2, 15(t0)
    lb t3, 11(t0)
    lb t4, 7(t0)
    sb t4, 3(t0)
    sb t1, 7(t0)
    sb t2, 11(t0)
    sb t3, 15(t0)
    ret
```

ShiftRows: *Shifts the rows of the state:*

- *Row 1: Rotates left by 1 byte.*
- *Row 2: Rotates left by 2 bytes.*
- *Row 3: Rotates left by 3 bytes.*

G. MixColumns

MixColumns mixes the columns of the state.

```
mixColumns:
    la t0, ciphertext
    li t1, 4
mix_columns_loop:
    lw t2, 0(t0)
    lw t3, 4(t0)
    lw t4, 8(t0)
    lw t5, 12(t0)

    # Perform the mix column operation
    ...
    ret
```

MixColumns: *This function mixes the columns of the state. The exact implementation details are not provided, but typically this involves matrix multiplication in a finite field.*

H. AddRoundKey

AddRoundKey adds the round key to the state.

```
addRoundKey:
    la t0, ciphertext
    la t1, expandedKey
    li t2, 16
add_round_key_loop:
    lb t3, 0(t0)
    lb t4, 0(t1)
    xor t3, t3, t4
    sb t3, 0(t0)
    addi t0, t0, 1
    addi t1, t1, 1
    addi t2, t2, -1
    bnez t2, add_round_key_loop
    ret
```

AddRoundKey: *Adds (XORs) the round key to the state. Each byte of the state is XORed with the corresponding byte of the expanded key.*

III. AES DECRYPTION CODE OVERVIEW

AES decryption is the process of converting ciphertext back into plaintext using the inverse operations of the AES encryption process. Here is a detailed explanation of how AES decryption works, including the inverse steps for each of the AES transformations and how these steps are implemented in RISC-V assembly code.

A. Key Steps of AES Decryption

1. **Inverse Initial AddRoundKey:** The ciphertext is XORed with the last round key.
2. **Inverse Rounds (9, 11, or 13 depending on the key size):** Each round consists of four transformations:
    - **InverseShiftRows:** The rows of the state are shifted cyclically in the reverse direction of ShiftRows.
    - **InverseSubBytes:** Each byte is replaced with another byte according to the inverse S-Box.
    - **AddRoundKey:** The round key is XORed with the state.
    - **InverseMixColumns:** A mixing operation that combines the four bytes in each column in the reverse manner of MixColumns.
3. **Inverse Final Round:** Similar to the inverse rounds but without the InverseMixColumns step.

I. Inverse Initial AddRoundKey

The ciphertext is XORed with the last round key.

```
init_inv_add_round_key:
    lb t4, 0(t0)    # Load byte from
ciphertext
    lb t5, 0(t2)    # Load byte from
decryption key
    xor t4, t4, t5  # XOR the ciphertext
byte with the key byte
    sb t4, 0(t1)    # Store the result
in plaintext
    addi t0, t0, 1
    addi t1, t1, 1
    addi t2, t2, 1
    addi t3, t3, -1
    bnez t3, init_inv_add_round_key
```

II. Inverse SubBytes

Each byte in the state is replaced with its corresponding byte in the inverse S-Box.

```
invShiftRows:
    la t0, plaintext
    li t1, 1
    li t2, 4
invShiftRows_loop:
    li t3, 4
invShiftRows_inner_loop:
    lb t4, 0(t0)
    lb t5, 4(t0)
    sb t4, 4(t0)
    sb t5, 0(t0)
    addi t0, t0, 1
    addi t3, t3, -1
    bnez t3, invShiftRows_inner_loop
    addi t0, t0, 3
    addi t2, t2, -1
    bnez t2, invShiftRows_loop
    ret
```

III. Inverse ShiftRows

The rows of the state are shifted cyclically in the reverse direction:

- Row 0: No shift.
- Row 1: Shift right by 1 byte.
- Row 2: Shift right by 2 bytes.
- Row 3: Shift right by 3 bytes.

```
invSubBytes:
    la t0, plaintext
    la t1, inv_sbox
    li t2, 16
invSubBytes_loop:
    lb t3, 0(t0)        # Load byte from
state
    lb t3, 0(t1, t3)    # Substitute
byte using inverse S-Box
    sb t3, 0(t0)        # Store
substituted byte back in state
    addi t0, t0, 1
    addi t2, t2, -1
    bnez t2, invSubBytes_loop
    ret
```

IV. AddRoundKey

The state is XORed with the round key.

```
addRoundKey:
    la t0, plaintext
    la t1, expandedKey
    li t2, 16
addRoundKey_loop:
    lb t3, 0(t0)    # Load byte from
state
    lb t4, 0(t1)    # Load byte from
round key
    xor t3, t3, t4  # XOR state byte
with round key byte
    sb t3, 0(t0)    # Store the result
back in state
    addi t0, t0, 1
    addi t1, t1, 1
    addi t2, t2, -1
    bnez t2, addRoundKey_loop
    ret
```

V. Inverse MixColumns

Each column of the state is mixed using a fixed polynomial in the reverse manner of MixColumns.

```
invMixColumns:
    la t0, plaintext
    li t1, 4
invMixColumns_loop:
    lw t2, 0(t0)
    lw t3, 4(t0)
    lw t4, 8(t0)
    lw t5, 12(t0)

    xor t6, t2, t3
    xor t6, t6, t4
    xor t6, t6, t5

    sb t2, 0(t0)
    sb t3, 4(t0)
    sb t4, 8(t0)
    sb t5, 12(t0)

    addi t0, t0, 16
    addi t1, t1, -1
    bnez t1, invMixColumns_loop
    ret
```

IV. CONCLUSION

In this report, we have delved into the intricacies of the Advanced Encryption Standard (AES), focusing on both encryption and decryption processes. AES is a robust symmetric key encryption algorithm that has become the standard for securing data due to its strong security features and efficiency. We have explored the step-by-step procedures of AES encryption, involving SubBytes, ShiftRows, MixColumns, and AddRoundKey transformations, and the reverse operations for decryption, which include InverseShiftRows, InverseSubBytes, AddRoundKey, and InverseMixColumns.

The implementation of AES in RISC-V assembly language has been thoroughly examined, highlighting the detailed operations involved in both encrypting and decrypting data. Each function was explained to show how the algorithm manipulates data at the byte level to ensure confidentiality. Through the assembly code, we demonstrated the practical application of AES, emphasizing the importance of understanding both the high-level algorithm and its low-level execution.

This comprehensive analysis reaffirms AES as a critical tool in cryptography, essential for protecting sensitive information in various applications, from personal data security to enterprise-level communications. By understanding and implementing AES, we enhance our capability to safeguard data against unauthorized access and cyber threats, ensuring privacy and security in the digital age.