ECE 212
LAB-1 REPORT

Introduction to Arm Assembly Language

Name: Misbah Ahmed Nauman
CCID: misbahah
Student ID: 1830574

Partner: Muhammad Ayaan Hafeez
CCID: mhafeez1
Student ID: 1805075

Section: H25
Lab Technician: Wing Hoy
Lab Number: 1
Lab Date: 02.04.2025

Table of Contents

1. Introduction

This lab focused on developing two ARM assembly programs using the NUCLEO-L432KC board and STM32CubeIDE software. The objectives were:

- **Gain practical experience in THUMB2 assembly programming.**
- **Implement memory operations and data conversion in low-level programming.**
- **Use ARM assembly instructions to process ASCII characters and manage memory efficiently.**

The lab consisted of two programs:

- **Part A: Converts ASCII characters (digits 0-9, letters A-F/a-f) into their hexadecimal/decimal equivalent, storing results in memory or flagging invalid inputs (-1).**
- **Part B: Converts uppercase ASCII letters to lowercase and vice versa, handling invalid characters with an error code ('*').**

Both programs terminate when encountering the ASCII 'Enter' character (0x0D).

2. Design

2.1 Part A: ASCII to Hex/Decimal Conversion

1. **Initialize memory at `0x20001000` for input and `0x20002000` for output.**
2. **Load ASCII characters from memory.**
3. **Check for termination (ASCII 'Enter' character `0x0D`).**
4. **Validate the input:**
    - **Digits (0-9): `0x30–0x39`**
    - **Uppercase (A-F): `0x41–0x46`**
    - **Lowercase (a-f): `0x61–0x66`**
    - **Invalid characters: Anything else.**
5. **Convert based on category:**
    - **Digits: `value = ASCII - 0x30`**
    - **Uppercase: `value = ASCII - 0x41 + 10`**
    - **Lowercase: `value = ASCII - 0x61 + 10`**
    - **Invalid: Store `-1 (0xFFFFFFFF)`.**
6. **Store result and increment memory addresses.**

2.2 Part B: Case Conversion

1. Initialize memory at `0x20001000` for input and `0x20003000` for output.
2. Load ASCII characters from memory.
3. Check for termination (ASCII 'Enter' character `0x0D`).
4. Validate input:
    - Uppercase (A-Z): `0x41–0x5A`
    - Lowercase (a-z): `0x61–0x7A`
    - Invalid: Any other character.
5. Convert case:
    - Uppercase → Lowercase: `ASCII + 0x20`
    - Lowercase → Uppercase: `ASCII - 0x20`
    - Invalid: Store `'*' (0x2A)`.
6. Store result and increment memory addresses.

3. Testing Strategy and Results

3.1 Testing Approach:

- Used MTTTY to interact with the Nucleo board.
- Verified correct execution with predefined test cases.
- Ensured accurate conversion for valid inputs.
- Checked proper handling of invalid characters.
- Confirmed termination upon receiving ASCII 'Enter'.

3.2 Results:

All test cases produced expected outputs. No errors were observed during testing.

4. Discussion and Questions

4.1 Discussion

- The program executed as expected, converting ASCII values correctly.
- The memory structure efficiently handled 4-byte storage per character.
- Challenges included debugging unexpected branching errors and ensuring correct pointer increments.

4.2 Questions

1. **Why is `0x20` used for case conversion?**
    - **Uppercase and lowercase letters in ASCII differ by `0x20`.**
    - **Adding/subtracting `0x20` swaps the case.**
2. **How does memory alignment work?**
    - **Each character uses 4 bytes for proper word alignment.**
    - **Pointers increment by 4 for sequential access.**
3. **Why are `-1` and `'*'` used as error codes?**
    - **`-1` (`0xFFFFFFFF`) is an invalid hexadecimal representation.**
    - **`'*'` (`0x2A`) is a non-alphabetic ASCII character used for flagging errors.**

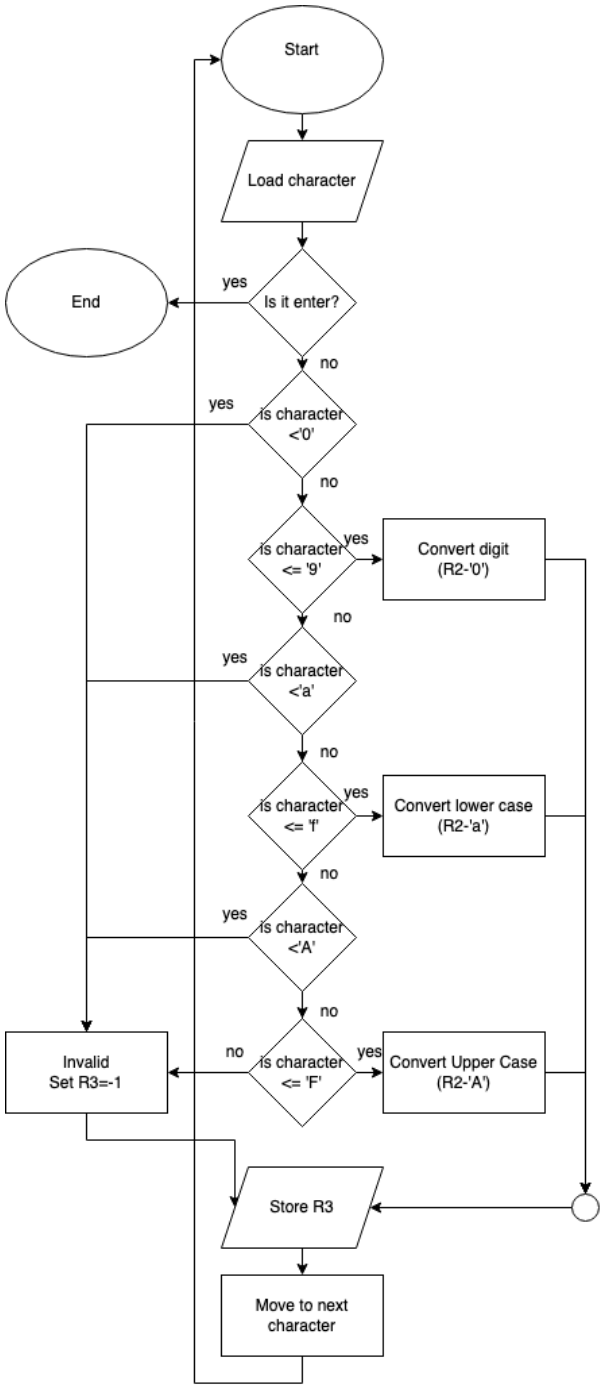
5. Conclusion

This lab provided hands-on experience in ARM assembly programming and memory operations. The programs met all functional requirements, correctly handling ASCII conversions. Challenges included ensuring precise input validation and optimizing memory access.

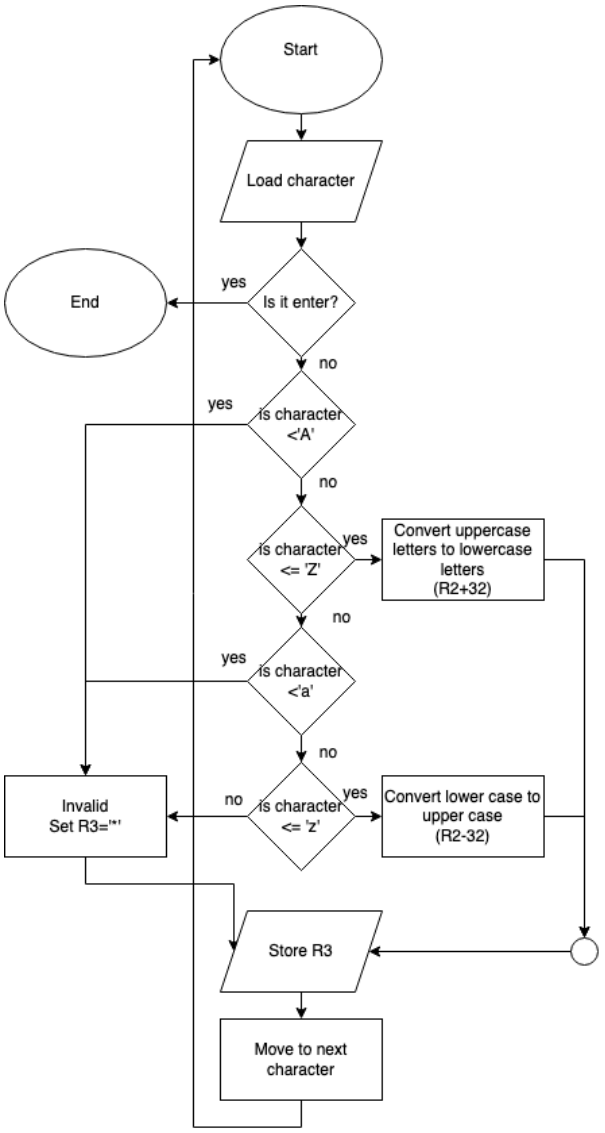
Future improvements could involve optimizing branching logic and expanding input validation for broader ASCII support.

6. Appendix

6.1 Flowchart



Flow Chart for Part A



Flow Chart for Part B

6.2 Assembler Code

```
/*-------------------------------------*/
/* Author: Misbah Ahmed Nauman, Student ID: 1830574 */
/* Author: Muhammad Ayaan Hafeez, Student ID: 1805075*/
/*-------------------------------------*/


/* Part A */
LDR R0, =0x20001000        // Load address of input data
LDR R1, =0x20002000        // Load address where results will be stored

loop:
    LDR R2, [R0]           // Load current character from input
    CMP R2, #0x0D          // Check if it is Enter key
    BEQ exit               // Exit if Enter is encountered

    CMP R2, #'0'           // Check if character is less than '0'
    BLT invalid_char       // If so, mark as invalid
    CMP R2, #'9'           // Check if character is between '0' and '9'
    BLE convert_digit      // Convert if valid digit

    CMP R2, #'A'           // Check if character is less than 'A'
    BLT invalid_char       // If so, mark as invalid
    CMP R2, #'F'           // Check if character is between 'A' and 'F'
    BLE convert_uppercase  // Convert if valid uppercase hex

    CMP R2, #'a'           // Check if character is less than 'a'
    BLT invalid_char       // If so, mark as invalid
    CMP R2, #'f'           // Check if character is between 'a' and 'f'
    BLE convert_lowercase  // Convert if valid lowercase hex

invalid_char:
    MOV R3, #-1            // Store error code -1 for invalid character
    STR R3, [R1]           // Store result
    B next_char            // Move to next character

convert_digit:
    SUB R3, R2, #'0'       // Convert ASCII digit to decimal
    B store_result

convert_uppercase:
    SUB R3, R2, #55        // Convert ASCII uppercase hex to decimal
    B store_result
```

```
convert_lowercase:
    SUB R3, R2, #87          // Convert ASCII lowercase hex to decimal

store_result:
    STR R3, [R1]             // Store converted result

next_char:
    ADD R0, R0, #4           // Move to next input character
    ADD R1, R1, #4           // Move to next output location
    B loop                   // Repeat loop

exit:
POP {PC}                     // Restore link register and return
```

```
/* Part B */
LDR R0, =0x20001000          // Load address of input data
LDR R1, =0x20003000          // Load address where results will be stored

loop:
    LDR R2, [R0]             // Load current character from input
    CMP R2, #0x0D            // Check if it is Enter key
    BEQ exit                 // Exit if Enter is encountered

    CMP R2, #'A'             // Check if character is less than 'A'
    BLT invalid_char         // If so, mark as invalid
    CMP R2, #'Z'             // Check if character is between 'A' and 'Z'
    BLE convert_uppercase    // Convert to lowercase

    CMP R2, #'a'             // Check if character is less than 'a'
    BLT invalid_char         // If so, mark as invalid
    CMP R2, #'z'             // Check if character is between 'a' and 'z'
    BLE convert_lowercase    // Convert to uppercase

invalid_char:
    MOV R3, #'*'             // Store error code '*' for invalid character
    STR R3, [R1]             // Store result
    B next_char              // Move to next character

convert_uppercase:
    ADD R3, R2, #32          // Convert uppercase to lowercase
```

```
    B store_result

convert_lowercase:
    SUB R3, R2, #32          // Convert lowercase to uppercase

store_result:
    STR R3, [R1]             // Store converted result

next_char:
    ADD R0, R0, #4           // Move to next input character
    ADD R1, R1, #4           // Move to next output location
    B loop // Repeat loop

exit:
POP {PC}                     // Restore link register and return
```