
Marmara University - Faculty of Engineering

Department of Computer Engineering

CSE4219 Principles of Embedded System Design (Fall 2024)

Submit Date: 03/11/2024.

Arm Cortex M4 Problems

Student Number (ID)	Name	Surname
150120998	Abdelrahman	Zahran
150120997	Mohamed Nael	Ayoubi
130319659	Cihan	Erdoğanılmaz

Sections Of the Report: -

- Section (1): Problem (1) - ARM Assembly Program for Repeated Digit Summation
- Section (2): Problem (2) - ARM Assembly Program for Matrix Column Swap
- Section (3): Problem (3) - ARM Assembly Program for Error Correcting Code (ECC)

All Inputs are given to programs as Data Memory Inputs – For Implementation Simplicity Purposes!!

This method is common in embedded systems, assembly language programming, and certain low-level software development contexts. It simplifies the process by avoiding complex input/output handling at runtime, enabling the program to fetch inputs directly from specified memory addresses in a predictable, structured way.

Section (1): Problem (1) - ARM Assembly Program for Repeated Digit Summation

This assembly program calculates the sum of a pattern based on two inputs: a (the base number) and n (the number of terms). The pattern is defined as:

$$F(a,n)=a+aa+aaa+\dots (n \text{ terms})$$

For example, if a = 3 and n = 5, it computes:

$$F(3,5)=3+33+333+3333+33333$$

The final sum is stored in r0.

Program Overview

1. Initialize Variables:

- ♦ r1 holds a, r2 holds n, r0 accumulates the result, r3 is the loop counter, r4 stores the current term, and r5 is the multiplication factor (powers of 10).

2. Calculate Multiplication Factor:

- ◆ We loop through a to set up r5 with the appropriate power of 10 to shift a left each time (e.g., 3 to 33, then 333).

3. Loop for Summation:

- ◆ Each iteration forms the term by multiplying r4 by 10 and adding a. The term is added to the total in r0.
- ◆ The loop stops after n terms.

4. Program End:

- ◆ The program enters an infinite loop at stop.

Inputs & Outputs:

- 1) A = 3, B = 5 → Output: 37035 (In Hex.)
 - 2) A = 45, B = 3 → Output: 459135 (In Hex.)
-

Section (2): Problem (2) - ARM Assembly Program for Matrix Column Swap

This assembly program swaps two columns in a 3x3 integer matrix stored as a single-dimensional array. The matrix is stored in row-major order, and we use zeroed memory (zMem) to store the swapped result. The column indexes are zero-based and provided as inputs.

Program Overview

1. Initialize Inputs:

- ◆ r0 and r1 hold the zero-based column indexes to be swapped.
- ◆ The address of the matrix is loaded into r2, and the address of zMem (output storage) is loaded into r3.

2. Column Index Calculation:

- ◆ We calculate the memory offset for each element in the two columns to be swapped. This is done by using the formula $(i * \text{Num of Columns} + j) * 4$ for addressing each element in a specific row i and column j.
- ◆ The program iterates over each row, swapping the values in the specified columns while copying the remaining column to zMem.

3. Main Loop:

- ◆ For each row in the matrix:
 - Calculate the address of elements in the two columns to be swapped.
 - Load the elements from these positions, then store them in zMem with swapped positions.
 - Copy the value from the remaining column into zMem.
- ◆ The loop iterates for all rows (3 in total), completing the swap across all rows.

4. End Program:

- ◆ After all rows are processed, the program halts by entering an infinite loop.

Inputs & Outputs:

In the program the 2D Matrix is stored in Data Memory Section as a one-dimensional array, each array element is stored in one byte memory size resulting into $9 \times 4 = 36$ bytes of memory for each matrix before swap and after swap.

For Memory offset Calculation for each element in the two columns to be swapped. The following formula is used

$$(i * \text{Num of Columns} + j) * 4$$

for addressing each element in a specific row i and column j .

Matrix:

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}$$

After Swapping:

$$\begin{bmatrix} 1 & 3 & 2 \\ 1 & 3 & 2 \\ 1 & 3 & 2 \end{bmatrix}$$

Column 1 (2, 2, 2) and Column 2 (3, 3, 3) are swapped.

Before: (1, 2, 3) (1, 2, 3) (1, 2, 3) in Data Memory.

After : (1, 3, 2) (1, 3, 2) (1, 3, 2) in Data Memory (Zero Memory Partition).

Section (3): Problem (3) - ARM Assembly Program for Error Correcting Code (ECC)

This assembly program generates and stores Error Correcting Codes (ECC) for an 8-bit input, expanding it to 13 bits by adding 5 parity bits. ECC helps in detecting and correcting data errors. Here, each parity bit follows even parity, meaning it is set to 1 if the number of 1s in specific bit positions is odd; otherwise, it's 0.

Program Overview

1. Initialize Inputs:

- ◆ The 8-bit input data (input) is stored in memory and loaded into a register.
- ◆ A zeroed 13-bit memory block (zMem) is reserved to store the expanded data with ECC.

2. Bit Extraction and Placement:

- ◆ The program extracts each data bit and shifts it to the appropriate position in a 13-bit register (r3).
- ◆ The bits are placed according to the specified 13-bit format with gaps left for the parity bits (p0, p1, p2, p4, and p8).

3. Parity Masks:

- ◆ Parity masks are loaded into registers (r4 to r8) to isolate the bits that each parity bit (p1, p2, p4, p8, p0) is responsible for.
- ◆ These masks represent the positions that each parity bit checks according to the problem specifications.

4. Parity Bit Calculation:

- ◆ For each parity bit, the program performs the following steps:
 - **Isolate and Count:** It performs an AND operation with the parity mask to isolate the relevant bits.

- **Even Parity Check:** It counts the 1s using XOR to determine even or odd parity. If the parity is odd, the parity bit is set to 1; if even, it remains 0.
- The parity bit is then stored in its designated position within r3.

5. **Store Result:**

- ◆ After calculating all parity bits, the final 13-bit expanded data is stored in zMem.
