CSCI 592

LAB ASSIGNMENT – 9

Written by

DINESH SEVETI

Date: 04-12-2025

**OBJECTIVE**

The main objective of the lab is to grasp the concept of Hamming code, which is a special technique for encoding and decoding information to enable error detection and correction.

**TECHNOLOGY USED**

* Simulator: EASy68K (Motorola 68000 Assembly)
* Programming Language: Assembly Language (Motorola 68K Syntax)
* Concepts: Hamming (7,4) Code, Parity Calculation, Bitwise Logic, Memory Access

**PROCEDURE**

* Store the 4-bit data input in memory location $002400 (encoder) and encoded bytes at $002500 (decoder).
* Use bitwise operations to extract individual bits a, b, c, and d.
* Calculate the parity bits r, s, and t using XOR logic.
* Construct the 7-bit encoded byte in the format a b c r d s t.
* For decoding, extract all bits from the encoded byte.
* Recalculate parity conditions and compare them to the original parity bits.
* Determine if a single-bit error occurred and store its position in register D0.

**OPERATIONS**

* Use MOVE.B to load input bytes and store encoded results.
* Use bit masking (AND.W) and shifting (LSR.W, LSL.W) to isolate and align bits.
* Compute parity using EOR.W (XOR) operations.
* Assemble encoded output in register D1 using sequential merging of bits.
* For decoding, repeat parity calculations and use conditional checks (CMP.B, BEQ, ADDI.B) to find error location.

**ALGORITHM**

Encoding Steps:

* Extract bits a, b, c, d from input nibble.
* Compute:
  + r = a ⊕ b ⊕ c
  + s = a ⊕ b ⊕ d
  + t = a ⊕ c ⊕ d
* Assemble encoded byte in 7-bit format: a b c r d s t

Decoding Steps:

* Extract bits a, b, c, r, d, s, t from input.
* Recompute:
  + r' = a ⊕ b ⊕ c ⊕ d
  + s' = a ⊕ b ⊕ d ⊕ s
  + t' = a ⊕ c ⊕ d ⊕ t
* Combine r', s', t' as a binary value to indicate error position (0 = no error)

**CODE LISTING**

ENCODER

*ORG $1000*

*START:*

*MOVE.L #0,D2*

*MOVE.L #0,D3*

*MOVE.L #0,D4*

*MOVE.L #0,D5*

*MOVE.L #0,D6*

*MOVE.L #0,D7*

*MOVE.B #11,$00002400*

*LEA.L $00002400,A0*

*MOVE.B (A0),D1*

*MOVE.L #1,D2*

*MOVE.L #2,D3*

*MOVE.L #4,D4*

*MOVE.L #8,D5*

*AND.W D1,D2*

*AND.W D1,D3*

*AND.W D1,D4*

*AND.W D1,D5*

*LSR.W #1,D3*

*LSR.W #2,D4*

*LSR.W #3,D5*

*MOVE.B D5,D0*

*EOR.W D4,D5*

*EOR.W D3,D5*

*LSR.W #1,D1*

*LSL.W #1,D1*

*EOR.W D5,D1*

*LSL.W #1,D1*

*EOR.W D2,D1*

*EOR.W D0,D4*

*EOR.W D2,D4*

*EOR.W D0,D3*

*EOR.W D2,D3*

*EOR.W D4,D1*

*LSL.W #1,D1*

*EOR.W D3,D1*

*SIMHALT*

*END START*

*DECODER*

*ORG $1100*

*DECODER:*

*MOVE.L #0,D0*

*MOVE.L #0,D1*

*MOVE.L #0,D2*

*MOVE.L #0,D3*

*MOVE.L #0,D4*

*MOVE.L #0,D5*

*MOVE.L #0,D6*

*MOVE.L #0,D7*

*MOVE.B #$55, $002500*

*LEA.L $002500,A0*

*MOVE.B (A0),D1*

*MOVE.B D1,D2*

*ANDI.B #$80,D2*

*LSR.B #7,D2*

*MOVE.B D1,D3*

*ANDI.B #$40,D3*

*LSR.B #6,D3*

*MOVE.B D1,D4*

*ANDI.B #$20,D4*

*LSR.B #5,D4*

*MOVE.B D1,D5*

*ANDI.B #$10,D5*

*LSR.B #4,D5*

*MOVE.B D1,D6*

*ANDI.B #$08,D6*

*LSR.B #3,D6*

*MOVE.B D1,D7*

*ANDI.B #$04,D7*

*LSR.B #2,D7*

*MOVE.B D1,D0*

*ANDI.B #$02,D0*

*LSR.B #1,D0*

*MOVE.B D2,D1*

*EOR.B D3,D1*

*EOR.B D4,D1*

*EOR.B D6,D1*

*CMP.B D1,D5*

*BEQ R\_OK*

*MOVE.B #4,D5*

*R\_OK: TST.B D5*

*MOVE.B D2,D1*

*EOR.B D3,D1*

*EOR.B D6,D1*

*CMP.B D1,D7*

*BEQ S\_OK*

*ADDI.B #2,D5*

*S\_OK: TST.B D5*

*MOVE.B D2,D1*

*EOR.B D4,D1*

*EOR.B D6,D1*

*CMP.B D1,D0*

*BEQ T\_OK*

*ADDI.B #1,D5*

*T\_OK: MOVE.B D5,D0*

*SIMHALT*

*END DECODER*

**DESCRIPTION**

This lab involves the implementation of the Hamming algorithm using assembly language to detect and correct single-bit errors in a transmitted message. The encoded data consists of four information bits and three calculated parity bits. The decoder examines these bits to detect any single-bit errors and outputs the position of the error.

**OBSERVATIONS**

* The encoder correctly converted data like 00001011 into 01010101.
* The decoder was able to identify errors introduced manually (e.g., flipping bit 5 in the encoded message resulted in correct error position 101).
* The simulation worked as expected with both provided and custom test cases.

**RESULTS**

**Encoding**

**Decoding**

**CONCLUSIONS**

This lab successfully demonstrated the theoretical and practical application of Hamming (7,4) codes using assembly programming. Encoding logic ensures even-parity-based protection, and the decoding mechanism can reliably detect and localize single-bit errors. The implementation strengthens the understanding of error correction in digital communication systems.