CSCI 320-54 – Assignment 6: Hamming Encoding

Created by: Thomas Hoerger

Objectives

Convert a given message into an encoded message using Hamming encoding.

Equipment Used

EASy 68K simulator

Procedure

You are to encode the contents of a byte in memory where the information bits a, b, c and d are located as shown in the following:

xxxxabcd

For example, suppose that you have the following memory configuration, your task would be to compose the encoded bytes as shown in Table 4.1.

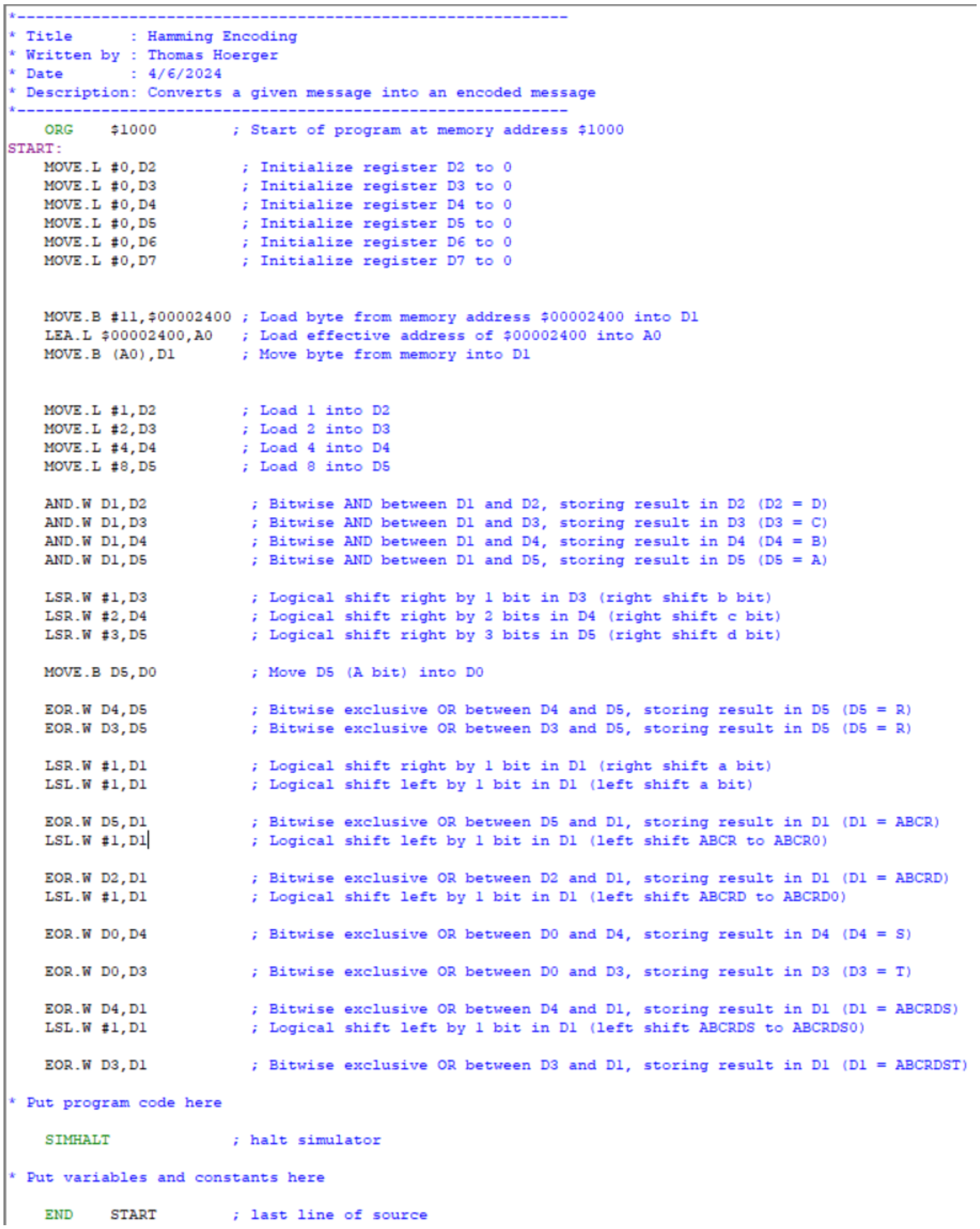
New Operations Learned

MOVE.L
MOVE.B
LEA.L
AND.W
LSR.W
EOR.W
LSL.W

Program Description

This program encodes the contents of a byte in memory where the information bits a, b, c and d are located into an encoded message (xxxxabcd) using Hamming encoding.

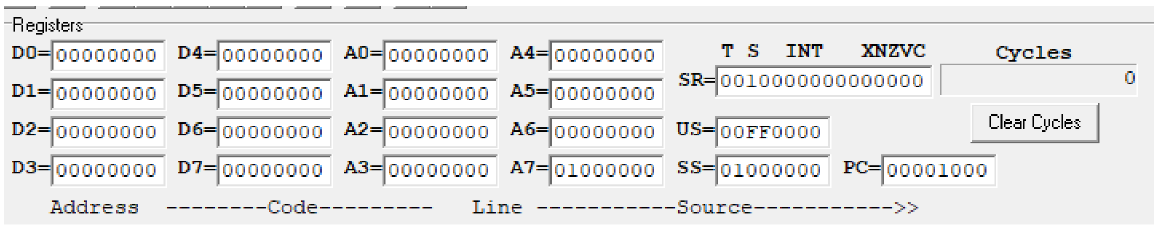
SOURCE CODE



```
*-----------------------------------------------------------
* Title      : Hamming Encoding
* Written by : Thomas Hoerger
* Date       : 4/6/2024
* Description: Converts a given message into an encoded message
*-----------------------------------------------------------
    ORG    $1000        ; Start of program at memory address $1000
START:
    MOVE.L #0,D2         ; Initialize register D2 to 0
    MOVE.L #0,D3         ; Initialize register D3 to 0
    MOVE.L #0,D4         ; Initialize register D4 to 0
    MOVE.L #0,D5         ; Initialize register D5 to 0
    MOVE.L #0,D6         ; Initialize register D6 to 0
    MOVE.L #0,D7         ; Initialize register D7 to 0


    MOVE.B #11,$00002400 ; Load byte from memory address $00002400 into D1
    LEA.L $00002400,A0   ; Load effective address of $00002400 into A0
    MOVE.B (A0),D1       ; Move byte from memory into D1


    MOVE.L #1,D2         ; Load 1 into D2
    MOVE.L #2,D3         ; Load 2 into D3
    MOVE.L #4,D4         ; Load 4 into D4
    MOVE.L #8,D5         ; Load 8 into D5

    AND.W D1,D2           ; Bitwise AND between D1 and D2, storing result in D2 (D2 = D)
    AND.W D1,D3           ; Bitwise AND between D1 and D3, storing result in D3 (D3 = C)
    AND.W D1,D4           ; Bitwise AND between D1 and D4, storing result in D4 (D4 = B)
    AND.W D1,D5           ; Bitwise AND between D1 and D5, storing result in D5 (D5 = A)

    LSR.W #1,D3           ; Logical shift right by 1 bit in D3 (right shift b bit)
    LSR.W #2,D4           ; Logical shift right by 2 bits in D4 (right shift c bit)
    LSR.W #3,D5           ; Logical shift right by 3 bits in D5 (right shift d bit)

    MOVE.B D5,D0          ; Move D5 (A bit) into D0

    EOR.W D4,D5           ; Bitwise exclusive OR between D4 and D5, storing result in D5 (D5 = R)
    EOR.W D3,D5           ; Bitwise exclusive OR between D3 and D5, storing result in D5 (D5 = R)

    LSR.W #1,D1           ; Logical shift right by 1 bit in D1 (right shift a bit)
    LSL.W #1,D1           ; Logical shift left by 1 bit in D1 (left shift a bit)

    EOR.W D5,D1           ; Bitwise exclusive OR between D5 and D1, storing result in D1 (D1 = ABCR)
    LSL.W #1,D1           ; Logical shift left by 1 bit in D1 (left shift ABCR to ABCR0)

    EOR.W D2,D1           ; Bitwise exclusive OR between D2 and D1, storing result in D1 (D1 = ABCRD)
    LSL.W #1,D1           ; Logical shift left by 1 bit in D1 (left shift ABCRD to ABCRD0)

    EOR.W D0,D4           ; Bitwise exclusive OR between D0 and D4, storing result in D4 (D4 = S)

    EOR.W D0,D3           ; Bitwise exclusive OR between D0 and D3, storing result in D3 (D3 = T)

    EOR.W D4,D1           ; Bitwise exclusive OR between D4 and D1, storing result in D1 (D1 = ABCRDS)
    LSL.W #1,D1           ; Logical shift left by 1 bit in D1 (left shift ABCRDS to ABCRDS0)

    EOR.W D3,D1           ; Bitwise exclusive OR between D3 and D1, storing result in D1 (D1 = ABCRDST)

* Put program code here

    SIMHALT             ; halt simulator

* Put variables and constants here

    END    START        ; last line of source
```

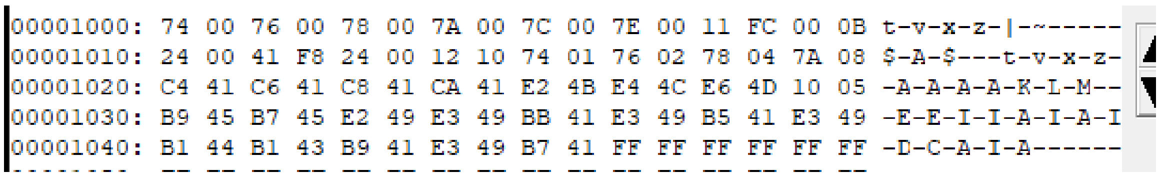
Figure 1 shows the code properly entered in the simulator.

Before Execution



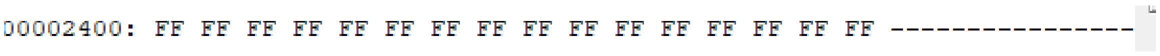
```
Registers
D0=00000000  D4=00000000  A0=00000000  A4=00000000     T S  INT   XNZVC        Cycles
D1=00000000  D5=00000000  A1=00000000  A5=00000000  SR=0010000000000000                0
D2=00000000  D6=00000000  A2=00000000  A6=00000000  US=00FF0000              Clear Cycles
D3=00000000  D7=00000000  A3=00000000  A7=01000000  SS=01000000  PC=00001000
   Address   --------Code---------    Line  -----------Source----------->>
```

Figure 2 Shows the registers before execution.



```
00001000: 74 00 76 00 78 00 7A 00 7C 00 7E 00 11 FC 00 0B t-v-x-z-|-~------
00001010: 24 00 41 F8 24 00 12 10 74 01 76 02 78 04 7A 08 $-A-$---t-v-x-z-
00001020: C4 41 C6 41 C8 41 CA 41 E2 4B E4 4C E6 4D 10 05 -A-A-A-A-K-L-M--
00001030: B9 45 B7 45 E2 49 E3 49 BB 41 E3 49 B5 41 E3 49 -E-E-I-I-A-I-A-I
00001040: B1 44 B1 43 B9 41 E3 49 B7 41 FF FF FF FF FF FF -D-C-A-I-A------
```

Figure 3 Shows memory starting at 00001000 before execution.

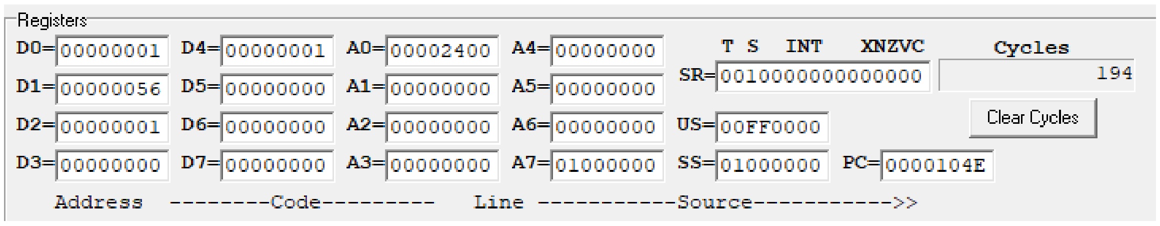


```
00002400: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
```

Figure 4 Shows memory at 00002400 before execution.

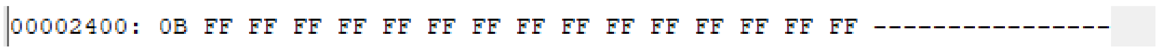
Results

After Execution of the program



```
Registers
D0=00000001  D4=00000001  A0=00002400  A4=00000000     T S  INT   XNZVC        Cycles
D1=00000056  D5=00000000  A1=00000000  A5=00000000  SR=0010000000000000              194
D2=00000001  D6=00000000  A2=00000000  A6=00000000  US=00FF0000              Clear Cycles
D3=00000000  D7=00000000  A3=00000000  A7=01000000  SS=01000000  PC=0000104E
   Address   --------Code---------    Line  -----------Source----------->>
```

Figure 5 Shows the registers after execution.



```
00002400: 0B FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ----------------
```

Figure 6 Shows memory at 00002400 after execution.