CE Lab Report 4

Encoding and Decoding Morse Code

Brandon Gomez

Lab Section: 2

11.14.14

PROCEDURE:

The purpose of this lab is to get familiar of using JSR and Stacks in LC-3 assembly. We should also get familiar of converting Morse hex to a string and string to Morse binary. Our program will be starting at x3000 and will be storing the input at x3201 through x3220 and the output at x3221 through x3240. Moreover, if the user encodes SOS into Morse, the program will produce the output: FOSOFO. If the user wants to decode a Morse message of B810B8, the output will be LOL.

ASCII	Morse	Morse	Binary	Hex
A	.-	10	10100000	A0
B	-...	0111	01111000	78
C	-.-.	0101	01011000	58
D	-..	011	01110000	70
E	.	1	11000000	C0
F	..-.	1101	11011000	D8
G	–.	001	00110000	30
H	....	1111	11111000	F8
I	..	11	11100000	E0
J	.—	1000	10001000	88
K	-.-	010	01010000	50
L	.-..	1011	10111000	B8
M	–	00	00100000	20
N	-.	01	01100000	60
O	—	000	00010000	10
P	.–.	1001	10011000	98
Q	–.-	0010	00101000	28
R	.-.	101	10110000	B0
S	...	111	11110000	F0
T	-	0	01000000	40
U	..-	110	11010000	D0
V	...-	1110	11101000	E8
W	.–	100	10010000	90
X	-..-	0110	01101000	68
Y	-.–	0100	01001000	48
Z	–..	0011	00111000	38
1	.—-	10000	10000100	84
2	..—	11000	11000100	C4
3	...–	11100	11100100	E4
4	....-	11110	11110100	F4
5	.....	11111	11111100	FC
6	-....	01111	01111100	7C
7	–...	00111	00111100	3C
8	—..	00011	00011100	1C
9	—-.	00001	00001100	0C
0	——	00000	00000100	04

MATERIALS:

Orgin / .ORIG xXXXX / 0110 xXXXX

Tells the program where to start

AND DR, SR1, SR2 / 0101 DR SR1 0 00 SR2

This AND ands SR1 and SR2 and stores the result into DR

AND DR, SR1, imm5 / 0101 DR SR1 1 imm5

This AND ands SR1 and the digit and stores the result into DR

ADD DR, SR1, SR2 / 0001 DR SR1 0 00 SR2

This ADD adds SR1 and SR2 and stores the result into DR

ADD DR, SR1, imm5 / 0001 DR SR1 1 imm5

This ADD adds SR1 and the digit and stores the result into DR

Store Direct / ST SR, LABEL / 0011 SR offset9

Gets whatever data is in SR and stores it into LABEL

Load Direct / LD DR, LABEL / 0010 DR PCoffset9

Loads the data inside of LABEL and puts it in DR

Load Base + Index / LDR DR, BaseR, offset / 0110 DR BaseR offset6

Loads whatever is in BaseR and puts it into DR

Load Effective Address / LEA DrR LABEL / 1110 DR offset9

Loads the address of LABEL and puts it into DR

Conditional Branch / BRx LABEL / 0000 nzp PCoffset9

If the condition is true then it will go to the address LABEL, else it will go to the next line

System Routine / TRAP x25 / 1111 0000 trapvect8

Halts the program

GETC / Gets user input and stores it in R0

PUTC / Gets R0 and prints it on to the monitor

STACKS / Stores data and you are able to push or pop it

JSR / Takes you to a subroutine where R7 contains the address of where you called the JSR so you can return.

PARTA: Choosing

```
; checks to see if the user wants to decode or encode,
; else redo and print error messege4.
        LD R1, DECODE
        ADD R2, R0, R1
        BRz DECODER
        LD R1, ENCODE
        ADD R2, R0, R1
        BRz ENCODER
        LEA R0, ILLEGAL
        PUTS
        BRnzp INPUT
```

My program right away asks for the input of the user and then outputs it (not in picture), it then checks to see if it is the letter E of D for Encode and Decode. If the user selects to E, it is expected that the user inputs items like SOS. If the user selects D, it is expected that the user inputs items like B810B8.

PARTB: Decoding

I first store my values inside a stack that is located inside x3201 and only 30 values can be stored inside the stack so I keep a counter to make sure it never gets to that value. Once the user presses ENTER, the program continues.

```
; Load variables
STOP2   LD R4, STRING
        LD R2, STORE
GOTO    LD R1, THIRTY
        LD R7, FORTY
        ADD R6, R6, #-1
        LDR R5, R4, 0
        ADD R5, R5, R7
; DETERMINES IF THE FIRST STRING IS IN A-F OR 0-9
        BRp POS
        BRnz OKAY
POS     LDR R5, R4, 0
        ADD R5, R5, R1
        ADD R5, R5, #-6
        ADD R4, R4, #1
        BRnzp SKIP7
OKAY    LDR R5, R4, 0
        ADD R5, R5, R1
        ADD R4, R4, #1
SKIP7   LDR R3, R4, 0
        ADD R3, R3, R7
```

After getting instructions and going into a JSR subroutine, my program stores variable inside some registers in order to check for some tests. The first string that the user inputs is stored inside R5, and we have to check to see if that value lies inside 0-9 or A-F so we can determine the offset. If it is inside 0-9 we only subtract x30 from the first hex value, else we subtract x41 from the hex value. This is done by first subtracting x3A from the hex and if it is positive it lies in A-F if it is negative, it lies inside 0-9. We used a temp in order to retain our real value.

```
; SUBTRACT BOTH HEXES AND ADD TO SINGLE BIT R1
; ADD 16 FROM FIRST HEX AND 1 FROM SECOND
; HEX 1 = R5, HEX 2 = R3
SKIP8   AND R1, R1, #0
        ADD R5, R5, #-1
        BRz ADDONE
        BRp FOR
ADDONE  ADD R1, R1, #8
        ADD R1, R1, #8
FOR     ADD R5, R5, #-1
        BRn SKIP3
        ADD R1, R1, #8
        ADD R1, R1, #8
        BRnzp FOR
SKIP3   ADD R3, R3, #-1
        BRn SKIP4
        ADD R1, R1, #1
        BRnzp SKIP3
SKIP4   STR R1, R2, #0
        ADD R2, R2, #1
        ADD R6, R6, #-2
        BRp GOTO
```

My program then takes the a pair of hexes and then decrements them and adds 16 if it is the first hex and adds 1 if it is the second hex to a single byte.

```
; STORE STRING
        LD R3, STORE
UP      LD R1, LL
        LDR R4, R3 #0
        ADD R2, R4, R1
        BRz SKIPLS
        LD R1, OO
        ADD R2, R4, R1
        BRz SKIPOS
        BRnzp END
```

Then, the program stores the new values into another stack inside x3221. Last, the program looks though a table to see which of the new single byte matches with the corresponding letter in order to print out the letter.

PARTC: Encoding

```
; ENCODER
; insert string into x3201 + n
; if the user presses enter, it stop asking to store information
; only thirty elements can be stored
ENCODER ADD R4, R4, #-15
        ADD R4, R4, #-15
        LEA R0, INSERT
        PUTS
        LD  R3, STRING
INP     GETC
        PUTC
        ADD R6, R6, #1
        ADD R5, R4, R6
        BRz STOP
        STR R0, R3, #0 ; STORE STRING
        ADD R3, R3, #1
        LD R1, ENTER
        ADD R2, R0, R1
        BRnp INP
```

The first part of the encoder is very similar to the decoder were the program stores the users input inside a stack in x3201. If the user inputs ENTER, the program will stop asking for more inputs. The user can only enter 30 items. Once the user is done, the program checks the results and compares the item to a table. Once one matches, the program stores the HEX values into another stack in x3221 and prints the hex values.

DECODER RESULT:

If the user inputs B810B8[ENTER], then the program will take the first value and see it is a letter so it will deduct x41 from B to give the value 12. (12-1) * 16 give a value of 176. Then the program takes the second value of 8 and it sees that it is a digit so it will only deduct x30 from the hex to give a value of 8. 176 + 8 = 184, which corresponds to L on the table.

Since there are still hex values to check, the program will go back up and get the next to values until there is no more in the stack.

ENCODER RESULTS:
If the user inputs SOS[ENTER], then the program will get the first value and go through a table in order to check to see which Hex value it corresponds to. Once it finds it, it will print out the Hex value of F0. Since there are still values inside of the stack, the program goes back up and gets the next value until the stack is empty.

CONCLUSION:
In conclusion, I found that the algorithms were much easier to solve, but took quite a while to implement them. If I were to type in "HELLO WORLD" into the program, it will output F8C0B8B8109010B0B870. If I were to type in "40101020A06048F0C058B0C040F0", the program will output TOOMANYSECRETS. Error checking that may be good to have is checking that the Morse code is in multiples of two. If it was not then it will report an error. The subroutine should save and restore the registers so that they can use the registers for arithmetic reasons. My conversion table is 72 bytes long. To convert a two ASCII hex to a binary byte, will take a certain amount of cycles. For example if the input was A8. It will take 18 cycles since 10 + 8 = 18. To convert a binary byte to a two ASCII hex character will take the a certain amount of cycles too. If it was 17, it will take two cycles. One cycle to determine how many 16s there are and one cycle to determine how many 1s there are.