ECEN 3320-002

Assembly Language Programming

Lab Assignment # 2.4

Read, Random, and Write

Cameron Biniamow

University of Nebraska-Lincoln

Department of Electrical and Computer Engineering

Peter Kiewit Institute

Due: 10/30/2020

Summary

Lab 2.4 Read/Random/Write, uses understandings learned in past labs in conjunction with new concepts to help produce a program that will be uploaded to the 8051 trainer. The goal of Lab 2.4 is to input a value into the 8051, randomize the value twice, and output the two random values. With the use of the equipment of an 8-switch DIP, 8051 trainer, LCD display, and Keil uVision, Lab 2.4 will be completed with expected results. The program will work by reading the value on the 8-switch DIP, which will be connected through PORT3, and randomize the value through a randomize procedure in the program. After the value has been randomized using an algorithm given in the lab handout, the now randomized value will be saved in a register but will also be randomized again. Since there are two random numbers that have been generated, the values will be output through PORT1, which is connected to an LCD display. The LCD display will be configured to display the two randomized values with the first value on the first line and the second value on the second line. Following the completion of Lab 2.4, a solid understanding of the 8051 architecture and assembly code format will be gained.

Background

Since Lab 2.4 is the final lab in a four-part series revolving around programming the 8051 microprocessor, many parts of this lab use previously created code to operate. This includes the use of the LCD driver created in Lab 2.1, which is needed in order to initialize and control the LCD connected to the 8051. Additionally, portions of a BCD conversion program are used in order to unpack packed BCD values inputted through DIP switches and convert the values to binary for display on the LCD.

Procedure

Lab 2.4 begins by referring back to code that was previously written in Lab 2.1, which was built to introduce an LCD display that is connected to the 8051 trainer. The code in Lab 2.1 includes procedures that initialize the LCD display as well as write the necessary data to the LCD within the required specifications of the LCD. The code implemented in Lab 2.1 will be copied exactly as it was written and pasted into the Read, Random, Write program. By utilizing the previously written code, a great amount of time is saved and far less code is subject to being filled with errors. In addition to the code from Lab 2.1 that initializes the LCD, three procedures will need to be implemented in order for Lab 2.4 to function properly. The program will implement the procedures READ, RANDOM, and WRITE to achieve the desired results. Prior to implementing these three procedures, the program will need to setup the I/O ports which will be used to transfer data from the 8-switch DIP as well as to the LCD display. This means that in addition to the already written code from Lab 2.1, which utilized PORT1 and PORT2 to connect to the LCD display, PORT3 will need to be setup as an input port to read the states of the DIP switches. Now that the configuration of the ports is complete, the three procedures can be written.

Starting with the READ procedure, only two instructions will be included. The READ procedure will clear the accumulator and input the data from PORT3, which is the value from the DIP switches. At this point, the actual value set on the DIP switches is held in the accumulator and is ready to be randomized through the RANDOM procedure. The RANDOM procedure can now be written and implemented.

The RANDOM procedure needs to be built to follow four instructions which will cause the input value to be output as “random”. To begin the RANDOM procedure, the first instruction will need to be completed which is shifting the data left by 1-bit. This will be done by the accumulator being logically left shifted using the RLC instruction. The second instruction of the random algorithm is to exclusive-OR bits 6 and 7 held in the accumulator, replace bit 0 with the result, and clear bit 7. To do so, the accumulator value will next be moved to R1 and R2 then the accumulator is logically shifted left using the RLC instruction. Using the XRL instruction, an exclusive OR of accumulator and R2 is performed and placed in the accumulator. Using the ANL instruction, the accumulator will have bit 7 masked by using the value #80H. At this point R1 holds the input value from the DIP switches and is left-shifted 1-bit and the accumulator holds the exclusive-OR of bits 6 and 7 of the input value in bit 7. Since the exclusive-OR of bits 6 and 7 need to replace bit 0, the upper and lower nibbles of the accumulator are swapped and the RR instruction is used three times, which moves the value from bit 7 to bit 0. The accumulator value will now be moved into R2 and the original DIP switch value left shifted once, held in R1, is moved into the accumulator. Using the ANL instruction on the accumulator with value #7EH clears bits 7 and 0 and a logical OR with the accumulator and R2 replaces bit 0 of the accumulator with the exclusive OR of bits 7 and 6.

The WRITE procedure will be used to print the values that were randomized onto the LCD with the first random value on the first line and the second random value on the second line. Before writing the WRITE procedure, 18 strings will need to be defined at the bottom of the program. These strings will be named FIRST, SECOND, ZERO, ONE, TWO,…, FIFETEEN . As expected, FIRST will hold the string “First: ”, SECOND will hold the string “Second: ”, ZERO will hold the string “0000”, ONE will hold the string “0001” and so on and so forth. This is done to write the values on the LCD display. For the WRITE procedure, the now unpacked BCD value is stored in the accumulator and is compared to values #00H - #0FH using the CJNE instruction. This works by comparing the accumulator to #00H initially and if it is equal, loads the ROM pointer to point at the ZERO string. Given the accumulator does not equal #00H, the procedure jumps to the next compare, which compares the accumulator to the value #01H. Again, if the two are equal, the ROM pointer will point at the ONE string. Otherwise, the next compare will occur.

Since the procedures have been defined, the main portion of code will be explained. The program will operate by first initializing the LCD through the LCD_INIT procedure. The READ procedure is called and reads the value of the DIP switch followed by the RANDOM procedure being called, which creates the first random number. The first random number is stored in R6 and the RANDOM procedure is called again to create the second random value. The second random value is stored in register B. The ROM pointer then points to the FIRST string and is written to the LCD through the LCD_STRING1 procedure. The first random number is returned to the accumulator from R6 and the CONVERT procedure is called which converts the value to binary and displays it on the LCD. Next, the cursor on the LCD is moved to the second line through the LCD_2NDLINE procedure and the ROM pointer points to the SECOND string. The SECOND string is written to the LCD through calling the LCD_STRING1 procedure. The second random number is moved back to the accumulator from register B and a copy is stored in R6. Finally, the CONVERT procedure is called which converts the BCD value to binary and displays it on the LCD followed by ending the program.

Results

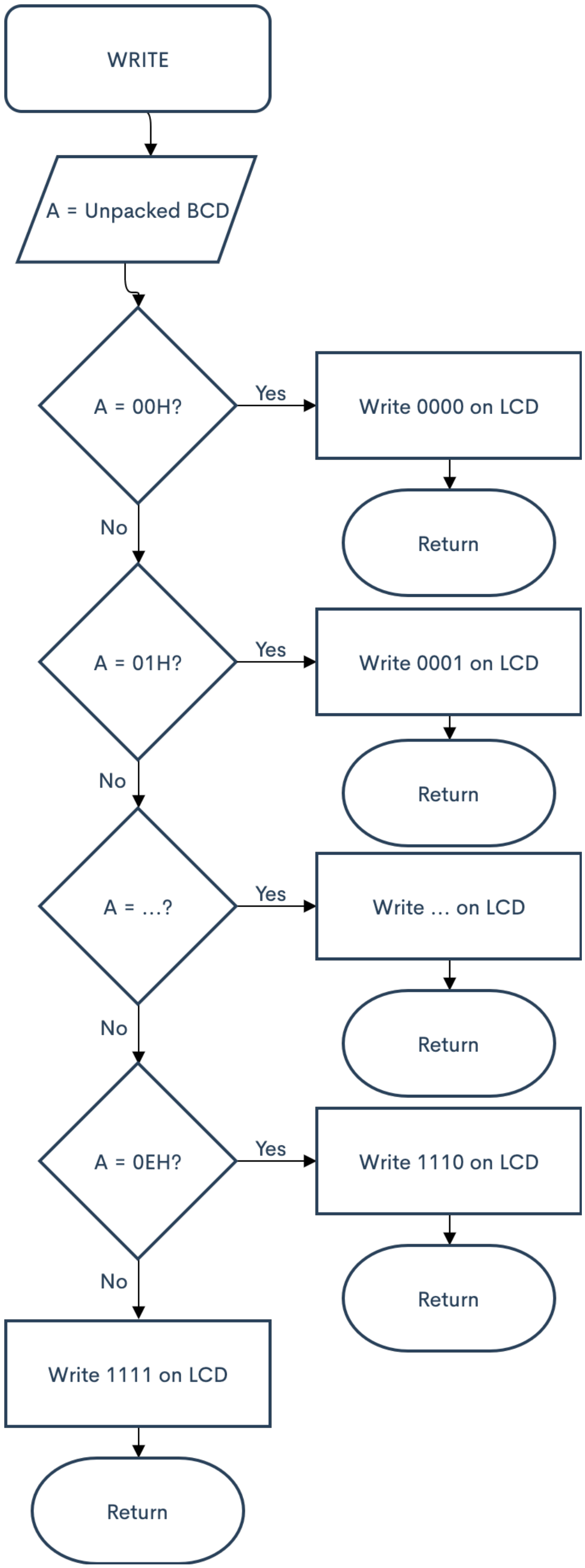
Testing for Lab 2.4 was completed through uploading the assembly code to an 8051 trainer and inputting various values while observing the “randomized” output values on a connected LCD. It was determined that throughout testing that the program did initially operate as expected in some respects. The LCD was displaying the “First: “ and “Second: “ strings properly as well as the lower nibbles of the random numbers, however the upper nibbles of the random numbers were not displaying. This was found to be caused by an oversight in which registers were used to hold the random values thus causing only half of the values to be displayed. Once the program was debugged and problem was fixed, the updated program was uploaded to the 8051 trainer and tested for accuracy. It was determined that the program operated as expected and without error as all tested input values from the DIP switches did output the proper random values on the LCD. The fully functioning Read, Random, Write program can be seen uploaded onto the 8051 trainer operating properly in Figure II.

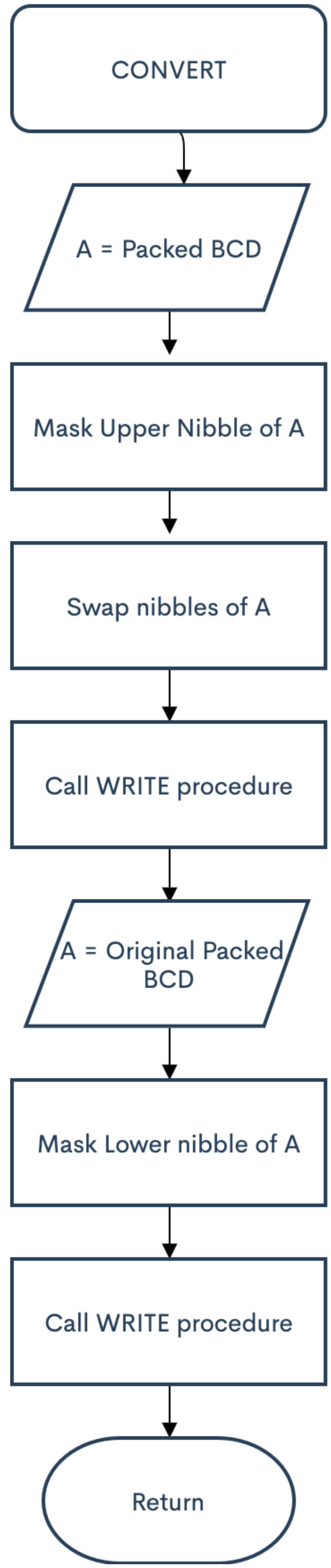
Answers to Posted Questions

1. The engineering methods used to write the procedures needed for Lab 2.4 included the following:
    - Referencing the ECEN 1060 lab that required the creation of a program that produces two random values and displays them on an LCD.
    - Referencing code created in Labs 2.1 – 2.3

    The ECEN 1060 lab proved to be the most beneficial as the same concepts were covered. The only difference between the labs being the randomize algorithm and the conversion of code from AVR to 8051 assembly code. Methods used to improve code logic consisted of better use of loops and fewer moves of data between registers. Additionally, adding more in depth comments throughout the code.
2. Flowcharts for the READ, RANDOM, WRITE, and CONVERT procedures can be seen below.


Read
Clear Accumulator
Read PORT3 & move value to Accumulator
Return
Random
Shift Accumulator left by 1-bit
Copy value in accumulator into R1 & R2
Shift Accumulator left by 1-bit
A = A ^ R2
Mask A.7
X000 0000
Swap nibbles in A
0000 X000
Shift A right 3 times
0000 000X
Move A to R2
Move R1 to A
Mask A.1 - A.6
0XXX XXX0
A = A | R2
0XXX XXX0 | 0000 000X
Return


WRITE
A = Unpacked BCD
A = 00H?
Yes
Write 0000 on LCD
Return
No
A = 01H?
Yes
Write 0001 on LCD
Return
No
A = ...?
Yes
Write ... on LCD
Return
No
A = 0EH?
Yes
Write 1110 on LCD
Return
No
Write 1111 on LCD
Return


CONVERT
A = Packed BCD
Mask Upper Nibble of A
Swap nibbles of A
Call WRITE procedure
A = Original Packed BCD
Mask Lower nibble of A
Call WRITE procedure
Return

Conclusion

Lab 2.4, the final lab of a four-part series, tied together all the concepts introduced in project 2. This lab used the LCD driver created in Lab 2.1 and BCD conversion created in Lab 2.2. Following the completion of Lab 2.4, a full understanding of the 8051 architecture and assembly language has been gained. It was determined that through compiling and uploading the assembly code to an 8051 trainer, the program worked and produced results consistent with what was expected. The information within this lab will prove to be beneficial in future labs within ECEN 3320 as well as future courses.

Appendix

Figure I: 8051 Read, Random, Write Assembly Source Code

```
; CAMERON BINIAMOW
; ECEN 3320
; LAB 2.4: 8051 ASSEMBLY LANGUAGE PROGRAMMING
; DUE: 10/30/2020

      ORG         0H
      ACALL       LCD_INIT
      ACALL       READ                ; READ INPUT VALUE
      ACALL       RANDOM              ; CREATE RANDOM NUMBER
      MOV         R6, A               ; HOLD FIRST RANDOM NUMBER IN R6
      ACALL       RANDOM              ; CREATE SECOND RANDOM NUMBER
      MOV         B, A                ; REG B = SECOND RANDOM NUMBER
      MOV         DPTR, #FIRST        ; LOAD ROM POINTER
      ACALL       LCD_STRING1         ; WRITE "FIRST: "
      MOV         A, R6               ; REG A = FIRST RANDOM NUMBER
      ACALL       CONVERT
      ACALL       LCD_2NDLINE         ; MOVE CURSOR THE 2nd LINE OF LCD
      MOV         DPTR, #SECOND       ; LOAD ROM POINTER
      ACALL       LCD_STRING1         ; WRITE "SECOND: "
      MOV         A, B                ; LOAD 2nd RAND NUM INTO REG A
      MOV         R6, A               ; HOLD COPY OF SECOND RANDOM NUMBER
      ACALL       CONVERT

HERE:
      SJMP        HERE                ; STAY HERE

;---------------------------------------------------------------------
; UNPACKS BCD

CONVERT:
      MOV         R1, #0F0H
      ANL         A, R1               ; MASK UPPER NIBBLE
      SWAP        A                   ; PLACE UPPER NIBBLE INTO LOWER NIBBLE
      ACALL       WRITE               ; WRITE BIN. FOR UPPER NIBBLE OF RAND NUM
      MOV         A, R6               ; ORIGINAL UNMASKED VALUE
      MOV         R1, #0FH
      ANL         A, R1               ; MASK LOWER NIBBLE
      ACALL       WRITE               ; WRITE BIN. FOR LOWER NIBBLE OF RAND NUM
      RET

;---------------------------------------------------------------------
; READS PACKED BCD VALUE FROM PORT3

READ:
      CLR         A
      MOV         A, P3               ; STORE VALUE READ FROM PORT3
      RET

;---------------------------------------------------------------------
; CREATES RANDOM VALUE

RANDOM:
      RLC         A                   ; SHIFT LEFT ONCE
      MOV         R1, A               ; R1 = READ VALUE SHIFTED LEFT ONCE
      MOV         R2, A               ; R2 = READ VALUE SHIFTED LEFT ONCE
      RLC         A                   ; A = READ VALUE SHIFTED LEFT TWICE
      XRL         A, R2
      ANL         A, #80H             ; A.7 = XOR OF R1.6 & R1.7 (X000 0000)
      SWAP        A                   ; A = 0000 X000
      RR          A                   ; A = 0000 0X00
      RR          A                   ; A = 0000 00X0
```

```
        RR          A                   ; A = 0000 000X
        MOV         R2, A
        MOV         A, R1
        ANL         A, #7EH             ; A = 0XXX XXX0
        ORL         A, R2               ; A = 0XXX XXXX (RANDOM NUMBER – FINAL)
        RET

;----------------------------------------------------------------------
; WRITES A BYTE OF BINARY VALUES

WRITE:
        CJNE        A, #00H, L1         ; LOWER NIBBLE OF A = 0? NO – CHECK NEXT
        MOV         DPTR, #ZERO         ; YES – WRITE 0000
        ACALL       LCD_STRING
        RET
        L1:
        CJNE        A, #01H, L2         ; LOWER NIBBLE OF A = 1? NO – CHECK NEXT
        MOV         DPTR, #ONE          ; YES – WRITE 0001
        ACALL       LCD_STRING
        RET
        L2:
        CJNE        A, #02H, L3         ; LOWER NIBBLE OF A = 2? NO – CHECK NEXT
        MOV         DPTR, #TWO          ; YES – WRITE 0010
        ACALL       LCD_STRING
        RET
        L3:
        CJNE        A, #03H, L4         ; LOWER NIBBLE OF A = 3? NO – CHECK NEXT
        MOV         DPTR, #THREE        ; YES – WRITE 0011
        ACALL       LCD_STRING
        RET
        L4:
        CJNE        A, #04H, L5         ; LOWER NIBBLE OF A = 4? NO – CHECK NEXT
        MOV         DPTR, #FOUR         ; YES – WRITE 0100
        ACALL       LCD_STRING
        RET
        L5:
        CJNE        A, #05H, L6         ; LOWER NIBBLE OF A = 5? NO – CHECK NEXT
        MOV         DPTR, #FIVE         ; YES – WRITE 0101
        ACALL       LCD_STRING
        RET
        L6:
        CJNE        A, #06H, L7         ; LOWER NIBBLE OF A = 6? NO – CHECK NEXT
        MOV         DPTR, #SIX          ; YES – WRITE 0110
        ACALL       LCD_STRING
        RET
        L7:
        CJNE        A, #07H, L8         ; LOWER NIBBLE OF A = 7? NO – CHECK NEXT
        MOV         DPTR, #SEVEN        ; YES – WRITE 0111
        ACALL       LCD_STRING
        RET
        L8:
        CJNE        A, #08H, L9         ; LOWER NIBBLE OF A = 8? NO – CHECK NEXT
        MOV         DPTR, #EIGHT        ; YES – WRITE 1000
        ACALL       LCD_STRING
        RET
        L9:
        CJNE        A, #09H, L10        ; LOWER NIBBLE OF A = 9? NO – CHECK NEXT
        MOV         DPTR, #NINE         ; YES – WRITE 1001
        ACALL       LCD_STRING
        RET
        L10:
        CJNE        A, #0AH, L11        ; LOWER NIBBLE OF A = 10? NO – CHECK NEXT
        MOV         DPTR, #TEN          ; YES – WRITE 1010
        ACALL       LCD_STRING
        RET
```

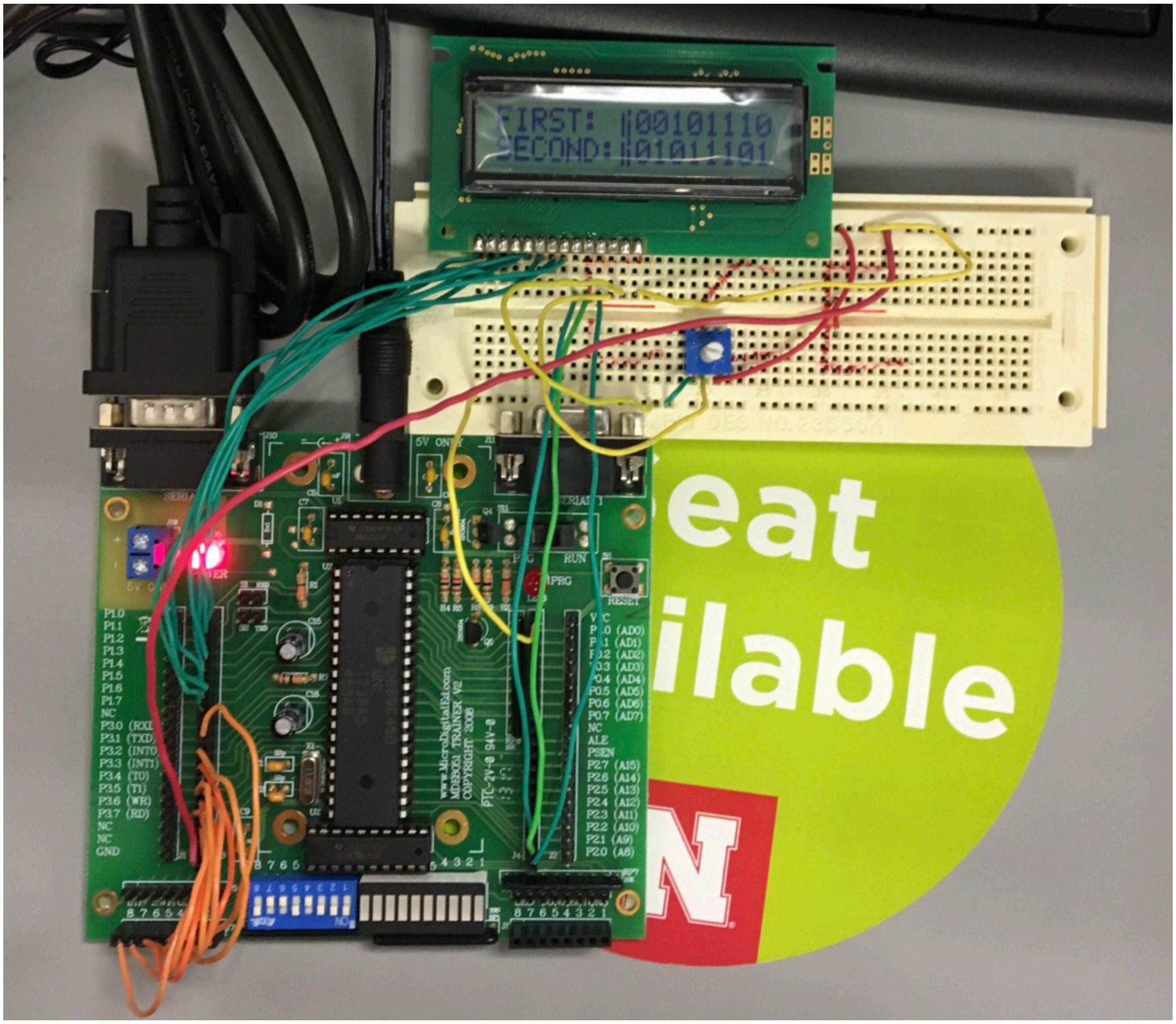
```
    L11:
    CJNE        A, #0BH, L12        ; LOWER NIBBLE OF A = 11? NO – CHECK NEXT
    MOV         DPTR, #ELEVEN       ; YES – WRITE 1011
    ACALL       LCD_STRING
    RET
    L12:
    CJNE        A, #0CH, L13        ; LOWER NIBBLE OF A = 12? NO – CHECK NEXT
    MOV         DPTR, #TWELVE       ; YES – WRITE 1100
    ACALL       LCD_STRING
    RET
    L13:
    CJNE        A, #0DH, L14        ; LOWER NIBBLE OF A = 13? NO – CHECK NEXT
    MOV         DPTR, #THIRTEEN     ; YES – WRITE 1101
    ACALL       LCD_STRING
    RET
    L14:
    CJNE        A, #0EH, L15        ; LOWER NIBBLE OF A = 14? NO – CHECK NEXT
    MOV         DPTR, #FOURTEEN     ; YES – WRITE 1110
    ACALL       LCD_STRING
    RET
    L15:
    MOV         DPTR, #FIFETEEN     ; WRITE 1111
    ACALL       LCD_STRING
    RET

;---------------------(n)ms DELAY PROCEDURE--------------------------
;CALLS DELAY PROCEDURE n # OF TIMES. n = VALUE LOADED INTO REG A

DELAY_ms:
    MOV         R5, A               ;REGISTER A HOLDS # OF ms DELAY NEEDED
    JUMP:
    ACALL       DELAY
    DJNZ        R5, JUMP            ;CONITUALLY DECREMENT R5 UNTIL 0
    RET

;---------------------1ms DELAY PROCEDURE----------------------------
;DECREMENTS R4 FROM 255 TO 0 THEN DECREMENTS R3.
;CONTINUES LOOPS UNTIL R3 IS 0. CREATES 12750 CLOCKS ~1ms.

DELAY:
    MOV         R3, #50             ;LOAD 50 DEC INTO R3
    HERE0:
    MOV         R4, #255            ;LOAD 255 DEC INTO R4
    HERE2:
    DJNZ        R4, HERE2           ;CONTINUALLY DECREMENT R4 UNTIL 0
    DJNZ        R3, HERE0           ;DECREMENTS R3 UNTIL 0
    RET

;---------------------LCD INITIALIZE PROCEDURE-----------------------
;INITIALIZES THE LCD IN ACCORDANCE TO THE LCD DATA SHEET

LCD_INIT:
    MOV         A, #15              ;15ms DELAY
    ACALL       DELAY_ms

    MOV         A, #000H
    MOV         P1, A               ;SET PORT 1 AS OUTPUT
    MOV         P3, A               ;SET PORT 3 AS OUTPUT

    MOV         A, #30H             ;FUNCTION SET COMMAND
    ACALL       LCD_CMD

    MOV         A, #5               ;5ms DELAY
    ACALL       DELAY_ms
```

```
        MOV         A, #30H             ;FUNCTION SET COMMAND
        ACALL       LCD_CMD

        ACALL       DELAY               ;1ms DELAY

        MOV         A, #30H             ;FUNCTION SET COMMAND
        ACALL       LCD_CMD

        MOV         A, #3CH             ;FUNCTION SET COMMAND
        ACALL       LCD_CMD             ;8-BIT INTERFACE

        MOV         A, #08H             ;DISPLAY OFF
        ACALL       LCD_CMD

        MOV         A, #06H             ;ENTRY MODE SET
        ACALL       LCD_CMD

        MOV         A, #0FH             ;DISPLAY ON
        ACALL       LCD_CMD

        ACALL       LCD_CLEAR           ;CLEAR DISPLAY

        RET

;---------------------LCD DATA PROCEDURE------------------------------
;TRANSFERS DATA TO THE LCD

LCD_DATA:
        SETB        P2.0                ;RS = 1
        CLR         P2.1                ;RW = 0
        ACALL       DELAY               ;1ms DELAY
        SETB        P2.2                ;EN = 1
        ACALL       DELAY               ;1ms DELAY
        MOV         P1, A               ;REGISTER A -> PORT 1
        ACALL       DELAY               ;1ms DELAY
        CLR         P2.2                ;EN = 0
        RET

;---------------------LCD COMMAND PROCEDURE---------------------------
;USED TO EXECUTE ALL COMMANDS ON THE LCD

LCD_CMD:
        CLR         P2.0                ;RS = 0
        CLR         P2.1                ;RW = 0
        ACALL       DELAY               ;1ms DELAY
        SETB        P2.2                ;EN = 1
        ACALL       DELAY               ;1ms DELAY
        MOV         P1, A               ;REGISTER A -> PORT 1
        ACALL       DELAY               ;1ms DELAY
        CLR         P2.2                ;EN = 0
        RET

;---------------------LCD CLEAR PROCEDURE-----------------------------
;CLEARS THE LCD

LCD_CLEAR:
        MOV         A, #01H
        ACALL       LCD_CMD
        RET

;---------------------LCD CHAR PROCEDURE------------------------------
;WRITE A SINGLE CHARACTER ON THE LCD

LCD_CHAR:
        ACALL       LCD_DATA
```

```
        RET

;---------------------LCD 2ND LINE PROCEDURE-------------------------
;MOVES CURSORS TO THE SECOND LINE OF THE LCD

LCD_2NDLINE:
        MOV             A, #0C0H
        ACALL           LCD_CMD
        RET

;---------------------LCD STRING PROCEDURE---------------------------
;WRITES A STRING ON THE LCD

LCD_STRING1:
        L3S:
        CLR             A
        MOVC            A, @A+DPTR
        ACALL           LCD_DATA
        ACALL           DELAY
        INC             DPTR
        JZ              L4S
        SJMP            L3S
        L4S:
        RET

LCD_STRING:
        MOV             R7, #4
        L3B:
        CLR             A
        MOVC            A, @A+DPTR
        ACALL           LCD_DATA
        ACALL           DELAY
        INC             DPTR
        DJNZ            R7, L3B
        RET
;---------------------STRINGS----------------------------------------

        ORG             300H
FIRST:
        DB              "FIRST: ", 0
SECOND:
        DB              "SECOND:", 0
ZERO:
        DB              "0000"
ONE:
        DB              "0001"
TWO:
        DB              "0010"
THREE:
        DB              "0011"
FOUR:
        DB              "0100"
FIVE:
        DB              "0101"
SIX:
        DB              "0110"
SEVEN:
        DB              "0111"
EIGHT:
        DB              "1000"
NINE:
        DB              "1001"
TEN:
        DB              "1010"
ELEVEN:
```

```
        DB              "1011"
TWELVE:
        DB              "1100"
THIRTEEN:
        DB              "1101"
FOURTEEN:
        DB              "1110"
FIFETEEN:
        DB              "1111"

        END
```

Figure II: Demonstration of the Read, Random, Write Program on an 8051 Trainer



CAMERON BINIAMOW

"LAB 2.4" 10-12-20

READ RANDOM WRITE

READ:
- DIP SWITCHES – PORT 3
- STORE INPUT IN REG A

RANDOM:
- SHIFT A LEFT ONCE
- REPLACE BØ WITH
  EOR B6 & B7
- CLEAR B7

WRITE: LCD – PORT1 (DATA)
PORT2 (CONTROL)

- WRITE 1ST RANDOM # ON 1ST LINE
  "FIRST: XXXXXXXX"
- WRITE 2ND RANDOM # ON 2ND LINE
  "SECOND: XXXXXXXX"

CAMERON BINIAMOW

"LAB 2.4" 10/12/20

READ PORT 3 — 1011 0011

↓

MOVE INPUT TO REG A

↓

SHIFT A LEFT ONCE — A: 0110 0110

↓

MOV R1, A — R1: 0110 0110

↓

MOV R2, A — R2: 0110 0110

↓

SHIFT R2 LEFT ONCE — R2: 1100 1100

↓

EOR R1 & R2 — R1: 1010 1010

↓

AND ~~OR~~ R1 & 0x~~7F~~ 80 ~~MASK 87 OF R1~~ — ~~MASK LOWER 7 BITS OF R1~~ — ~~R1: 1000 0000~~ ~~R1: 1111 1111~~ R1: 1000 0000

↓

AND ~~OR~~ A & 7F — A: 0110 0110

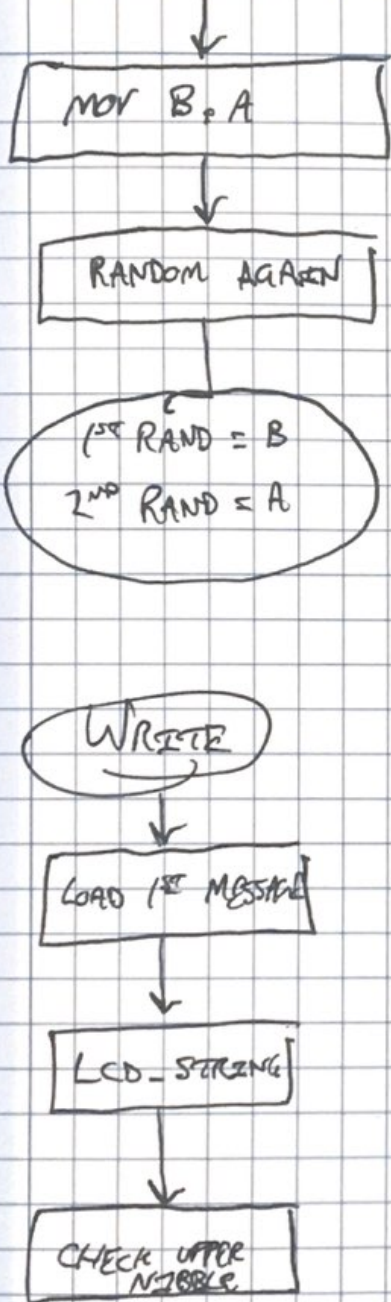
↓

OR A & R1 — A: 1110 0110

↓ CONTINUE

ECEN 3320 Cameron Biniamon

"Lab 2.4"



CAMERON BINIAMOW

EN 3320

"LAB 2.4"

0000 0000
0000 0000
0000 0000

0000 0001
0000 0010
0000 0010

0000 0100

0000 1001
- 0001 0010

0010 0100

1111 1111
1111 1110
0111 1110

1111 1100
0111 1100

1001 0110
- 0010 1100

0101 1000
- 0101 1001

TOTAL TIME: 6.5 HOURS

Individual report:

1. Describe the engineering learning plan you used to write the three new procedures and integrate the LCD procedures from previous labs. How was the plan developed and implemented? What methods did you use to improve your code logic? What ideas have you generated to improve your coding skills?

2. Include the flowcharts of your three new procedures on the yellow sheet copies of your lab notebook pages.

Include this verification with your report.

`READ`, `RANDOM`, and `WRITE` verified:

TA initials: MJB   Date: 10/23/20