ECEN 3320-002
Assembly Language Programming
Lab Assignment # 3.2

Cameron Biniamow
University of Nebraska-Lincoln
Department of Electrical and Computer Engineering
Peter Kiewit Institute
Due: 11/13/2020

Summary

Lab 3.2, *I/O Ports, LCDs, and Read/Random/Write*, includes two activities that work with the STM32F103 Blue Pill. The first activity will introduce writing an ARM assembly program that uses previously determined logic in order to create an LCD driver program. This LCD driver will be able to initialize an LCD, send commands, send data, write characters, write strings, move the cursor to the second line, and clear the LCD. As for the second activity, the LCD driver program will be expanded to use the LCD for display of two random numbers that are determined through reading 8 DIP switches and sending the value read from the switches through a predetermined algorithm. Again, the second activity has already been designed and implemented in previous labs in AVR and 8051 assembly language. Both activities in this lab will be implemented using Keil µVision5.

Background

Lab 3.2 will take advantage of PORTA and PORTB on the STM32F103C8 for data transfer required to use an LCD and DIP switches. PORTA of the STM32F103C8 will connect to pins D0 – D7 of the LCD using pins A0 – A7. PORTB of the STM32F103C8 will connect to the 8 DIP switches using pins B0 – B7. On the LCD, pins RS, RW, and EN are controlled through pins C13, C14, and C15 of the STM32F103C8.

The purpose of Lab 3.2 is to create an LCD driver, read the state of DIP switches and perform arithmetic. Lab 3.2 consists of two activities where the first activity builds an LCD driver program and the second activity uses the LCD driver to read data, convert it, and display the data on the LCD.

Activity 1 requires the implementation of the following nine procedures in order to control the LCD:

- LCD_INIT – Initializes the LCD
- LCD_CMD – Sends a byte to the LCD command register
- LCD_DATA – Sends a byte to the LCD data register
- LCD_CHAR – Writes a character to the LCD
- LCD_CLEAR – Clears the LCD
- LCD_2NDLINE – Moves the LCD cursor to the second line
- LCD_STRING – Writes a string to the LCD
- DELAY_1ms – Performs a 1ms delay

- DELAY_ms – Performs a *n* ms delay

Following the implementation of the nine procedures, a program is created that will write the users first name on the first line of the LCD and the course on the second line.

Activity 2 builds off the LCD driver created in Activity 1 in order to create a Read, Random, Write program. This program will read the states of eight connected DIP switches and stores the data in R0. After reading the data, the program will "randomize" the value through the following algorithm:

1. Shift left by 1 bit
2. Replace bit 0 with the exclusive OR of bits 6 and 7
3. Clear bit 7
4. Leave the result in R0

The program will create the first random number from the data read off the DIP switches and store the value temporarily. Using the first random number, the program will create a second random number by using the same algorithm used to obtain the first random number. Following the creation of both random numbers, the program will write the first random number in binary on the first line of the LCD and the second random number in binary on the second line of the LCD. The LCD will display the random numbers similar to the example below.

1ST: XXXXXXXX

2ND: XXXXXXXX

Procedure

For Activity 1, an LCD driver is designed and implemented in ARM assembly code using Keil µVision5. The LCD driver begins by copying provided code in the lab handout that assigns the clock and port addresses of the STM32F103 as well as the control pins for the LCD. Additionally, code provided in Lab 3.1 for the vector area and startup area are copied into the LCD driver program. As for the main portion of code, the following line of code is written to begin:

```
AREA MAIN, CODE, READONLY
THUMB
```

This allows use of the Thumb instruction set and defines the main portion of code in the program. The main code begins by enabling the clocks on each port by loading the clock enable address into R1. The data at the clock enable address is then loaded into R0 and a logical OR is performed with 0xFC and R0 with the result stored in R0. This allows for only the needed bits to be affected. The value stored in R0 is then stored at the clock enable address and the respective clocks are enabled.

Port setup is executed next as PORTA is set as inputs, PORTB is set as outputs, and PORTC pins 13, 14, and 15 are set as outputs. Starting with PORTA, R1 is loaded with the PORTA CRH register (GPIOA_CRH), which writes a 1 to the corresponding bits and sets the I/O line to an input. Register R0 is loaded with the value 0x44444444 and is then stored at the GPIOA_CRH address. The PORTA CRL register (GPIOA_CRL) address, which sets the direction for I/O lines, is loaded into R1 and the value 0x33333333 is loaded into R0. R0 is then stored at the GPIOA_CRL address. PORTA is now set as inputs for bits 0 through 7.

For PORTB, the GPIOB_CRL address is loaded into R1 and the value 0x88888888 is loaded into R0. R0 is then stored at the GPIOB_CRL address. R1 is then loaded with the GPIOB_ODR address, which allows writing to an I/O line configured as an output, and R0 is loaded with the value 0x0000. The value in R0 is then stored in at the GPIOB_ODR address to configure PORTB as an output port.

The final port and bits needing configured is PORTC bits 13, 14, and 15. These will be set as outputs and be used for control of the RS, RW, and EN pins of the LCD. The GPIOC_CRH address is loaded into R1 and the value 0x33344444 is loaded into R0. R0 is then stored at the GPIOC_CRH address and PORTC pins 13, 14, and 15 are set as outputs.

Now that the clocks and ports are configured, the following LCD procedures are designed and implemented:

- LCD_INIT
- LCD_CMD
- LCD_DATA
- LCD_CHAR
- LCD_CLEAR
- LCD_2NDLINE
- LCD_STRING
- DELAY_1ms
- DELAY_ms

LCD_INIT begins by pushing the link register value onto the stack as other procedures are called within the LCD_INIT procedure and upon finishing the LCD_INIT procedure, the program would not return to the correct instruction with pushing the link register onto the stack. The following instructions follow the information stated in the LCD datasheet for proper initialization. The value 15 (dec) is loaded into R0 and a branch with link to the DELAY_ms procedure occurs. Since within the DELAY_ms procedure R0 is used as the register that holds the number of ms delay needed, following execution, 15ms will have passed. Next, the function set command is sent to the LCD by loading 0x30 into R0 and branching with a link to the LCD_CMD procedure. A 5ms delay is then executed by loading 5 (dec) into R0 and branching with a link to the DELAY_ms procedure. Once again, the function command set is sent to the LCD followed by a 1ms delay, which is performed by a branch with link to the DELAY_1ms procedure. For a third time, the function set command is sent to the LCD followed by the 8-bit interface command (0x3C). The display is turned off by writing 0x08 to the LCD through the LCD_CMD procedure then entry mode is set by writing 0x06 through the LCD_CMD procedure. The LCD is cleared through the LCD_CLEAR procedure and the link register is popped off the stack to return to correct instruction. Finally, the procedure returns to the instruction after where the procedure was originally called.

LCD_CMD begins by pushing the link register onto the stack to ensure proper operation after the procedure is executed. Using the GPIOC_BSRR register, the address is loaded into R3, RS and RW are set to 0 then loaded into R4, and the value in R4 is stored at the GPIOC_BSRR address. A 1ms delay occurs by using the DELAY_1ms procedure and the EN pin is set HIGH by loading the GPIOC_BSRR address into R3, the EN address into R4, and storing R4 at the GPIOC_BSRR address. A 1ms delay occurs by using the DELAY_1ms procedure followed by outputting the command held in R0 through PORTA by loading the GPIOA_ODR address into R5 and storing the command in R0 at the GPIOA_ODR address. A 1ms delay occurs and the EN pin is set to 0 followed by popping the link register off the stack and returning to where the procedure was originally called.

LCD_DATA begins by pushing the link register onto the stack in order to provide to correct return address when the procedure is complete. Using the GPIOC_BSRR register, the RS pin is set HIGH and the RW pin is set LOW. A 1ms delay occurs through the DELAY_1ms procedure and then the EN pin is set HIGH through the GPIOC_BSRR register. Another 1ms delay occurs followed by sending the data held in R0 to PORTA, which is then displayed on the LCD, through the GPIOA_ODR register. After another 1ms delay, the EN pin is set LOW through the GPIOC_BSRR register and the link register is popped from the stack to restore the correct return address.

LCD_CHAR simply operates by pushing the link register onto the stack in order to provide the correct return address followed by branching with link to the LCD_DATA procedure. The link register is restored by popping it from the stack and the procedure is complete.

LCD_CLEAR operates by pushing the link register onto the stack followed by moving the LCD clear command, 0x01, into R0. The command is sent to the LCD through the LCD_CMD procedure and then the link register is restored by popping it from the stack.

LCD_2NDLINE operates similarly to the LCD_CLEAR procedure; however, instead of moving the clear command into R0, the second line command, 0x0C, is moved into R0. The command is sent to the LCD through the LCD_CMD procedure and the link register is popped from the stack to restore the correct return address.

LCD_STRING operates by initially pushing the link register onto the stack as other procedures are branched to within this procedure. Using the GPIOA_ODR register, the string address is loaded into R10 and the first byte of the string is loaded into R0. R0 is then compared to the value 0 to check for the terminator and given the terminator is present, the procedure jumps to pop the link register from the stack and return to where it was originally branched to. However, if the terminator is not present in R0, the LCD_DATA procedure is branched to followed by a 1ms delay. R10 is then incremented by 1 to move to the next byte of the string and the procedure loops back to load the next byte into R0 and check for the terminator.

DELAY_1ms operates by initially moving the value 50 into R6 for counter 2 and 255 into R7 for counter 1. R7 is decremented using the SUBS instruction to ensure that the zero flag is affected. Given that the zero flag is 0 a branch occurs back to decrementing R7. This loop continues until R7 equals 0 and the zero flag is 1. Once the zero flag is 1, R6 is decremented using the SUBS instruction. Given R6 is not 0 and the zero flag is 0, the procedure loops back to restoring R7 with 255 and performing the process over again. This will continue until both R6 and R7 are both 0. Following both registers being 0, the procedure is complete and returns to where it was originally branched from.

DELAY_ms operates by initially pushing the link register onto the stack and moving the desired number of ms, which is loaded into R0 before branching to the procedure, into R8. The procedure then branches to the DELAY_1ms procedure and decrements R8 with the SUBS instruction following the return of the branch. This will continue looping until R8 is 0 and then the link register will be popped from the stack.

The strings needed for activity 1 are placed in a new ‘area’ with the directives “DATA” and “READONLY”. The strings are defined by the labels “NAME” and “COURSE” and use 0 as the terminator.

For activity 2, the LCD driver created in Activity 1 is copied exactly as is into the second program. For the main code, the ports are set appropriately, which is the same as in Activity 1. The LCD_INIT procedure is called followed by the READ and RANDOM procedures. Since the RANDOM procedure produces the random number in R0, R0 is moved to R2 and the RANDOM procedure is called once again to obtain the second random number. For the second random number, R0 is moved to R11 and the WRITE procedure is called to display the values on the LCD. The program is complete after displaying the values on the LCD.

READ operates by reading the states of the 8 DIP switches on PORTB through the GPIOB_IDR register and storing the value in R0 before returning.

RANDOM operates by moving the value held in R0 into R1, shifting R0 to the left once using the LSL instruction, placing the exclusive OR of R1 and R0 into R1, and right shifting R1 6 times. A logical AND of R0 and 0x7E is performed and placed in R0 in order to mask bits 1 – 6 and a logical AND of R1 and 0x01 is placed into R1 to mask bit 0. The logical OR of R0 and R1 occurs and the result is placed into R0 to produce the random value.

WRITE operates by pushing the link register onto the stack and loading the “FIRST” string address into R10 and calling LCD_STRING. R2, where the first random number is held is moved into R3 and shifted right four times to place the upper nibble in the lower nibble. The lower nibble is masked, and the CONVERT procedure is called followed by the LCD_STRING procedure. At this point, the first random number is written to the LCD. For the second random number, the “SECOND” string address is loaded into R10 and the LCD_STRING procedure is called. The remaining process follows the same as writing the first random number to the LCD.

CONVERT operates by initially comparing a byte of the random number in R3 to 0 and given they are equal, writes the binary 0 string to the LCD. Given R3 does not equal 0, the procedure moves to check if R3 equals 1. The procedure will compare R3 until a match is found and then the procedure is complete.

Results

Results for this lab were obtained through following the steps listed in the procedure section of this report. Lab 3.2 included two activities that start with the implementation of an ARM assembly program that is an LCD driver used to initialize and control the connected LCD. The second activity uses the LCD driver from activity two and requires further implementation in order to read that state of 8 DIP switches and create two random numbers that are then displayed on the LCD.

For activity 1, the LCD driver was compiled and debugged within the Keil µVision software. It was initially determined that proper operation was not achieved as branches to subroutines that included branches to other subroutines would not return to the correct instruction when finished. After further research as to how the link register and the BL instruction operate, a solution was discovered. In order to achieve nested branches, the link register must be pushed onto the stack upon entering the subroutine that includes the BL instruction within itself. Once the subroutine is finished, the link register is popped from the stack and BX LR instruction returns to the correct instruction. Following the implementation of pushing and popping the link register on and off the stack, the program was once again debugged. It was then determined that the program operated as expected and uploading to the Blue Pill could occur. The program was then uploaded to the Blue Pill and testing began. Upon testing the LCD driver, viewing "CAMERON" on the first line and "ECEN 3320" on the second line proved successful operation. Initially, this did not occur, and the LCD only turned ON, but did not display any characters. After reviewing the assembly code, an issue was observed where the value needing to be written through the LCD_CMD subroutine was held in R0 and needed to be output through PORTA, but the LCD_CMD procedure was outputting the value held at the RAM address of the value held in R0 to PORTA. Due to this error, none of the correct data was being sent to the LCD. After fixing the error, the program was uploaded to the Blue Pill once again and testing occurred. It was then determined that the program operated as expected and without error as the LCD displayed that strings stated earlier.

For activity 2, the LCD driver was built on to create a Read, Random, Write program that reads the states of 8 DIP switches and uses an algorithm to display two random numbers on an LCD. The program was implemented without issue as all the logic had already been determined in previous labs. Once compiling the program in Keil and debugging, it was determined that the program operated as expected. The program was then uploaded to the Blue Pill and testing occurred. While testing, an undetermined issue occurred that kept pins 2 and 4 on PORTB HIGH regardless of if the DIP switch was ON or OFF. The program was reviewed, and the problem could not be discovered. Although the issues with pins 2 and 4 of PORTB did not allow the program to operate exactly as expected, the program could still be tested by accounting for the pins always being HIGH. Following testing, it was determined that other than the issues with pins 2 and 4 of PORTB, the program worked as expected and produced the correct results.

Answers to Posted Questions

1. How many port groups are present in the STM32F103C8?
   - 2 port groups. PORTA and PORTB.
2. How many bits are present in any port group?
   - 16 bits
3. When manipulating I/O port lines configured as OUTPUTS, what is the difference between using the ODR register, versus using the BSRR and BRR? Under what circumstances would you prefer one approach over the other?
   - ODR is used to write data to I/O lines configured as outputs while BSRR and BRR are used to set corresponding bits on an I/O line either HIGH or LOW. This means that if only one bit is needing changed, using either BSRR or BRR is the most effective whereas changing multiple bits on an I/O line would be more efficiently done by using the ODR register.
4. What does the 'S' prefix mean in an instruction such as MOVS, or ADDS, or ANDS?
   - The 'S' indicates that following the execution of the instruction, the respective flags will be affected.
5. What is the difference between LDR R4, =0x01234567, vs MOVS R4, #0x80?
   - LDR R4, =0x01234567 places the value at memory address 0x01234567 into R4. MOVS R4, #0x80 places the value 0x00000080 into R4 and affects the respective flags.
   - Why do we have to use LDR to load a constant into a register, and sometimes we can (and cannot) use the MOV instruction?
     - MOV can only be used for constants within the range of the instruction whereas LDR can be used to load any 32-bit number.

- What is the ‘=’ sign for?
    - Loads any 32-bit number or the value at the address of a label.
- What is the ‘#’ for?
    - Written in front of an immediate value.

Conclusion

Lab 3.2 introduced a deeper understanding of ARM assembly language programming through the design and implementation of an LCD driver and Read, Random, Write program. Since both of these programs have already been designed and implemented previously in other languages, this lab revolved around translating the code and becoming familiar with the ARM assembly instruction set. As the lab was completed, a greater understanding of ARM and its instruction set were gained. Although few issues occurred during the implementation and testing portions of the lab, it was determined that both the LCD driver and Read, Random, Write programs operated as expected. The information and understandings gained through this lab will prove to be beneficial in coming labs in ECEN 3320 and future courses.

Appendix

Figure I: LCD Driver ARM Assembly Source Code

```
; CAMERON BINIAMOW
; ECEN 3320
; LAB 3.2.2: I/O PORTS, LCDs, & READ/RANDOM/WRITE
; DUE: 11/05/2020


;========================Port Locations============================

; CLOCK ENABLE
RCC_APB2ENR  EQU 0x40021018

; Port C Register Addresses
GPIOC_CRL    EQU 0x40011000
GPIOC_CRH    EQU 0x40011004
GPIOC_IDR    EQU 0x40011008
GPIOC_ODR    EQU 0x4001100C
GPIOC_BSRR   EQU 0x40011010
GPIOC_BRR    EQU 0x40011014
GPIOC_LCKR   EQU 0x40011018

; Port B Register Addresses
GPIOB_CRL    EQU 0x40010C00
GPIOB_CRH    EQU 0x40010C04
GPIOB_IDR    EQU 0x40010C08
GPIOB_ODR    EQU 0x40010C0C
GPIOB_BSRR   EQU 0x40010C10
GPIOB_BRR    EQU 0x40010C14
GPIOB_LCKR   EQU 0x40010C18

; Port A register Addresses
GPIOA_CRL    EQU 0x40010800
GPIOA_CRH    EQU 0x40010804
GPIOA_IDR    EQU 0x40010808
GPIOA_ODR    EQU 0x4001080C
GPIOA_BSRR   EQU 0x40010810
GPIOA_BRR    EQU 0x40010814
GPIOA_LCKR   EQU 0x40010818

RS           EQU 0x2000                ;Pin 13
RW           EQU 0x4000                ;Pin 14
EN           EQU 0x8000                ;Pin 15

       EXPORT Reset_Handler
       EXPORT __Vectors

;=======================Vector Area=================================

       AREA VECTORS, DATA, READONLY
       THUMB

__Vectors
       DCD 0x20000190                  ;Points to top of stack
       DCD Reset_Handler               ;Points to our reset location

;=======================Startup Area=================================

       AREA STARTUP, CODE, READONLY
       THUMB

Reset_Handler       PROC
```

```
        LDR R5, =__main
        BX R5
        ENDP

;========================MAIN CODE================================
        AREA MAIN, CODE, READONLY
        THUMB

__main

        ; ENABLE CLOCKS ON EACH PORT
        LDR R1,=RCC_APB2ENR                 ;Setup address
        LDR R0,[R1]                         ;Read current value
        ORR R0,R0,#0xFC                     ;Only affect bits we want to change
        STR R0,[R1]                         ;Rewrite with clocks enabled

        ; SET PORT A AS INPUTS
        LDR R1, =GPIOA_CRH
        LDR R0, =0x44444444
        STR R0, [R1]
        LDR R1, =GPIOA_CRL
        LDR R0, =0x33333333
        STR R0, [R1]

        ; SET PORT B AS OUTPUTS
        LDR R1, =GPIOB_CRL
        LDR R0, =0x88888888
        STR R0, [R1]
        LDR R1, =GPIOB_ODR
        LDR R0, =0x0000
        STR R0, [R1]

        ; SET PC.13, PC.14, PC.15 AS OUTPUTS
        LDR R1,=GPIOC_CRH
        LDR R0,=0x33344444
        STR R0, [R1]

        BL      LCD_INIT                    ; INITIALIZE LCD

        LDR     R10,=NAME
        BL      LCD_STRING                  ; PRINT FIRST LINE ON LCD
        BL      LCD_2NDLINE                 ; MOVE CURSOR TO 2ND LINE
        LDR     R10,=COURSE
        BL      LCD_STRING                  ; PRINT SECOND LINE ON LCD

HERE
        B       HERE

;=======================LCD_INIT=======================
; INITITALIZES LCD

LCD_INIT
        PUSH    {LR}                        ; STORE LINK ADDRESS

        MOV     R0, #15                     ; 15ms DELAY
        BL      DELAY_ms

        MOV     R0, #0x30                   ; FUNCTION SET COMMAND
        BL      LCD_CMD

        MOV     R0, #5                      ; 5ms DELAY
        BL      DELAY_ms

        MOV     R0, #0x30                   ; FUNCTION SET COMMAND
        BL      LCD_CMD
```

```
        BL      DELAY_1ms                   ; 1ms DELAY

        MOV     R0, #0x30                   ; FUNCTION SET COMMAND
        BL      LCD_CMD

        MOV     R0, #0x3C                   ; 8-BIT INTERFACE
        BL      LCD_CMD

        MOV     R0, #0x08                   ; DISPLAY OFF
        BL      LCD_CMD

        MOV     R0, #0x06                   ; ENTRY MODE SET
        BL      LCD_CMD

        MOV     R0, #0x0F                   ; DISPLAY ON
        BL      LCD_CMD

        BL      LCD_CLEAR                   ; CLEAR DISPLAY

        POP     {LR}                        ; RESTORE LINK ADDRESS

        BX      LR                          ; RETURN


;========================LCD_CMD========================
; EXECUTES ALL COMMANDS ON THE LCD

LCD_CMD
        PUSH    {LR}                        ; STORE LINK ADDRESS

        LDR     R3, =GPIOC_BSRR             ; RS = 0 / RW = 0
        LDR     R4, =(RS << 16 :OR:RW << 16)
        STR     R4, [R3]

        BL      DELAY_1ms                   ; 1ms DELAY

        LDR     R3, =GPIOC_BSRR             ; EN = 1
        LDR     R4, =EN
        STR     R4, [R3]

        BL      DELAY_1ms                   ; 1ms DELAY

        LDR     R5, =GPIOA_ODR              ; OUTPUT COMMAND THROUGH PORT A
        STR     R0, [R5]

        BL      DELAY_1ms                   ; 1ms DELAY

        LDR     R3, =GPIOC_BSRR             ; EN = 0
        LDR     R4, =(EN << 16)
        STR     R4, [R3]

        POP     {LR}                        ; RESTORE LINK ADDRESS

        BX      LR                          ; RETURN


;========================LCD_DATA========================
; TRANSFERS DATA FROM R0 TO THE LCD

LCD_DATA
        PUSH    {LR}                        ; STORE LINK ADDRESS

        LDR     R3, =GPIOC_BSRR             ; RS = 1 / RW = 0
        LDR     R4, =(RS :OR: RW << 16)
```

```
        STR     R4, [R3]

        BL      DELAY_1ms                   ; 1ms DELAY

        LDR     R3, =GPIOC_BSRR             ; EN = 1
        LDR     R4, =EN
        STR     R4, [R3]

        BL      DELAY_1ms                   ; 1ms DELAY

        LDR     R5, =GPIOA_ODR              ; R0 => PORT A
        STR     R0, [R5]

        BL      DELAY_1ms                   ; 1ms DELAY

        LDR     R3, =GPIOC_BSRR             ; EN = 0
        LDR     R4, =(EN << 16)
        STR     R4, [R3]

        POP     {LR}                        ; RESTORE LINK ADDRESS
        BX      LR                          ; RETURN


;========================LCD_CHAR========================
; WRITE A SINGLE CHARACTER FROM R0 TO THE LCD

LCD_CHAR
        PUSH    {LR}                        ; STORE LINK ADDRESS
        BL      LCD_DATA                    ; SEND DATA TO LCD
        POP     {LR}                        ; RESTORE LINK ADDRESS
        BX      LR                          ; RETURN


;========================LCD_CLEAR========================
; CLEARS THE LCD

LCD_CLEAR
        PUSH    {LR}                        ; STORE LINK ADDRESS
        MOV     R0, #0x01                   ; CLEAR COMMAND
        BL      LCD_CMD                     ; SEND COMMAND TO LCD
        POP     {LR}                        ; RESTORE LINK ADDRESS
        BX      LR                          ; RETURN


;========================LCD_2NDLINE========================
; MOVES CURSOR TO THE SECOND LINE OF THE LCD

LCD_2NDLINE
        PUSH    {LR}                        ; STORE LINK ADDRESS
        MOV     R0, #0xC0                   ; 2ND LINE COMMAND
        BL      LCD_CMD                     ; SEND COMMAND TO LCD
        POP     {LR}                        ; RESTORE LINK ADDRESS
        BX      LR                          ; RETURN


;========================LCD_STRING========================
; WRITES A STRING FROM R0 ON THE LCD

LCD_STRING
        PUSH    {LR}                        ; STORE LINK ADDRESS

        LDR     R5, =GPIOA_ODR              ; SETUP ADDRESS
LOOP
        LDRB    R0, [R10]                   ; LOAD BYTE OF STRING INTO R0
        CMP     R0, #0                      ; CHECK FOR TERMINATOR
```

```
        BEQ     LOOP1                       ; END IF TERMINATOR
        BL      LCD_DATA                    ; SEND DATA TO LCD
        BL      DELAY_1ms                   ; 1ms DELAY
        ADD     R10, R10, #1                ; NEXT BYTE OF STRING
        B       LOOP                        ; CONTINUE FOR STRING LENGTH
LOOP1
        POP     {LR}                        ; RESTORE LINK ADDRESS
        BX      LR                          ; RETURN


;========================DELAY_1ms========================
; 1ms DELAY

DELAY_1ms
        MOV     R6, #50                     ; LOAD COUNTER 2
L1
        MOV     R7, #255                    ; LOAD COUNTER 1
L2
        SUBS    R7,R7,#1                    ; DECREMENT COUNTER 1
        BNE     L2                          ; LOOP UNTIL COUNTER 1 = 0
        SUBS    R6,R6,#1                    ; DECREMENT COUNTER 2
        BNE     L1                          ; LOOP UNTIL COUNTER 2 = 0
        BX      LR                          ; RETURN


;========================DELAY_ms========================
; (n)ms DELAY

DELAY_ms
        PUSH    {LR}                        ; STORE LINK ADDRESS
        MOV     R8, R0                      ; LOAD # OF ms
JUMP
        BL      DELAY_1ms                   ; 1ms DELAY
        SUBS    R8,R8,#1                    ; DECREMENT COUNTER
        BNE     JUMP                        ; LOOP UNTIL COUNTER = 0
        POP     {LR}                        ; RESTORE LINK ADDRESS
        BX      LR                          ; RETURN


;========================STRINGS========================

        AREA    STRINGS, DATA, READONLY
NAME
        DCB     "CAMERON",0
COURSE
        DCB     "ECEN-3320",0

        END
```

Figure II: Read, Random, Write ARM Assembly Source Code

```
; CAMERON BINIAMOW
; ECEN 3320
; LAB 3.2.3: I/O PORTS, LCDs, & READ/RANDOM/WRITE
; DUE: 11/05/2020


;========================Port Locations============================

; CLOCK ENABLE
RCC_APB2ENR EQU 0x40021018

; Port C Register Addresses
GPIOC_CRL     EQU 0x40011000
GPIOC_CRH     EQU 0x40011004
GPIOC_IDR     EQU 0x40011008
GPIOC_ODR     EQU 0x4001100C
GPIOC_BSRR    EQU 0x40011010
GPIOC_BRR     EQU 0x40011014
GPIOC_LCKR    EQU 0x40011018

; Port B Register Addresses
GPIOB_CRL     EQU 0x40010C00
GPIOB_CRH     EQU 0x40010C04
GPIOB_IDR     EQU 0x40010C08
GPIOB_ODR     EQU 0x40010C0C
GPIOB_BSRR    EQU 0x40010C10
GPIOB_BRR     EQU 0x40010C14
GPIOB_LCKR    EQU 0x40010C18

; Port A register Addresses
GPIOA_CRL     EQU 0x40010800
GPIOA_CRH     EQU 0x40010804
GPIOA_IDR     EQU 0x40010808
GPIOA_ODR     EQU 0x4001080C
GPIOA_BSRR    EQU 0x40010810
GPIOA_BRR     EQU 0x40010814
GPIOA_LCKR    EQU 0x40010818

RS            EQU 0x2000                ;Pin 13
RW            EQU 0x4000                ;Pin 14
EN            EQU 0x8000                ;Pin 15

      EXPORT Reset_Handler
      EXPORT __Vectors

;=======================Vector Area=================================

      AREA VECTORS, DATA, READONLY
      THUMB

__Vectors
      DCD 0x20000190                    ;Points to top of stack
      DCD Reset_Handler                 ;Points to our reset location

;=======================Startup Area=================================

      AREA STARTUP, CODE, READONLY
      THUMB

Reset_Handler       PROC

      LDR    R5, =__main
      BX     R5
```

```
        ENDP

;========================MAIN CODE===============================
        AREA MAIN, CODE, READONLY
        THUMB

__main

        ; ENABLE CLOCKS ON EACH PORT
        LDR     R1,=RCC_APB2ENR             ;Setup address
        LDR     R0,[R1]                     ;Read current value
        ORR     R0,R0,#0xFC                 ;Only affect bits we want to change
        STR     R0,[R1]                     ;Rewrite with clocks enabled

        ; SET PORT A AS INPUTS
        LDR     R1, =GPIOA_CRH
        LDR     R0, =0x44444444
        STR     R0, [R1]
        LDR     R1, =GPIOA_CRL
        LDR     R0, =0x33333333
        STR     R0, [R1]

        ; SET PORT B AS OUTPUTS
        LDR     R1, =GPIOB_CRL
        LDR     R0, =0x88888888
        STR     R0, [R1]
        LDR     R1, =GPIOB_ODR
        LDR     R0, =0x0000
        STR     R0, [R1]

        ; SET PC.13, PC.14, PC.15 AS OUTPUTS
        LDR     R1,=GPIOC_CRH
        LDR     R0,=0x33344444
        STR     R0, [R1]

        BL      LCD_INIT                    ; INITIALIZE LCD

        BL      READ                        ; READ DIP SWITCHES
        BL      RANDOM                      ; FIRST RANDOM NUMBER
        MOV     R2, R0                      ; STORE FIRST RANDOM NUMBER IN R2
        BL      RANDOM                      ; SECOND RANDOM NUMBER
        MOV     R11, R0
        BL      WRITE                       ; WRITE BINARY VALUES ON LCD

HERE
        B       HERE


;========================READ============================
; READS DIP SWITCHES

READ
        LDR     R1, =GPIOB_IDR
        LDR     R0, [R1]                    ; READ PORTA. STORE VALUE IN R0

        BX      LR                          ; RETURN


;========================RANDOM==========================
; PRODUCES RANDOM 8-BIT VALUE

RANDOM
        MOV     R1, R0                      ; COPY PORTA DATA INTO R1
        LSL     R0, R0, #1                  ; LEFT SHIFT ONCE
        EOR     R1, R1, R0                  ; XOR BITS 6 & 7
```

```
        ASR     R1, R1, #6                  ; RIGHT SHIFT XOR BIT TO BIT 0
        AND     R0, R0, #0x7E               ; MASK BITS 1 - 6
        AND     R1, R1, #0x01               ; MASK BIT 0
        ORR     R0, R0, R1                  ; REPLACE BIT 0 WITH XOR BIT
        BX      LR                          ; RETURN

;========================WRITE============================
; WRITES RANDOM VALUES ON LCD

WRITE
        PUSH    {LR}                        ; STORE LINK ADDRESS ON STACK

        LDR     R10,=FIRST                  ; WRITE "FIRST: " STRING
        BL      LCD_STRING

        MOV     R3, R2
        ASR     R3, R3, #4                  ; MOVE UPPER NIBBLE TO LOWER NIBBLE
        AND     R3, R3, #0x0F               ; MASK LOWER NIBBLE

        BL      CONVERT
        BL      LCD_STRING

        MOV     R3, R2
        AND     R3, R3, #0x0F               ; MASK LOWER NIBBLE

        BL      CONVERT
        BL      LCD_STRING

        MOV     R2, R11

        LDR     R10,=SECOND                 ; WRITE "SECOND: " STRING
        BL      LCD_STRING

        MOV     R3, R2
        ASR     R3, R3, #4                  ; MOVE UPPER NIBBLE TO LOWER NIBBLE
        AND     R3, R3, #0x0F               ; MASK LOWER NIBBLE

        BL      CONVERT
        BL      LCD_STRING

        MOV     R3, R2
        AND     R3, R3, #0x0F               ; MASK LOWER NIBBLE

        BL      CONVERT
        BL      LCD_STRING


        POP     {LR}
        BX      LR


;========================CONVERT==========================
; CONVERTS TO BINARY

CONVERT
        CMP     R3, #0
        BNE     CK1
        LDR     R10,=ZERO
        BX      LR
CK1     CMP     R3, #1
        BNE     CK2
        LDR     R10,=ONE
        BX      LR
CK2     CMP     R3, #2
        BNE     CK3
```

```
        LDR     R10,=TWO
        BX      LR
CK3     CMP     R3, #3
        BNE     CK4
        LDR     R10,=THREE
        BX      LR
CK4     CMP     R3, #4
        BNE     CK5
        LDR     R10,=FOUR
        BX      LR
CK5     CMP     R3, #5
        BNE     CK6
        LDR     R10,=FIVE
        BX      LR
CK6     CMP     R3, #6
        BNE     CK7
        LDR     R10,=SIX
        BX      LR
CK7     CMP     R3, #7
        BNE     CK8
        LDR     R10,=SEVEN
        BX      LR
CK8     CMP     R3, #8
        BNE     CK9
        LDR     R10,=EIGHT
        BX      LR
CK9     CMP     R3, #9
        BNE     CKA
        LDR     R10,=NINE
        BX      LR
CKA     CMP     R3, #10
        BNE     CKB
        LDR     R10,=TEN
        BX      LR
CKB     CMP     R3, #11
        BNE     CKC
        LDR     R10,=ELEVEN
        BX      LR
CKC     CMP     R3, #12
        BNE     CKD
        LDR     R10,=TWELVE
        BX      LR
CKD     CMP     R3, #13
        BNE     CKE
        LDR     R10,=THIRTEEN
        BX      LR
CKE     CMP     R3, #14
        BNE     CKF
        LDR     R10,=FOURTEEN
        BX      LR
CKF     LDR     R10,=FIFETEEN
        BX      LR

;========================LCD_INIT========================
; INITITALIZES LCD

LCD_INIT
        PUSH {LR}                       ; STORE LINK ADDRESS

        MOV     R0, #15                 ; 15ms DELAY
        BL      DELAY_ms

        MOV     R0, #0x30               ; FUNCTION SET COMMAND
        BL      LCD_CMD
```

```
        MOV     R0, #5                      ; 5ms DELAY
        BL      DELAY_ms

        MOV     R0, #0x30                   ; FUNCTION SET COMMAND
        BL      LCD_CMD

        BL      DELAY_1ms                   ; 1ms DELAY

        MOV     R0, #0x30                   ; FUNCTION SET COMMAND
        BL      LCD_CMD

        MOV     R0, #0x3C                   ; 8-BIT INTERFACE
        BL      LCD_CMD

        MOV     R0, #0x08                   ; DISPLAY OFF
        BL      LCD_CMD

        MOV     R0, #0x06                   ; ENTRY MODE SET
        BL      LCD_CMD

        MOV     R0, #0x0F                   ; DISPLAY ON
        BL      LCD_CMD

        BL      LCD_CLEAR                   ; CLEAR DISPLAY

        POP     {LR}                        ; RESTORE LINK ADDRESS

        BX      LR                          ; RETURN

;========================LCD_CMD========================
; EXECUTES ALL COMMANDS ON THE LCD

LCD_CMD
        PUSH    {LR}                        ; STORE LINK ADDRESS

        LDR     R3, =GPIOC_BSRR             ; RS = 0 / RW = 0
        LDR     R4, =(RS << 16 :OR:RW << 16)
        STR     R4, [R3]

        BL      DELAY_1ms                   ; 1ms DELAY

        LDR     R3, =GPIOC_BSRR             ; EN = 1
        LDR     R4, =EN
        STR     R4, [R3]

        BL      DELAY_1ms                   ; 1ms DELAY

        LDR     R5, =GPIOA_ODR              ; OUTPUT COMMAND THROUGH PORT A
        STR     R0, [R5]

        BL      DELAY_1ms                   ; 1ms DELAY

        LDR     R3, =GPIOC_BSRR             ; EN = 0
        LDR     R4, =(EN << 16)
        STR     R4, [R3]

        POP     {LR}                        ; RESTORE LINK ADDRESS

        BX      LR                          ; RETURN

;========================LCD_DATA========================
; TRANSFERS DATA FROM R0 TO THE LCD
```

```
LCD_DATA
        PUSH   {LR}                          ; STORE LINK ADDRESS

        LDR    R3, =GPIOC_BSRR               ; RS = 1 / RW = 0
        LDR    R4, =(RS :OR: RW << 16)
        STR    R4, [R3]

        BL     DELAY_1ms                     ; 1ms DELAY

        LDR    R3, =GPIOC_BSRR               ; EN = 1
        LDR    R4, =EN
        STR    R4, [R3]

        BL     DELAY_1ms                     ; 1ms DELAY

        LDR    R5, =GPIOA_ODR                ; R0 => PORT A
        STR    R0, [R5]

        BL     DELAY_1ms                     ; 1ms DELAY

        LDR    R3, =GPIOC_BSRR               ; EN = 0
        LDR    R4, =(EN << 16)
        STR    R4, [R3]

        POP    {LR}                          ; RESTORE LINK ADDRESS
        BX     LR                            ; RETURN


;========================LCD_CHAR========================
; WRITE A SINGLE CHARACTER FROM R0 TO THE LCD

LCD_CHAR
        PUSH   {LR}                          ; STORE LINK ADDRESS
        BL     LCD_DATA                      ; SEND DATA TO LCD
        POP    {LR}                          ; RESTORE LINK ADDRESS
        BX     LR                            ; RETURN


;========================LCD_CLEAR========================
; CLEARS THE LCD

LCD_CLEAR
        PUSH   {LR}                          ; STORE LINK ADDRESS
        MOV    R0, #0x01                     ; CLEAR COMMAND
        BL     LCD_CMD                       ; SEND COMMAND TO LCD
        POP    {LR}                          ; RESTORE LINK ADDRESS
        BX     LR                            ; RETURN


;========================LCD_2NDLINE========================
; MOVES CURSOR TO THE SECOND LINE OF THE LCD

LCD_2NDLINE
        PUSH   {LR}                          ; STORE LINK ADDRESS
        MOV    R0, #0xC0                     ; 2ND LINE COMMAND
        BL     LCD_CMD                       ; SEND COMMAND TO LCD
        POP    {LR}                          ; RESTORE LINK ADDRESS
        BX     LR                            ; RETURN


;========================LCD_STRING========================
; WRITES A STRING FROM R0 ON THE LCD

LCD_STRING
        PUSH   {LR}                          ; STORE LINK ADDRESS
```

```
        LDR    R5, =GPIOA_ODR              ; SETUP ADDRESS
LOOP
        LDRB   R0, [R10]                   ; LOAD BYTE OF STRING INTO R10
        CMP    R0, #0                      ; CHECK FOR TERMINATOR
        BEQ    LOOP1                       ; END IF TERMINATOR
        BL     LCD_DATA                    ; SEND DATA TO LCD
        BL     DELAY_1ms                   ; 1ms DELAY
        ADD    R10, R10, #1                ; NEXT BYTE OF STRING
        B      LOOP                        ; CONTINUE FOR STRING LENGTH
LOOP1
        POP    {LR}                        ; RESTORE LINK ADDRESS
        BX     LR                          ; RETURN

;========================DELAY_1ms========================
; 1ms DELAY

DELAY_1ms
        MOV    R6, #50                     ; LOAD COUNTER 2
L1
        MOV    R7, #255                    ; LOAD COUNTER 1
L2
        SUBS   R7,R7,#1                    ; DECREMENT COUNTER 1
        BNE    L2                          ; LOOP UNTIL COUNTER 1 = 0
        SUBS   R6,R6,#1                    ; DECREMENT COUNTER 2
        BNE    L1                          ; LOOP UNTIL COUNTER 2 = 0
        BX     LR                          ; RETURN

;========================DELAY_ms========================
; (n)ms DELAY

DELAY_ms
        PUSH   {LR}                        ; STORE LINK ADDRESS
        MOV    R8, R0                      ; LOAD # OF ms
JUMP
        BL     DELAY_1ms                   ; 1ms DELAY
        SUBS   R8,R8,#1                    ; DECREMENT COUNTER
        BNE    JUMP                        ; LOOP UNTIL COUNTER = 0
        POP    {LR}                        ; RESTORE LINK ADDRESS
        BX     LR                          ; RETURN

;========================STRINGS========================

        AREA   STRINGS, DATA, READONLY
FIRST
        DCB    "FIRST: ",0
SECOND
        DCB    "SECOND: ",0
ZERO
        DCB    "0000", 0
ONE
        DCB    "0001", 0
TWO
        DCB    "0010", 0
THREE
        DCB    "0011", 0
FOUR
        DCB    "0100", 0
FIVE
        DCB    "0101", 0
SIX
        DCB    "0110", 0
```

```
SEVEN
       DCB    "0111", 0
EIGHT
       DCB    "1000", 0
NINE
       DCB    "1001", 0
TEN
       DCB    "1010", 0
ELEVEN
       DCB    "1011", 0
TWELVE
       DCB    "1100", 0
THIRTEEN
       DCB    "1101", 0
FOURTEEN
       DCB    "1110", 0
FIFETEEN
       DCB    "1111", 0


       END
```

CAMERON BINIAMOW

"LAB 3.2"

ECEN 3320

START

↓

STR R#, [R#]

PUSHES

0x08000190

C	0x43
A	41
M	4D
E	45
R	52
O	4F
N	4E
E	45
C	43
E	45
N	4E
-	2D
3	33
3	33
2	32
0	30

~~6 5 4 3 2 1 0~~

R0: X X X X X X X X

R0: ~~7~~ 6 5 4 3 2 1 0

R1: X 7 6 5 4 3 2 1

MASK

TOTAL HOURS

5.5

Note: This sign-off sheet must be printed prior to requesting a sign-off from instructor or TA. You must submit this form with your typewritten report to receive credit. NO EXCEPTIONS.

ACTIVITY 1: Successful port of LCD Routines

- Demonstration of running program on Blue Pill showing name/course on LCD.

Instructor/TA Initials: Matthew J Bentz Date: 11/20/20

ACTIVITY 2: Read/Random/Write

- Demonstration of READ/RANDOM/WRITE with few test cases on DIP as input.

Instructor/TA Initials: Matthew J Bentz Date: 11/20/20