# Homework 7

**Student**

Devonte Dane Billings

**Total Points**

99 / 100 pts

**Question 1**

## Overview

**0** / 0 pts

✔ **+ 0 pts** Correct

**+ 0 pts** Incorrect

**Question 2**

User I/0

**5** / 5 pts

2.1 — .stringz

**2** / 2 pts

✔ **+ 2 pts** Correct (12)

**+ 0 pts** Incorrect

2.2 — (no title)

**3** / 3 pts

✔ **+ 3 pts** Correct (GETC)

**+ 0 pts** Incorrect

**Question 3**

Subroutine Basics                                                               **20** / 21 pts

**3.1**  **BR vs. JMP**                                                          **3** / 3 pts

  ✔  **+ 1 pt** Correctly explains functionality of **BR**

  ✔  **+ 1 pt** Correctly explains functionality of **JMP**

  ✔  **+ 1 pt** Describes a **valid/correct** instance where one would use **BR** over **JMP**

    **+ 0 pts** Incorrect

**3.2**  **JSR vs. JSRR**                                                        **2** / 3 pts

  ✔  **+ 1 pt** Explains the functionality of **JSR**

  ✔  **+ 1 pt** Explains the functionality of **JSRR**

    **+ 1 pt** Describes a **valid/correct** instance to use one instruction over the other

    **+ 0 pts** Incorrect

**3.3**  **RET**                                                                **4** / 4 pts

  ✔  **+ 4 pts** Correct (JMP R7)

    **+ 0 pts** Incorrect

**3.4**  **JSR**                                                                **6** / 6 pts

  ✔  **+ 6 pts** Correct - uses one of the following strategies:
        (1)
        LEA RX, FOO
        JSRR RX

        (2)
        LEA R7, 1
        BR FOO

    **+ 3 pts** Partially correct - does one of the following:
        (1) Attempts to use LEA + JSRR strategy, but implements incorrectly
        (2) Attempts to use LEA + BR strategy, but implements incorrecctly

    **+ 0 pts** Incorrect

**3.5**  **Caller and Callee**                                                   **5** / 5 pts

  ✔  **+ 2 pts** Selects FOO as the caller

  ✔  **+ 3 pts** Provides valid explanation (ex. since foo calls bar, foo must be the caller)

    **+ 0 pts** Incorrect

**Question 4**

Subroutine Tracing                                                    **18** / 18 pts

**4.1**  (no title)                                                    **6** / 6 pts

> ✔  **+ 6 pts** Correct (-7)

> **+ 0 pts** Incorrect

**4.2**  (no title)                                                    **6** / 6 pts

> ✔  **+ 6 pts** Correct (-2)

> **+ 0 pts** Incorrect

**4.3**  (no title)                                                    **6** / 6 pts

> ✔  **+ 6 pts** Correct (0x3005)

> **+ 0 pts** Incorrect

**Question 5**

Subroutine Fill-in-the-Blanks                                         **18** / 18 pts

**5.1**  (no title)                                                    **6** / 6 pts

> ✔  **+ 6 pts** Correct (ADD R1, R1, 1)

> **+ 0 pts** Incorrect

**5.2**  (no title)                                                    **6** / 6 pts

> ✔  **+ 6 pts** Correct (LDR R3, R3, 0)

> **+ 0 pts** Incorrect

**5.3**  (no title)                                                    **6** / 6 pts

> **+ 0 pts** Incorrect

> ✔  **+ 6 pts** Correct
>     (1) ADD R1, R1, -1
>     (2) AND R1, RX, 0

**Question 6**

The Stack                                                                    **8** / 8 pts

**6.1** | **Stack and Return Addresses**                                      **4** / 4 pts

✔  **+ 4 pts** Correct - calling multiple subroutines would clobber R7

**+ 2 pts** Partially correct - example: (losing system state but no explicit mention of R7 and/or losing return address)
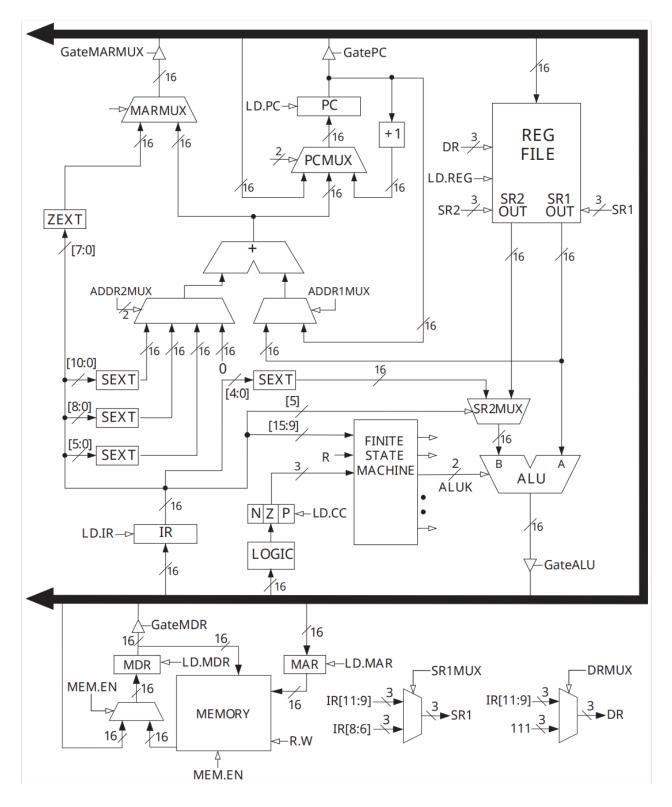
**+ 0 pts** Incorrect

**6.2** | **(no title)**                                                     **4** / 4 pts

✔  **+ 4 pts** Correct (ADD R6, R6, 3)

**+ 0 pts** Incorrect

**Question 7**

I/O and the Trap Vector Table                                                    **30** / 30 pts

**7.1**  **(no title)**                                                              **3** / 3 pts

✔  **+ 3 pts**  Correct (true)

**+ 0 pts**  Incorrect

**7.2**  **(no title)**                                                              **3** / 3 pts

✔  **+ 3 pts**  Correct (false)

**+ 0 pts**  Incorrect

**7.3**  **(no title)**                                                              **3** / 3 pts

✔  **+ 3 pts**  Correct (false)

**+ 0 pts**  Incorrect

**7.4**  **(no title)**                                                              **3** / 3 pts

✔  **+ 3 pts**  Correct (as a memory location)

**+ 0 pts**  Incorrect

**7.5**  **(no title)**                                                              **3** / 3 pts

✔  **+ 3 pts**  Correct (the address of the trap service subroutines)

**+ 0 pts**  Incorrect

**7.6**  **(no title)**                                                            **15** / 15 pts

✔  **+ 3 pts**  (1) Code section 1 correct:

LDI RX, KBSRADDR (RX any register except R1)

✔  **+ 3 pts**  (2) Code section 2 correct:

LDI R0, KBDRADDR

✔  **+ 5 pts**  (3) Correct explanation of code:

stores characters starting from R1/BUFFER and terminates when 'z' is pressed

✔  **+ 4 pts**  (4) Correct explanation of stop condition changes:

would need to check against the ASCII value of 'Z' (optional: 0x5A) in addition to 'z' which we already have)

**+ 0 pts**  Incorrect

## Q1 Overview
0 Points



| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| **ADD** | | 00 | 01 | | | DR | | | SR1 | | 0 | | 00 | | SR2 | |

| ADD | 0001 | DR | SR1 | 1 | imm5 |
|---|---|---|---|---|---|

| | 15 14 13 12 | 11 10 9 | 8 7 6 | 5 | 4 3 | 2 1 0 |
|---|---|---|---|---|---|---|
| AND | 0101 | DR | SR1 | 0 | 00 | SR2 |
| AND | 0101 | DR | SR1 | 1 | imm5 | |

| | 15 14 13 12 | 11 10 9 | 8 7 6 | 5 4 3 2 1 0 |
|---|---|---|---|---|
| NOT | 1001 | DR | SR | 111111 |

| | 15 14 13 12 | 11 | 10 | 9 | 8 7 6 5 4 3 2 1 0 |
|---|---|---|---|---|---|
| BR | 0000 | N | Z | P | PCoffset9 |

| | 15 14 13 12 | 11 10 9 | 8 7 6 | 5 4 3 2 1 0 |
|---|---|---|---|---|
| JMP | 1100 | 000 | BaseR | 000000 |
| JSR | 0100 | 1 | PCoffset11 | |
| JSRR | 0100 | 0 00 | BaseR | 000000 |

| | 15 14 13 12 | 11 10 9 | 8 7 6 | 5 4 3 2 1 0 |
|---|---|---|---|---|
| LD | 0010 | DR | PCoffset9 | |
| LDI | 1010 | DR | PCoffset9 | |
| LDR | 0110 | DR | BaseR | offset6 |
| LEA | 1110 | DR | PCoffset9 | |

| | 15 14 13 12 | 11 10 9 | 8 7 6 | 5 4 3 2 1 0 |
|---|---|---|---|---|
| ST | 0011 | SR | PCoffset9 | |
| STI | 1011 | SR | PCoffset9 | |
| STR | 0111 | SR | BaseR | offset6 |

| | 15 14 13 12 | 11 10 9 | 8 7 6 5 4 3 2 1 0 |
|---|---|---|---|
| TRAP | 1111 | 0000 | trapvect8 |

This homework is worth a total of 100 points.

We have provided LC-3 datapath and instruction set here, but LC-3 reference materials can **also be found in Canvas > Files**.

This question ( Q1 ) cannot be answered. It's used for formatting instructions. Do not worry about Gradescope saying you haven't answered one question. It's this one!

Please complete the following problems. The collaboration policy for the course still applies. Refer to the syllabus for details regarding this policy.

**Please answer these questions without copying the provided assembly code into LC-3 Tools. Why? You WILL NOT have access to LC-3 Tools for the quizzes.**

## Q2 User I/0
**5 Points**

### Q2.1 .stringz
**2 Points**

How many memory locations does the pseudo-op `.stringz "Hello World"` allocate in memory?

- ○ 10
- ○ 11
- ○ 9
- ● 12

**Q2.2**
**3 Points**

Choose the correct TRAP instruction that receives some user input and sets R7 to contain the value 1 if it is the character 'A' and 0 otherwise.

```
.orig x3000
          ;; TODO: Your code here
          AND R7, R7, 0
          LD R1, CHAR
          NOT R1, R1
          ADD R1, R1, 1
          ADD R2, R1, R0
          BRnp END
          ADD R7, R7, 1
END       HALT

CHAR .fill x41 ;; ASCII 'A'

.end
```

○ PUTS

◉ GETC

○ OUT

○ HALT

## Q3 Subroutine Basics
21 Points

### Q3.1 BR vs. JMP
3 Points

What is the difference between `BR` and `JMP`?

**Explain** the functionality of each instruction and describe **one** instance where you would use `BR` over `JMP`.

> While BR sets PC to the value of PC* + PCOffset9 upon a condition, JMP sets PCC to the value of the base register given in the instruction unconditionally. I would want to use BR instead of JMP if I wanted to create a while loop that continually branches to a previous line until a condition is fulfilled.

### Q3.2 JSR vs. JSRR
3 Points

What is the difference between `JSR` and `JSRR`?

**Explain** each of the instructions and describe **one** instance where you would use one over the other.

> Although both JSR and JSRR set R7 to the value of PC*, JSR will set PC to equal PC* + PCOffset11 meanwhile JSRR will set PC to the value of the base register given in the instruction.

**Q3.3 RET**
**4 Points**

Which one of the following is equivalent to `RET` ?

🔘 JMP R7

⚪ LEA R0

⚪ JSRR R7

⚪ BRnzp R7

**Q3.4 JSR**
**6 Points**

Consider the code snippet below:

```
.orig x3000
    AND R0, R0, 0
    JSR FOO ;REPLACE THIS LINE
    PUTS
    HALT


FOO
    ;CODE FOR FOO
    RET
.end
```

Let's say I want to replace `JSR FOO` (line 3 in the code above) with other LC-3 instructions.

Write **exactly two** LC-3 instructions that preserve the functionality of `JSR FOO` but do **not** use the `JSR` instruction.

```
LEA R7, 1
BR FOO
```

## Q3.5 Caller and Callee
### 5 Points

Consider the code snippet below that has two functions, `foo` and `bar`:

```
1       .orig x3000
2
3       FOO
4               LEA R0, PROMPT
5               PUTS
6               GETC
7               JSR BAR
8               ADD R1, R1, 0
9               BRn ELSE
10              LEA R0, WIN
11              PUTS
12              BR DONE
13      ELSE    LEA R0, LOSE
14              PUTS
15      DONE    HALT
16
17      BAR
18              LD R2, CHAR
19              NOT R2, R2
20              ADD R1, R1, 1
21              ADD R1, R1, R0
22              RET
23
24      PROMPT  .stringz "Enter a character and win some money: "
25      WIN     .stringz "You won $10,000,000 dollars!\n"
26      LOSE    .stringz "You didn't win any money :(\n"
27
28      CHAR .fill x51
29
30      .end
31
```

Which function is the **caller**?

- ⦿ FOO
- ○ BAR
- ○ neither

**Justify your answer** in three sentences or less.

Usually, the program that contains the subroutine is referred to as the callee is called "the caller." Conversely, the subroutine that calls a return to the program is called the "callee." While BAR acts as a Callee by returning back to FOO, FOO acts as the caller as it jumps to the subroutine BAR.

## Q4 Subroutine Tracing
### 18 Points

Given the following LC-3 Assembly code, answer the questions below

```
.orig x3000
        AND R0, R0, 0
        ADD R0, R0, 7
        JSR FOO
        ADD R0, R1, R0
        AND R1, R1, 0
        HALT
.end

.orig x300F
FOO     NOT R0, R0
        ADD R0, R0, 1
        LD R1, MYLABEL
        ADD R7, R7, R1
        NOT R1, R1
        ADD R1, R1, 1
        RET

MYLABEL .fill x0002
.end
```

## Q4.1
**6 Points**

What is the value in **R0** (as a decimal number) after the program is finished running?

-7

## Q4.2
**6 Points**

What is the value in **R1** (as a decimal number) after the program is finished running?

-2

## Q4.3
**6 Points**

What is the value of **R7** (as a hexadecimal number) after the program is finished running?

Use the prefix `0x` in your answer.

0x3005

## Q5 Subroutine Fill-in-the-Blanks
18 Points

Consider the following pseudocode translated into LC-3 Assembly code.

**Pseudocode**:

```
String str = "aibohphobia";
boolean isPalindrome = true
int length = 0;
while (str[length] != '\0') {
    length++;
}
int left = 0
int right = length - 1
while(left < right) {
    if (str[left] != str[right]) {
        isPalindrome = false;
        break;
    }
    left++;
    right--;
}
mem[mem[ANSWERADDR]] = isPalindrome;
```

**LC-3 Assembly Code:**

```
AND R1, R1, 0     ;;R1 = isPalindrome = 0 = false
;CODE SECTION 1   - we need to set isPalindrome to true
AND R2, R2, 0     ;;R2 = length = 0

WHILE
ADD R3, R2, R0    ;R3 = length + starting address of string
;CODE SECTION 2   - we want R3 = str[length]
BRz ENDWHILE
ADD R2, R2, 1     ;;R2 = length++
BR WHILE


ENDWHILE
AND R3, R3, 0     ;;R3 = left = 0
ADD R4, R2, -1    ;;R4 = right = length - 1

LOOP
NOT R5, R4
ADD R5, R5, 1     ;;R5 = -right
ADD R5, R3, R5    ;;R5 = left - right
BRzp END

ADD R5, R0, R3
LDR R5, R5, 0     ;;R5 = str[left]|

ADD R6, R0, R4
LDR R6, R6, 0
NOT R6, R6
ADD R6, R6, 1     ;;R6 = -str[right]

ADD R5, R5, R6    ;;str[left] - str[right]
BRz EQUAL
;;CODE SECTION 3 - set isPalindrome to false
BR END

EQUAL
ADD R3, R3, 1     ;;R3 = left++
ADD R4, R4, -1    ;;R4 = right--
BR LOOP

END
HALT
```

**Notes/Assumptions:**

- the **address** of the first letter of the string has **ALREADY** been placed in **R0**
- a value of **1 is true**, and a value of **0 is false**
- the .orig and .end tags are not pictured, assume the program functions as expected

**Given the pseudocode and the commented LC-3 Assembly code above, you will fill in the missing lines of code:** CODE SECTION 1 **,** CODE SECTION 2 **, and** CODE SECTION 3 **.**

**Q5.1**
**6 Points**

Code Section 1:

```
ADD R1, R1, 1
```

**Q5.2**
**6 Points**

Code Section 2:

```
LDR R3, R3, 0
```

**Q5.3**
**6 Points**

Code Section 3:

```
AND R1, R1, 0
```

## Q6 The Stack
**8 Points**

### Q6.1 Stack and Return Addresses
**4 Points**

What is the purpose behind storing the return address on the stack if we already have a register ( R7 ) dedicated to holding the return address?
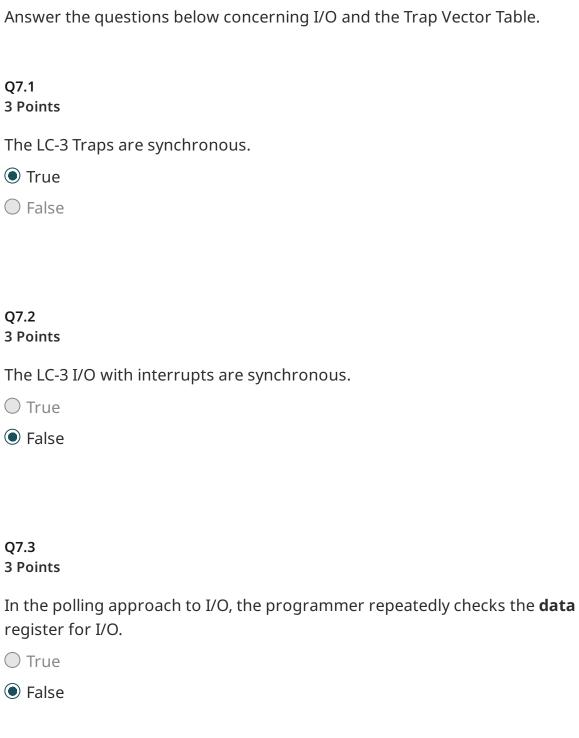
> During the second call to JSR/JSRR, R7 may hold a different value than the intended return address. Thus we save the return address from R7 onto the stack frame

### Q6.2
**4 Points**

Write exactly 1 LC-3 instruction that pops 3 elements off the stack without restoring values into any registers. You may assume popping will not not move R6 beyond the boundaries of the stack memory region.

> ADD R6,R6,3

**Q7 I/O and the Trap Vector Table**
30 Points

Answer the questions below concerning I/O and the Trap Vector Table.

**Q7.1**
3 Points

The LC-3 Traps are synchronous.

- ⦿ True
- ○ False

**Q7.2**
3 Points

The LC-3 I/O with interrupts are synchronous.

- ○ True
- ⦿ False

**Q7.3**
3 Points

In the polling approach to I/O, the programmer repeatedly checks the **data** register for I/O.

- ○ True
- ⦿ False

**Q7.4**
**3 Points**

How does the LC-3 allow you to access device registers?

○ As a general purpose register

○ As read-only memory in the FSM

○ As a datapath register (ex. PC, IR)

◉ As a memory location

**Q7.5**
**3 Points**

What does the trap vector table hold?

○ The code for trap service subroutines

○ An instruction that invokes the specific behavior of traps

◉ The addresses of the trap service subroutines

## Q7.6
**15 Points**

Consider the code provided below:

```
.orig x3000
            LEA R1, BUFFER
START
            ;;CODE SECTION 1

            BRzp START

            ;;CODE SECTION 2

            STR R0, R1, 0

            LD R2, TERMINATOR
            NOT R2, R2
            ADD R2, R2, 1

            ADD R0, R0, R2
            BRz END
            ADD R1, R1, 1
            BR START

END
            HALT

KBSRADDR    .fill xFE00
KBDRADDR    .fill xFE02
TERMINATOR  .fill x7A
BUFFER      .blkw 100
.end
```

The goal of this program is to read characters from the keyboard and stop after meeting a certain condition. You will insert the missing code at CODE SECTION 1 and CODE SECTION 2.

They should each be **one line of code**.

Code Section 1:

```
LDI R3, KBSRADDR
```

Code Section 2:

```
LDI R0, KBDRADDR
```

---

Briefly explain what the code above does, **aside** from the fact that it should read characters from the keyboard.

After reading a character, the character is added into an array of size 100. It stops adding characters to the array if the character is equal to "z".

Now, let's say we want our **stop condition** for the program to **no longer be case sensitive**. Describe what changes we would need to make to our program above.

I would inserting a new segment of code after x300A that will check if the character is "Z." If it is true, I will branch to END. Then I will reset R0 to its proper value to begin checking for "z" and allowing the remaining code to run.

I would do this by starting after line x300A, and inserting the lines:
LD R3, TERMINATOR
ADD R3, R3, -16
ADD R3, R3, -16
NOT R3, R3
ADD R3, R3, 1

LDI R0, KBDRADDR
ADD R0, R0, R2
BRz END