1. Give a simple example (a segment of LC-2 code) of how to use the command TRAP x23. (4)

**Answer**

```
LEA    R0, prompt
TRAP   x23
; ASCII code of the key pressed is in R0

prompt .STRINGZ "Enter a character> "
```
2. Assemble the following code. (4)

```
.ORIG x3000
LD   R3, one
LD   R1, two
again TRAP x20
ADD  R7, R0, R1
BRNZP branch
ST   R3, branch
branch BR again
HALT
one  .FILL x0E07
two  .FILL x-30
.END
```

**Answer**

```
x3000  00100111000001000  x2608  LD    R3, one
x3001  00100010000010001  x2209  LD    R1, two
x3002  1111000001000000  xF020  again TRAP GETC
x3003  0001111000000001  x1E01  ADD  R7, R0, R1
x3004  0000101000000110  x0A06  BRNZP branch
x3005  0011011000000110  x3606  ST   R3, branch
x3006  0000111000000010  x0E02  branch BRNZP again
x3007  1111000001010111  xF025  TRAP HALT
x3008  0001111000000011  x0E07  one  BRNZP x3007
x3009  1111111111101000 xFFD0  two  TRAP x00
```

3. What does the code in question #2 do? (4)

**Answer**

Repeat inputting a character until the character “0” is entered and the program halts.
4. Implement the following high-level pseudo code in LC-2 assembly language. You have to make use of the RET command for the branching to the different cases.

\[
\begin{align*}
n = 4 \\
\text{switch}(n) \{ \\
\text{case 1: cout } & \ll \text{ “case 1”; break;} \\
\text{case 2: cout } & \ll \text{ “case 2”; break;} \\
\text{case 3: cout } & \ll \text{ “case 3”; break;} \\
\text{case 4: cout } & \ll \text{ “case 4”; break;} \\
\text{otherwise: cout } & \ll \text{ “invalid case”; break;} \\
\}
\end{align*}
\]
Answer

.ORIG x3000

LD R1,n ; get n

; check that n is between 1 to 4
ADD R7,R1,#-1 ; check lower bound 1
BRN zero
ADD R7,R1,#-4 ; check upper bound 4
BRNZ checked
zero AND R1,R1,#0 ; n > 4. Set R1 to 0

; calculate case address in jump table
checked LEA R7,cases ; get base address of jump table
ADD R7,R7,R1 ; add n to base address
RET ; jump to correct case

; cases jump table
cases BR other
BR case1
BR case2
BR case3
BR case4
case1 LEA R0,msg1 ; get message for case
TRAP x22 ; print message
BR continue ; end of case statement
msg1 .STRINGZ "case 1"
case2 LEA R0,msg2 ; get message for case
TRAP x22 ; print message
BR continue ; end of case statement
msg2 .STRINGZ "case 2"
case3 LEA R0,msg3 ; get message for case
TRAP x22 ; print message
BR continue ; end of case statement
msg3 .STRINGZ "case 3"
case4 LEA R0,msg4 ; get message for case
TRAP x22 ; print message
BR continue ; end of case statement
msg4 .STRINGZ "case 4"
other LEA R0,msgother ; get message for case
TRAP x22 ; print message
BR continue ; end of case statement
msgother .STRINGZ "invalid case"

n .FILL #4 ; end of case statement
continue
HALT
.END
5. Assume that a one-dimensional array of size 100 starting at location ARRAY has already
been setup and filled with integers. Write a complete LC-2 subroutine that will find the
largest and second largest of all the numbers in this array. These two numbers are returned to
the caller in memory locations FIRST and SECOND, which has already been setup by the
caller. (4)

**Answer**

```
.ORIG x3000
; save all the registers
ST R0,reg0
ST R1,reg1
ST R2,reg2
ST R3,reg3
ST R4,reg4
ST R5,reg5
ST R6,reg6
ST R7,reg7

LEA R6, array ; for indexing into array

; calculate the end of the array
LD R0, ninetynine ; add 99 to start of array
ADD R3, R6, R0
NOT R3, R3 ; negate it
ADD R3, R3, #1

; R1 to store smallest and R2 to store 2nd smallest
; assume the largest number is the smallest
LD R1, smallest
LD R2, smallest

; main loop to check each entry in the array
next LDR R5, R6, #0 ; load array entry into R5
NOT R0, R5 ; negate entry
ADD R0, R0, #1

; is it larger than the largest?
ADD R7, R1, R0
BRN newlargest ; yes, it's the new largest

; is it larger than the 2nd largest?
ADD R7, R2, R0
BRN newsecond ; yes, it's the new 2nd largest

cont ; get next entry in array
ADD R6, R6, #1 ; increment array index
ADD R7, R6, R3 ; 100 times yet?
BRNZ next

; done
ST R1, first ; store largest and 2nd largest
ST R2, second ; in the two memory locations

; restore all the registers
LD R0, reg0
```
LD R1,reg1
LD R2,reg2
LD R3,reg3
LD R4,reg4
LD R5,reg5
LD R6,reg6
LD R7,reg7
RET

newlargest ; found a new largest
ADD R2,R1,#0 ; second <- first
ADD R1,R5,#0 ; first <- current entry
BR cont

newsecond ; found a new 2nd largest
ADD R2,R5,#0 ; second <- current entry
BR cont

smallest .FILL #0 ; smallest possible positive number.
Assume only for positive numbers
ninetynine .FILL #99
reg0 .FILL #0
reg1 .FILL #0
reg2 .FILL #0
reg3 .FILL #0
reg4 .FILL #0
reg5 .FILL #0
reg6 .FILL #0
reg7 .FILL #0
first .FILL #0
second .FILL #0
array .BLKW 100

.END