EE306 Introduction to Computing

Lab Assignment # 5

All programming assignments are individual. You are allowed to seek help only from TAs and the instructor.

The purpose of this assignment is to show how interrupt-driven Input/Output can interrupt a program that is running, execute the interrupt service routine, and return to the interrupted program, picking up exactly where it left off (just as if nothing had happened). In this assignment, we will use the Keyboard as the input device for interrupting the running program.

To demonstrate this, we will need three things:

1. The interrupt vector table has to be generated, interrupts will need to be enabled, and there will have to be space on the interrupt stack to save the state (PC and PSR) of interrupted routines while they are waiting to resume execution. The operating system will allocate this space, set up the interrupt vector table and enable the interrupts.

2. A program that will run at priority level 0.

3. An interrupt service routine that will be invoked as a result of someone hitting a key on the keyboard.

<u>**The user program**</u>.

Our user program will simply fill the screen with dots, as follows: The user program will print a newline character (ascii x0A), then a dot (ascii x2E), then a blank space (ascii x20), then a second blank space (ascii x20), and repeat this sequence of dot, space, space, until there are 20 dots on the line. Then go to the next line and repeat. The user program is to do this in an infinite loop.

One more thing: In order to make sure the dots don't get written to the screen too fast to be seen by the naked eye, the user program should include a piece of code that will count down from 2500 each time a character is output to the screen. A simple way to do this is with the following:

```
          LD R1, COUNT
DELAY     ADD R1, R1, #-1
          BRp DELAY
          ...
          ...

COUNT     .FILL #2500
```

<u>**The keyboard interrupt service routine**</u>.

The keyboard interrupt service routine will simply write to the screen ten times whatever key the person sitting at the keyboard typed.

VERY IMPORTANT: You are not allowed to use any TRAP instructions in your interrupt service routine. To display a character to the screen, you must poll the DSR and then write to the DDR, you may not call TRAP x21 (OUT), or use any of the other TRAP routines. If you use TRAP in the interrupt service routine or if you do not properly poll the DSR before writing to the DDR, your program is not correct and will fail our testing *even though it may appear to work when you test it*. You may not use TRAP instructions in the interrupt service routine only, you are free to use them in the user program if you wish.

Hint: Don't forget to save and restore any registers that you use in the interrupt service routine.

The operating system enabling code.

Unfortunately, we have not installed Windows or Linux on the LC-3, so we are going to require you to do the enabling actions that the operating system would do before you start to execute the User program.

1. Normally, the operating system will set up stack space for pushing the state of the running process, so it can return to it after the service routine has finished executing. We will assume the base of the stack is x2FFF, so the stack pointer (i.e. R6) must be initialized to x3000, indicating an empty stack.

2. Normally, the operating system establishes the interrupt vector table to contain the starting addresses of the corresponding interrupt services routines. You will have to do that for the keyboard interrupt. The starting address of the interrupt vector table is x0100 and the interrupt vector for the keyboard is x80.

3. Normally, the operating system would set KBSR[14] to enable the keyboard to interrupt user programs. You will have to do that.

You must perform these three tasks as part of your user program.

Your job.

Your job will be to write the user program augmented with the interrupt enabling code described above and the keyboard interrupt service routine.

The user program must be named **main.asm** and will be of the form:

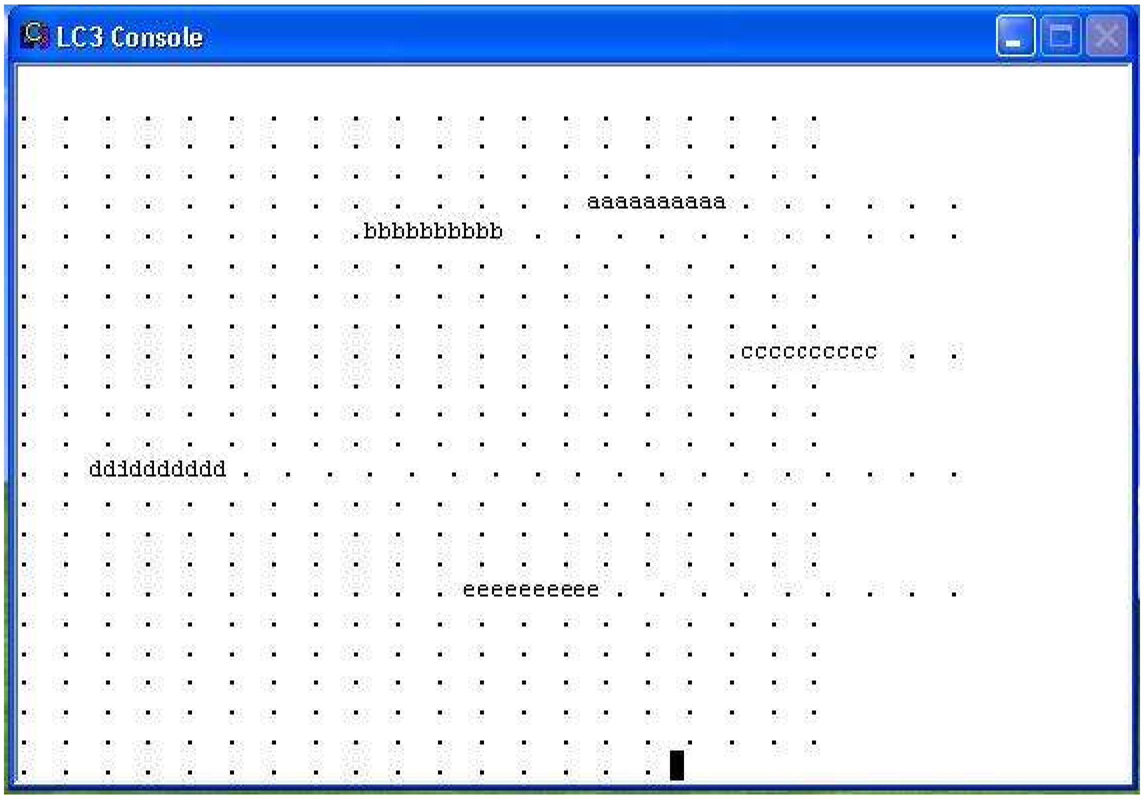
```
.ORIG    x3000
  --      ---     ; initialize the stack pointer
     ...
  --      ---
```

```
                  --       ---      ; set up the keyboard interrupt vector
table entry
                     ...
                  --       ---
                  --       ---      ; enable keyboard interrupts
                     ...
                  --       ---
                  --       ---      ; start of actual user program to print
dots
                     ...
                  --       ---
                .END
```

The interrupt service routine must be named **isr.asm** and will be of the form:

```
                .ORIG     x2000
                  --      ---       ; the code
                     ...
                  --      ---
                 RTI
                  --      ---       ; buffer space as required
                      ...
                  --      ---
                .END
```

An example of how the console should look when you run the program:



```
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . aaaaaaaaaa . . . . . .
. . . . . . . . . .bbbbbbbbbb . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . .cccccccccc . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . dddddddddd . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . eeeeeeeeee . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . .
```

Submission:
Name your programs as **main.asm** and **isr.asm**.