Fig. 3.8 ISR (Solution)

Interrupt-service routine

```
ILOC:   Subtract    SP, SP, #8          Save registers.
        Store       R2, 4(SP)
        Store       R3, (SP)
        Load        R2, PNTR            Load address pointer.
        LoadByte    R3, KBD_DATA        Read character from keyboard.
        StoreByte   R3, (R2)            Write the character into memory
        Add         R2, R2, #1            and increment the pointer.
        Store       R2, PNTR            Update the pointer in memory.
ECHO:   LoadByte    R2, DISP_STATUS     Wait for display to become ready.
        And         R2, R2, #4
        Branch_if_[R2]=0    ECHO
        StoreByte   R3, DISP_DATA       Display the character just read.
        Move        R2, #CR             ASCII code for Carriage Return.
        Branch_if_[R3]≠[R2] RTRN        Return if not CR.
        Move        R2, #1
        Store       R2, EOL             Indicate end of line.
        Clear       R2                  Disable interrupts in
        StoreByte   R2, KBD_CONT          the keyboard interface.
RTRN:   Load        R3, (SP)            Restore registers.
        Load        R2, 4(SP)
        Add         SP, SP, #8
        Return-from-interrupt
```

4. Where is the Frame Pointer?

4. No FP used in interrupts

6. From specification, KBD read is done by interrupts while DISP is done by polling within KBD ISR

5. Why the PTR is incremented by only one address location?

5. Each keystroke is an ASCII so only need 1 byte

6. Why do we wait-loop to echo, while there is no wait in getting the Keyboard input?

7. Why DI at keyboard interface, and not in PS or IENABLE registers?

8. Comment on the difference in saving/restoring registers in ISR vs Call Subroutine?

7. If DI in PS or IENABLE or both, then an interrupt may still be generated from KBD; DI at the KBD local level prevents all 3 interrupt signals

8. In subroutine, registers are saved using PUSH, while in interrupt, registers are saved using SP+offset, after SP is adjusted