Program Code

```
;**************************************************************
;*
;* Lab 5 Template
;*
;**************************************************************

;--------------------------------------------------
; Export Symbols
; KEEP THIS!!
;--------------------------------------------------
        XDEF      Entry       ; export 'Entry' symbol
        ABSENTRY  Entry       ; for absolute assembly: mark this as application entry point

;--------------------------------------------------
; Derivative-Specific Definitions
; KEEP THIS!!
;--------------------------------------------------
        INCLUDE   'derivative.inc'

;--------------------------------------------------
; Constants Section
; KEEP THIS!!
;--------------------------------------------------
ROM       EQU     $0400
DATA      EQU     $1000
PROG      EQU     $2000

;--------------------------------------------------
; Variable/Data Section
; KEEP THIS!!
;--------------------------------------------------
        ORG     DATA
OUT      FCB     $3C

;--------------------------------------------------
; Code Section
; KEEP THIS!!
;--------------------------------------------------
        ORG     PROG

; Insert your code following the label "Entry"
Entry:     ; KEEP THIS LABEL!!
        LDS     #PROG
        BSR     INIT
MAIN:
        LDAB    OUT
        STAB    PORTB

        BRA     MAIN

;--------------------------------------------------
; Init Subroutine
;--------------------------------------------------
INIT:
        SEI                 ; Disable maskable interrupts

        ; Disable 7-Segment Display
        BSET    DDRP,#$0F   ; Set Port P pins 0-3 to output
        BSET    PTP, #$0F     ; Disable 7-Segment Display

        ; LED
        BSET    DDRB,$FF
        BSET    DDRJ,$02
        BCLR    PTJ,$02

        ; Set Port H, pin 0 (PB1/DIP1)
        BCLR    DDRH, $0F

        ; Enable interrupts on Port H, pin 0 (PB1/DIP1)
        ; Disable interrupts on Port H for other pins
        MOVB    #$0F, PIEH
        MOVB    #$0F, PIFH    ; Clear flag for pin 0, set the rest

        CLI                 ; Enable maskable interrupts

        RTS

;--------------------------------------------------
; ISR
;--------------------------------------------------
PB_ISR:
        MOVB    #$0F,PIFH           ; Acknowledge interrupt
        LDAA    PTH
        COMA
        LDAB    OUT
        CMPA    #01
        BEQ     RIGHT_SHIFT
        BRA     SKIP1
RIGHT_SHIFT:
        LSRB
        STAB    OUT
        RTI
SKIP1:
        CMPA    #02
        BEQ     LEFT_SHIFT
        BRA     SKIP2
LEFT_SHIFT:
        LSLB
        STAB    OUT
        RTI
SKIP2:
        CMPA    #04
        BEQ     COMPLEMENT
        BRA     SKIP3
COMPLEMENT:
        COMB
        STAB    OUT
        RTI
SKIP3:
        CMPA    #08
        BEQ     RESET
RESET:
        LDAB    #$3C
        STAB    OUT
        RTI

;---------------------------------------------------
; Interrupt Vector
;---------------------------------------------------
        ; Port H Vector
        ; $3E4C = $FFCC - $C180
        ORG     $3E4C
        FDB     PB_ISR

;---------------------------------------------------
; End program
; KEEP THIS!!
;---------------------------------------------------
FINISH:
        NOP
                    END
```

Questions

1. What would happen if you have the SEI instruction at the beginning of your INIT subroutine but neglect to include CLI at the end if you are performing a maskable interrupt? What about a nonmaskable interrupt?

   SEI disables the maskable interrupts so you can make changes without causing any problems. At the end we must include a CLI to enable interrupts, otherwise all of the interrupts will be disabled and unable to be triggered.

A nonmaskable interrupt is not affected by the SEI or CLI commands would have no effect on them.

2. What is the purpose of the interrupt vector?

    An interrupt vector points to the memory location of the ISR to handle that specific interrupt. When an interrupt occurs, the MCU saves the values of the registers and jumps to the interrupt vector table to determine the address of the ISR that is going to handle that particular interrupt. Once the ISR reaches the RTI, the register values are restored and the control flows back to the next instruction after the point where the interrupt had occurred.

3. What gets put on the stack and why when an interrupt occurs?

When an interrupt occurs, before the ISR is executed, the MCU pushes all the register values to the stack and then it continues to the ISR. Once the ISR is done or completed, the MCU pulls the register values from the stack and continues the normal execution from where it left off.