Microcomputers 1

Lab 8

Edge Triggered Interrupt Driven Program

Concepts:

- Using the HCS12 **edge-triggered interrupts**

Objectives:

- Use the HCS12 I/O ports connected to the Dragon12+ board.
- Implement **edge-triggered interrupts** for pushbuttons.

Files Needed:

- Lab8.zip

Assignment:

Write an interrupt-driven program that displays a number (in the range of 0000 - 9999) on the four 7-segment LEDs. You will use the main program as a loop to cycle through the 7-segment LEDs. The pushbutton is a valid interrupt at any point, so the interrupt for the pushbuttons should be enabled initially. An interrupt-driven program implements the following requirements.

1. The 7-segment LEDs should display “0000” and not increment when the program begins.
2. When Pushbutton #0 is pressed, the LED display should increment by 1.
3. When Pushbutton #1 is pressed, the LED display should decrement by 1.
4. When Pushbutton #2 is pressed, the display should reset to “0000”.

I. Data declaration:

a) **The Number to display:** The BCD number displayed on the four 7-segment LEDs are defined by using an assembly directive ‘DS.B’ in the data section:

```
; The memory allocation for the BCD number. Four bytes are allocated.

BCDNum:  DS.B  4
```

b) **PORT P Data:** When displaying the BCD digits onto the four 7-segment LEDs, only one 7-segment LED is activated at a time. To enable one 7-segment LED display at a time, only one bit in P0 to P3 in PORT P should be set to zero, and the other bits are ones. As shown in Figure 1, Port B's bits (pins) are connected to *all four 7-segment LEDS* and 8 flashing LEDs in the HCS 12+ board. **This means it cannot output two different patterns on two 7-segment displays at the same time. A time-multiplexing technique is used to display multiple digits simultaneously. Repeatedly, enable one 7-segment LED display for a short period (ex: 1 ms) and then disable this display to enable the other 7-segment LED.**

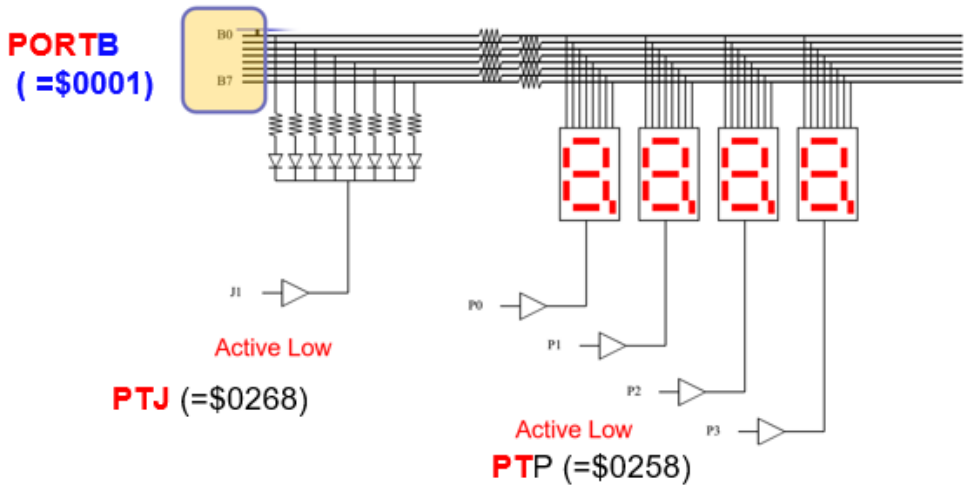


Fig. 1. Four 7-segment LEDs and eight flashing LEDs are connected to PORT B.

```
;PORT P data to enable LED#0 -  LED #3
PORTP_Data:  DC.B %00001110, %00001101, %00001011, %00000111
```

c) **Data for PORT B to display digit on the 7-segment LED**

The table below shows how to set bits in PORT B to display the digit from 0-9.

Decimal digit	Segments 6 5 4 3 2 1 0	Value to send to port B
0	0 1 1 1 1 1 1	$3F
1	0 0 0 0 1 1 0	$06
2	1 0 1 1 0 1 1	$5B
3	1 0 0 1 1 1 1	$4F
4	1 1 0 0 1 1 0	$66
5	1 1 0 1 1 0 1	$6D
6	1 1 1 1 1 0 1	$7D
7	0 0 0 0 1 1 1	$07
8	1 1 1 1 1 1 1	$7F
9	1 1 0 1 1 1 1	$6F

```
; The table contains the hex-value for PORT B to represent the digit 0-9. For example, $3F
will display the digit  '0 ' on the 7-seg LED, $6F will display the digit ' 9' on the 7-seg LED

LED_Data:   DC.B $3F, $06, $5B, $4F, $66, $6D, $7D, $07, $7F, $6F
```

d) **Interrupt Vector Address and Interrupt Vector:** Provide the beginning address of the Interrupt Service Routine (Interrupt Vector) to the Interrupt Vector address.

```
ORG     $FFCC          ; $FFCC is a Push button interrupt vector address
DC.W    ISR_PSHBUTTN   ; ISR_PSHBUTTN  is an interrupt vector
                       ; It is the beginning address of your interrupt service routine
```

II. Main Program:

Steps involved in the main task are defined as below:

1) Enable global interrupt system using CLI instruction

2) Configure I/O port using Data Directions Registers for PORT B, PORT P, PORT J, and Port H

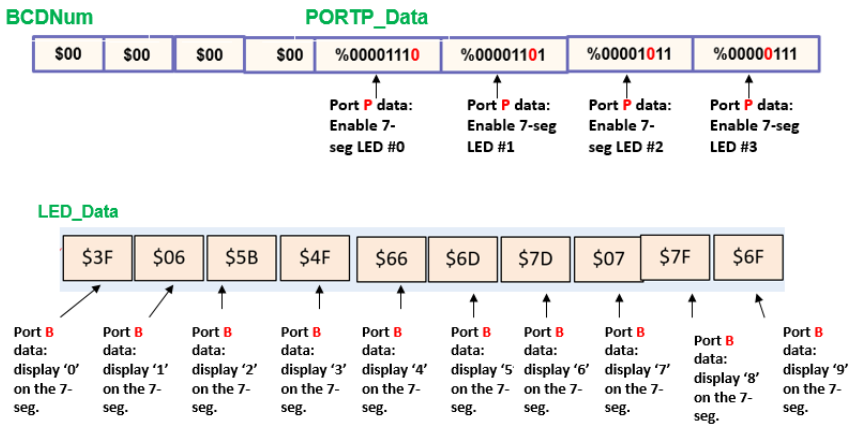
3) Disable 8 flashing LEDs by setting a value 1 in Bit 1 in PORT J.

4) **Enable Port H interrupts for Push Button 0, Button 1, and Button 2.**

5) Initialize the digits to display on the four 7-segment LEDs as '0000' by assigning '$00' to each BCD digit.

```
MOVB    #$00, BCDNum
MOVB    #$00, BCDNum+1
MOVB    #$00, BCDNum+2
MOVB    #$00, BCDNum+3
```

The data for the -7-segment LED display is declared ad below:



6) Loop forever: Use your Lab 7 code to display the BCD digits on the four 7-segment LEDs.

```
Forever:  LDX    #BCDNum        ;Register X holds the beginning address of BCD number
          LDY    #LED_Data

LEDLoop:  LDAA   0,X
          LDAB   A,Y
          STAB   PORTB

          MOVB   4,X, PTP

          PSHY
          LDY    #1
          JSR    Delay_Yms
          PULY

          INX
          CPX    #BCDNum+4
          BLO    LEDLoop
          BRA    Forever
```

III. Interrupt Service Routine:

ISR_PSHBUTTN: ; this is the Interrupt vector=the beginning address of your ISR

Button0: ; Module when Push Button #0 is pressed

a) You need to find out which button is pressed by checking the interrupt flag bit in **PIFH**.
Check Bit #0 in **PIFH** to determine whether Push Button #0 is pressed or not.
If the Push Button #0 is not pressed, branch to **CHKButton1** module else continue.

```
BRCLR  PIFH,  %00000001, CHKButton1
```

b) De-bouncing the signal for 20 ms.

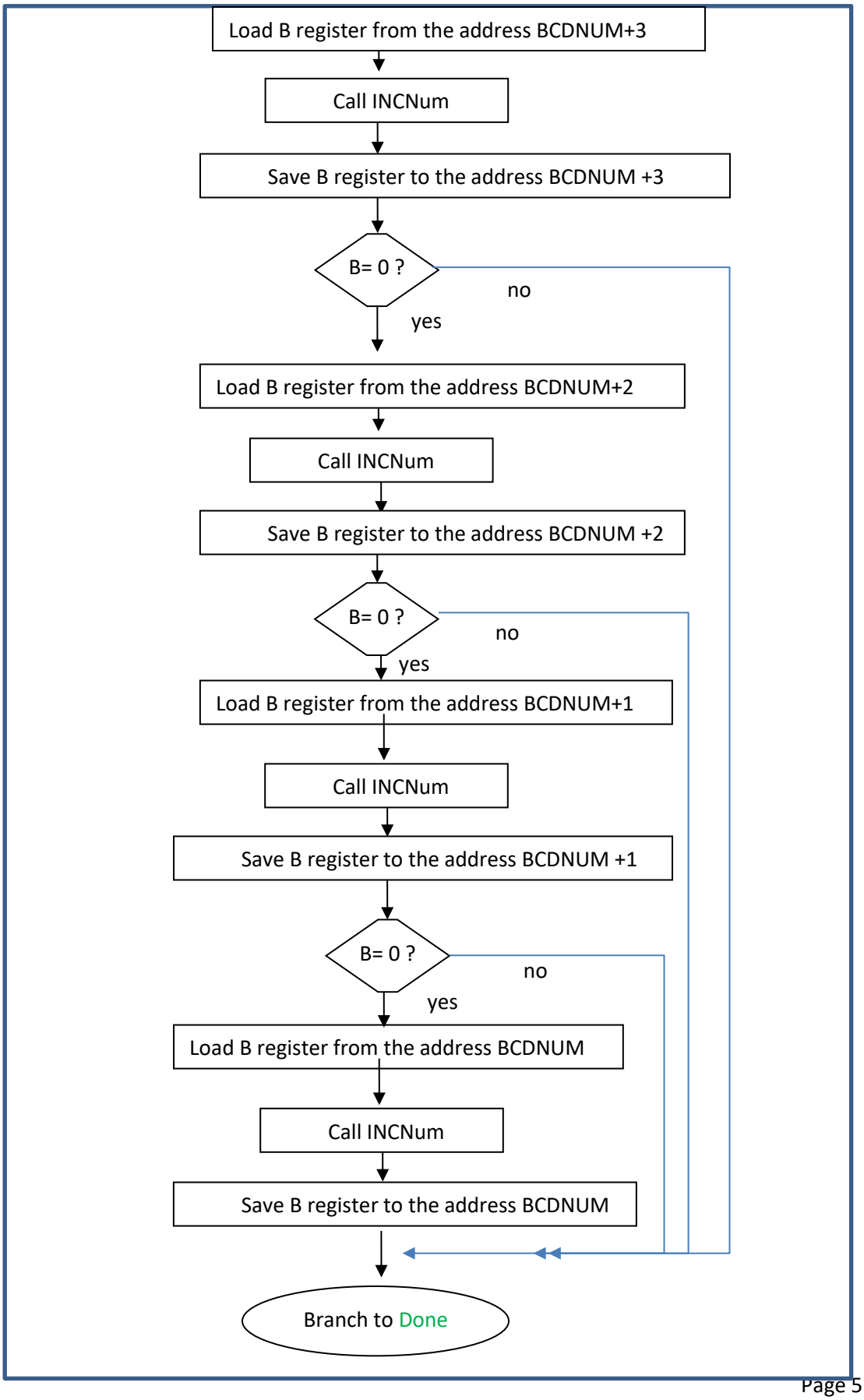
-Load the amount of the delay into register Y, call the subroutine, Delay_Yms

c) Clear Button 0 interrupt flag bit in PIFH by writing 1 to Bit 0

```
BSET   PIFH, %00000001
```

d) Increment the number **starting from the least significant byte** and branch to **Done**. The Flowchart for the 4- digit number increment is presented in Fig. 2. Implement the code to increment the number by one by following the Flowchart in Fig. 2.

Fig. 2. Flowchart for increment of the 4- digit number



CHKButton1: **; Module when Push Button #1 is pressed**

a) Check the interrupt flag bit for push BUTTON #1 in **PIFH**.
If the push Button #1 is not pressed, branch to **CHKButton2** else continue.

```
BRCLR  PIFH,  %00000010, CHKButton2
```

b) Debouncing Pushbutton signals for 20ms.

-Load the amount of the delay into register Y, call the subroutine Delay_Yms

c) Clear Interrupt flag Bit 1 for Button #1 in PIFH by writing 1 .

```
BSET   PIFH,  %00000010
```

d) Decrement the number on the four 7-segment LEDs by one and branch to **Done**. The Flowchart for the 4- digit number decrement is presented in Fig. 3. Implement the code to decrement the number by one by following the Flowchart in Fig. 3.

CHKButton2:

a) Debouncing Pushbutton signals for 20ms.

Load the amount of the delay into register Y, call the subroutine, Delay_Yms

b) Clear Interrupt flag bit Bit#2 for Button #2 in PIFH by writing 1

```
BSET   PIFH,  %00000100
```

c) Clear the BCD digits for four 7-segment LEDs

```
Done:    RTI        ;end of the interrupt service routine
```

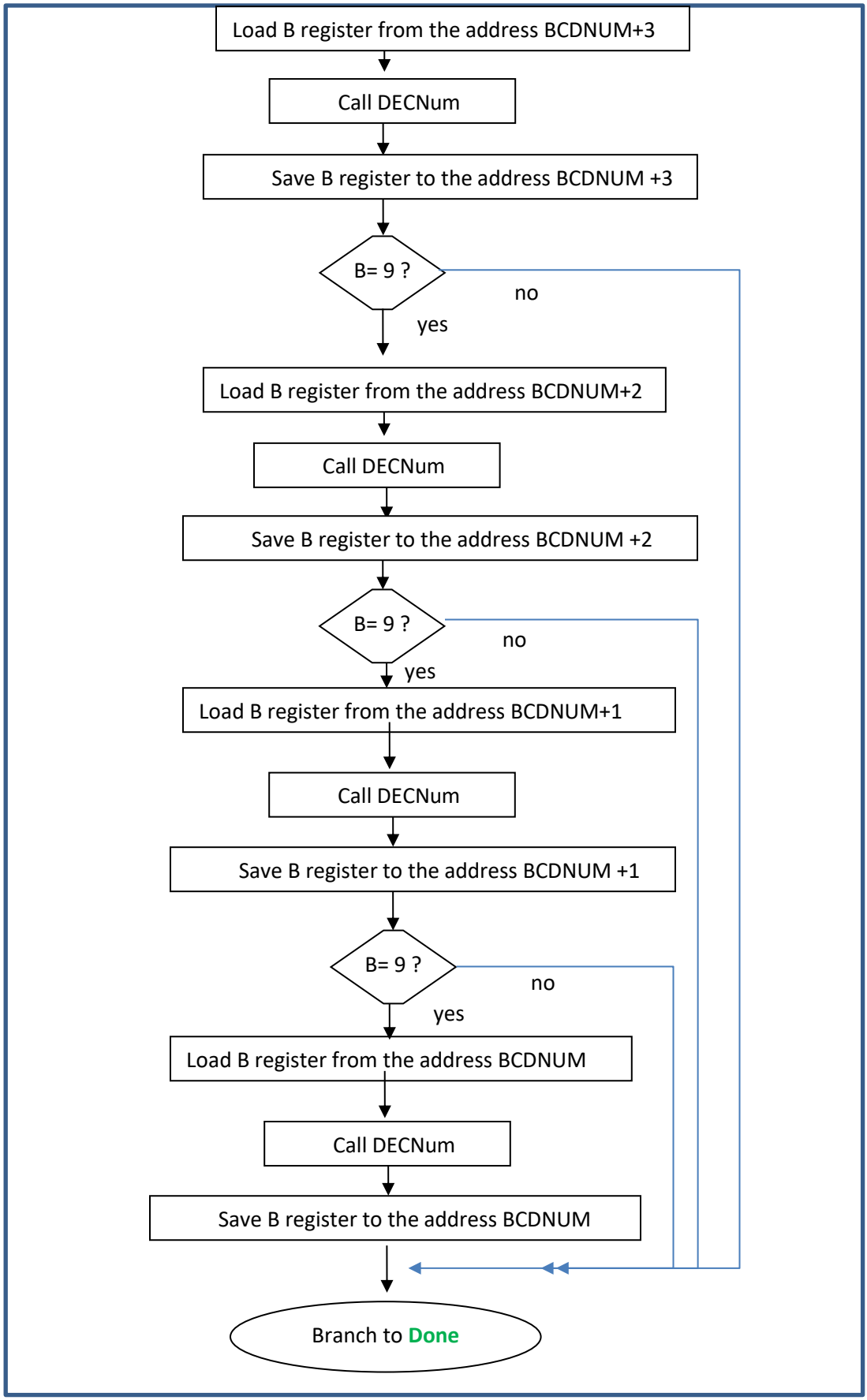
IV. Subroutines:

The subroutines for increment and decrement of the digit (0-9) are provided as below:

```
;*****************************************************
; Input: B: BCD number,    ; Output: B: (B)+1 if (B) < 9, else B=0
;*****************************************************
INCNum:          INCB
                 CMPB    #9
                 BHI      CARRY1
                 BRA     END_SUBI
CARRY1:          LDAB    #0
END_SUBI :   RTS

;*************************************************
; Input: B: BCD number,    ; Output: B: (B)-1 if (B) > 0 else B= 9
;*************************************************
DECNum       CMPB          #0
              BLS    BORROW1
              DECB
             BRA   END_SUBD
BORROW1:    LDAB  #9
 END_SUBD:   RTS
```

Fig. 3. Flowchart for decrement of the 4- digit number


Load B register from the address BCDNUM+3
Call DECNum
Save B register to the address BCDNUM +3
B= 9 ?
no
yes
Load B register from the address BCDNUM+2
Call DECNum
Save B register to the address BCDNUM +2
B= 9 ?
no
yes
Load B register from the address BCDNUM+1
Call DECNum
Save B register to the address BCDNUM +1
B= 9 ?
no
yes
Load B register from the address BCDNUM
Call DECNum
Save B register to the address BCDNUM
Branch to Done

Task1: Component Identification: Identify the components (the peripheral modules on the HCS12 microcontroller) that you need to solve this problem. Please be as specific as possible; e.g., if you need 8 pins of PORT B, mention it here along with their directions. Also, briefly explain what they are used for, e.g., PTJ enables or disables eight flashing LEDs. Also, show any data declaration and initialization that you need for the initialization of your system.

Task2:De-bouncing Solution: A common problem in mechanical switches (or buttons) is bouncing. When a pushbutton is pressed or released, the actual signal bounces for a short period. Explain how you solve the bouncing problem. Generate the demo video clip that includes the interrupts events by Pushbutton #0, Pushbutton #1, and Pushbutton #2.

Task3: To improve your program readability, add comments appropriately in your program

What to Submit through Blackboard

1) **Per each member: 50pts**

   Complete Lab8_WorkBook.doc :

   - **(20 pts) Task1: Component Identification**
   - **(30 pts) Task2: De-bouncing solution**

2) **One copy per group : 100 pts**

- **50 pts: Video demo file:**
  - Lab8 _Student1 Lastname_Student2 Lastname.mp4
- **40 pts:** The assembly source code main.asm
- **10 pts:** Your program styling and readability.

Note: Lab8 file name convention:

- Individual file: Lab8_Student_Firstnname_Lastname.pdf
  Ex) Lab8_Smith_Green.pdf

- **Group file:**
  Source file: main.asm
  Video file: Lab8 _Student1 Lastname_Student2 Lastname.mp4