Lab 4: Parallel Input / Output

ECE-3731

INTRODUCTION

Parallel input/output ports

The HCS12 has numerous parallel I/O ports. For this lab we are mainly concerned with PORTB (has LED's) and PORTH. The ports are eight-bit ports; where each bit corresponds to a pin on the HC12 board itself. The pins use TTL logic levels (logical 0 = 0 volts, logical 1 = 5 volts). Each port is memory mapped. (All I/O in the HCS12 is controlled by a set of registers reside in locations $0000-$03FF.) Simply accessing a memory location can access a memory-mapped port. That is, each port is assigned a memory address. This makes it easy to work with parallel ports, since we do not have to use special instructions to access them. The same instructions for accessing memory can access I/O ports. The use of the parallel ports is explained in the course lecture notes. In general, any bit on a General Purpose I/O port can be an input or an output. Freescale uses the concept of Data Direction. An I/O port has a corresponding data direction register. E.g. PORTB has data direction register DDRB. Each bit in DDRB controls the direction of its corresponding bit in PORTB. The convention used is 1 for output and 0 for input. E.g. DDRB = $0F sets the upper four bits of PORTB to inputs and the lower four bits to outputs.

PORT B (PORTB)
This is a General Purpose I/O (GPIO) register located at $0001.
(We do not need to write an equate for this because it is defined in an include file in your Codewarrior Project.) On the Board, it is connected to the LEDs.

PORT H (PTH)
This is a General Purpose I/O (GPIO) register located at $0260. On the Board, it is connected to the DIP Switches and Pushbuttons (the last 4 bits). To use the DIP Switch and the Pushbuttons on PTH all of the bits must be inputs.

NOTE: Port H always has switches on it and therefore all Port H bits must always be inputs.
Upon reset all Ports are set to inputs.
So to avoid problems, simply leave the data direction for Port H (PTH) alone.
Writing to Port H with switches closed and data direction set to output will damage your board.

PORT P (PTP)
This is a General Purpose I/O (GPIO) register located at $0258. In the Board, the bits 3 to 0 are connected to the cathodes of the 7-segment displays. The bits 4 to 6 are connected to the RGB LED. To use it with the 7-segment display and the RGB LED,

we must configure PTP so that the bits are outputs. This is done via the DDRP register located at $025A.

PORTM (PTM)
We will only use Port M , bit 2 to control the common anode of the RGB LED display.

Writing a value to PORTB is quite straightforward:
For example, if we want to light up all eight LED's we write:

```
    ldaa   #%11111111   ; all 8 LED's lit
    staa   PORTB        ; light the LED's
```

alternatively we could use:

```
    movb   #$FF, PORTB
```

We can also read back the value that is stored on PORTB and modify it if we wish.
For example, to <u>toggle</u> the leftmost bit on PORTB we can write:

```
   ldaa   PORTB   ; read value on LED's into A register
   eora   #%10000000  ; use bitwise exclusive OR to toggle the leftmost bit in A reg.
   staa   PORTB   ; write the modified value back into PORTB
```

Pre-Lab Questions:

1) Write a program which inputs a character from user, if the user enters a character ‘A” from the keyboard. The value of the 4-bit counter displayed on PORTB (LED's) is incremented by one. When a value of four ones is reached (1111), the counter rolls over to zero (0000).

2) Draw the flowchart of the sample program LAB4ASM provided on Canvas.

Initialization:

Due to complexities involved with your particular board, you must initialize PORTB first before reading or writing to it. I have written a subroutine that does this for you and placed it in an include file in your project.
To initialize PORTB simply write:

```
        jsr    led_enable
```

Setting Bits:
Sometimes we want to set one or more bit positions (in a memory operand) while leaving the others alone.
e.g. To set the lower two bits of PORTB we write:

```
      bset   PORTB,  %00000011  ; sets bits for the "1" valued bits in PORTB
                                ;  …. (in this case the lower two bits)
                                ; … leaving the others unchanged
```

Clearing Bits:
Similarly, sometimes we want to clear one or more bit positions (in a memory operand) while leaving the others alone.

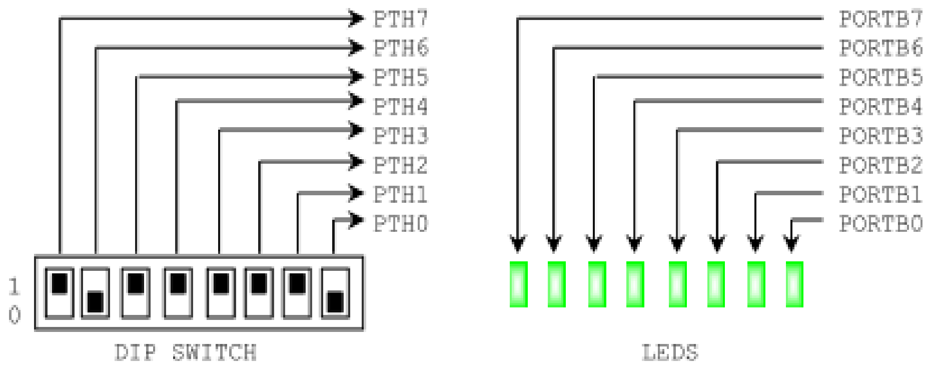
e.g. To clear the lower two bits of PORTB we write:

```
      bclr   PORTB,  %00000011  ; clears bits for the "1" valued bits in PORTB
                                ;  …. (in this case the lower two bits)
                                ; … leaving the others unchanged
```

Example : Read DIP Switches and place the result on the LEDs

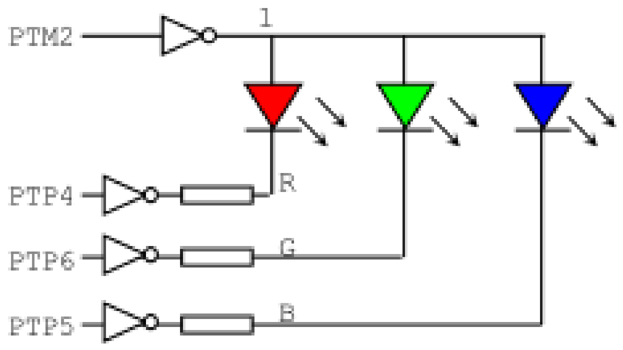
Pseudo -code:

```
led_enable();
 while (1)  {
  A = PTH
  PORTB = A }
```



Using the RGB LED Display:

PORTP4 = RED LED cathode, PORTP5 = BLUE LED cathode, PORTP6 = GREEN LED cathode. The anodes of these 3 LEDs are connected to PTM2. All the pins that control the LEDs are in negative logic (Active Low).



<u>Example</u>: Flash RED LED then GREEN LED repeatedly

```
RED:    EQU   $10   ; PP4
BLUE:   EQU   $20   ; PP5
GREEN:  EQU   $40   ; PP6

        bset   DDRP,  RED+GREEN+BLUE  ; make pp4-pp6  outputs
        bset   DDRM, 4    ; make PM2 an output
        bclr   PTM, 4   ; make PM2 low to enable RGB
LOOP1
        bclr   PTP,  RED+GREEN+BLUE  ; clear all
        bset   PTP,  RED   ; turn on red
; delay for 1 second
        ldd    #1000      ; 1000ms delay
        jsr    ms_delay
        bclr   PTP,  RED+GREEN+BLUE  ; clear all
        bset   PTP,  GREEN   ; turn on green
; delay for 1 second
        ldd    #1000      ; 1000ms delay
        jsr    ms_delay
        jmp    LOOP1  ; repeat endlessly
```

ASSIGNMENT

1. Bit Manipulation, Voting Logic - Pushbuttons, LED's:

To be done in Assembly

This program will receive inputs from pushbuttons. The board has four push buttons on PORTH that you can use:
PTH0, PTH1, PTH2 and PTH3. (Please note the switches on the red DIP switch block must be up in the lower four-bit positions for the pushbutton switches to work.)
Your program should manipulate the value on the LED's based on user input.
Your program should do the following depending on the user input.:

i. User presses PTH3: The program should load a "random" value display it on PORTB

ii. User presses PTH2: The program should logically shift left the bit positions by one position of PORTB and flash blue LED 3 times with 1 second on time and 1 second off time.

iii. User presses PTH1: The program should arithmetic shift right the bits by one position of PORTB

iv. User presses PTH0:
In this case the program implements **5-input majority voting logic**:
Reads the switches on PORT H (assuming that the 5 leftmost bits have their values set through the DIP switches).
The value of the digital number represented by the five leftmost bits on Port H is tested. If the number of logic '1' bits (in the 5 leftmost bits) is greater than the number of logic '0' bits, the program lights the GREEN LED on the RGB (while the other LED's (colors) on the RGB are off). Otherwise, the RED LED on the RGB is turned on (while the other LED's on the RGB are off).

Note:

1) The program should pause for approximately 200 milliseconds between checking user inputs. This is for debouncing delay.
2) There is a subroutine "RAND" provided in Sample Project. This subroutine generates a random number and stores it in variable "SEED".

Post-Lab Question:

Demo all the 4 questions of homework 5 to lab instructor