**Khalifa University of Science, Technology and Research**
**Electronic Engineering Department**

**ELCE333Microprocessor Systems laboratory**

# Laboratory Experiment 3

# HCS12 INPUT AND OUTPUT PORTS

**Lab Partners**
Dana Bazazeh, 100038654
Moza Al Ali 100038604

**Date Experiment Performed**: 11/02/2015
**Date Lab Report Submitted**: 18/02/2015

**Lab Instructor:**Dr.Mohamed Al Zaabi/ Dr.MahmoudKhonji

**SPRING 2015**

# Table of Contents

# List of Figures and Tables

## Figures:

## Summary

In this lab experiment four different tasks are provided in order to grain the maximum knowledge regarding the Dragon board. In the first task, students were asked to derive output lines through by using the LEDs and the 8 bits in accumulator A. Apart from that, they were asked to read input lines using Dip switches. In task 3, reading DIP switches+ writing them to LEDs was required from students. This is mainly the combination of both the previous tasks. Task 4 asked students to implement a delay using subroutines. In the end, aim of this laboratory experiment is fulfilled. Later in this report detailed explanation and analysis of each step in the tasks are provided.

# 1. Introduction

In this lab session students are mainly dealing with a M68HCS12 microcontroller. This contains many components imbedded within it.

Its basic branches consist of fully integrated suite of I/O capabilities which are parallel and serial I/O, analog input, and timer functions. Each pin in the microcontroller is responsible for a different function and all of them are done using 1024 control registers. These registers are initially located at memory locations $0000-$03FF.

Mainly, there are 3 components students should understand: parallel ports ,light-emitting diode and  dip-switches and push buttons.

First of all, parallel ports function is to operate in either output or input and are available as general-purpose I/O ports. Each port has a specific usage. There are nearly nine parallel ports which are: Port A, B, E, AD, H, J, K, P and port T.  This laboratory experiment will focus on the use of LEDs controlled by ports B, J and P and the Dip Switches and finally the push buttons controlled by port H.

Secondly, light emitting diodes that exist in the HCS12 board are actually the LEDs (eight light-emitting diodes) and 7-segment anodes that are both controlled by port B.

Moreover, the ports of the HCS12 board will only be turned on and used under some required conditions and those conditions will be shown in the tasks of this laboratory experiment.

**Aim:**

To introduce the students to the read and write data from the input and output ports and how delays can be implemented using loops.

**Objectives:**

1- Understand the microcontroller IO ports.

2- Configure the ports as inputs or output.

3- Read and write data from input and output ports.

4- To examine the DIP switches of PTH for I/O programming on Dragon12+ Board.

5- To do I/O bit programming in HCS12 Assembly language.

6- To create binary counter on Dragon12+ Board.

7- Download, run, and test code on a Dragon12+ Board.

# 2. Design and Results

## 2.1 Task1: Deriving Output Lines

This task allows us to modify the state of the 8 LEDs found on the microcontroller using the 8 bit accumulator A, where each bit represents one of the LEDs. An LED can either be ON(high bit 1) or OFF(low bit 0). In order to run the program code on the microcontroller, we have to make sure that we are in HCS12 Serial Monitor mode. The figure below shows the full code used.

```
        ORG $4000 ;Flash ROM address for Dragon12+
Entry:
        LDAA #$FF
        STAA DDRB ;Make PORTB output
        ;PTJ1 controls the LEDs connected to PORTB
        LDAA #$FF
        STAA DDRJ ;Make PORTJ output
        STAA DDRP ;Make PORTP output
        LDAA #$00
        STAA PTJ ;Turn off PTJ1 to allow the LEDs to show data
        LDAA #$0F
        STAA PTP ; Disable the 7-segment display
        ;-------Switch on LEDs connected to PORTB based on the Acc A value
        LDAA #$55
        STAA PORTB ;Store A into PORTB
BRA Entry
```

**Figure 1: Program of task1**

From the lab introduction section, we know that port B is used to control both the 8 LEDs and the 7 segment display. In order to set the LEDs to output mode, we need to set PORTB data direction register (DDRB) to $FF using the store instruction. We do the same for PORTJ and PORTP. In order to set the LEDs, we store the hexadecimal value $55 in PORTB and this is equivalent to 0101 0101 in binary. We run the program and notice that the LEDs turn on representing hexadecimal 55 in binary. Next, we comment the instruction STAA DDRP which means we did not initialize PORTP as output, the result of doing this is shown below in figure 2 which also shows the state of the LEDs holding value of accumulator A.

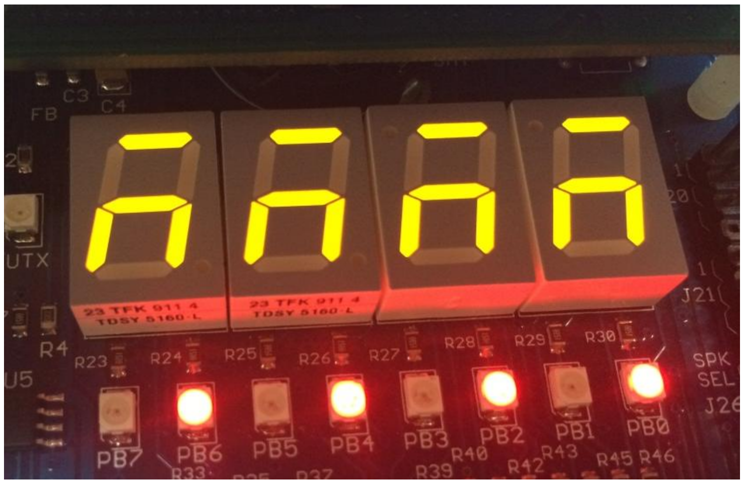


**Figure 2: LEDs representing output of Acc A=$55**

## 2.2 Task2: Reading Input Lines

In this task, we want to use the 8 Dip Switches as inputs where the value of the switch states is stored in accumulator A. If the switch is flipped upwards then it represents a binary value of 1 and if it's down it represents binary value 0. We know that the switches are controlled using port H (PTH). We use the code below:

```
        ORG $4000 ;Flash ROM address for Dragon12+
Entry:
        LDAA #$0
        STAA DDRH
        LDAA PTH
        BRA Entry
```

**Figure 3: code for reading from DIP Switches**

We can see from the code that we set the data direction register of PTH to $0 in order to set it to input mode. Then we load the value of PTH representing the switches state into accumulator A. We run the program and change the Dip Switches state and notice how this affects accumulator a value. The accumulator will store the binary values of the 8 switches in hexadecimal. Figure 4 below shows the state of the switches (00001111) while figure 5 shows the value of accumulator A at that time which is in hex.

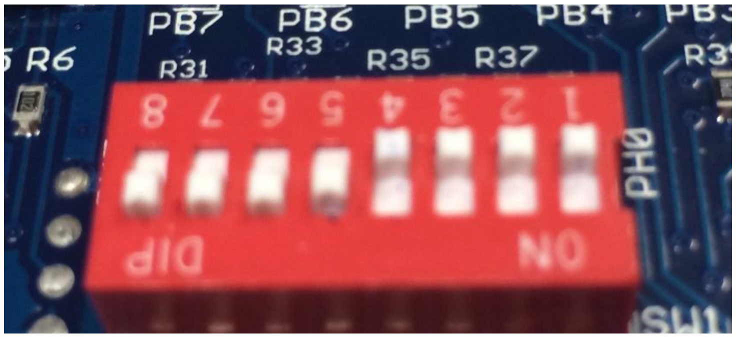
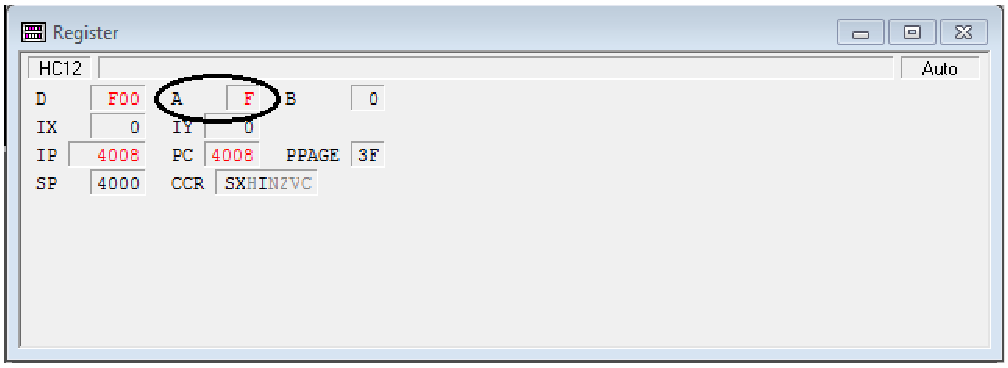


**Figure 4: Dip Switches states (00001111)**



```
D    F00   A      F  B      0
IX     0   IY     0
IP  4008   PC  4008  PPAGE 3F
SP  4000   CCR SXHINZVC
```

**Figure 5: Output of AccA when input as in Figure 4**

Now we must modify the code to read only bit 4 of PTH. This can be done by masking accumulator A after loading the value of PTH. This can be done by ANDing the contents of accumulator A with the immediate value #$04 and storing the result in A. The code is modified by inserting the instruction ANDA #$04 after LDAA PTH. Figure 6 below show the effect of this modification where the switches are the same as in figure 4 above. Instead of reading 1111 from the switches, the accumulator value is (1111)AND(0100) which gives $04 as seen stored in accumulator A.

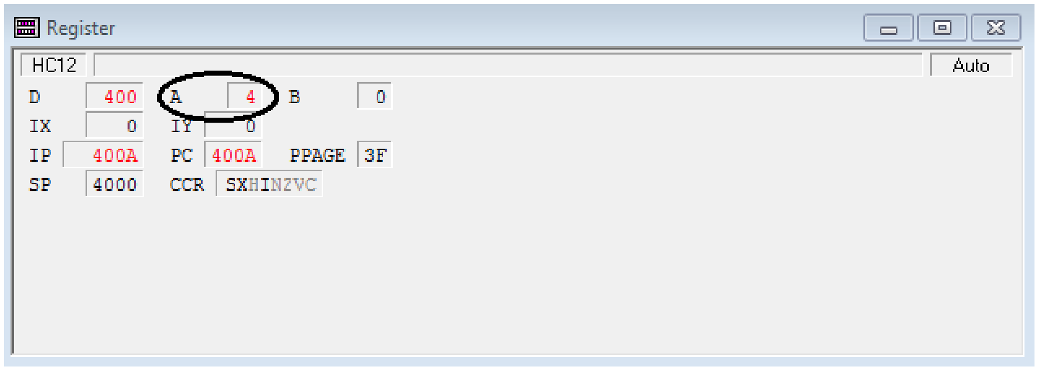

```
HC12                                   Auto
D    400     A     4   B      0
IX     0     IY    0
IP  400A     PC 400A   PPAGE 3F
SP  4000     CCR SXHINZVC
```

**Figure 6: Output of Acc A after masking bit 4**

## 2.3 Task3: Reading Dip Switches and Writing Them to LEDs

This task asks us to write a program that will allow the 8 Dip Switches to be continuously read and their state represented as output on the 8 LEDs. This can be achieved by combining the codes from task 1 and 2 together and modifying it a little. The combined code is shown below in figure 7.

```
 ORG $4000 ;Flash ROM address for Dragon12+
Entry:
  LDAA #$FF
  STAA DDRB ;Make PORTB output
;PTJ1 controls the LEDs connected to PORTB
  LDAA #$FF
  STAA DDRJ ;Make PORTJ output
  STAA DDRP ;Make PORTP output
  LDAA #$00
  STAA PTJ ;Turn off PTJ1 to allow the LEDs to show data
  LDAA #$0F
  STAA PTP ; Disable the 7-segment display

  LDAA #$0
  STAA DDRH
  LDAA PTH
  STAA PORTB
  BRA Entry
```

**Figure 7: reading DIP switches and writing to LEDs code**

Here, we set PORTB, PORTJ and PORTP as outputs by setting the value of their data direction registers to $FF. Next, we store $00 in PRJ1 to allow the LEDs show the data and disable the 7 segment since we do not need by setting the value of PTP to $0F.

Now that the LED outputs are set, we set the Dip Switches as inputs by setting the value of Port H data direction registers(DDRH) to $00 and then storing the value of PTH which represents the switches states in accumulator A. Finally, we store this value in PORTB which in turn modifies the LEDs representing the state of switches. In order to read and store the switches state continuously, we need to use branch always instruction BRA and repeat this whole read/store cycle.

After executing the program, figure 8below show the LEDs representing the current switches state.

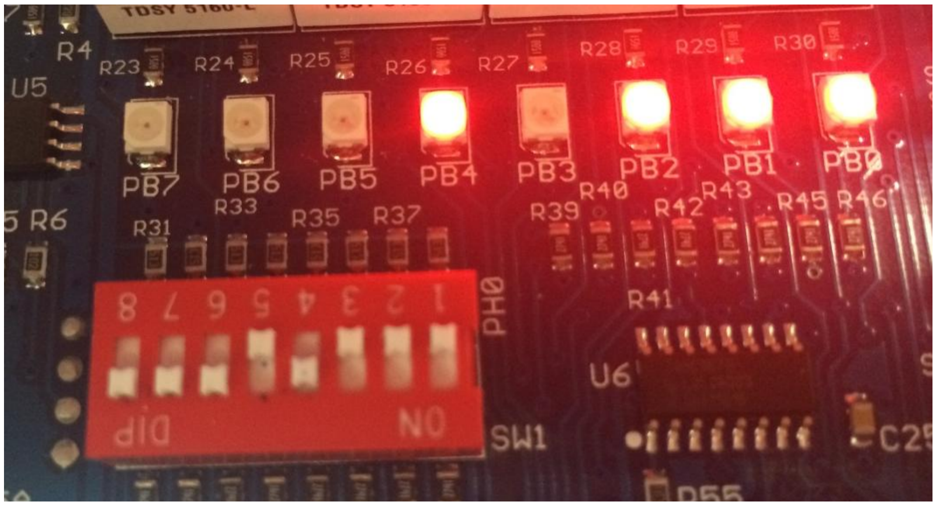


**Figure 8: input from dip switches shown in LEDs**

## 2.4 Task4: Using Subroutine to Implement Delay

This task is similar to task 3; the difference here is that we should introduce a delay between reading the input switches and representing their value on the LEDs. We can do this using a subroutine. A delay subroutine is provided in the lab script.

This delay subroutine uses 3 nested loops along with counters to execute a large number of instructions causing a specific time delay before moving on to store the DIP switches state to the LEDs. Therefore, we need to execute this subroutine between the reading of the switches state and writing to PORTB (LEDs). This can be done using jump instruction JSR.

The complete code is shown below in figure9.

```
INCLUDE 'mc9s12dg256.inc'

 ORG $4000 ;Flash ROM address for Dragon12+
Entry:
  LDAA #$FF
  STAA DDRB ;Make PORTB output
;PTJ1 controls the LEDs connected to PORTB
  LDAA #$FF
  STAA DDRJ ;Make PORTJ output
  STAA DDRP ;Make PORTP output
  LDAA #$00
  STAA PTJ ;Turn off PTJ1 to allow the LEDs to show data
  LDAA #$0F
  STAA PTP ; Disable the 7-segment display
;-------Switch on LEDs connected to PORTB based on the Acc A value

  LDAA #$0
  STAA DDRH
  LDAA PTH
  JSR DELAY
R1 EQU $1000
R2 EQU $1001
R3 EQU $1002
DELAY:

PSHA ;SaveReg A on Stack
  LDAA #100 ;Change this value to see
  STAA R3 ;how fast LEDs shows data coming from PTH DIP switches
;--10 msec delay. The Serial Monitor works at speed of 48MHz with
XTAL=8MHz on Dragon12+ board
;Freq. for Instruction Clock Cycle is 24MHz (1/2 of 48Mhz).
;(1/24MHz) x 10 Clk x240x100=10 msec. Overheads are excluded in this
calculation.
    LDAA #10
    STAA R4
L3 LDAA #100
    STAA R2
L2 LDAA #240
    STAA R1
```

```
L1 NOP ;1IntructionClk Cycle
NOP ;1
NOP ;1
  DEC R1 ;4
  BNE L1 ;3
  DEC R2 ;TotalInstr.Clk=10
  BNE L2
  DEC R3
  BNE L3
;--------------
PULA ;RestoreReg A
  RTS

  STAA PORTB
  BRA Entry
```

**Figure 9: reading from switches and writing to LEDs with delay code**

In the code above, we see that the delay subroutine is executed by using JSR DELAY just before storing the switches values to PORTB. In the Delay subroutine, we initialize 3 counters R1, R2 and R3. These counters determine how many times each loop will execute and is used by decrementing counters and branching to repeat when counter is not yet 0. The NOP instruction found in the inner most loop L1 is a no operation instruction that does not do anything and takes 1 clk cycle to execute. It is used to increase delay time. The inner loop L1 is the one executed R1*R2*R3 times. We can increase the delay time by either increasing one of these 3 values or by inserting more instructions such as NOP into the inner loop.  Using the delay formula provided, we can approximately calculate the delay time as follows:

$$Delay = \frac{1}{24MHz} xR1xR2xR3$$

$$Delay = \frac{1}{24MHz} x240x100x10 = 10msec$$

A clearer description of the delay subroutine is provided using the flow chart in figure 10 below:

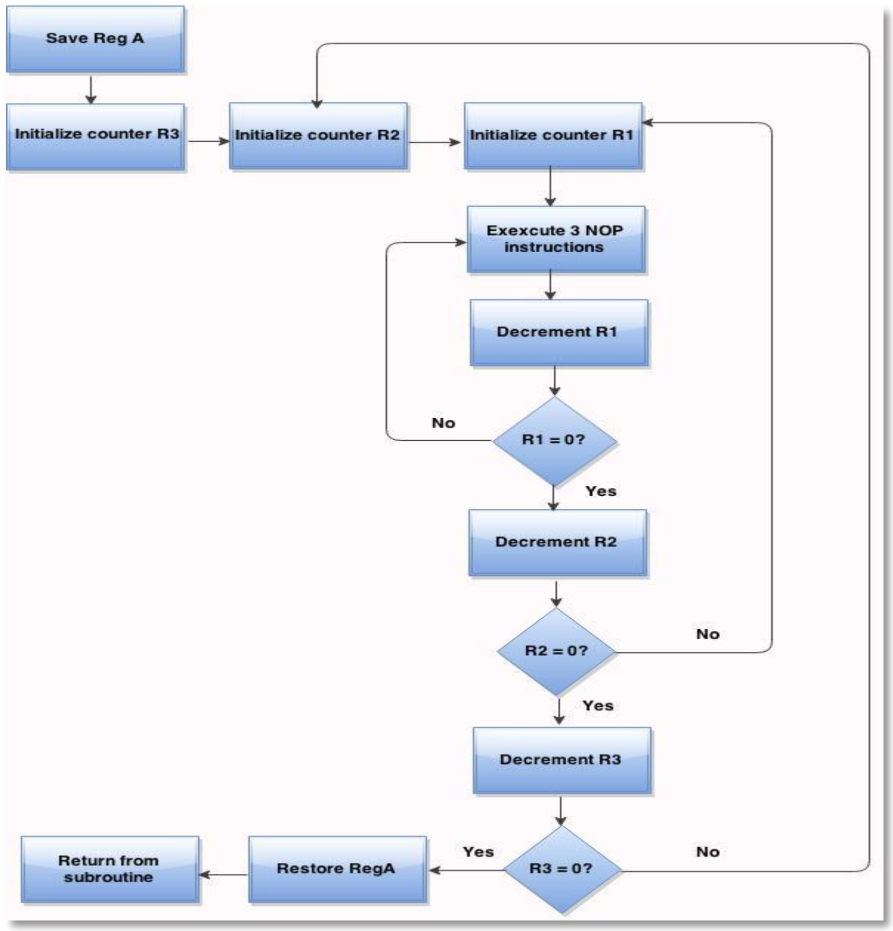


**Figure 10: Delay subroutine flow chart**

Finally, we execute the program and count the time between setting the input using the switches and displaying the output on the LEDs. We then compared the implementation delay with the calculated one, and it turned out to be very close, since our calculation is an approximation. Figure 11 below shows a case where switch 3 is set but the LED 3 is still OFF representing a delay

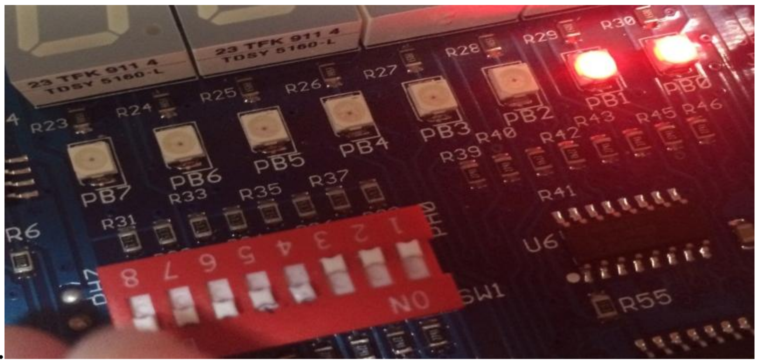
.

**Figure 11: showing delay in writing switch 3 to LED 3**

## 3. Assignment Questions

**1) Create a program that implements a binary counter. Use the delay subroutine given in Task-4 to display the counting sequence on the LEDs connected to PORTB.**

```
ORG $4000 ;Flash ROM address for Dragon12+
R1 EQU $1000
R2 EQU $1001
R3 EQU $1002
Entry:
  LDAA #$FF
  STAA DDRB ;Make PORTB output
;PTJ1 controls the LEDs connected to PORTB
  LDAA #$FF
  STAA DDRJ ;Make PORTJ output
  STAA DDRP ;Make PORTP output
  LDAA #$00
  STAA PTJ ;Turn off PTJ1 to allow the LEDs to show data
  LDAA #$0F
  STAA PTP ; Disable the 7-segment display
;-------Switch on LEDs connected to PORTB based on the Acc A value
  CLRA
  COUNT:
        INCA
        JSR DELAY
        STAA PORTB
        BRA COUNT
DELAY:
PSHA ;SaveReg A on Stack
  LDAA #100 ;Change this value to see
  STAA R3
  LDAA #10
  STAA R3
L3 LDAA #100
   STAA R2
L2 LDAA #240
   STAA R1
L1 NOP ;1IntructionClk Cycle
NOP ;1
NOP ;1
   DEC R1 ;4
   BNE L1 ;3
   DEC R2 ;TotalInstr.Clk=10
   BNE L2
   DEC R3
   BNE L3
;--------------
PULA ;RestoreReg A
  RTS
```

**Figure 12: Binary counter code**

As can be seen in figure 12 above, the only difference between this program and the one in task 4 is that here we do not use PTH Dip switches as input but instead we replace it with a counter loop that increments accumulator A, which is initialized with 0 using CLRA and store the value of the accumulator into PORTB allowing it to be displayed on the LEDs. However, just before storing the accumulator value, we jump to the delay subroutine used earlier to create some time delay so that the counter is not too fast.

**2) Create a program that flash all the LED's with a delay of 0.1 sec in between**

```
ORG $4000 ;Flash ROM address for Dragon12+

R1 EQU $1000
R2 EQU $1001
R3 EQU $1002
Entry:
  LDAA #$FF
  STAA DDRB ;Make PORTB output
;PTJ1 controls the LEDs connected to PORTB
  LDAA #$FF
  STAA DDRJ ;Make PORTJ output
  STAA DDRP ;Make PORTP output
  LDAA #$00
  STAA PTJ ;Turn off PTJ1 to allow the LEDs to show data
  LDAA #$0F
  STAA PTP ; Disable the 7-segment display
;-------Switch on LEDs connected to PORTB based on the Acc A value

REPEAT:
LDAA #%00000001
  STAA PORTB
  JSR DELAY

COUNT
   LSLA
   CMPA #$0
   BEQ REPEAT
   STAA PORTB
   JSR DELAY
   BRA COUNT
```

```
DELAY
PSHA ;SaveReg A on Stack
  LDAA #100 ;Change this value to see
  STAA R3
L3 LDAA #100
   STAA R2
L2 LDAA #240
   STAA R1
L1 NOP ;1IntructionClk Cycle
NOP ;1
NOP ;1
   DEC R1 ;4
   BNE L1 ;3
   DEC R2 ;TotalInstr.Clk=10
   BNE L2
   DEC R3
   BNE L3
;--------------
PULA ;RestoreReg A
  RTS
```

**Figure 13: code to flash all the LED's with a delay of 0.1 sec**

In this question, we first initialized the accumulator A with a binary value 00000001 and then used the instruction LSLA to logically shift the content of A to the left. This will create multiples of 2, where we will have 1 high bit and 7 low bits. When we store theses accumulator values, only one LED will light turn ON at a time, moving from the last LED on the right up to the first LED on the left. After each shift and store, the program will jump to the delay subroutine. In the delay subroutine, we change the value of R3 from 10 to 100 to get a delay of 0.1 secs as found using the delay formula. We made the shifting and storing loop in another loop such that when the shift bit 1 reaches the right most bit causing the accumulator A to have value of $0, we re-insert the shift bit 1 by going to the REPEAT loop and re-initializing the accumulator A.

**3) Re-examine the toggle program in assignment question 2 which flashes the LEDs of PORTB with 0.1 sec delay. Now, modify that program to get the byte of data from PTH switches and give it to R3 register of the DELAY loop. Run the program to show how you can set the time delay size using the PTH switches.**

```
ORG $4000 ;Flash ROM address for Dragon12+

R1 EQU $1000
R2 EQU $1001
R3 EQU $1002
Entry:
  LDAA #$FF
  STAA DDRB ;Make PORTB output
;PTJ1 controls the LEDs connected to PORTB
  LDAA #$FF
  STAA DDRJ ;Make PORTJ output
  STAA DDRP ;Make PORTP output
  LDAA #$00
  STAA PTJ ;Turn off PTJ1 to allow the LEDs to show data
  LDAA #$0F
  STAA PTP ; Disable the 7-segment display
;-------Switch on LEDs connected to PORTB based on the Acc A value
  LDAA #$0
  STAA DDRH

REPEAT:
  LDAA #%00000001
  STAA PORTB
  LDAB PTH
  JSR DELAY

COUNT
   LSLA
   CMPA #$0
   BEQ REPEAT
   STAA PORTB
   LDAB PTH
   JSR DELAY
   BRA COUNT
```

```
DELAY

PSHA ;SaveReg A on Stack
  STAB R3
L3 LDAA #100
   STAA R2
L2 LDAA #240
   STAA R1
L1 NOP ;1IntructionClk Cycle
NOP ;1
NOP ;1
   DEC R1 ;4
   BNE L1 ;3
   DEC R2 ;TotalInstr.Clk=10
   BNE L2
   DEC R3
   BNE L3
;--------------
PULA ;RestoreReg A
  RTS
```

**Figure 14: code to read delay byte from DIP Switches**

In this question, we modify the code from the previous question so that the value of the delay byte R3 is taken from the DIP Switches. To do so, we must set PTH to input mode by loading the value $00 to DDRH. Next we use the accumulator B to store the value of the DIP Switches before every Delay subroutine. Finally, we modify the delay subroutine by using STAB R3 instead of the previous pre-defined value of R3.

## 4. Analysis and Interruption

This lab session was a bit different from what students used to do in their previous sessions. That’s because now they are simulating the program able code to the hardware and the hardware is the microcontroller board in our case. Students after this lab experiment got an idea of the usage of ports to control LEDs. They also came to know that and PTJ should be turned off with the use of Accumulator A. Apart from that, Dip switches also used as inputs in the second task and it was controlled using PTH. In the third task, codes of task 1 & 2 are combined together so that the dip switches are used as inputs and the LEDs as outputs. Finally, a subroutine of a delay in order to use it in other codes and it showed how the delay was created between the LEDs.

## 5. Conclusion and Recommendations

This session contains 4 tasks that requires and good understanding of the use of the Dragon12 board. In this experiment students understood the use of different ports on the board. Students moreover learned how to use Dip switches as inputs and the LEDs lights as output. Finally the implementation of delay was the major part in task 4.This had a precise code to add on each program in order to see the effect on the output. After the completion of this lab students built a solid knowledge about the usage of the board and they believe that all the objectives are met.