LABORATORY EXPERIMENT 1

LABORATORY HARDWARE AND SOFTWARE FOR THE DEVELOPMENT OF TMS320LF2407 DSP-BASED SYSTEMS

Objectives

The objective of this lab is to introduce the students to the hardware and software used in the laboratory for developing TMS320LF2407 DSP-based systems. The students will learn about the TMS320LF2407 evaluation module (EVM), and familiarize themselves with the various development software required. At the end of this lab, the students should be able to do the following:

- Understand the working principle of the evaluation module and explain how it can be used to assist in the development of the DSP system.
- Become familiar with the MS-Windows based integrated development environment IDE – Code Composer 4.12, which is used to run the various software needed including TMS320C2000 Assembler, Compiler and Linker, TMS320C2xx C Source debugger.
- Write simple assembly language programs, assemble and link them, download the assembled code to the evaluation board, and execute it.

Equipment Required

Hardware :

- PC Specifications -
  - PC running Windows 95/98/NT/XP
  - 1.44Mb 3.5-inch floppy drive
  - 4-bit standard parallel port (SPP4)/ 8-bit bi-directional standard parallel port (SPP8) / enhanced parallel port (EPP)
  - Minimum 4Mb memory
  - Color VGA Monitor

- TMS320LF2407 Evaluation Module (EVM)
- XDS510PP-Plus Emulator
- +5V power supply for the EVM, converted to 3.3V for the 2407 CPU.
- 5-pin DIN connector
- DB25 connector printer cable
- Power supply adapter cable

Software :

- Windows 95/98/NT/XP
- Code Composer 4.12

EXAMPLE PROGRAM 1:

Write a program to turn on LEDs 2 and 4 with LEDs 1 and 3 off on the EVM board.

The EVM has 4 LEDs. They are mapped at address 000Ch of I/O memory space. Thus each LED can be turned on or off by setting or clearing the corresponding bit in the register that is mapped at 000Ch in the I/O space.

7	6	5	4	3	2	1	0	Bit No. in reg 000Ch
				LED4 (DS7)	LED3 (DS6)	LED2 (DS5)	LED1 (DS4)	

Thus to turn on LEDs 2and 4 the register contents should be 000Ah. Thus our task is to write these contents to the register 000Ch in I/O space. To write to I/O space, the OUT instruction is used.

The OUT instruction writes a 16-bit value from a data memory location to a specified I/O location. Only direct and indirect addressing can be used for this instruction. Thus, the data first has to be written to a register in data memory and then output to the I/O space. The instruction to store data to a data memory register is the SPLK instruction that used. Thus the program segment to write 000Ah to the location 000Ch is:

```
SPLK      #000Ah, LED_STATUS  ;Load value into the
                              ;uninitialized register
                              ;LED_STATUS
OUT       LED_STATUS, LEDS    ;Write the value in LEDS to
                              ;address 000Ch in the I/O
                              ;memory space
```

LED_STATUS is defined as an uninitialized register i.e 16-bits of space is reserved for it in memory. This is similar to the concept of a variable in programming languages. The definition of LED_STATUS is done as follows -

```
.bss LED_STATUS,1   ;LED Status Register
```

The register LEDS refers to the address 000Ch. It is defined as a symbol with constant with the .set directive as follows. Unlike the .bss, the .set defines a memory location with a constant/ initialized value.

```
LEDS      .set    000Ch       ;LEDs Register
```

A listing of the complete program is shown below:

```
;**********************************************************************
; File Name:       ch2_e1.asm
; Target System: C240x Evaluation Board
; Description:  This sample program helps you get familiar with
;                manipulating the I/O mapped LEDS (DS4-DS7) on the
;                F2407 EVM Development Board
```

```
;*********************************************************************

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Global symbol declarations
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        .def _c_int0,PHANTOM,GISR1,GISR2,GISR3,GISR4,GISR5,GISR6

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Address definitions
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        .include f2407.h

LEDS    .set    000Ch           ;EVM LED bank (I/O space)


;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Uninitialized global variable definitions
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        .bss    GPR0,1          ;general purpose variable
        .bss    LED_STATUS,1    ;LED Status Register

;=====================================================================
; M A I N   C O D E  - starts here
;=====================================================================
            .text
_c_int0
            NOP
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Configure the System Control and Status Registers
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_PF1         ;set data page

        SPLK    #0000000011111101b, SCSR1
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15        0:      reserved
* bit 14        0:      CLKOUT = CPUCLK
* bit 13-12     00:     IDLE1 selected for low-power mode
* bit 11-9      000:    PLL x4 mode
* bit 8         0:      reserved
* bit 7         1:      1 = enable ADC module clock
* bit 6         1:      1 = enable SCI module clock
* bit 5         1:      1 = enable SPI module clock
* bit 4         1:      1 = enable CAN module clock
* bit 3         1:      1 = enable EVB module clock
* bit 2         1:      1 = enable EVA module clock
* bit 1         0:      reserved
* bit 0         1:      clear the ILLADR bit

        LACC    SCSR2                   ;ACC = SCSR2 register
        OR      #0000000000001011b      ;OR in bits to be set
        AND     #0000000000001111b      ;AND out bits to be cleared
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15-6      0's:    reserved
* bit 5         0:      do NOT clear the WD OVERRIDE bit
* bit 4         0:      XMIF_HI-Z, 0=normal mode, 1=Hi-Z'd
* bit 3         1:      disable the boot ROM, enable the FLASH
* bit 2     no change   MP/MC* bit reflects the state of the MP/MC* pin
* bit 1-0      11:      11 = SARAM mapped to prog and data (default)

        SACL    SCSR2                   ;store to SCSR2 register

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
;Other setup
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup the core interrupts
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #0h                  ;set data page
        SPLK    #0h,IMR              ;clear the IMR register
        SPLK    #111111b,IFR         ;clear any pending core interrupts
        SPLK    #000000b,IMR         ;disable interrupts

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup the event manager interrupts
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_EVA               ;set data page
        SPLK    #0FFFFh, EVAIFRA      ;clear all EVA group A interrupts
        SPLK    #0FFFFh, EVAIFRB      ;clear all EVA group B interrupts
        SPLK    #0FFFFh, EVAIFRC      ;clear all EVA group C interrupts
        SPLK    #00000h, EVAIMRA      ;disable EVA group A interrupts
        SPLK    #00000h, EVAIMRB      ;disable EVA group B interrupts
        SPLK    #00000h, EVAIMRC      ;disable EVA group C interrupts

        LDP     #DP_EVB               ;set data page
        SPLK    #0FFFFh, EVBIFRA      ;clear all EVB group A interrupts
        SPLK    #0FFFFh, EVBIFRB      ;clear all EVB group B interrupts
        SPLK    #0FFFFh, EVBIFRC      ;clear all EVB group C interrupts
        SPLK    #00000h, EVBIMRA      ;disable EVB group A interrupts
        SPLK    #00000h, EVBIMRB      ;disable EVB group B interrupts
        SPLK    #00000h, EVBIMRC      ;disable EVB group C interrupts

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Enable global interrupts
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        CLRC    INTM                  ;enable global interrupts

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Disable the watchdog timer
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #DP_PF1               ;set data page

        SPLK    #0000000011101000b, WDCR
*                ||||||||||||||||
*                FEDCBA9876543210
* bits 15-8     0's      reserved
* bit 7         1:       clear WD flag
* bit 6         1:       disable the dog
* bit 5-3       101:     must be written as 101
* bit 2-0       000:     WDCLK divider = 1

;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;Setup external memory interface for LF2407 EVM
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        LDP     #GPR0          ;set current data page to
                               ;the data page of variable GPR0

        SPLK    #0000000001000000b, GPR0
*                ||||||||||||||||
*                FEDCBA9876543210
* bit 15-11     0's:    reserved
* bit 10-9      00:     bus visibility off
* bit 8-6       001:    1 wait-state for I/O space
* bit 5-3       000:    0 wait-state for data space
* bit 2-0       000:    0 wait state for program space
```

```
        OUT     GPR0, WSGR


        SPLK    #000ah,LED_STATUS   ;Turn on LEDs DS5, DS7
        OUT     LED_STATUS,LEDS     ;Turn off LEDs DS4, DS6
                                    ;0ah=01010b

END         B     END

;======================================================================
; I S R  -  PHANTOM and unused GISRs
;
; Description:    Dummy ISR, used to trap spurious interrupts.
;
; Modifies: Nothing
;======================================================================
PHANTOM     B     PHANTOM
GISR1       RET
GISR2       RET
GISR3       RET
GISR4       RET
GISR5       RET
GISR6       RET
```

File “2407.h” in the `.include` line is the header file for the TMS320LF2407 processor. It contains all peripheral register declarations as well as other useful definitions. This file must be included for all programs.

Code from the line with lable “`_c_int0`” to the line of “`OUT GPR0, WSGR`” is used to initialize the DSP registers and parameters, which must be included in any programs. In the following examples in this chapter, the initialization part of the code is omitted to save space.

EXAMPLE PROGRAM 2:

Write a program to check the status of the DIP switches and accordingly manipulate the corresponding LEDs i.e. if DIP switch 1 is ON, turn LED1 (DS4) ON and vice versa etc.

The EVM has 4 DIP switches that are mapped at address 0008h of I/O memory space. As mentioned earlier, the LEDs are mapped at address 000Ch. Thus, we first define these two registers so that they can be referred to as symbols as follows:

```
LEDS      .set 000Ch          ;LEDs Register
SWITCHES  .set 0008h          ;DIP SWITCH Register
```

The first task is to read the status of the DIP switches. This is done using the IN instruction. The status is read into a register DIP_STATUS which is defined as an uninitialized variable. To write this data to the I/O space assigned to the LEDs, the OUT instruction is used. The relevant code segment is as follows:

```
.bss DIP_STATUS,1              ;DIP SWITCH Register

IN   DIP_STATUS, SWITCHES      ; Get the status of each DIP
                               ; switch status
OUT  DIP_STATUS, LEDS          ; Turn LEDs on/off depending
                               ; on status of corresponding
                               ; switch
```

To run the program, do the following:

Assemble, link and load the program as described earlier. Adjust the DIP switches such that switched 1 and 3 are ON and switches 2 and 4 are OFF. Now press the F5 key to run the program. If the execution is successful, LEDs DS4 and DS6 will be ON and DS5 and DS7 will be OFF.

EXAMPLE PROGRAM 3:

Write a program to add 2 numbers that are stored as uninitialized variables.

We will use the ADD instruction with the direct addressing in this example. Let the variables to be added, be initialized as 'var1' and 'var2'. The ADD instruction adds the contents of a register to the contents of the accumulator and stores the result in the accumulator. The relevant code segment is:

```
     .bss var1,1
     .bss var2,1

     SPLK  #0002h,var1   ;Store a value 2h in var1
```

```
     SPLK  #0003h,var2   ;Store a value 3h in var2
     LACC  var1          ;Load contents of var1 in
                         ;accumulator
     ADD   var2          ;Add contents of var2 to
                         ;contents of  accumulator.
                         ;Store results in accumulator
```

Running the program:

Assemble, link and load the program. To check the program, the values of var1, var2 and accumulator need to be checked. The CPU window of the debugger shows the contents of the accumulator (ACC). To observe the values of var1 and var2, open the watch window of Code Composer with the following command:

wa *var1 ↵

wa *var2 ↵

wa is the command to add a variable in the watch window. The ***** tells the debugger that it is the data value of the variable that you wish to observe. If the ***** is omitted, then the debugger keeps a watch on the address of the variable rather than its value. A detailed discussion of the various debugger commands can be found in the appendix to this chapter.

Now run the program by pressing the F5 key. Now observe the contents of the accumulator and variables. For successful execution of the program the following values are expected:

```
ACC       0x0005
var1      0x0002
var2      0x0003
```

EXAMPLE PROGRAM 4:

Write a program to add two numbers stored at specific memory locations. Store the result in a third memory location. All these memory locations should be on the external SARAM.

Refer the memory map of the EVM as shown in Figure 11. The address range for SARAM is 8000h - FFFFh. Let us select the addresses 8000h, 8002h and 8004h for this program. The first task is to define symbols for these memory locations - var1, var2 and res. When we use direct addressing, the address is formed with the 9 MSBs taken from the data pointer and the 7 LSBs are taken from the operand of the instruction. This break-up is as shown below-

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	var1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	var2
1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	res

9 LSBs of Data Pointer   Symbol address

Thus the DP value will be 0100h. The relevant code is:

```
var1       .set    0000h
var2       .set    0002h
res        .set    0004h

           LDP     #0100h ;Load Data Pointer
           LACC    var1   ;Load value of var1 in accumulator
           ADD     var2
           SACC    res    ;Store result in res i.e. memory
                          ;location 8004h
```

Run the program. The checking of variables can be done by checking the memory window in Code Composer: Menu->View->Memory->Address=0x8000.

EXAMPLE PROGRAM 5:

Write a program to multiply two numbers. Both the numbers are stored in memory location in the SARAM.

Let the memory locations be 8000h and 8002h. For the multiply instruction, the multiplicand needs to be in the TREG and the result is stored in PREG. Accordingly the relevant code is:

```
mtplr      .set 0000h
mtplcnd    .set 0002h

LDP        #0100h
SPLK       #0002h,mtplr
SPLK       #0003h,mtplcnd
LT         mtplcnd         ;Load multiplicand into TREG
MPY        mtplr           ;Multiply contents of TREG
                           ;with multiplier
```

To view the contents of TREG and PREG, check the CPU window of the debugger. The values of mtplr and mtplcnd can be checked with the **wa** command. The contents should be:

	After Instruction
8000h / mtplr	0x0002
80002h/mtplcnd	0x0003
TREG	0x0003
PREG	0x00000006

EXAMPLE PROGRAM 6:

Check the value of a temporary register TEMP. If TEMP = 1, add the contents of *var1* and *var2*. Store the result in *res*. If TEMP=2, subtract contents of *var2* from *var1* and store result in *res*. For any other value of TEMP, multiply the contents of *var1* and *var2* and store result in *res*.

The purpose of this example is to introduce conditional .if/.elseif/.else/.endif directives. The relevant code is:

```
          TEMP    .set     1
          var1    .set     000eh
          var2    .set     000fh
          res     .set     0010h
          one     .set     1
          two     .set     2

          LDP    #0100h
          SPLK   #0005h,var1
          SPLK   #0002h,var2
lbl_if:   .if       TEMP = one
           LACC      var1
           ADD       var2
           SACL      res              ;res = var1 + var2
           .elseif   TEMP = two
           LACC      var1
           SUB       var2
           SACL      res         ;res = var1 - var2
           .else
           LT       var1
           MPY      var2
           SPL      res          ;res = var1 * var2
           .endif
```

EXAMPLE PROGRAM 7:

Write a program to increment a variable 10 times.

The main purpose of this program is to introduce the .loop/.break/.endloop directive.

The relevant code is:

```
          .bss ctr,1
          .bss var,1
```

```
            SPLK      #0,var
            .eval     0,ctr     ;Set counter = 0
lbl         .loop
            LACC #1
            ADD  var
            SACL var            ;var = var + 1
            .eval     ctr+1,ctr ;Increment counter
            .break    ctr=10    ;If counter=10, exit loop
            .endloop
```

Since the initial value of var is 0, the value of var after the completion of the program will be 10 or 000Ah.

EXAMPLE PROGRAM 8:

Write a program to turn on the LEDs DS7 to DS4, one after the other.

The purpose of this program is to introduce the bit-shift operation. The Shift Right SFR instruction will be employed. The key part of the code is listed below with infinite iteration.

```
LEDS        .set  000ch        ; Address of LEDs
            .bss  ctr,1        ; LED number counter
            .bss  LED_STATUS,1       ; LED status
          .bss RPT_NUM,1             ; for delay subroutine
          .bss mSEC,1          ; for delay subroutine

strt        SPLK #0080h,LED_STATUS

            .eval 0,ctr        ; Set counter = 0
lbl         .loop
            OUT     LED_STATUS,LEDS ; Turn on LEDs
                                    ; based on status
            LACC  LED_STATUS        ; Load status into ACC
            SFR                     ; Shift right ACC
            SACL  LED_STATUS        ; Update LED status
            CALL    mS_DELAY        ; Delay for 1 sec
```

```
            .eval ctr+1,ctr          ; Increment counter
          .break  ctr=8              ; If counter=8, exit loop
            .endloop
            B       strt             ; Repeat shifting
                                     ; from beginning
END         B      END

;============================================================
; Subroutine: mS_DELAY,
; Discription: implement a delay of approximately 1 sec
;============================================================
mS_DELAY:       LDP     #RPT_NUM     ; Set data page
                LACC    #6000        ; Load repeat number
                SACL    RPT_NUM      ; Store repeat number
                SPLK    #5000,mSEC   ; Initialize loop counter
mS_LOOP:
                RPT     RPT_NUM      ; Repeat next instruction
                NOP                  ; 4000 cycles = 0.2ms
                LACC    mSEC         ; Load value of counter
                SUB     #1           ; Decrement ACC
                SACL    mSEC         ; Update loop counter
                BCND    mS_LOOP,NEQ
                                     ; Jump to mS_LOOP
                                     ; if not zero
                RET                  ; Return
```

Look-up tables play a very important role in any programming language. In the next few examples, we shall see how to access data from a look-up table, how to write data in tabular form etc.

EXAMPLE PROGRAM 9:

Write a program that reads data from the program memory and writes it to the address 8000h of external data memory. The total number of words written is 10.

The TBLR instruction allows a word from a location in program memory to be transferred to a specific location in data memory. We will use this instruction in order to achieve the above objective. The table in program memory is defined as TABLEA and the destination table in data memory is defined as TABLEB. A counter (CTR) is setup in order to transfer 10 words. The BNZ (Branch if ACC > 0) conditional branch is maintains the loop for 10 word transfer. The key part of the program is as below -

```
TABLEB      .set   8000h          ;Starting address of the
                                  ;destination table
COUNT .set  10                    ;Defines the number of entries
                                  ;in the table
            .bss   SRCTBL,1
            .bss   CNT,1


            LACC   #COUNT
            SACL   CNT            ;Store the no. of data entries
                                  ;in CNT
            LARP   1              ;Select AR1 as the current AR
            LDP    #SRCTBL        ;Set data page
            LAR    AR1,#TABLEB ;Load the starting address of
                                  ;the destination table in AR1
            LACC   #TABLEA
            SACL   SRCTBL         ;Point the data pointer SRCTBL
                                  ;to the top of the source data LOOP
      LACC  SRCTBL
            TBLR   *+             ;Read the value from the table
                                  ;and store the destination
                                  ;table. Increment AR1 to point
                                  ;to the next address in the
                                  ;destination table
            ADD    #1
            SACL   SRCTBL         ;Increment source data pointer
            LACC   CNT
            SUB    #1
            SACL   CNT            ;Decrement loop count
            BNZ    LOOP           ;Continue if CNT > 0; i.e.
                                  ;until the end of the source
                                  ;data table is reached.
END         B      END            ;End Program

;--------------------------------------------------------
; Data look-up table
; No. Entries     : 10
;--------------------------------------------------------
TABLEA      .word  0
```

```
            .word    1h
            .word    2h
            .word    3h
            .word    4h
            .word    5h
            .word    6h
            .word    7h
            .word    8h
            .word    9h
```

To check this program, open a memory window in Code Composer as mentioned before.

EXAMPLE PROGRAM 10:

Write a program that reads data from a location in data memory (8310h) and writes it to another location in data memory (8410h). The total number of words written is 10.

The BLDD instruction allows a word in data memory pointed to by *source* to be copied into another data memory location pointed by *destination*. The various addressing modes allowed for this instruction are -

BLDD *#lk, dma*

BLDD *#lk,ind[,* **AR***n]*

BLDD **dma,** *#lk,*

BLDD *ind, #lk [,* **AR***n]*

In this example, we'll use the **BLDD** *#lk,ind[,* **AR***n]* mode. The RPT instruction is employed to repeat the transfer 10 times. The number of words to be transferred is stored in the RPTCNT. When the BLDD instruction is repeated, the source address specified by the long immediate constant is stored in the PC. Since, the PC is incremented after every repetition, the source address is also incremented. In case of the destination, the auto-increment option of indirect addressing is used. The key part of the code listing is given below-

```
TABLEA          .set  8310h ;Starting address of the
                            ;source table
TABLEB          .set  8410h ;Starting address of the
                            ;destination table
```

```
RPTCNT          .set   10      ;Defines the number of entries
                               ;in the table
            LARP  1            ;Select AR1 as the current AR
            LDP   #0100h       ; DP for addresses 8000h-807Fh
            LAR   AR1,#TABLEB  ;Load the starting address of
                               ;the destination table in AR1
            RPT   #COUNT       ;Perform the following
                               ;operation 10 times
            BLDD  #TABLEA,*+   ;Transfer data from TABLEA to
                               ;TABLEB. After the
                               ;instruction, pointers to data
                               ;in both the tables are
                               ;incremented.

END         B     END          ;End Program
```

Laboratory Assignments

1. Read and run all the example programs. Draw flow charts for all the programs with reasonable details.
2. Write a program to turn on the LEDs from DS1 to DS8 or inversely, one after the other and repeat for N (N<8) times (cycles). Use DIP switches to set the value of N. If a variable TEMP=0, turn on the LEDs form DS1 to DS8, otherwise, reverse the sequence. Use assembler directives and branch instructions to control the flow.
3. Write a program to store integer vector $A=[1, \ldots, 9]^T$ into data memory starting from address 8000h and integer vector $B=[9, \ldots, 1]^T$ into data memory starting from address 8100h. Then compute the inner product of these two vectors (A^TB) and store the result in 8200h. (Hint: use indirect addressing mode)