1. Write a program to multiply two 16 bit binary numbers.

```
; VALUE1:   1900H (6400)          (IN R1)
; VALUE2:   0C80H (3200)          (IN R2)
; RESULT:   1388000H(20480000)    (IN R3)
; Set a breakpoint at NOP instruction, run the program & check the result

        AREA  MULTIPLY , CODE, READONLY

ENTRY                           ;mark first instruction to execute

START
        MOV R1,#6400            ; store first number in R0
        MOV R2,#3200            ; store second number in R1
        MUL R3,R1,R2            ; multiplication

        NOP
        NOP
        NOP

        END                      ;mark end of file
```

2. Write a program to find the sum of first 10 integer numbers.

```
        AREA SUM,CODE,READONLY

ENTRY
        MOV R1,#10              ; load 10 to regester
        MOV R2,#0               ; empty the register to store result
LOOP
        ADD R2,R2,R1            ; add the content of R1 with result at R2
        SUBS R1,#0x01           ; decreament R1 by 1
        BNE LOOP                ; repeat till R1 goes 0
BACK B BACK                     ; jumps back to C code
        END
```

3. Write a program to find factorial of a number.

```
; In this example we have taken N=7
; Check the result in R0/R3 register =13B0H (5040)
; Set a breakpoint at NOP instruction, run the program & check the result

        AREA  FACTORIAL , CODE, READONLY

ENTRY                           ;mark first instruction to execute

START
        MOV r0, #3              ; store factorial number in R0
        MOV r1,r0               ; move the same number in R1

FACT  SUBS r1, r1, #1           ; subtraction
        CMP r1, #1              ; comparison
        BEQ STOP
        MUL r3,r0,r1            ; multiplication
        MOV   r0,r3             ; Result
        BNE FACT                ; branch to the loop if not equal
STOP
        NOP
        NOP
        NOP

        END                     ;mark end of file
```

4. Write a program to add an array of 16 bit numbers and store the 32 bit result in internal RAM

```
; Array of 6 numbers  0X1111, 0X2222, 0X3333, 0XAAAA, 0XBBBB, 0XCCCC
; The sum is 29997H. The result can be viewed in location 0X40000000 & also in R0

        AREA  ADDITION , CODE, READONLY

ENTRY                           ;Mark first instruction to execute

START
        MOV R5,#6               ; INTIALISE COUNTER TO 6(i.e. N=6)
        MOV R0,#0               ; INTIALISE SUM TO ZERO
        LDR R1,=VALUE1          ; LOADS THE ADDRESS OF FIRST VALUE
LOOP
        LDRH R3,[R1],#02        ; READ 16 BIT DATA
        ADD R0,R0,R3            ; ADD R2=R2+R3
        SUBS R5,R5,#1           ; DECREMENT COUNTER
        CMP R5,#0
        BNE LOOP                ; LOOK BACK TILL ARRAY ENDS
        LDR R4,=RESULT          ; LOADS THE ADDRESS OF RESULT
        STR R0,[R4]             ; STORES THE RESULT IN R1
JMP B JMP

VALUE1 DCW 0X1111,0X2222,0X3333,0XAAAA,0XBBBB,0XCCCC
                                ;ARRAY OF 16 BIT NUMBERS(N=6)

        AREA DATA2,DATA,READWRITE  ; TO STORE RESULT IN GIVEN ADDRESS
RESULT DCD 0X0

        END                     ; Mark end of file
```

5. Write a program to find the square of a number (1 to 10) using look-up table.

```
; Given number is 6 (R1) then result is in R3=24H(36)
; Set a breakpoint at NOP instruction, run the program & check the result

        AREA  SQUARE , CODE, READONLY

ENTRY                           ; mark first instruction to execute

START
        LDR R0, = TABLE1  ; load start address of Lookup table
        MOV R1,#6               ; load number whose square is to be find
        MOV R1, R1, LSL#0x2     ; generate address corresponding to square of given number
        ADD R0, R0, R1          ; load address of element in Lookup table
        LDR R3, [R0]            ; get square of given number in R3

        NOP
        NOP
        NOP

; Lookup table contains Squares of numbers from 0 to 10 (in hex)

TABLE1      DCD 0X00000000    ; SQUARE OF 0=0
            DCD 0X00000001    ; SQUARE OF 1=1
            DCD 0X00000004    ; SQUARE OF 2=4
            DCD 0X00000009    ; SQUARE OF 3=9
            DCD 0X00000010    ; SQUARE OF 4=16
            DCD 0X00000019    ; SQUARE OF 5=25
            DCD 0X00000024    ; SQUARE OF 6=36
            DCD 0X00000031    ; SQUARE OF 7=49
            DCD 0X00000040    ; SQUARE OF 8=64
            DCD 0X00000051    ; SQUARE OF 9=81
            DCD 0X00000064    ; SQUARE OF 10=100

        END                     ; mark end of file
```

6. Write a program to find the largest/smallest number in an array of 32 numbers .

; **Program to find largest number in an array & store in internal ram**
; Array of 7 numbers 0X44444444, 0X22222222, 0X11111111, 0X33333333, 0XAAAAAAAA,
; 0X88888888, 0X99999999
; Result can be viewed in location 0X40000000 & also in R2
; Set a breakpoint at NOP instruction, run the program & check the result

```
        AREA LARGEST , CODE, READONLY

ENTRY                       ; mark first instruction to execute

START
        MOV R5,#6           ; initialize counter to 6(i.e. N=7)
        LDR R1,=VALUE1      ; loads the address of first value
        LDR R2,[R1],#4      ; word align to array element
LOOP
        LDR R4,[R1],#4      ; word align to array element
        CMP R2,R4           ; compare numbers
        BHI LOOP1           ; if the first number is > then goto LOOP1

        MOV R2,R4           ; if the first number is < then move content R4 to R2
LOOP1
        SUBS R5,R5,#1       ; decrement counter
        CMP R5,#0           ; compare counter to 0
        BNE LOOP            ; loop back till array ends

        LDR R4,=RESULT      ; loads the address of RESULT
        STR R2,[R4]         ; stores the result in R2

        NOP
        NOP
        NOP

        ; Array of 32 bit numbers(N=7)
VALUE1
        DCD 0X44444444
        DCD   0X22222222
        DCD   0X11111111
```

```
        DCD   0X33333333
        DCD   0XAAAAAAAA
        DCD   0X88888888
        DCD   0X99999999

        AREA DATA2,DATA,READWRITE  ; to store result in given address
RESULT DCD 0X0

        END                        ; mark end of file
```

; **Program to find smallest number in an array & store in internal ram**

```
; Array of 7 numbers 0X44444444, 0X22222222, 0X11111111, 0X22222222, 0XAAAAAAAA,
; 0X88888888, 0X99999999
; Result can be viewed in location 0X40000000 & also in R2
; Set a breakpoint at NOP instruction, run the program & check the result

        AREA SMALLEST, CODE, READONLY

ENTRY                          ; mark first instruction to execute

START
        MOV R5,#6              ; initialize counter to 6(i.e. N=7)
        LDR R1,=VALUE1         ; loads the address of first value
        LDR R2,[R1],#4         ; word align to array element
LOOP
        LDR R4,[R1],#4         ; word align to array element
        CMP R2,R4              ; compare numbers
        BLS LOOP1              ; if the first number is < then goto LOOP1

        MOV R2,R4              ; if the first number is > then move content R4 to R2
LOOP1
        SUBS R5,R5,#1          ; decrement counter
        CMP R5,#0              ; compare counter to 0
        BNE LOOP               ; loop back till array ends

        LDR R4,=RESULT         ; loads the address of result
```

```
        STR R2,[R4]                 ; stores the result in R1

        NOP
        NOP
        NOP

; Array of 32 bit numbers(N=7)
VALUE1
        DCD 0X44444444
        DCD   0X22222222
        DCD   0X11111111
        DCD   0X22222222
        DCD   0XAAAAAAAA
        DCD   0X88888888
        DCD   0X99999999

        AREA DATA2,DATA,READWRITE  ; to store result in given address
RESULT DCD 0X0

        END                         ; mark end of file
```

7. Write a program to arrange a series of 32 bit numbers in ascending/descending order.

```
; Program to sort in ascending order
; Array of 4 numbers 0X44444444, 0X11111111, 0X33333333, 0X22222222
; Set a breakpoint at START1 label & run the program
; Check the unsorted numbers at location 0X40000000 next
; Set a breakpoint at NOP instruction, run the program & check the result
; Result can be viewed at location 0x40000000

        AREA  ASCENDING , CODE, READONLY

ENTRY                       ; mark first instruction to execute

START
        MOV R8,#4           ; initialize counter to 4(i.e. N=4)
        LDR R2,=CVALUE      ; address of code region
        LDR R3,=DVALUE      ; address of data region
LOOP0
        LDR R1,[R2],#4      ; loading values from code region
        STR R1,[R3],#4      ; storing values to data region

        SUBS R8,R8,#1       ; decrement counter
        CMP R8,#0           ; compare counter to 0
        BNE LOOP0           ; loop back till array ends

START1
        MOV R5,#3           ; initialize counter to 3(i.e. N=4)
        MOV R7,#0           ; flag to denote exchange has occured
        LDR R1,=DVALUE      ; loads the address of first value

LOOP
        LDR R2,[R1],#4      ; word align to array element
        LDR R3,[R1]         ; load second number
        CMP R2,R3           ; compare numbers
        BLT LOOP2           ; if the first number is < then goto LOOP2

        STR R2,[R1],#-4     ; interchange number R2 & R3
        STR R3,[R1]         ; interchange number R2 & R3
        MOV R7,#1           ; flag denoting exchange has taken place
        ADD R1,#4           ; restore the ptr
```

```
LOOP2
        SUBS R5,R5,#1           ; decrement counter
        CMP R5,#0               ; compare counter to 0
        BNE LOOP                ; loop back till array ends
        CMP R7,#0               ; comparing flag
        BNE  START1             ; if flag is not zero then go to START1 loop

        NOP
        NOP
        NOP

; array of 32 bit numbers(N=4) in code region

CVALUE
        DCD   0X44444444
        DCD   0X11111111
        DCD   0X33333333
        DCD   0X22222222

        AREA DATA1,DATA,READWRITE
; array of 32 bit numbers in data region
DVALUE
        DCD 0X0000000

        END                     ; mark end of file
```

; Program to sort in Descending order

```
; Array of 4 numbers 0X44444444, 0X11111111, 0X33333333, 0X22222222
; Set a breakpoint at START1 lable & run the program
; Check the unsorted numbers at location 0X40000000 next
; Set a breakpoint at NOP instruction, run the program & check the result
; Result can be viewed at location 0X40000000

        AREA DESCENDING, CODE, READONLY

ENTRY                       ;Mark first instruction to execute

START
        MOV R8,#4           ; intialise counter to 4(i.e. N=4)
        LDR R2,=CVALUE      ; address of code region
        LDR R3,=DVALUE      ; address of data region

LOOP0
        LDR R1,[R2],#4      ;  loading values from code region
        STR R1,[R3],#4      ;  storing values to data region

        SUBS R8,R8,#1       ; decrement counter
        CMP R8,#0           ; compare counter to 0
        BNE LOOP0           ; loop back till array ends

START1
        MOV R5,#3           ; intialise counter to 3(i.e. N=4)
        MOV R7,#0           ; flag to denote exchange has occured
        LDR R1,=DVALUE      ; loads the address of first value

LOOP
        LDR R2,[R1],#4      ; word align t0 array element
        LDR R3,[R1]         ; load second number
        CMP R2,R3           ; compare numbers
        BGT LOOP2           ; if the first number is > then goto LOOP2
        STR R2,[R1],#-4     ; interchange number R2 & R3
        STR R3,[R1]         ; interchange number R2 & R3
        MOV R7,#1           ; flag denoting exchange has taken place
        ADD R1,#4           ; restore the ptr
```

```
LOOP2
        SUBS R5,R5,#1           ; decrement counter
        CMP R5,#0               ; compare counter to 0
        BNE LOOP                ; loop back till array ends
        CMP R7,#0               ; comparing flag
        BNE  START1             ; if flag is not zero then go to START1 loop

        NOP
        NOP
        NOP

; array of 32 bit numbers(N=4) in code region
CVALUE
        DCD 0X44444444
        DCD   0X11111111
        DCD   0X33333333
        DCD   0X22222222

; Array of 32 bit numbers in data region
        AREA DATA1,DATA,READWRITE
DVALUE
        DCD 0X00000000
        END                     ; mark end of file
```

8. Write a program to count the number of ones and zeros in two consecutive memory locations.

```
; Program to count the number of ones & zeros in two consecutive memory locations
; We took two numbers i.e. 0X11111111,0XAA55AA55  (R0)
; check the result in R2 for ones & R3 for zeros
; Set a breakpoint at NOP instruction, run the program & check the result
        AREA  ONEZERO , CODE, READONLY
ENTRY                           ; mark first instruction to execute
START
        MOV R2,#0               ; counter for ones
        MOV R3,#0               ; counter for zeros
        MOV R7,#1               ; counter to get two words
        LDR R6,=VALUE           ; loads the address of value

LOOP
        MOV R1,#32              ; 32 bits counter
        LDR R0,[R6],#4          ; get the 32 bit value

LOOP0
        MOVS R0,R0,ROR #1       ; right shift to check carry bit (1's/0's)
        BHI ONES                ; if carry bit is 1 goto ones branch otherwise next zeros
        ADD R3,R3,#1            ; if carry bit is 0 then increment the counter by 1(R3)
        B LOOP1                 ; branch to LOOP1

ONES
        ADD R2,R2,#1            ; if carry bit is 1 then increment the counter by 1(R2)

LOOP1
        SUBS R1,R1,#1           ; counter value decremented by 1
        BNE LOOP0               ; if not equal goto to loop0 checks 32bit
        SUBS R7,R7,#1           ; counter value decremented by 1
        CMP R7,#0               ; compare counter R7 to 0
        BNE LOOP                ; if not equal goto to LOOP
        NOP
        NOP
        NOP

VALUE DCD 0X11111111,0XAA55AA55 ; two values in an array
        END                     ; mark end of file
```

9. Display "Hello World" message using Internal UART.

```
#include <lpc214x.h>

void uart_interrupt(void)__irq ;

unsigned char temp , temp1 = 0x00 ;
unsigned char rx_flag = 0 , tx_flag = 0 ;

int main(void)
{
        PINSEL0=0X0000005;          //select TXD0 and RXD0 lines
        U0LCR = 0X00000083;         //enable baud rate divisor loading and
        U0DLM = 0X00;               //select the data format
        U0DLL = 0x13;               //select baud rate 9600 bps
        U0LCR = 0X00000003;
        U0IER = 0X03;               //select Transmit and Recieve interrupt
        VICVectAddr0 = (unsigned long)uart_interrupt;      //UART 0 INTERRUPT
        VICVectCntl0 = 0x20|6;      // Assign the VIC channel uart-0 to interrupt priority 0
        VICIntEnable = 0x00000040;          // Enable the uart-0 interrupt
        rx_flag = 0x00;
        tx_flag = 0x00;
        while(1)
        {
                while(rx_flag == 0x00);         //wait for receive flag to set
                rx_flag = 0x00;         //clear the flag
                U0THR = temp1 ;
                while(tx_flag == 0x00);         //wait for transmit flag to set
                tx_flag = 0x00;         //clear the flag
        }
}
void uart_interrupt(void)__irq
{
        temp = U0IIR;
        temp = temp & 0x06;             //check bits,  data sending or receiving
         if(temp  == 0x02)              //check data is sending
        {
                tx_flag = 0xff;         // flag that indicate data is sending via UART0
                VICVectAddr=0;
```

```
        }
        else if(temp  == 0x04)          // check any data available to receive
        {
                //   U0THR = U0RBR;
                temp1 = U0RBR ;      // copy data into variable
                rx_flag = 0xff;          // set flag to indicate that data is received
                VICVectAddr=0;
        }
}
```

10. Interface and Control a DC Motor.

```
/*Description  :Direction of the DCM is cotrolled in this software by alternatively inter- changing the supply with the help of Relay.          Port lines: P1.16 and P1.17.  */

#include<lpc214x.h>

void clock_wise(void);
void anti_clock_wise(void);

unsigned int j=0;

int main()
{
        PINSEL2 = 0XFFFFFFF0;
        //IO1CLR = 0X0000ff00;
        IO1DIR= 0X00030000;       //p1.16 and p1.17 are selected as outputs.
        IO1SET= 0X00010000;       //P1.16 should always high.
        while(1)
        {
                clock_wise();
                for(j=0;j<500000;j++);          //delay
                anti_clock_wise();
                for(j=0;j<500000;j++);          //delay
        }       //End of while(1)
}       //End of Main
void clock_wise(void)
{
        IO1CLR = 0x00030000;      //stop motor and also turn off relay
        for(j=0;j<500000;j++);          //small delay to allow motor to turn off
        IO1SET = 0X00030000;      //Selecting the P1.17 line for clockwise and turn on motor
 }
void anti_clock_wise(void)
{
        IO1CLR = 0X00030000;      //stop motor and also turn off relay
        for(j=0;j<1000000;j++);         //small delay to allow motor to turn off
        IO1SET = 0X00010000;      //not selecting the P1.17 line for Anti clockwise
}
```

11. Interface a Stepper motor and rotate it in clockwise and anti-clockwise direction.

/*A stepper motor direction is controlled by shifting the voltage across the coils. Port lines : P1.20 to P1.23 */

```
#include <LPC21xx.h>

void clock_wise(void) ;
void anti_clock_wise(void) ;

unsigned  int var1 ;
unsigned long int i = 0 , j = 0 , k = 0 ;

int main(void)
{
        PINSEL2 = 0x00000000;       //P1.20 to P1.23 GPIO
        IO1DIR |= 0x00F00000 ;      //P1.20 to P1.23 made as output
        while(1)
        {
                for( j = 0 ; j < 50 ; j++ )         // 50 times in Clock wise Rotation
                        clock_wise() ;              // rotate one round clockwise
                for( k = 0 ; k < 65000 ; k++ ) ;          // Delay to show  anti_clock Rotation
                for( j=0 ; j < 50 ; j++ )           // 50 times in  Anti Clock wise Rotation
                        anti_clock_wise() ;     // rotate one round anticlockwise
                for( k = 0 ; k < 65000 ; k++ ) ;          // Delay to show ANTI_clock Rotation
          }
}       // End of main
void clock_wise(void)
{
        var1 = 0x00080000;   //For Clockwise
        for( i = 0 ; i <= 3 ; i++ )         // for A B C D Stepping
        {
                var1 <<= 1 ;
                IO1CLR =0x00F00000 ;        //clearing all 4 bits
                IO1SET = var1 ;         // setting perticular bit
                for( k = 0 ; k < 3000 ; k++ );  //for step speed variation
        }
}
void anti_clock_wise(void)
{
```

```
        var1 = 0x00800000 ;             //For Anticlockwise
        IO1CLR =0x00F00000 ;        //clearing all 4 bits
        IO1SET = var1 ;
        for( k = 0 ; k < 3000 ; k++ ) ;
        for( i = 0 ; i < 3 ; i++ )          // for A B C D Stepping
        {
                var1 >>=1;      //rotating bits
                IO1CLR =0x00F00000 ;        // clar all bits before setting
                IO1SET = var1           // setting perticular bit
                for( k = 0 ; k < 3000 ; k++ ) ;             //for step speed variation
        }
}
```

12. Determine Digital output for a given Analog input using Internal ADC of ARM controller.

```
/*Description : This example scans the channel ADC0.4. Voltage is varied by varying the pot R2. Since ref
voltage is 3.3, ADC output range is 000 to 3FF (10 bit). Short JP2 to enable the hardware.*/
//10-bit internal ADC
//AIN0 pin is selected
//you can change the channel by changing PINSEL1 and ADCR value

#include <lpc214x.h>
#include <Stdio.h>
#define vol 3.3          //Reference voltage
#define fullscale 0x3ff        //10 bit adc fullscale

unsigned int data_lcd=0,i=0,n=0;
unsigned int adc_value=0,temp_adc=0,temp1,temp2,adc[8];
float temp,adc1[8];
unsigned char var[15],var1[15],fst_flag=0xff;
unsigned char *ptr,arr[]= "ADC O/P= ";
unsigned char *ptr1,dis[]="A I/P  = ";

void lcd_init(void);
void wr_cn(void);
void clr_disp(void);
void delay(unsigned int);
void lcd_com(void);
void wr_dn(void);
void lcd_data(void);

int main()
{
        PINSEL1 = 0X04000000;      //AD0.4 pin is selected
        IO0DIR = 0x000000FC;       //configure o/p lines for lcd
        delay(3200);
        lcd_init();        //LCD initialization
        delay(3200);
        clr_disp();        //clear display
        delay(3200);   //delay
        ptr = dis;
        temp1 = 0x80;          //Display starting address of 1st line on LCD
```

```
lcd_com();
delay(800);
while(*ptr!='\0')
{
        temp1 = *ptr;
        lcd_data();
        ptr ++;
}
ptr1 = arr;
temp1 = 0xC0;          //Display starting address of 2nd line on LCD
lcd_com();
delay(800);
while(*ptr1!='\0')
{
        temp1 = *ptr1;
        lcd_data();
        ptr1 ++;
}


while(1)   //infinite loop
{
        temp = 0;
        adc_value = 0;
         AD0CR = 0x01200004;       //CONTROL register for ADC-AD0.4
        while(((temp_adc = AD0GDR) &0x80000000)  == 0x00000000);  //to check the interrupt bit
        adc_value = AD0GDR;         //reading the ADC value
        adc_value >>=6;
        adc_value &= 0x000003ff;
        temp = ((float)adc_value * (float)vol)/(float)fullscale;
        if(fst_flag)
        {
                fst_flag = 0x00;
                for(i=0;i<8;i++)
                {
                        adc[i] = adc_value;
                        adc1[i] = temp;
                }
```

```
}
else
{
        n=7;
        for(i=n;i>0;i--)
        {
                adc[i] = adc[i-1];
                adc1[n] = adc1[n-1];
                n = n-1;
        }
        adc[0] = adc_value;
        adc1[0] = temp;
}
temp=0;
adc_value=0;
for(i=0;i<8;i++)
{
        temp += adc1[i];
        adc_value += adc[i];
}
 temp = (temp/8);
adc_value = (adc_value/8);
sprintf(var1,"%4.2fV",temp);
sprintf(var,"%3x",adc_value);
temp1 = 0x89;
lcd_com();
delay(1200);
ptr1 = var1;
while(*ptr1!='\0')
{
        temp1=*ptr1;
        lcd_data();
        ptr1++;
}
temp1 = 0xc9;
lcd_com();
delay(1200);
ptr1 = var;
```

```
                while(*ptr1!='\0')
                {
                        temp1=*ptr1;
                        lcd_data();
                        ptr1++;
                }
        }       // end of while(1)
} //end of main()

//**** LCD initialization ****//
void lcd_init()
{
        temp2=0x30;
        wr_cn();
        delay(800);

        temp2=0x30;
        wr_cn();
        delay(800);

        temp2=0x30;
        wr_cn();
        delay(800);

        temp2=0x20;
        wr_cn();
        delay(800);

        temp1 = 0x28;
        lcd_com();
        delay(800);

        temp1 = 0x0c;
        lcd_com();
        delay(800);

        temp1 = 0x06;
        lcd_com();
```

```
        delay(800);

        temp1 = 0x80;
        lcd_com();
        delay(800);
}
void lcd_com(void)
{
        temp2= temp1 & 0xf0;
        wr_cn();
        temp2 = temp1 & 0x0f;
        temp2 = temp2 << 4;
        wr_cn();
        delay(500);
}
// command nibble o/p routine
void wr_cn(void)            // write command reg
{
        IO0CLR  = 0x000000FC;     // clear the port lines.
        IO0SET        = temp2;        // Assign the value to the PORT lines
        IO0CLR  = 0x00000004;     // clear bit  RS = 0
        IO0SET        = 0x00000008;   // ENABLE=1
        delay(10);
        IO0CLR  = 0x00000008;
}

// data nibble o/p routine
void wr_dn(void)
{
        IO0CLR  = 0x000000FC;     // clear the port lines.
        IO0SET = temp2;               // Assign the value to the PORT lines
        IO0SET = 0x00000004;      // set bit  RS = 1
        IO0SET = 0x00000008;      // ENABLE=1
        delay(10);
        IO0CLR = 0x00000008;
}
// data o/p routine which also outputs high nibble first and lower nibble next
void lcd_data(void)
```

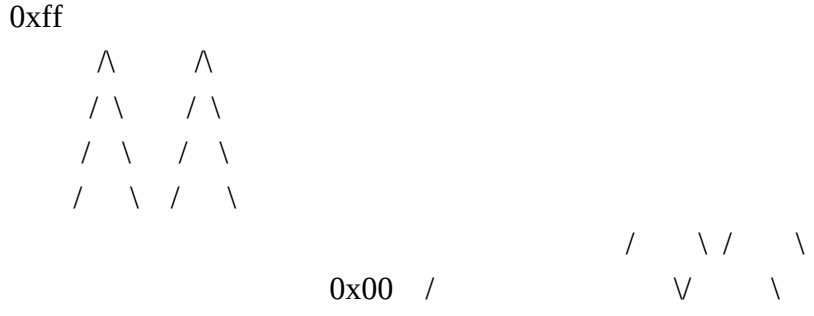
```
{
        temp2 = temp1 & 0xf0;
        wr_dn();
        temp2= temp1 & 0x0f;
        temp2= temp2 << 4;
        wr_dn();
        delay(100);
}
void delay(unsigned int r1)
{
        unsigned int r;
        for(r=0;r<r1;r++);
}
void clr_disp(void)
{
        temp1 = 0x01;
        lcd_com();
        delay(500);
}
```

13. Interface a DAC and generate Triangular and Square waveforms.

// program to generate Triangular wave with DAC interface

/* Description : This example explains about how Triangular Wave is generated.P0.4 to P0.11 are used to get the Digital values.*/

/*



*/

```
#include <LPC21xx.h>

int main ()
{
        unsigned long int temp=0x00000000;
        unsigned int i=0;

        IO0DIR=0x00FF0000;
        while(1)
        {
                // output 0 to FE
                for(i=0;i!=0xFF;i++)
                {
                        temp=i;
                        temp = temp << 16;
                        IO0PIN=temp;
                }

                // output FF to 1
                for(i=0xFF; i!=0;i--)
                {
```

```
                temp=i;
                temp = temp << 16;
                IO0PIN=temp;
            }
        }       //End of while(1)
}       //End of main()
```

// program to generate square wave with DAC interface

/* Description : This example explains about how Sqaure Wave is generated.P0.0 to P0.15 are used to get the Digital values.*/

```
 /*

        0xff  _________
         |                          |                                  |
         |                          |                                  |
         |                          |                                  |
                                            |                          |
         |
                              0x00 |                        |_________|

*/

#include <lpc21xx.h>

void delay(void);

int main ()
{
        PINSEL0 = 0x00000000 ;      // Configure P0.0 to P0.15 as GPIO
        PINSEL1 = 0x00000000 ;      // Configure P0.16 to P0.31 as GPIO
        IO0DIR  = 0x00FF0000 ;
        while(1)
        {
                IO0PIN = 0x00000000;
                delay();
                IO0PIN = 0x00FF0000;
                delay();
        }
```

```
}
void delay(void)
{
        unsigned int i=0;
        for(i=0;i<=3000;i++);
}
```

14. Interface a 4x4 keyboard and display the key code on an LCD.

```
/* Description :4x4 matrix keyboard has total 16 keys.
There are 4 rows and 4 cols Identity of key pressed (0 to F) will be displayed on LCD. Port lines used : P0.8 to
P0.11 row1 to row4,P0.12 to P0.15 - col1 to col4 */

#include<lpc21xx.h>
#include<stdio.h>

void scan(void);
void get_key(void);
void display(void);
void delay(unsigned int);
void init_port(void);

void lcd_init(void);
void clr_disp(void);
void lcd_com(void);   // LCD routines
void lcd_data(void);
void wr_cn(void);
void wr_dn(void);

unsigned long int scan_code[16]= { 0x0000EE00,0x0000ED00,0x0000EB00,0x0000E700 ,
                                   0x0000DE00,0x0000DD00,0x0000DB00,0x0000D700 ,
                                   0x0000BE00,0x0000BD00,0x0000BB00,0x0000B700 ,
                                   0x00007E00,0x00007D00,0x00007B00,0x00007700 };

unsigned char ASCII_CODE[16]= {'0','4','8','C',
                               '1','5','9','D',
                               '2','6','A','E',
                               '3','7','B','F'};
unsigned char  row,col;

unsigned char temp,flag,i,result,temp1;
unsigned int r,r1;
unsigned long int var,var1,var2,res1,temp2,temp3,temp4;
unsigned char *ptr;
unsigned char disp0[] = "KEYPAD TESTING";
unsigned char disp1[] = "KEY = ";
```

```
int main()
{
        PINSEL0 = 0X00000000;      // configure P0.0 TO P0.15 as GPIO
        init_port();      //port intialisation
        delay(3200);    //delay
        lcd_init();       //lcd intialisation
        delay(3200);    //delay
        clr_disp();       //clear display
        delay(500);     //delay
        clr_disp();
        ptr = disp0;
        temp1 = 0x80;           // Display starting address of 1st line on LCD
        lcd_com();

        while(*ptr!='\0')
        {
                temp1 = *ptr;
                lcd_data();
                ptr ++;
        }
        ptr = disp1;
        temp1 = 0xC0;           // Display starting address of 2nd line on LCD
        lcd_com();

        while(*ptr!='\0')
        {
                temp1 = *ptr;
                lcd_data();
                ptr ++;
        }


        while(1)
        {
                get_key();
                display();
        }
```

```
} //end of main()

void get_key(void)     //get the key from the keyboard
{
        unsigned int k;
        flag = 0x00;
        IO0PIN=0x0000F000;
        while(1)
        {
                for(row=0X00;row<0X04;row++)     //Writing one for col's
                {
                        if( row == 0X00)
                        {
                                temp3=0x00000E00;
                        }
                        else if(row == 0X01)
                        {
                                temp3=0x00000D00;
                        }
                        else if(row == 0X02)
                        {
                                temp3=0x00000B00;
                        }
                        else if(row == 0X03)
                        {
                                temp3=0x00000700;
                        }
                        var1 = temp3;
                        IO0PIN = var1;          // each time var1 value is put to port1
                        IO0CLR =~var1;          // Once again Confirming (clearing all other bits)
                        scan();
                        delay(100);     //delay
                        if(flag == 0xff)
                                break;

                }       // end of for loop
                if(flag == 0xff)
                        break;
```

```
        }       // end of while
        for(k=0;k<16;k++)
        {
                if(scan_code[k] == res1)      //equate the scan_code with res1
                {
                        result =  ASCII_CODE[k];    //same position value of ascii code
                        break;                  //is assigned to result
                }
        }
}       // end of get_key();

void scan(void)
{
        unsigned long int t;
        temp2 = IO0PIN;                  // status of port1
        temp2 = temp2 & 0x0000F000;      // Verifying column key
        if(temp2 != 0x0000F000)          // Check for Key Press or Not
        {
                delay(3000);             //delay(100)//give debounce delay check again
                temp2 = IO0PIN; //IO0
                temp2 = temp2 & 0x0000F000;       //changed condition is same
                if(temp2 != 0x0000F000)           // store the value in res1
                {
                        flag = 0xff;
                        res1 = temp2;
                        t = (temp3 & 0x00000F00);     //Verfying Row Write
                        res1 = res1 | t;        //final scan value is stored in res1
                }
                else
                {
                        flag = 0x00;
                }
        }
}       // end of scan()


void display(void)
{
```

```
        temp1 = 0xC6;          //display  address for key value
        lcd_com();
        temp1 = result;
        lcd_data();
}
void lcd_init (void)
{
        temp = 0x30;
        wr_cn();
        delay(3200);

        temp = 0x30;
        wr_cn();
        delay(3200);

        temp = 0x30;
        wr_cn();
        delay(3200);

        temp = 0x20;
        wr_cn();
        delay(3200);

        temp = 0x28;           // load command for lcd function setting with lcd in 4 bit mode,
        lcd_com();             // 2 line and 5x7 matrix display
        delay(3200);

        temp1 = 0x0C;          // load a command for display on, cursor on and blinking off
        lcd_com();
        delay(800);

        temp1 = 0x06;          // command for cursor increment after data dump
        lcd_com();
        delay(800);

        temp1 = 0x80;
        lcd_com();
```

```
        delay(800);
}
void lcd_data(void)
{
        temp = temp1 & 0xf0;
        wr_dn();
        temp= temp1 & 0x0f;
        temp= temp << 4;
        wr_dn();
        delay(100);
}
void wr_dn(void)                        //write data reg
{
        IO0CLR  = 0x000000FC;       // clear the port lines.
        IO0SET = temp;                          // Assign the value to the PORT lines
        IO0SET = 0x00000004;        // set bit  RS = 1
        IO0SET = 0x00000008;        // Enable=1
        delay(10);
        IO0CLR = 0x00000008;
}
void lcd_com(void)
{
        temp = temp1 & 0xf0;
        wr_cn();
        temp = temp1 & 0x0f;
        temp = temp << 4;
        wr_cn();
        delay(500);
}
void wr_cn(void)       //write command reg
{
        IO0CLR  = 0x000000FC;                   // clear the port lines.
        IO0SET          = temp;                         // Assign the value to the PORT lines
        IO0CLR  = 0x00000004;                   // clear bit  RS = 0
        IO0SET          = 0x00000008;           // Enable=1
        delay(10);
        IO0CLR  = 0x00000008;
}
```

```
void clr_disp(void)
{
        temp1 = 0x01;                   // command to clear lcd display
        lcd_com();
        delay(500);
}
void delay(unsigned int r1)
{
        for(r=0;r<r1;r++);
}
void init_port()
{
        IO0DIR = 0x00000FFC; //Configured LCD Lines and Rows as O/P(P0.8-P0.11) and Columns as I/P(P0.12-P0.15)
        IO0SET = 0x0000FF00;  //Set the Rows high.
}
```

15. Demonstrate the use of an external interrupt to toggle an LED On/Off.

```
#include <LPC21xx.h>

void EINT0_Init(void);
void Extint0_Isr(void)__irq;

unsigned char int_flg=0x00, flag=0x00;

int main ( void )
{
	PINSEL2 = 0x00000000;	//made P0.16 - P0.31 as GPIO
	IO1PIN = 0x00000000;
	EINT0_Init();			// initialise external int0

	while(1)
	{
		if(int_flg == 0xFF)	//check interrupt occur or not
		{
			int_flg = 0x00;
			if(flag == 0x00)
			{	// when flag is '0x00' ON the LED
				IO1SET = 0x02000000;
				flag = 0xFF;
			}
			else
			{	// when flag is '0xFF' OFF the LED
				IO1CLR = 0X02000000;
				flag = 0x00;
			}
		}
	}
}
void EINT0_Init(void)
{
	IO1DIR |= 0X02000000;	// P1.25 for LED indication
	PINSEL1 &= ~0x00000003;
	PINSEL1	|= 0X00000001;	// Setup P0.16 to alternate function EINT0
	EXTMODE	= 0x01;		// edge i.e falling egge trigger and active low
```

```
        EXTPOLAR = 0X00;
        VICVectAddr0 = (unsigned long) Extint0_Isr;        // Assign the EINT0 ISR function
        VICVectCntl0 = 0x20 | 14;              // Assign the VIC channel EINT0 to interrupt priority 0
        VICIntEnable |= 0x00004000;    // Enable the EINT0 interrupt
}

void Extint0_Isr(void)__irq             // whenever there is a low level on EINT0
{
        EXTINT |= 0x01;                 // Clear interrupt
        int_flg = 0xFF;
        VICVectAddr = 0;                // Acknowledge Interrupt
}
```

16. Display the Hex digits 0 to F on a 7-segment LED interface, with an appropriate delay in between

```
/*Description: DISPLAY ARE CONNECTED IN COMMON CATHODE MODE
Port0 Connected to data lines of all 7 segement displays*/
/*
          a
         ----
        f|  g |b
         |----|
        e|    |c
         ----  . dot
          d
        a = P0.16
        b = P0.17
        c = P0.18
        d = P0.19
        e = P0.20
        f = P0.21
        g = P0.22
        dot = P0.23


        Select lines for two 7 Segments
        DIS1   P0.28
        DIS2   P0.29
        DIS3   P0.30
        DIS4   P0.31

        Values Correspoinding to Alphabets 1, 2, 3 and 4
*/

#include <LPC21XX.h>
unsigned int delay,j;
unsigned int Switchcount=0;
unsigned int Disp[16]={0x003F0000, 0x00060000, 0x005B0000, 0x004F0000, 0x00660000,0x006D0000,
                       0x007D0000, 0x00070000, 0x007F0000, 0x006F0000, 0x00770000,0x007C0000,
                       0x00390000, 0x005E0000, 0x00790000, 0x00710000 };
int main (void)
{
        PINSEL0 = 0x00000000;
```

```
        PINSEL1 = 0x00000000;
        IO0DIR  = 0x00FF0000;
        IO1DIR  = 0x00000000;

        while(1)
        {
                IO0CLR = 0x00FF0000;        // clear the data lines to 7-segment displays
                IO0SET = Disp[Switchcount];          // get the 7-segment display value from the array
                for(j=0;j<20;j++)
                        for(delay=0;delay<30000;delay++);   // 1s delay
                                Switchcount++;
                                if(Switchcount == 0x10)        // 0 to F has been displayed  go back to 0
                                {
                                        Switchcount = 0;
                                        IO0CLR  =     0x00FF0000;
                                }

        }
}
```