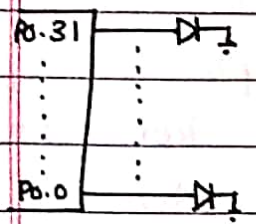
... and 9 will be written in the register.

Q) Write a program to turn on and off 8 LEDs connected to the port 0.

The circuit:



ON ← 0xFFFFFFFF

OFF ← 0x00000000

i) PINSEL0 ← 0x00000000

ii) FIO0DIR ← 0xFFFFFFFF

⟹ CODE example:

```
#include<lpcxx.h>
#include<lpc17xx.h>  >> void delay (void);
void main (void) {
    SystemInit();                          } initializing the program for
    SystemCoreClockUpdate();               } working properly. proper functioning
    LPC_PINCON -> PINSEL0 = 0;
    // LPC_ Pin Connect Block.
    LPC_PINCON -> PINSEL1 = 0;
    // Pins Configured for function 0.
    // Now putting the direction information.
    LPC_GPIO0 -> FIODIR = 0xFFFFFFFF;
    while(1){
        LPC_GPIO0 -> FIOPIN = 0xFFFFFFFF;
        // Turn all the LEDS on.
        delay();
```

// then one by one other off

```
        for(j=0; j<5000; j++);
        LED <<= 1;
    }
} // end while loop
} // end the main
```

Q.) Write a program to turn on LEDs connected to P0.4 to P0.11 whenever switch connected to P2.12 is pressed, turn off whenever the switch is released.

```
#include<lpc17xx.h>
unsigned int i,j;
unsigned long LED;
void main (void){
    SystemInit();
    SystemCoreClockUpdate();
    LPC_PINCON -> PINSEL0 = 0;
    LPC_PINCON -> PINSEL4 = 0;
    LPC_GPIO0 -> FIODIR = 0x00000FF0;
    LPC_GPIO2 -> FIODIR &= ~(1<<12);
    while(1){
        if(!(LPC_GPIO2 -> FIOPIN & (1<<12))){ // Pressed Condition
            LPC_GPIO0 -> FIOPIN = 0x00000FF0;
        }else{
            LPC_GPIO0 -> FIOPIN = 0;
        }
    }
}
```

P0.4 - P0.11 and P2.0 - P2.15 as GPIO.

1 → Not pressed
0 → pressed

This means that a voltage will be generated in that particular wire, and then there will be a potential difference across the register that is connected.

* So if the value of the column is zero, that means no key has been pressed in that particular row, and then move to the next row.

The exit condition for the key scan will be that a key is pressed. So it will stay in a sort of infinite loop until the key is pressed.

* Say a key is pressed now.

The key Scan function will just return the value of the column. Row value is not required because we know what row we are reading, then using an encoder convert the row number and column number to get the value of the key.

Q.) Write a program to display the key code on the segment digit 1.

Connector A → A–H

Connector B → Decoder.

The first four of lines of the connector will be connected to the the decoder.

That is why Connector A → h–a

Connector B → 4x4 keyboard

Connector C → Decoder.

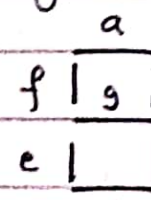
Multiplexing is not required because only one digit is to be displayed.

Connector A and Connector C are of the same port. Hence, we will have to use the required mask twice once to write to connector A and other while writing the connector C.

```
#include <lpc17xx.h>
#define FIRST_SEG 0<<15
// The above line says that, port 15 onwards them will be 0s.
// Otherway is to assign 0 to the whole port.

void Scan (void);
// Prototype of a function to scan the columns.

// declaration of the required variables.
unsigned char column, row, flag;
unsigned long int i, var1, temp, temp2, temp3,
unsigned char SevenCode[4][4] = {0x3F, 0x06,...}
// Inside the curly brackets SevenSegment codes must be written

// The main function starts from here.
// Pinsels not required by but direction must be configured.
int main (void) {
    SystemInit();
    SystemCoreClockUpdate();
    LPC_GPIO0 → FIODIR = 0xFFFFFFFF;
    // All the lines of the port are configured as output.
    LPC_GPIO1 → FIODIR = 0;
    // The full port has been configured as input.
    LPC_GPIO2 → FIODIR = 0x00003C00;
    // P2.10 - P2.13 is configured as output.

    // The infinite Loop Starts here.
    while (1) {
        for (row=0; row<4; row++) {
            if (row==0) {
                temp = 1<<10;
            } else if (row==1) {
                temp = 1<<11;
            }
```

```
else if (row == 2) {
    temp = 1<<12;
}else if (row==3) {
    temp = 1<<13;
}
LPC_GPIO2 → FIOPIN = temp;
flag = 0;
scan();    //Call the scan function.
//This will now read the columns.
if (flag == 1) {
    temp2 = SEVEN_CODE[row][column];
    //corresponding code is located for the given
    //row and column. & send that to the port.
    LPC_GPIO0 → FIOMASK = 0XFFF87FFF;
    //Unblock P0.15-P0.18 which is connected to the
    //decoder.
    LPC_GPIO0 → FIOPIN = FIRST_SEG;
    //Send dig 1 on this line.
    temp2 = temp2<<4;  //shifting the digits to the required pins.
    LPC_GPIO0 → FIOMASK = 0X FFFFF00F
    //Unblocking P0-4 - P0.11
    LPC_GPIO0 → FIOPIN = temp2;
    //It will now be displayed.
    break;
  } //if condition
 } //for condition
} //while loop
} //for main.
```

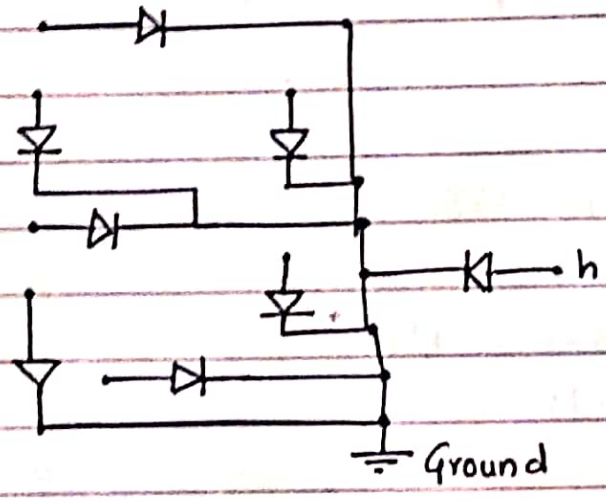
7-Segment Display:



All lines are LEDs.

h → Decimal Point

We have a common cathode configuration such as below. Anodes are open and cathodes are shorted to ground.



The common anode, all +ve are shorted and cathodes are open to Vcc.

- For common cathode, sending logic '1', The LED turns on.
- For common anode, sending logic '0', the LED turns on.
- For the 7-seg Display, sending 7-bit code we can display numbers

Ways of writing the codes for the display.

MSB → a b c d e f g h ←

OR

Similarly for all the other digits.
Now as this is being done at CPU work (clock) speed, it is faster than rate at which LED's decay intensity.
If we loop entire process, the LED's will keep renewing intensities and thus periodic refreshing is achieved. The numbers will stay on the display.

Q:) Display a 4 digit ~~number~~ counter on the multiplexed seven segment display.

```
#include <lpc17xx.h>
#define FirstSeg 0<<23
#define SecondSeg 1<<23
#define ThirdSeg 2<<23
#define FourthSeg 3<<23

unsigned int digitCount = 0;
unsigned int dig1=4, dig2=3, dig3=2, dig4=1;
unsigned char arrayCodes[ ]= { };
        // 7 Segment codes array.
unsigned long int temp1, temp2, temp3;
unsigned long int i=0;
void Display (void);
void Delay (void);

void main (void) {
    SystemInit();
    SystemCoreClockUpdate();
    LPC_GPIO0 → FIODIR = 0x00000FF0;
    LPC_GPIO1 → FIODIR = 0x07800000;

    while (1) {
        Delay();
        //optimal delay so that process is not overburdened
        digitCount += 1;
        if (digitCount == 5) {
            digitCount = 0;
        }
        if (flag == 1) {
            flag = 0;
            dig1 += 1;
            if (dig1 == 0x0A) {
                dig1 = 0;
                dig2 += 1;
                if (dig2 == 0x0A) {
                    dig2 = 0;
                    dig3 += 1;
                    if (dig3 == 0x0A) {
                        dig3 = 0;
                        dig4 += 1;
                        if (dig4 == 0x0A) {
                            dig4 = 0;
                        }
                    }
                }
            }
        }
        Display();
    } // for while loop
} // for main

// function for displaying.
void Display (void) {
    if (digitCount == 1) {
        temp1 = dig1;
        LPC_GPIO1 → FIOPIN = FirstSeg;
    }
```

```
    else if (digitCount == 2) {
        temp1 = dig2;
        LPC_GPIO1 → FIOPIN = SecondSeg;
    } else if (digitCount == 3) {
        temp1 = dig3;
        LPC_GPIO1 → FIOPIN = ThirdSeg;
    } else {
        temp1 = dig4;
        LPC_GPIO1 → FIOPIN = FourthSeg;
    }
    temp1 &= 0x0F
    //Just to check whether it is a digit or not.
    temp2 = arrayCode[temp1];
    LPC_GPIO0 → FIOSET = temp2<<4;
}

//function for the delay.
void Delay (void) {
    for(i=0; i<500; i++) {
        if (count == 1000) {
            count = 0;
            flag = 1;
        }
        count++;
    }
}
```

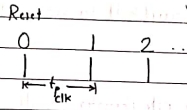
Timer and Counter Programing:

If we know the clock we can then calculate the time delay accordingly.

* If we use counter to provide the time delay then it is called a timer.

Timer 0, 1, 2, 3

(32-bit) counts from $0 \rightarrow (2^{32}-1)$



There is an oscillator of constant frequency on the chip.
from this the system clock is set
from the system clock, peripheral clock is set.

* t_{PCLK} = period of peripheral clock.

for most of the experiments we will be performing, the frequency of the peripheral clock is set to 3 MHz.

$\therefore T_{PCLK} = 1/3\times10^6 = 1/3\ \mu s.$

The clock counts at each positive edge.

* Each timer has a set of many SFRs.
  i) Timer Counter Register (TC)
  ii) Prescale Counter Register (PC)
  iii) Prescale Register, (PR).
  iv) Timer Control Register (TCR)

i) TCR

→ It is a 32 bit register.

→ Only bit 0 and 1 are useful.

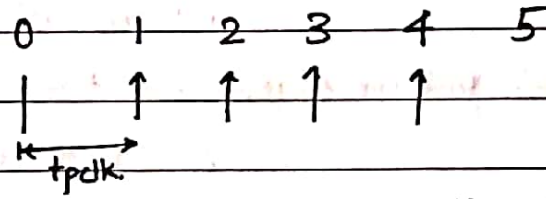
0 → Enable → (1) → Enable
(0) → Disable.

1 → Reset → (1) → Reset
(0) → Normal.

As long as the reset is one, even though we give the clock it won't count up.
So after reseting we have to switch it to 'no reset' state for the counter to count up.

* At each positive edge the Prescale Counter Counts up.
* PreScale register is used to scale the rate at which the counter counts up.

How it is done?
We store a particular value, say 4, in the prescale register. ~~when~~ PC counts from 0 ~~with count~~ till the PR's value. When the PC register value equals the PR's values then, as soon as it gets a positive cycle the PC register will be reset.



PC will be incremented till $(PR+1)t_{PCLK}$ positive edges.
The TC will count up when the PC resets.

NOTE:
* If we initialize the PR as 0, then at every positive edge the TC will count up.
* The rate at which TC counts depends on the PR value.

Now the time delay becomes,

$T_d = (PR+1)\, t_{p_{CLK}} \cdot (MR+1)$.

Q:) Calculate possible solution for getting the time ~~pr~~ delay.

$f_{PCLK} = 3$ MHz.

$$t_d = \frac{(PR+1)(MR+1)}{f_{PCLK}}$$

$(PR+1) \rightarrow 1000$ and $(MR+1)$ 3000

OR

$(PR+1) \rightarrow 3000$ and $(MR+1) \rightarrow 1000$

OR

$(PR+1) \rightarrow 1$ and $(MR+1) \rightarrow 3000000$.

The CTCR (Counter/Timer Control Register).

It is a 32 bit number.

When the 2 LSBs are 00, then it works as a timer i.e it uses the system clock.

When the 2 LSBs are 01, 10 or 11, it works at a counter and uses an external clock.

i) if 01 → counts ~~RF~~ at positive edge

ii) if 10 → counts ~~Down~~ at negative edge.

iii) if 11 → counts at both the edge.

Now, the 2 and 3rd bit tell us which pin to look for.

00 → CAPn.0 → P1.26 function3

01 → CAPn.1 → P1.27 function3

if 00, then it will count at capn.0's edge which is defined by the bits at 0 and 1 position.

where n can be the timer number.

If we apply 0000 → (pins 3210) then it will count at each positive edge of the system clock.

Match Control Register: (MCR)

It is a 32 bit register.

For each timer, there are four registers associated to them.

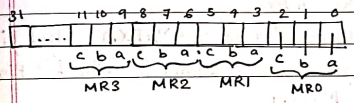
All the match register can be configured as per our requirement.

MR0 = TC ⟶ Match 0 event

MR1 = TC ⟶ Match 1 event

MR2 = TC ⟶ Match 2 event

MR3 = TC ⟶ Match 3 event.



where a = Interrupt on match

b = Reset TC on match

c = Stop TC and PC on match. (Down Reset)

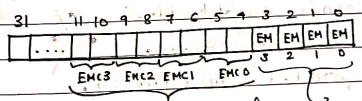
We will have to configure as per our requirement

NOTE:

Out of b and c we will use any one.

External Match Register: (EMR)

It is a 32 bit register.



External match control bit for match 0, 1, 2, 3.

match output for match 0, 1, 2, 3.

How to know whether we got our required delay or not?

In this situation we will use the ~~ME~~ EMC and EN bits ~~resp~~ respectively.

If we want to bring the status of the EMR flags to the external pins for some purpose then that is also possible. We can internally short the pins with the register.

So inorder to do the above thing, we use MATX.Y output pins.

In this only the EM pins are made available on the I/O Pins.

MATX.Y

In the above format X = Timer Number

and Y = match output.

eg:) Using Timer 0.

TC = MR0 ⟶ MAT0.0 ⟶ P1.28 function 3

TC = MR1 ⟶ MAT0.1 ⟶ P1.29 function 3.

TC = MR2 ⟶ MAT 0.2

TC = MR3 ⟶ MAT 0.3

We will have to look into the manufacturer manual to know which MAT is linked to which function of the pin.

EM1 can be brought to P1.28 iff configured.

This whole method/procedure is called PINOUT.

NOTE: Capture Event: to get the instantaneous value of the timer/counter after a particular event.

For this purpose there are two capture registers.

```
//At the end we will enable this otherwise it will start counting
//right away. We will load the values first.
LPC_TIM0 -> CTCR = 0;
LPC_TIM0 -> PR = 999;
LPC_TIM0 -> MR0 = 2999;          } //Setting these values for 1 second.
//Once the match occurs we'll stop the TC and PC counter.
LPC_TIM0 -> MCR = 0x04;
//We will capture this event in the EMR.
LPC-TIM0 -> EMR = 0x20; //Set EM0 upon match.
//The above statements can be written in any order.
//Keep-polling the EM0 in the while loop. Instruction execution
//delay must be neglected as circuit clock is way faster than
//circuit clock.
LPC-TIM0 -> TCR = 0x01; //Start the clock.
//Poll to check whether EM0 is one or not.
while(!(LPC_TIM0 -> EMR & 0x01)) { ; }
return;
}
```

We are ensuring that next time when we can the function we reset everything and hence we don't reset PC and TC instead stop them.

```
void main () {
    LPC_PINCON -> PINSEL3 = 8<<22;
    LPC_GPIO0 -> FIODIR = 0x04;
    while (1) {
        LPC_GPIO0 -> FIOPIN = ~(LPC_GPIO0 -> FIOPIN & 0x04);
        delay ();
        // Wait for 4 clock pulses. (positive edge)
    }
}
void delay () {
    LPC_TIM0 -> TCR = 0x02;
    LPC_TIM0 -> CTCR = 0x05; //positive edge of CAP0.1
    LPC_TIM0 -> PR = 0;
    LPC_TIM0 -> MR1 = 3;
    LPC_TIM0 -> MCR = 0x20;
    LPC_TIM0 -> EMR = 0x80;
    LPC_TIM0 -> TCR = 0x01;
```

cursor d/OFF

shift Display Cunor	0	0	0	1	S/C	R/L	-	-
function let	0	0	1	DL	N	f	-	-
Entry Mode	0	0	0	0	0	1	I/D	S

Where,

I/D → Increase/Decrease, D-Display ON.

S → Shift Display, C- Cursor ON.

↳ 0 — NO

1 — Yes.

B → Blinking ON

S/C → shift display/Cursor

R/L → Right/Left

DL → Display length

N → Number of lines.

F → Font Size

* 0x01 — Clear

0x02/0x03 -

Cursor goes to the first line, first character position.

0x06 — increment the cursor.

B — character at current cursor position blinks.

0x0C — display ON, cursor & blinking OFF.

0x0E → display ON, cursor ON, blinking OFF.

R/L → 0 — ~~left~~ left

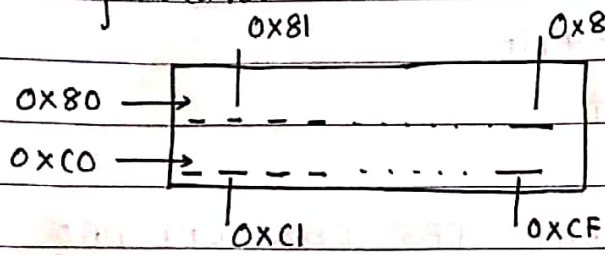
1 — Right

S/C — 0 — Cursor

1 - Display

initially the LCD will be configured with 8 bit mode, we change it to 4 bit mode by sending the command 0x20. The command is in 8 bit format, after sending the LCD will be configured into 4 bit. Then we send 0x28 which sets the font size and everything.

Setting the cursor:

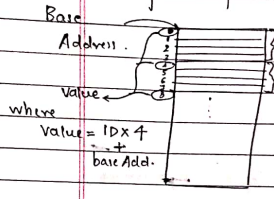



```
#include<lpc17xx.h>
#define RS_CTRL  1<<27
#define EN_CTRL  1<<28
#define DT_CTRL  15<<23
unsigned long int temp1=0, temp2=0, i, j;
unsigned char flag1=0, flag2=0;

// Declaring the message to be displayed on the LCD.
unsigned char msg[] = {"WELCOME"};

// Function Prototypes.
void lcdWrite(void);      // high level function sending the data
void portWrite (void);    // I/O level function, converts the data into bin.
void delayLCD (unsigned int);

// Configuring the LCD as per our requirement.
unsigned long int initCommand[] = {0x30, 0x30, 0x30, 0x20, 0x28,
                                   0x0C, 0x06, 0x01, 0x80};
```

```
    if (!flag2) {
        temp2 = temp1 & 0x0F;
        temp2 = temp2 << 23;
        portWrite();
    }
}

// portWrite Function.
/* The function takes the input, converts it into I/O level and sends it to
   the LCD as required */
void portWrite (void) {
    LPC_GPIO0 -> FIOPIN = 0;
    // Clear all the I/O lines.
    LPC_GPIO0 -> FIOPIN = temp2;
    // The pins 23-26 receive the required digit/character now.
    if (flag == 0) {
        LPC_GPIO0 -> FIOCLR = RS_CTRL;
    }else {
        LPC_GPIO0 -> FIOSET = RS_CTRL;
    }
```

get

After that we will the location of the vector in the interrupt Vector table for that particular requesting table.



After getting the address of the ISS, the NVIC will load that to the PC.

↳ Program Counter.

Q.) How to tell the system whether it has to use Interrupts or not?

Enabling the Interrupt:

Will have to enable at two locations

i) Inside the NVIC level (Globally)

ii) Inside the device. (locally).

After this the interrupt will be working.

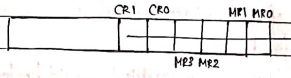
i) Whenever Request comes, Service it.

ii) Whenever you want a service, request it.

A.) Timer Interrupt:

The Interrupt Register (IR):

It is a 32 bit register.

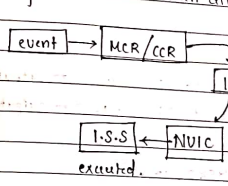


Tell the timer when MR0,... CR0 is toggled, toggle the flags in the IR as well.

When we enable in CCR and MCR then it is local that is it is within the timer.

Whenever match event occurs, the match flag in IR is set provided the MCR is set. Similarly for capture event enable it and it will be reflected here.

NOTE: NVIC just has to see the content of IR. It just has to be whether the value is 0 or not, if 0 then no request pending and if it is anything other than zero then atleast one request is pending.



NVIC in real time scans the IR register continuously.

* Upon completion of I.S.S., the IR must be cleared either automatically or manually else it will continuously executes the I.S.S.

Syntax for enabling timer in for NVIC:

NVIC_EnableIRQ(TIMERm_IRQn);

where m is the timer number.

→ Whichever device we want, just change the ~~name~~ timer.

NOTE: FunctionName for ISS is predefined.

void TIMER0_IRQHandler(void); // function Prototype.

Q.) Write a program to toggle an LED connected to P0.2 every second while displaying status of the switch connected to P1.0 on the LED connected to P2.0.

Assume PCLK = 3MHz.

Ground Work: P0.2 → output.

P1.0 → input

P2.0 → output.

We will be using timer0 ~~for ease~~.

```
#include<lpc17xx.h>
void TIMER0_IRQHandler (void);
void timer0_init (void);                    // User defined function.
int main (void){
    //Configuring the pins.
    LPC_GPIO0 -> FIODIR = 1<<2;      // P0.2 ← output
    LPC_GPIO2 -> FIODIR = 1<<0;      // P2.0 ← output
    // Enabling timer0 interrupts in NVIC.
timer0_init();  →
    NVIC_EnableIRQ (TIMER0_IRQn);
    while (1) {
        LPC_GPIO2 -> FIOPIN = LPC_GPIO1 -> FIOPIN & 0x01;
        // for switch, if pressed it will send 1 to GPIO2 else 0 to GPIO2
        // No shifting is required as the pins are the same, if they aren't
        // we will have to shift the bit to the required position.
    }
}
// Initializes the timer to generate interrupt each second.
//function for timer (local).
void timer0_init (void) {
    LPC_TIM0 -> TCR = 0x02; //Reset the timer.
    LPC_TIM0 -> RR = $0;
    LPC_TIM0 -> MR1 = 3000000-1; // timer for 1 second.
    LPC_TIM0 -> CTCR = 0;
    LPC_TIM0 -> MCR = 0x08; //Generate interrupt upon match 1 event.
    //The above step is called locally enabled interrupt (within the
    //timer).
    LPC_TIM0 -> TCR = 0x01; //start the timer.
}

//Interrupt Service Subroutine.
void TIMER0_IRQHandler (void) {
    //After entering this just toggle the LED.
    LPC_GPIO0 -> FIOPIN = ~(LPC_GPIO0 -> FIOPIN & 0x02);
```

```
void main(void) {
    LPC_PINCON -> PINSEL3 = 3<<22;
    counter0_init();
    timer1_init();
    NVIC_EnableIRQ(TIMER1_IRQn);
    while(1);
}
//If we don't write while(1), the program will terminate and
//nothing will happen. Any parallel operation can be put into the
//while loop.
```