demo of the project
-> simulator
-> problem statement

1. select the washing program
2. select the water level
3. start
4. function selected and time
machine fan rotation,

automatic gate opening system(pure hardware)
1.sensor -> ir
2.compator
3. relay ->
4. timers

uc -> hardware software

design ->

stand alone
take decision on its own
example tesla cars :-
location airport , road construction
reroute

real time;- ES which will complete the task within deadline
ex ;-missile system , air bag

networked
ES which can connect to other devices and able transmit data and recive

mobile
ES which you carry it from one place to another

hybrid ES

processing unit
up , uc , soc

memory

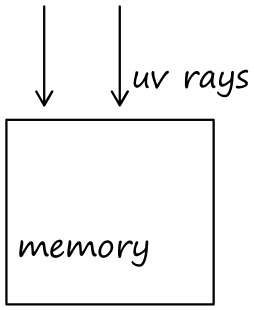
RAm: random acess memory
-> write/read = infinite times
volatile memory -> requires powersupply to retain the data
if there is no power supply the data will be erased
sram:- constant power to retain the data , cost more
dram:- more power
constant power supply + extra power supply to refresh the data

Rom:read only memory
-> write =one time read -> infinite times
non volatile memory
-> doest require the power supply to retain the data

rom

OTP/PROM ;-one time programable memory ,
W -> 1 , r -> infinite
ex: micro oven, toys -> mobile

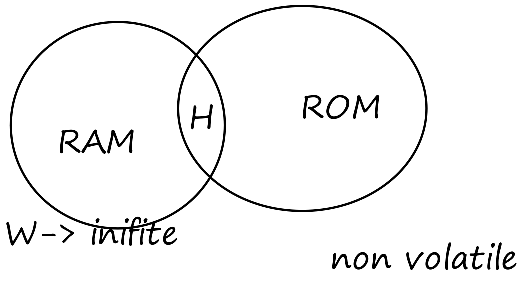
EPROM/UVROM;-
R-> infinite , W-> n times
ex : Rand D



masked ROM;-
data stored below this memory will never get currpted

ex man fact date , chip id

taken selfie :-
ram ->
deleted
rom->
i cannot delete



hybrid memories

EEPROM -
-non volatile memory
-persistant
-> r - infinite w -> 1 million times
byte acessable
size -> 256 bytes to 128 KB.
ex -> number or count or any state
Flash memories

Flash memories:-

nor-> byte accessable

non volatile memory

r- infinite w ->n

xip -> execute in place -> need not load code in ram for execution

store code in micro controllers

nand-> bolck acess able -> store data in blocks 256 bytes , 512 bytes

non volatile

r- infinite w ->n

less reliable

nand -> data is stored

ex: sd cards , pendrives

overview about the tools

simple program

switches ->

clcd ->

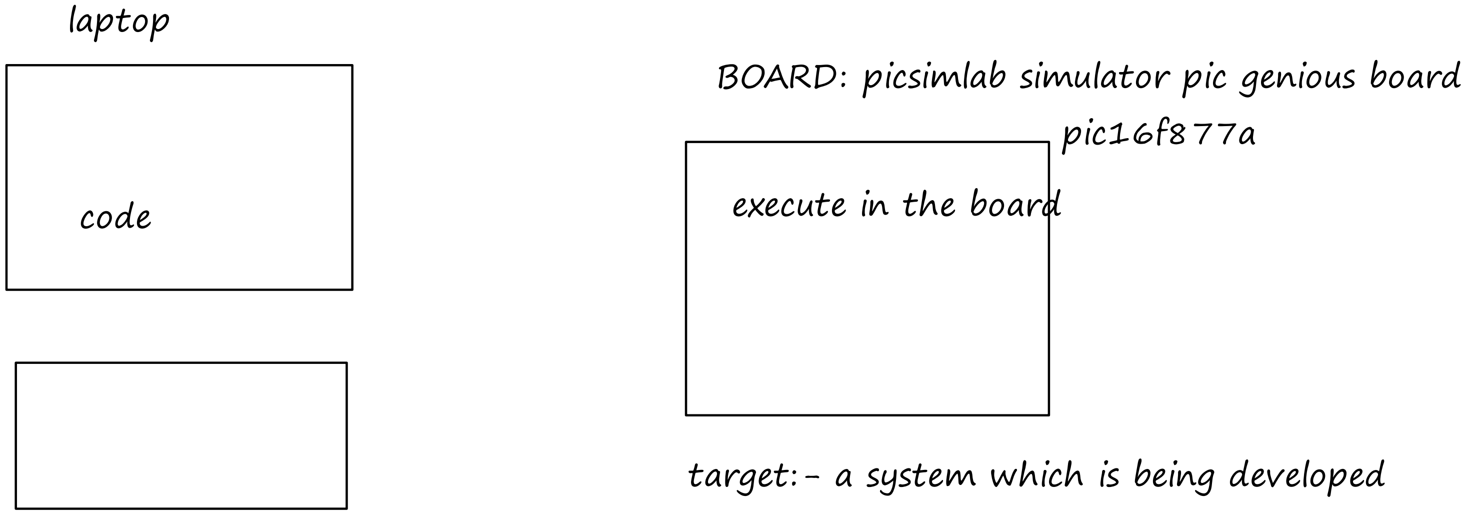
timer ->

buzzer->

fan/ motor ->

refer requirement document and start implementing the requirements one by one

host :- system which is used to develop the target



laptop

code

BOARD: picsimlab simulator pic genious board

pic16f877a

execute in the board

target:- a system which is being developed

C programing
wrote ->laptop
->execute laptop

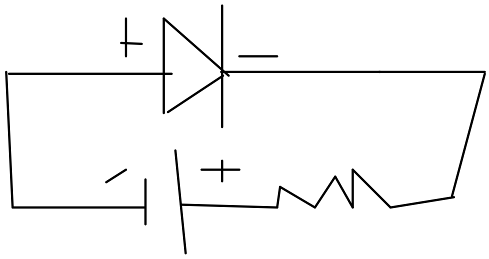
uc
development ->laptop -> c code = execute able file
being developed = board ->

host:
sytem which is used to develop the target
laptop
target :
system which is being developed for particular purpose
simulator -> picsimlab
board :- pic genious
controller : pic16f877a

First code :- simple pheripheral available board
less complication ,less overhead
code is simple
-> hardware is working or not
-> tools properly installed
-> conection host and target eshtablished or not

what pheripherals available on the board

FB
condtion
ON

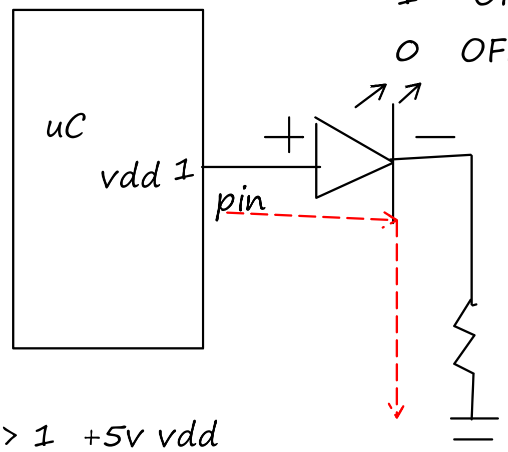


RB
OFF

interfacing an LED to uC

pin	LED
1	ON
0	OFF

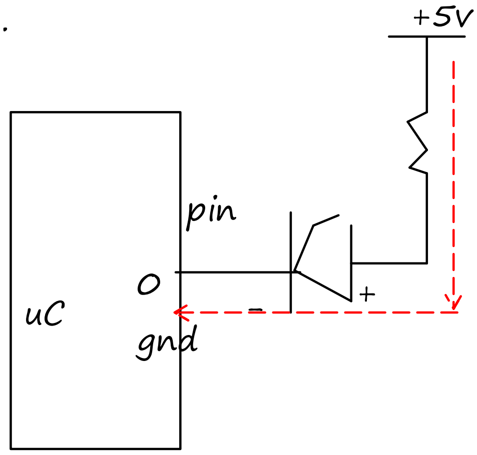
1



pin -> 1 +5v vdd
0 gnd

sourcing circuit

2.



pin	LED
1	OFF
0	ON

sinking circuit

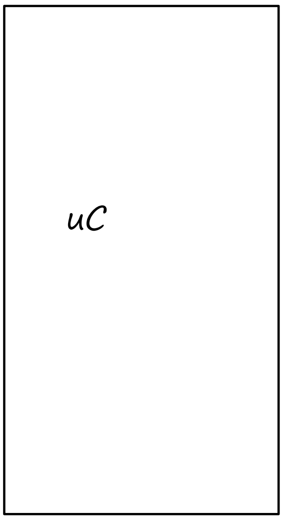
WAP to blink the LED?

-> 16 LEDs

where the LEDS are conected?

how they are connected?

```
main()
{
  int led;

  led = 1;

  led = 0;

}
```



i/O ports

external pheripheral to your micro controller

fixed number of I/O ports

pic16f877a

architecture of the uc

data sheet -> technical document information of the controller pic16f877a -> 5 ports

33i/o pin

led-> schematic -> blue print of the board

PORTD , PORTB

sourcing

1 -> ON , 0 -> OFF

bidirection out put / input port

PORTB -> 8 bit wide

RB7 RB6 .... RB0

PORTB -> leds

data sheet ->

DDR

TRISB -> 8 bit

TRISB7 6 0

TRSIB = 0000 0000

portb pin will be output pin

TRISB = 1111 1111

portb pin will be input pin

PORTB -> 0x06

TRISB -> 0x86

<xc.h>

include this header

TRISB

-> 0x00

unsigned char * portb = 0x06;

WAP to blink the LEDs connected to PORTB

8 leds -> poRTB , sourcing circuit

PORTB

DDR -> TRISB = 0x00 -> output pins
                    0xFF -> input pins

pointers -> <xc.h>

```
step 1
 config the led port as output port
step2
     turn on the leds
 delay
     turn off the leds
delay
  goto step2
```

led :-

mc -> 5 ports

led -> PORTB , sourcing circuit fashion

DDR -> TRISB

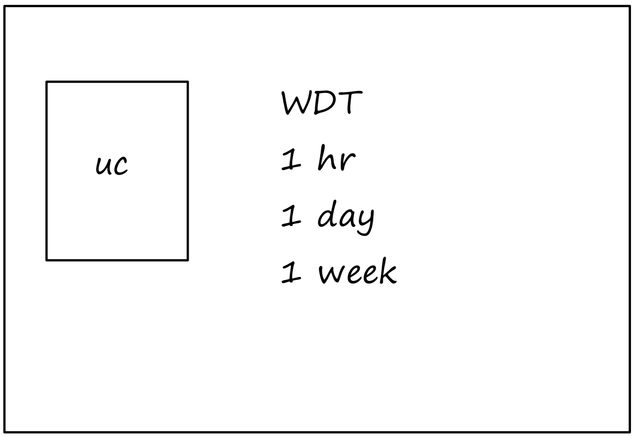
WAP to toogle the LED

```
#include <xc.h>
#pragma config WDTE = OFF

void init_config(void)
{
// one time initialisation code
}

void main(void)
{
    init_config();
    while (1)
    {
     //logic
    }

}
```

xc.h ->

watch dog timer

reseting the uc for configured time



pheripherals :-

*lcd

*switches

*buzzer

*fan

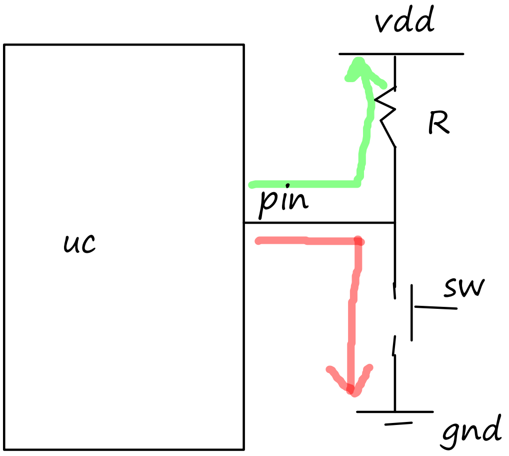
*timer

Switches: tactile switches

interfaced with micro controller

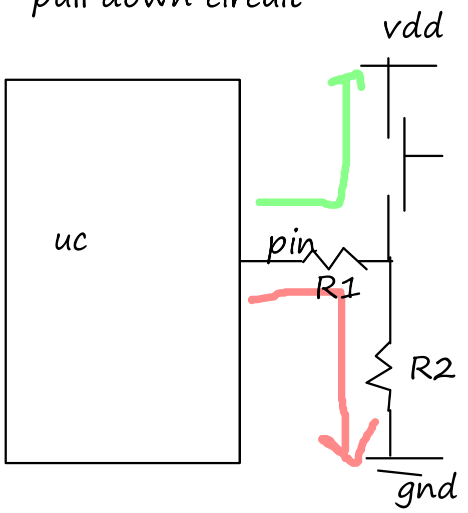
detecte the switch pressed on the mic

pull up circuit



pin	sw
0	prssed
1	released

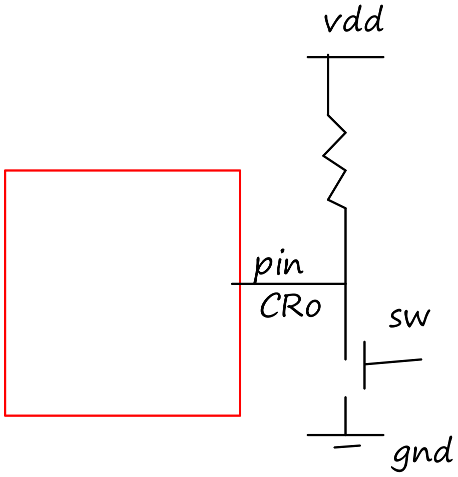
pull down circuit



R2> 10r1

pin	SW
0	R
1	P

detection type



level triggering

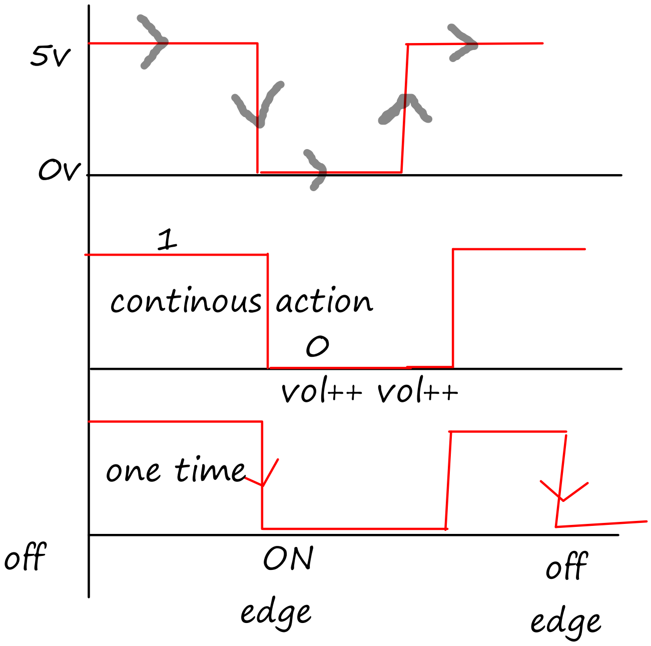
edge trigerring

vol button

-> vol++
level trig

power button

->on /off



level : to trigger the task based on the value

edge : to triggger the task based on the change in the value where the switches are connected and how they are connnected

6 switches -> rbo to rb5 , pull up circuit

WAp to toogle the led when sw is preesed

led -> rd0 , sw -> rb0

toogle as long as switch is pressed

Rb0 = 0

PORTB

TRISB =

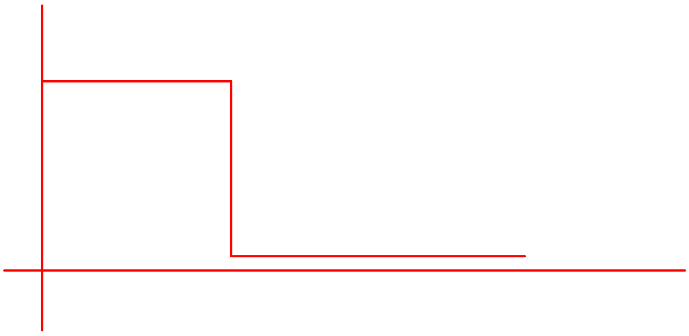
RB0 = input

TRISB0 = 1;

PORTD =

TRISD = 0x00

level triggering



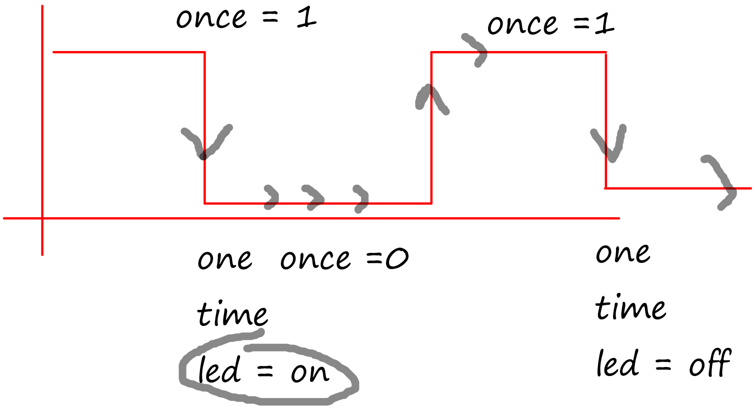
edge trigger

one time action

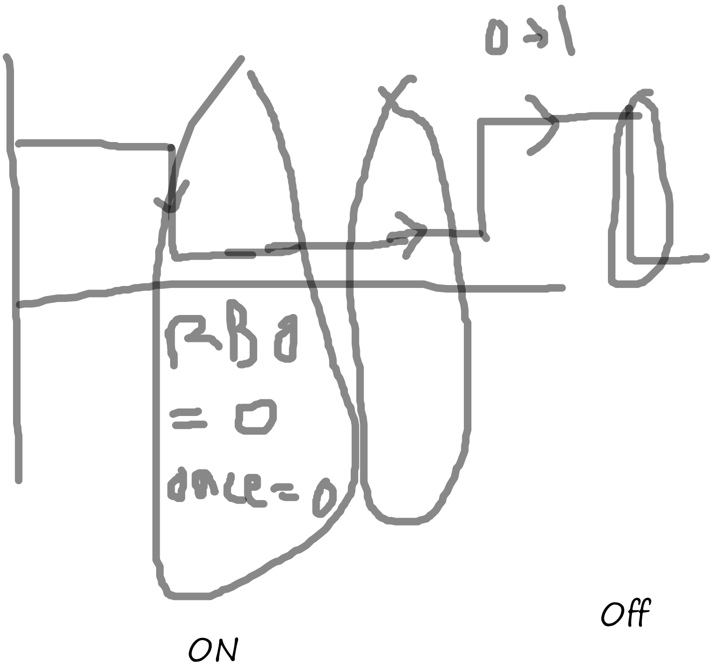
once = 1

change state led

once =0



```
while (1)
  {
      /*check if the switch is pressed*/
      if(RBO == 0 &&  once )    1
      {
          PORTD = ~PORTD;
          once = 0;
      }
      if(RBO == 1)
      {
          once = 1;
      }


  }
```

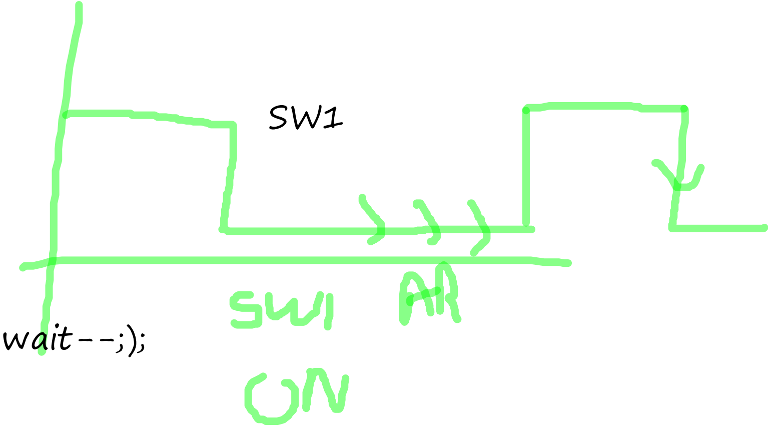


```
voidinit_digital_keypad(void)
{
/*to config RB0 to RB5 as input pins*/
KEYPAD_PORT_DDR = TRISB| 0x3F
}


void init_config(void)
{
    /*config led port as output port  PORTD*/
    TRISD = 0x00;
    PORTD = 0x00;

    /*config RB0 to RB5 pin as input pin*/
    init_digital_keypad();



}
```

```
PORTB = RB7  6   5   4   3   2   1   0
TRISB =      x   x   x   x   x   x   x   x
      =>     x   x   1   1   1   1   1   1

TRISB = x x x x x x x x
        |0 0 11  1111
        = x x 11 1111
     -> 0x3F
TRISB = TRISB | 0x3F


TRISB = 00 11  1111
        RB7 , RB6
```

```
#define LEVEL  0
#define STATE   1

void main(void)
{
    unsigned char once = 1, key ;
    init_config();
    while (1)
    {
        /*check if the switch is pressed*/
        key = read_switches(STATE)
        if(key == SWITCH1)
        {
            PORTD = ~PORTD;
           for(unsigned int wait = 50000; wait--;);
        }
    }
}
```

AR

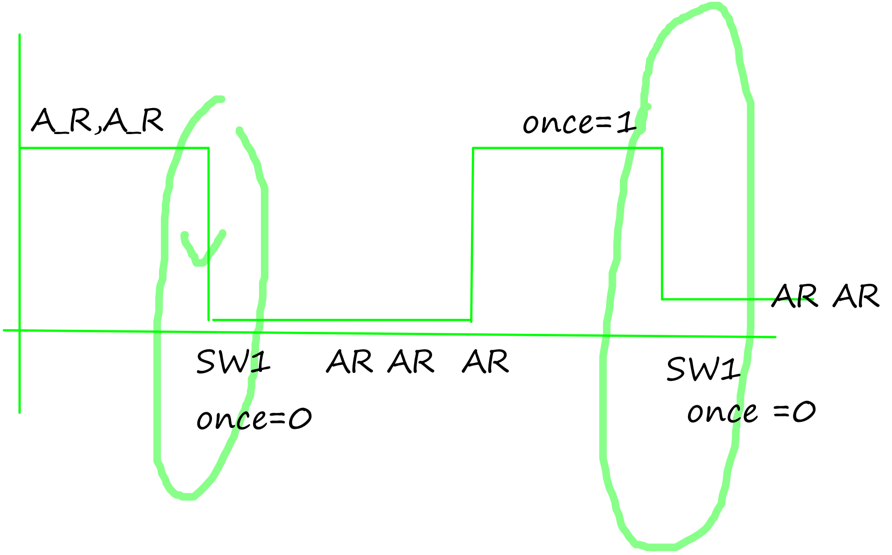


```
#define ALL_RELESED    0x3F
#define   SWiTCH1        0x3E
#define  SWITCH2        0x3D
#define  KEYPAD_PORT        PORTB
#define KEYPAD_PORT_DDR  TRISB
#define INPUT_LINES          0x3F
```

SWITCHES

```
PORTB = rb7 6 5 4 3 2 1 0  PORTB&0x3F
    &     =  x  x 1 1 1 1 1 1
             0  0 1 1 1 1 1 1
          => 0  0 1 1 1 1 1 1
no swit =  x  x  1 1 1 1 1 1   0x3F
switch1 =  x  x  1 1 1 1 1 0   0x3E
switch2 =  x  x  1 1 1 1 0 1   0x3D
```

```
unsigned char read_switches(detection_type)
{
     static unsinged char once = 1;
        if (detection_type == LEVEL)
         {
              return  (KEYPAD_PORT & INPUT_LINES);
         }
        if (detection_type == STATE)
        {   /* if any switch is pressed*/
            if((KEYPAD_PORT & INPUT_LINES) != ALL_RELEASED && once )
            {
                once = 0;
                return (PORTB & 0x3F)
           }
            if(PORTB &0x3F== ALL_RELEASED)
            {
                once = 1;
            }
             return ALL_RELESED;
        }
}
```



project

main.c

main.h

digital_keypad.h

switches

digital_keypad.c

init_digital_keypad()

read_digital_keypad()

sw 2 =

toogle alternate led

portb = 0xaa   1010 1010

portb = 0x55 0101 0101

```
main()
{
        init_config();
        while(1)
        {
             key= read_digital_keypad(LEVEL)
                if(key == SWITCH1)
                {
                        code
                }
                if(key == SWITCH2)
                {
                        code
                }
        }
}
```

read_digital_kepad->

level->

as long as switch is prresed

which switch is preesed

all_realsed

state:-

return one time

which switch is pressed

example

vol++ -> level

entering the password -> state

SW1
pattern1
alternate leds
sw2
toogle the nibble
4 led on
4 led off
sw3
toogle al the leds

PORTB = 0 1 0 1 0 1 0 1 -> 0x55
= 1 0 1 0 1 0 1 0 -> 0xAA
PORTB = 1 1 1 1 0 0 0 0 -> 0xF0
0 0 0 0 1 1 1 1 -> 0x0F
pORTB = 1 1 1 1 1 1 1 1 -> 0xFF
0 0 0 0 0 0 0 0 -> 0x00

0x55 - 0101 0101
~ 0xAA- 1010 1010

```
while(1)
{

// application code



}
```

LED->

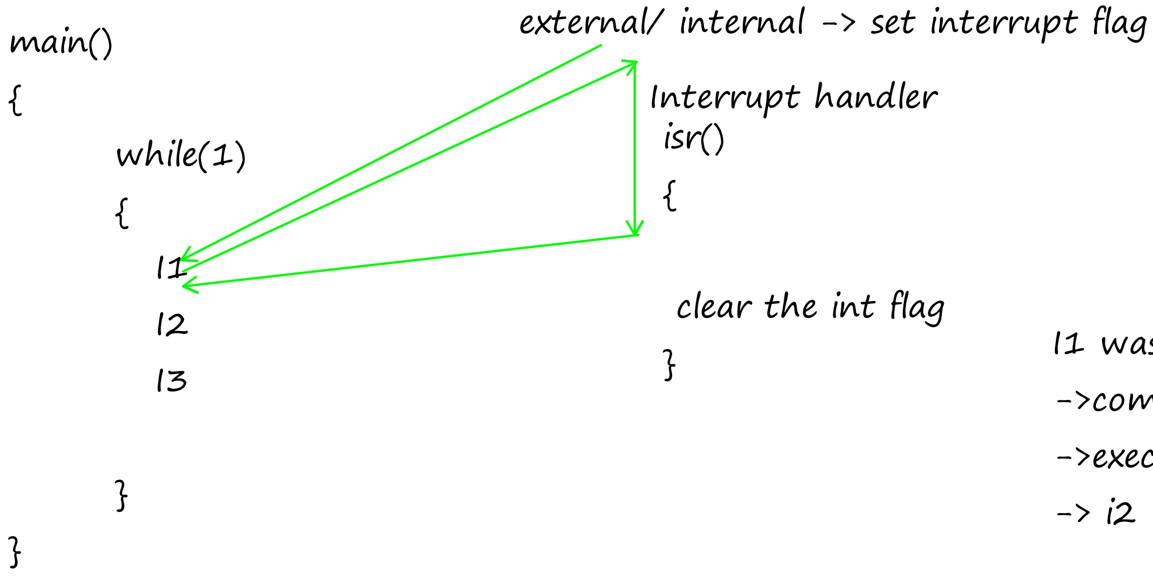
SWITCHES -> switches for washing machine

timer -> interrupt source

interrupt ->

high priority execution

timers ->



external/ internal -> set interrupt flag

```
main()
{
        while(1)
        {
            I1
            I2
            I3

        }
}
```

```
Interrupt handler
 isr()
 {

  clear the int flag
 }
```

I1 was being executed

->complete the i1 instruction

->execute the isR()

-> i2

interrupt handlers

-> IVT

timers	0x11
external	0x22

.

.

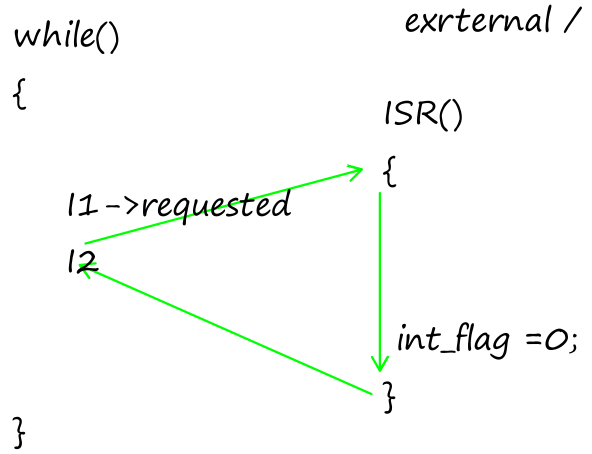
.

pic isr()-> all interrupt source

```
interrupt service routine()
{


}
```

-> ISR

exrternal / internal -> setting int_flag



```
while()
{
     I1->requested            ISR()
     I2                       {

                               int_flag =0;
                              }
}
```

1. i1 should be completed
2. PC -> stack
3. PC -> *(isr)

```
case1:      I1
      PC -> *(I2)

case2:      I1
        PC -> *(isr)
        *(I2)saved  on stack registers
```

interrupt latency :- delay in execution of the isr.

ISR will not be execute as soonas interrupt is requested

delay

-> completeion I1 dealy

-> PC

-> priorty execution

-> setting int flag

-> isr() function wiil be called

-> clear int_flag

timers: default phperpheral uC

calculate or tack the time

timer

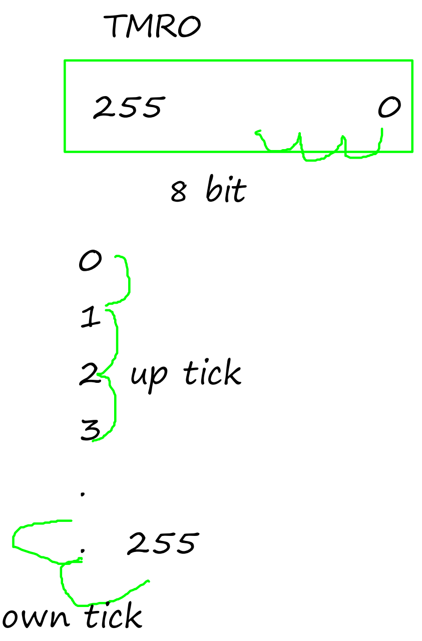
1

timer regiters

2

3

4



resolution : width of the timer register

ex : 8 bit , 16 bit, 32

tick : UP tick , down tick

Quantum : time taken by one tick

sytem clock setting-> F =

1 tick -> 4 clk pulse -> 4 * t

time -> 4 * 1/f

Q -> 4 * 1/ 20 * 10^6

= 0.2 us -> 200ns

1 tick -> 200ns

2 tick -> 400ns

time = no of ticks * Q

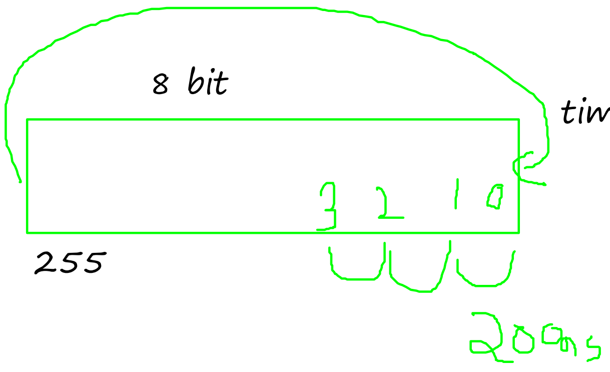
time= 255 * 200ns

start timer

start

counting ticking

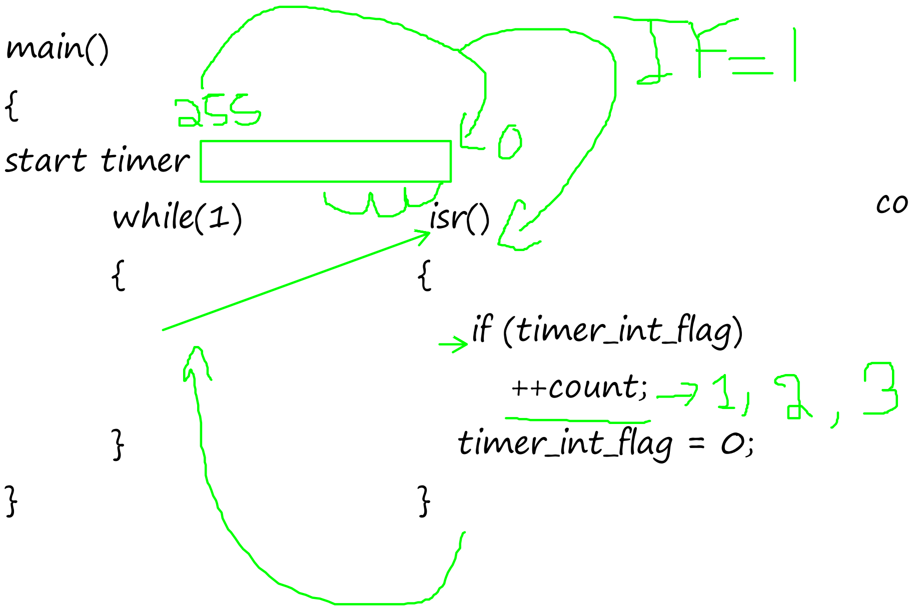
0



timer_int_flag

overflow-> 0

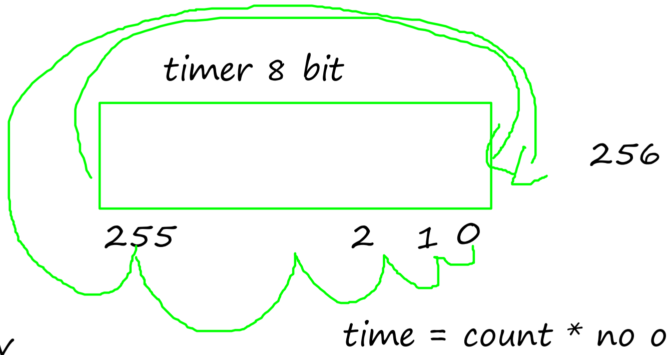
request interrupt-> isr()



```
main()
{
start timer
        while(1)          isr()
        {                 {
                           if (timer_int_flag)
                              ++count;
                          timer_int_flag = 0;
        }
}                         }
```

count = 1

time taken by one overrflow



= 256 *200ns

= 51200ns -> 51.2us

time taken count overflow

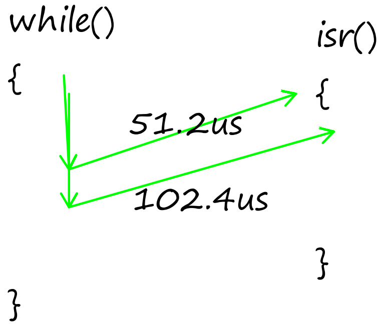
time = count *time taken 1 ov

time= count * no of tick in ov * Q

= 2 * 256 *200ns -> 102.4us

time = count * no of tick in ov * Q * P

scaling : way to increase the time to go to isr



prescaling: Q is scaled

1:1	Q= 1IC-> 200ns	ov-> 51.2us
1:2	Q= 2IC ->400ns	ov -> 102.4us

postscaling: after how many ov isr()

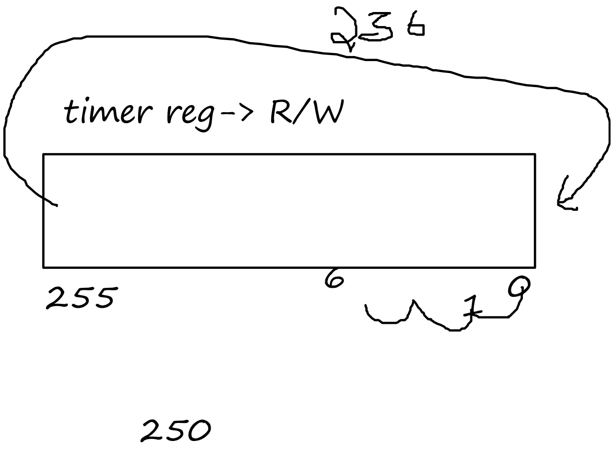
1:1	1ov-> 51.2us	1 ov isr () will be called
1:2	51.2us	2 ov isr() will be called
1:4		4 ov isr()

R = 8 bit -> 256

R = 16 -> 65536

time = count * no of tick in ov * Q * P

$$count = \frac{time}{P * Q * R}$$



example : calculate count for the folowing

P 1:1 , Q = 200ns , R -> 8 bit = 256 , time = 1 sec

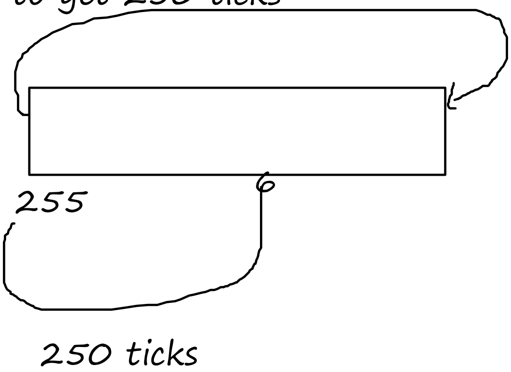
count = 1sec/ 1 * 200* 10^ -9 s * 250

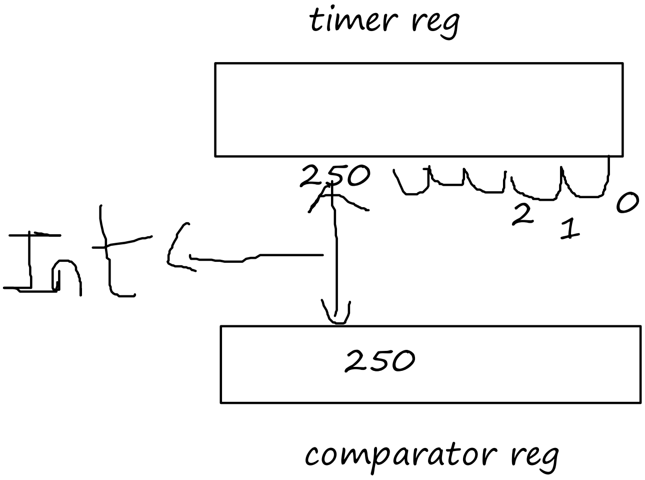
count = 19531 .25

count = 20000

timer -> over flow ->int

how to get 250 ticks





timer

resolution

tick

q

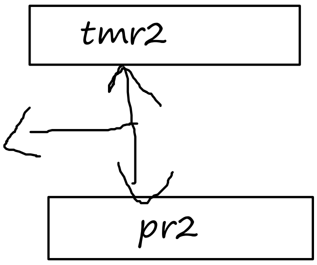
ov

int

count = time / p*q*r

WAP to toggle the led for every one second using timers

use timer2



resolution = 8 bit

TMR2 = timer register

PR2 => how tick for o.v

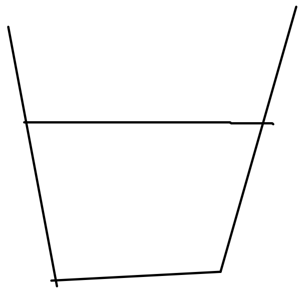
timer2 int flag

-> TMR2IF = 1

intcon

-> gie -

-> peie -



10 min to be full

5 min->