Real Time Operating System

Lecture (1)
RTOS Concepts

This material is developed by IMTSchool for educational use only
All copyrights are reserved

Super Loop(BackGround)

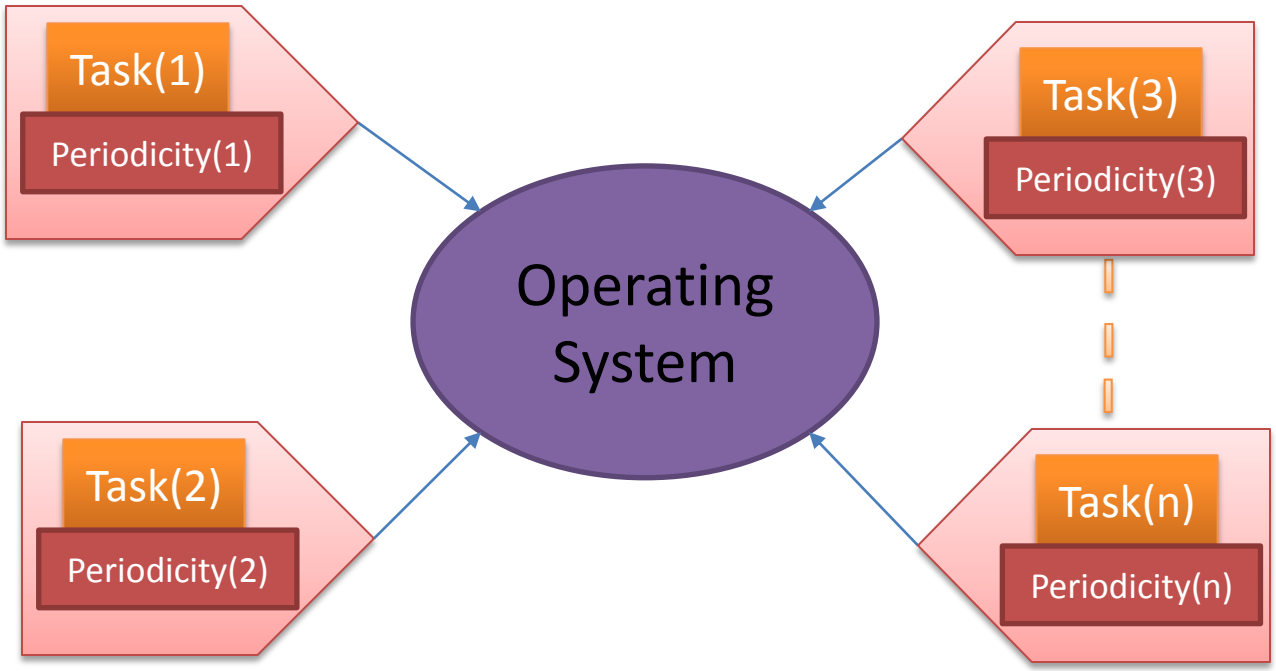
Consider we have 5 LEDs and each one has a blinking time (1 , 1.5 , 0.5 ,5 , 3.5)ms respectively what would we do to design this system ... ?

We can use a timer which fires every 0.5 ms. Inside the ISR we can increment a counter. According to the counter value we will blink the LEDs in a super while(1) loop.

What if we have 100 LEDs and every one has a different blinking time … ?
It will be more complex now to Synchronize between LEDs right !

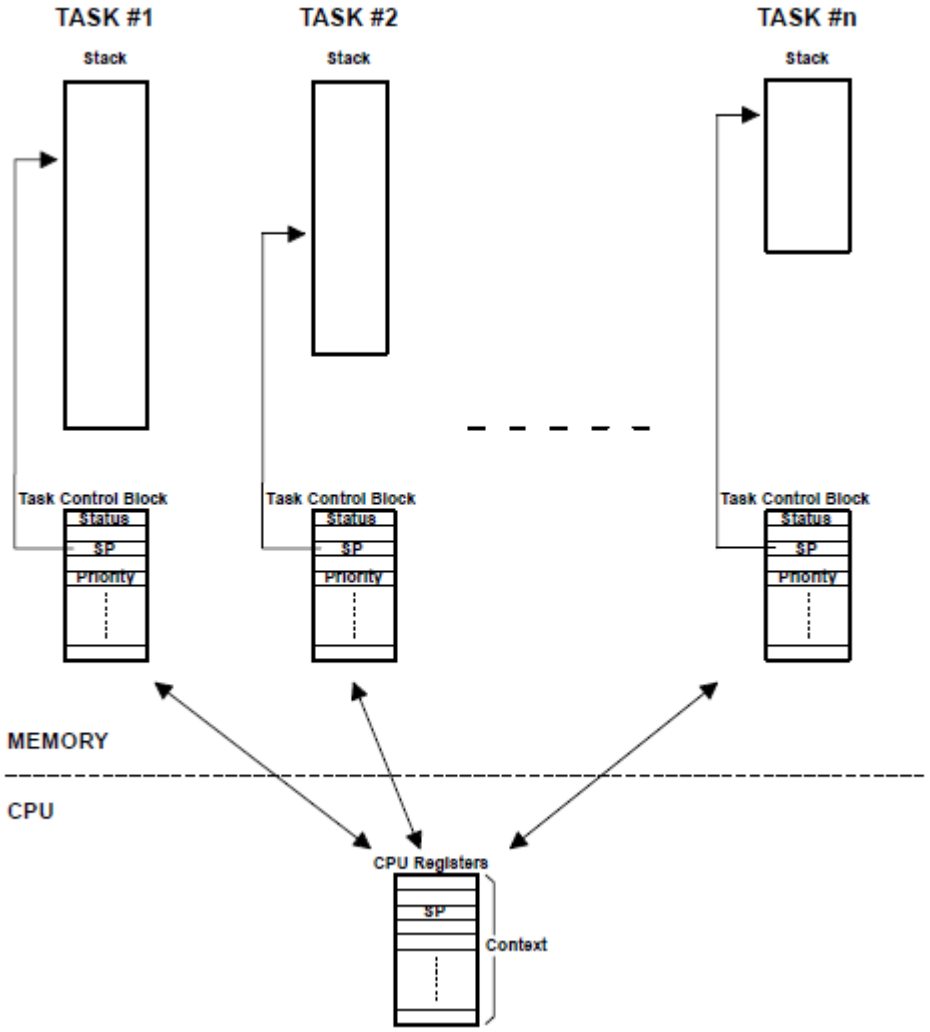
Operating System

The Operating System is a system which takes number of Tasks and perform them in a specific time called Periodicity. Task is nothing but a piece of code that repeated every specific time. Every task has its own periodicity . The OS will Synchronize between these task Simply.



Multitasking

- Multitasking is a process of scheduling and switching the Central processing unit (CPU) between several tasks .
- A single CPU switches its attention between several sequential tasks
- Application programs are typically easier to design and maintain if multitasking is used.



TASK

Task, or also called a *thread*, is a simple program that thinks it has the CPU all to itself.

Task is a function which Takes void and returns void only contains some lines of codes. Every task could be periodic task .

Now every LED will be a Task and the OS will Sync between them.

```
void LED1(void)
{

    LED1_COUNTS += 1;
    if(LED1_COUNTS == 1)
    {
        DIO_SetPinValue(PORTA , PIN0 , HIGH);
    }
    else if(LED1_COUNTS == 2)
    {
        DIO_SetPinValue(PORTA , PIN0 , LOW);
    }
    else if(LED1_COUNTS == 10)
    {
        LED1_COUNTS =0;
    }

}
```

TASK parameters

Tasks have main three parameters:

The function (pointer to function)

- The lines of code that will be executed by CPU.

The Periodicity

- The periodic time of the task to be executed .

The Priority

- the more importance the task , the higher priority given to it .

```
typedef struct
{
    u8 Periodicety ;
    void (*Fptr) (void);
}Task;
```

The priority of the Task will take place later.

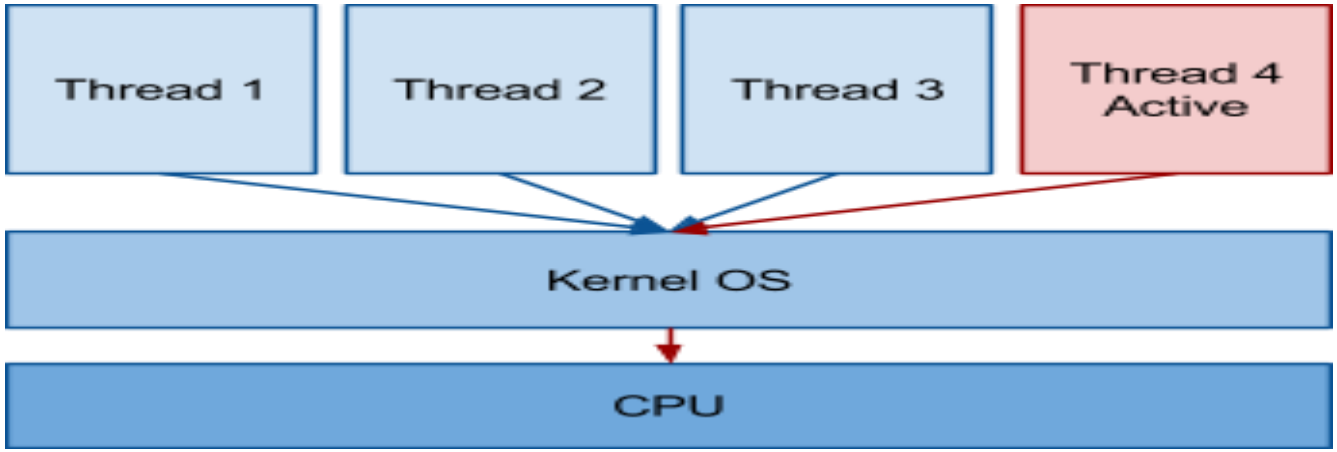
Task States

Each task can be in any one of five states .

Dormant	Task that resides in memory but has not been passed to the RTOS to start scheduling.
Ready	Task that can execute but its priority is less than the currently running task
Running	Task that has the control of CPU.
Waiting	Task that requires the occurrence of an event.(for ex “waiting for an I/o operation”
ISR	When an interrupt has occurred and the CPU is in the process of service the interrupt

Kernel

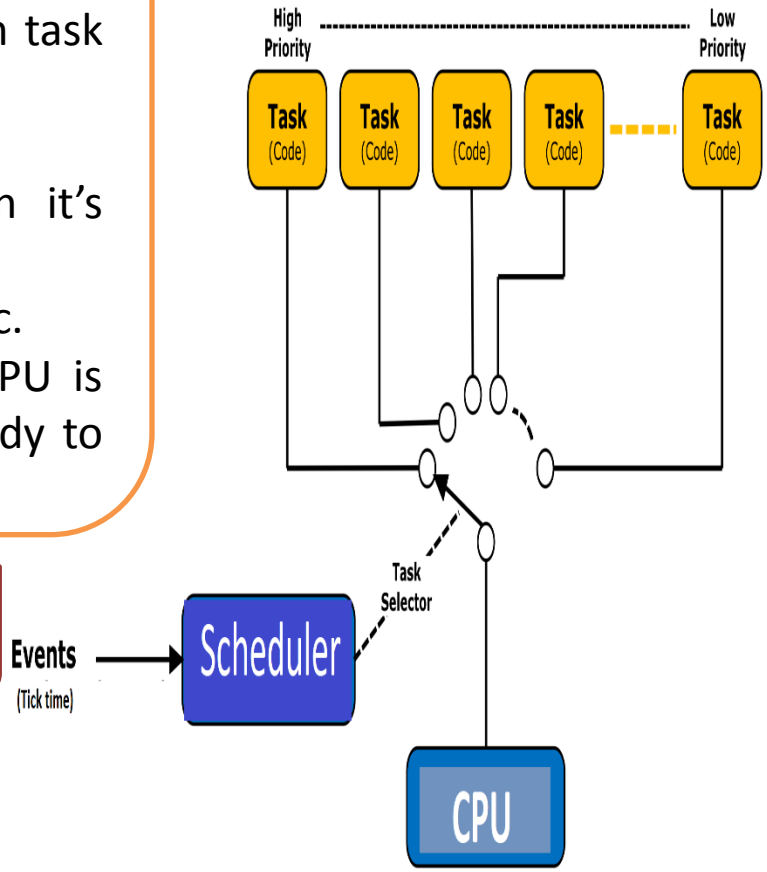
- A kernel is the part of multitasking system that software which responsible for management of tasks (managing the CPU time) and communicate between them.
- The use of Real-time Kernel Simplify the design of the system by allowing the application to be divided into multiple tasks that the kernel manages.



Scheduler

- ✓ Scheduler : also called dispatcher , is the part of the kernel responsible for determining which task runs next.
- ✓ Most Real time Kernel are priority passed.
- ✓ Each task is assigned a priority based on it's importance.
- ✓ the priority of every task is application specific.
- ✓ In a priority based kernel ,control of the CPU is always given to the highest priority task ready to run .

The Scheduler runs every a specific time called TICK time.



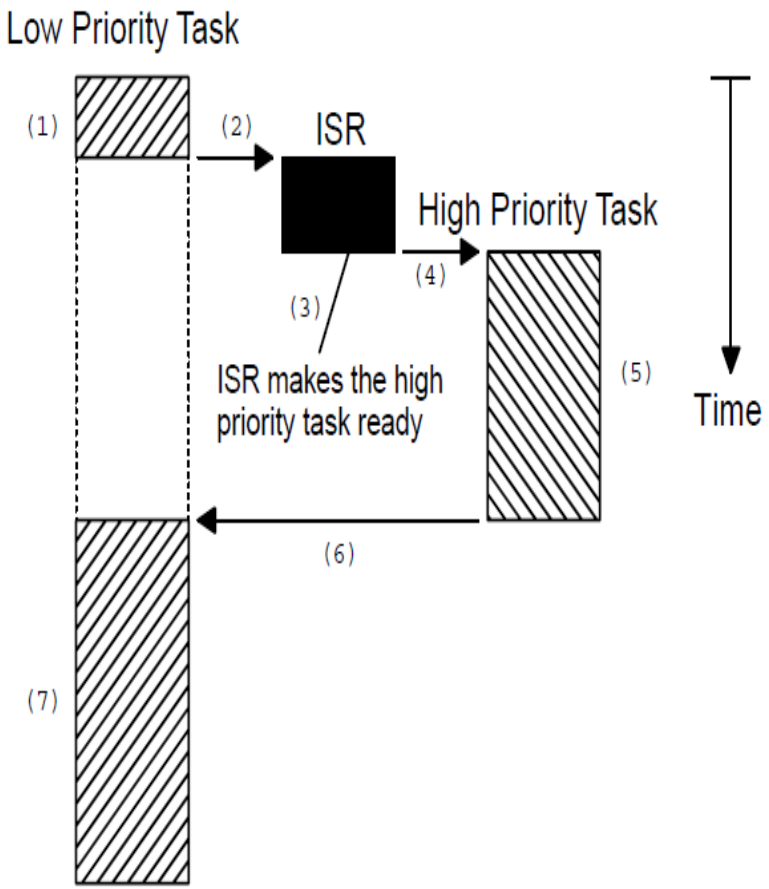
Two types of priority based kernel exist:

1. Preemptive kernel
2. Non-preemptive Kernel

Preemptive kernel

- A preemptive kernel is used when system responsiveness is important.
- The highest priority task ready to run is always given control of the CPU .
- The most commercial Real time Kernels are preemptive .

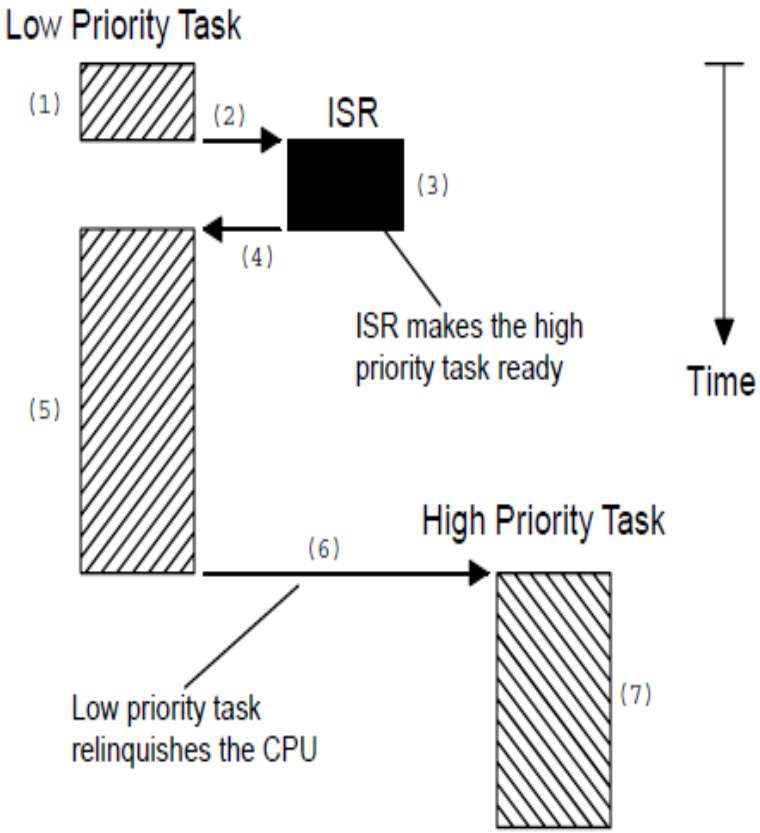
If the ISR a higher priority task ready , when the ISR completes , the interrupt task is suspended, and the new higher priority task is resumed



Non-Preemptive kernel

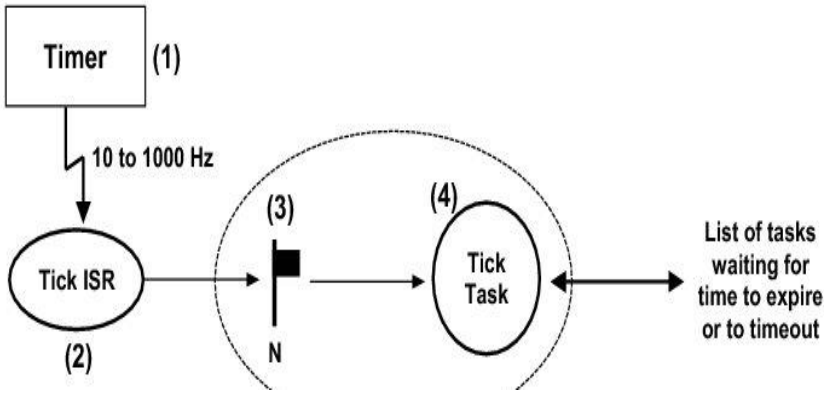
- Non-preemptive kernel requires that each task does something to explicitly give up control of the CPU

The ISR can make a higher priority task ready to run , but the ISR always returns to the interrupted task. The new higher priority task gains control of the CPU only when the current task give up the CPU.

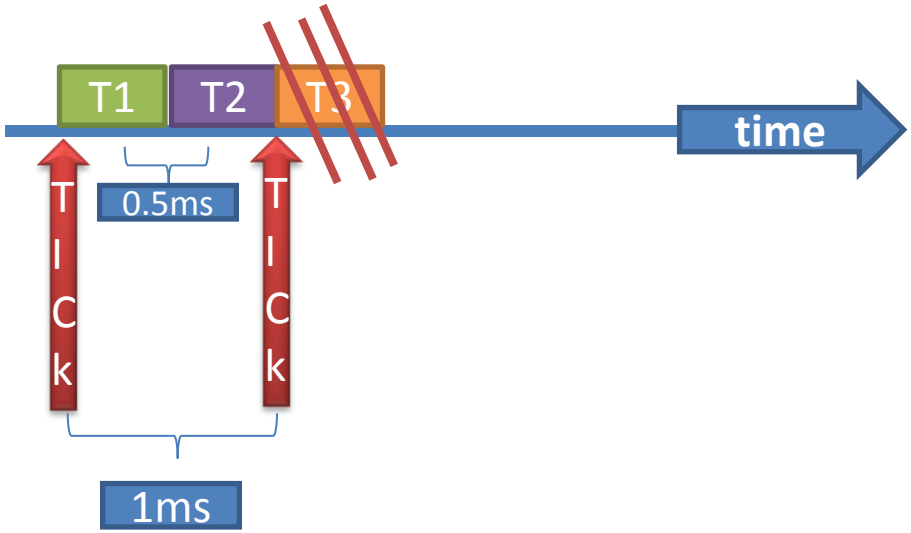


TICK time

Tick time is that time which the Scheduler takes place.
Tick time must be suitable for all tasks.



Consider that we have a scheduler with Tick time (1ms) and we have three tasks will run at next Tick every one of them takes 500us to be performed .
The scheduler will ignore the third task As if it did not exist and perform another one at next tick and that is a Disaster



CPU load

- ✓ CPU load is one of the most important parameters in the OS .
- ✓ CPU load is that how much is the CPU busy between two ticks.
- ✓ CPU load could be determined by a simple equation.

$$CPU\ load = \frac{\sum Exicution\ time\ in\ the\ worst\ case}{Tick\ time}\%$$

Notes

- ✓ CPU load have to be less than 100%.
- ✓ The lower the CPU load the better the system
- ✓ It is better for a CPU load to be between 60% and 80%.
- ✓ The tick time must be greater than the greatest execution time of a task.

priorities

Task Priority

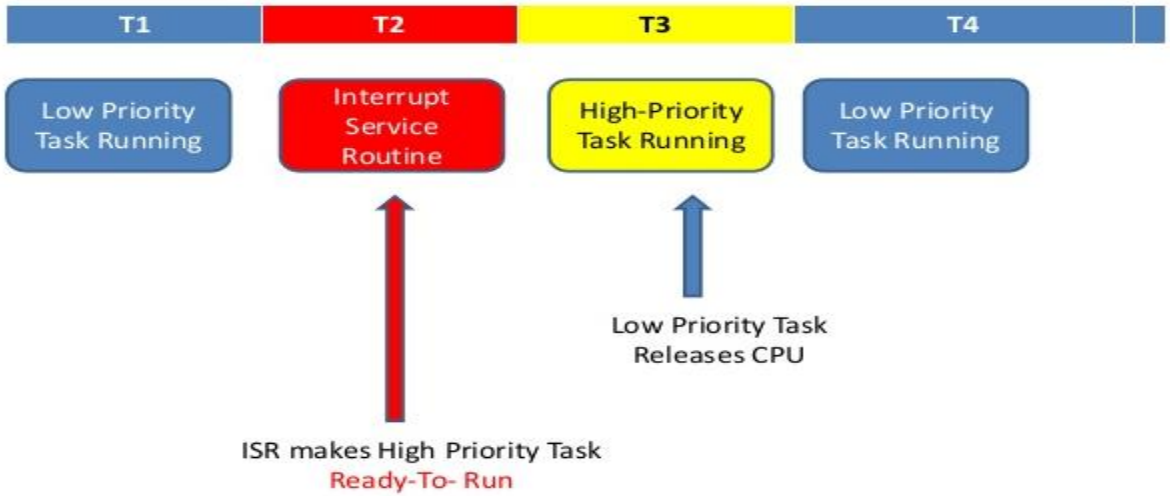
- A priority is assigned to each task . the more importance the task , the higher priority given to it .

Static Priority

- Task priorities are static when the priority of each task does not change during the execution time . Each task is given a fixed priority at a compile time.

Dynamic Priority

- Task priorities are dynamic if the priority of the task can be change during the run time this feature is used is used in Real-time kernel to avoid priority inversion(will discussed later)



let's implement our own simple RTOS

Task Definition

- We need to define a New struct for the task with it's parameters.

```
typedef struct
{
    u8 Periodicety ;        //the task periodicity
    void (*Fptr) (void);    //pointer to function of tank
}Task;
```

Array of Tasks

- We need to configure n array of the maximum no of tasks that system can handle.

```
#define NO_OF_TASKS  3

Task arr[NO_OF_TASKS] ;
```

The priority of every Task will be its Index in the array .

let’s implement our own simple RTOS

Create task

- We need to implement a function that create new task and fill the tasks’ array by it’s parameters .

```
void TASKS_CREATION(u8 priority , u8 periodicety ,void (*Fptr) (void) )
{
    arr[priority].Periodicety = periodicety;
    arr[priority].Fptr = Fptr;
}
```

let’s implement our own simple RTOS

Scheduler

- We need to make a function which decide which Task will be executed .

```
void SChEDULER(void)
{
    u8 i  ;
    for(i=0 ; i < NO_OF_TASKS ; i++)      //check every task periodicety
    {
        if (Tick_Counts % arr[i].Periodicety == 0)
        {
            arr[i].Fptr();                //execute the task function
        }
        else
        {
            //do nothing
        }
    }
}
```

let’s implement our own simple RTOS

Tick time

- We need to adjust a timer to configure the required tick time .

```
ISR (TIMER0_COMP_vect)        //runs every designed tick time
{
        Tick_Counts++;        //increment the tick_counts
        SChEDULER();          //call the scheduler
}
```

Lab 1

Design a system with these constrains:

1. RED LED blinks every 1ms.
2. Yellow LED blinks every 2ms.
3. Green LED blinks every 5ms.
4. Stepper motor steps every 30ms.
5. LCD Shifts your name right every 100ms.

Time To Code

IMT SCHOOL

www.imtschool.com

ww.facebook.com/imaketechnologyschool/

This material is developed by IMTSchool for educational use only
All copyrights are reserved