Basics of C and Microcontrollers

By Robosoft Systems

Index

C—How to program

Need for computing

Why do we use computers ? The most common answer we get to hear is that "The computer makes our life easier". Its simple. We cannot imagine how different and difficult our lives would be without computers. Giving a simple example, previously the use of typewriters was widespread to write letters & other official documents. But the problem with it was that it wasn't that flexible. One typing mistake even in the last line of the letter could compel you to type it all over again. This task was simplified with the introduction of MS-Word. This is one of the simplest example we can imagine. There are countless such applications where the computer has brought about a world of difference.

The computer is a general purpose device. Many people have computers which are dedicated only for entertainment via games, movies , internet and such other applications. These are nothing but customized applications developed by programming the computer. We can program the computer ourselves too to develop an application which we desire. Normally, the word 'programming ' is daunting for many people and thus seldom users try to develop their own code. However, we at Robosoft look at programming in a slightly different way.

Its nothing but common sense that if we need any person or machine to act as required, we first need to communicate what to do. Obviously, we need to use a language hat the computer understands. The only language that the computer understands is the Binary language which has only two distinct symbols i.e. "Logic 0" and "Logic 1". But then it becomes even more difficult for us to program the computer. Just imagine the size and complexity of programs if we were to write programs using 1's and 0's . Let alone the job of debugging the program if some mistakes were committed. To this problem, higher level programming languages have been developed over the past four decades. Some got very popular and some couldn't make it. We will talk about one of the most popular ones - the C programming language.

Although we assume that you must be aware of the different numbering systems and their significance, we have included this topic just to give an overview. It is important to have a sound knowledge of numbering systems as they play an important role in the representation and manipulation of data. Some of the most popular systems are the Decimal system used by human being, Binary system used by Computers and Hexadecimal system used for convenient representation of binary values.

Decimal Number System:

The Decimal System also known as Positional System is the most widely used numbering system in the world today. The distinctive feature of numbering systems is the Base of the numbering system. Each system has its unique base. Decimal has a base of 10. This system has 10 distinct symbols from 0 to 9, hence: Base of 10.

Ten Thousands	Thousands	Hundreds	Tens	Ones
10^4	10^3	10^2	10^1	10^0

The Decimal System is also called as the positional system because in this system, weight of each digit in a number depends on its position.

Example: In the decimal number 93152 the weightage of number 2 is 10^0, similarly the weight of 5 is 10^1 and so on. Therefore we write the equation

$$(9x10^4) + (3x10^3) + (1x10^2) + (5x10^1) + (2x10^0) = 93152$$

Binary Number System:

In the binary number system there are only two distinct symbols i.e. 0 and 1. Binary number system has been used in computers because the is a close proximity between the two. Just like the binary system even the computer has two distinct possibilities i.e. "ON" or "OFF", which can be easily represented by Logic 1 and Logic 0.

10^2	10^1	10^0
2^2	2^1	2^0

The binary system works exactly on the same principles as the decimal system, but has a base of 2 rather than base 10.

Example: In the binary number 11110 the weight of 0 is 2^0, and the weight of the next 1 is 2^1. Hence we can find the decimal equivalent as follows.

$$(1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) = (31)_{10}$$

Hexadecimal Number System:

The Hexadecimal Numbering System is a Base 16 system i.e. it has 16 unique symbols that can be used. Out of the 16 symbols the initial 10 symbols are same as in decimal numbering system from 0 to 9, after this we use the notations from A to F to represent values from 10 to 15 respectively. There is an important relationship between the binary & hexadecimal numbering systems. The hexadecimal numbering system is the preferred choice for representing binary data in any application. The reason being that conversion between binary & hexadecimal is the easiest and fastest. Even if you try to convert hexadecimal number into decimal number than it would require some calculation and a finite amount of time, whereas on the other hand you can just memorise a table and easily convert a binary number into an hexadecimal number.

Example: A binary number 111100000010, can be easily converted into a hexadecimal number by grouping the digits into groups of 4's.

1111 0000 0010 = F02

The C programming language

C is a general purpose programming language which was developed at the AT&T Bells Laboratory in the USA in 1972. It was originally designed for implementing on a machine called as the DEC PDP-11 which used the *UNIX OS* developed by Mr. Dennis Ritchie. C is not a very high level language nor a big one and it is not specialized for any particular area of application . Many parts of the popular operating systems like Windows, Linux and UNIX are still written in C. This is because when it comes to performance i.e. the speed of execution nothing beats C.

How to use C

To do anything in a programming language we would require some quantity or entity in which the user can first give the input, which will be processed and than the program will produce the desirable output. It works just like an open loop system as we all have studied.

This quantity or entity which we are talking about can be realized in the form of a VARIABLE in C language. Let it be a task as simple as printing your NAME or a complex one such as interfacing temperature sensor with your μc and measure the temperature. Everything begins with the use of a VARIABLE, hence it is the basic building block in learning C.

Constants and Variables:

A *constant* is a quantity that doesn't changes e.g. 20. This quantity can be stored in a location in the memory of the computer. The addresses of the memory are specified in Hexadecimal format which starts from 0000h and upper limit is determined by the size of the memory. Now suppose we are writing a program for temperature measurement. We first save the reading from the sensor in say memory location FF0Ah. Now after some time if you want to access that value, it is really inconvenient to remember and use the address FF0Ah to fetch the reading. You could imagine the difficulty if you were to develop an application which consists thousands of readings, a common practice at the industry level.

To overcome this difficulty most of the programming languages use the concept of variables. A quantity which may vary during program execution is called variable. A variable can be considered as a name given to the location in the memory of the computer where some value is stored. So this solves our problem of remembering the addresses of the locations, now we can choose a name which is easy to remember and relevant to its use.

Example: To store our age we can use the variable name "age", similarly we can use "temp" for storing the temperature.

Data Types in C:

When we are creating a variable we are actually allocating space for that variable in the memory. The standard sizes of variables are 8, 16 or 32 bits. The size is user dependent, depending upon his/her requirement. The same is achieved in C with the help of data types, summarised in the following table..

Type	Size (byte)	Min	Max	Format specifiers
signed char	1	-128	+127	%c
unsigned char	1	0	255	%c
signed int	2	-32768	+32767	%d
unsigned int	2	0	65535	%u
long int	4	-2147483648	+2147483647	%ld
float	4	-3.4e38	+3.4e38	%f

Note: "char" is by default taken as "signed char". Similarly "int" is by default taken as "signed int"

The above data types also falls in a categorisation called as "keywords". C has a set of 32 keywords. With the combination of a keyword and a variable name we can create a C instruction.

Example: int height; or char gender;

In the above e.g. height is a variable name stored in the memory and it type is "signed int" hence its size is 2 bytes. Similarly gender is a variable name having type "signed char" and size is 1 byte. In both the examples you must have observed the semicolon (;) sign. This a statement terminator in C. All the statements should be terminated by a semicolon.

Structure of a C program

C is also known as a Structured Programming Language, because it provides a structured approach towards solving problems. The programs in C have a pre-defined format as shown below.

- Header files (Pre-Processor directives)
- Prototype declaration
- Global declaration/definition
- Main function
- User defined functions

There has to be a main function in every C program. It is mandatory, the CPU always starts its operation from the main function.

Syntax of a Function:

```
return_type  function_name (parameters)
{
        Statement(s);
}
```

function_name is the identity of the function. The user can give a relevant name to the function. For instance, if we are writing a function to add two numbers we can call the function as add.

return_type specifies the type of value that a called function sends back to the calling function after executing. For e.g. in the add function after adding two integers the result will be an integer, hence the return type is int.

parameters are the values that we pass to the called function from the calling function. When we are calling the function add to add two numbers, we should also pass the two integers which are to be added, these are nothing but parameters.

Function Body:

The function body consists of different kinds of instructions like.

- Type declaration instructions (int age, char name)
- Arithmetic instructions (will consist of operators like +,-,/,* etc.)
- I/P and O/P instructions (printf , scanf)

Sample program

```
#include<stdio.h>        //Header File

void main(void)
{
     printf("Hello World");
}

Output :  Hello World
```

Note : The above e.xample explains one of the simplest programs possible in C language. The structure of this program is not at full potential of C, it doesn't even covers the "VARIABLE" aspect of C. The program starts with the header file ***#include<stdio.h>*** where we are telling the compiler to include the file stdio.h which consists of various library functions which we will be using quite often. For example the ***printf*** function which we are calling to print *"Hello World"* is a library function in stdio.h.

Types of variables

Variables can be classified based on various criterias. Here we will discuss variables with respect to the scope in which they are declared/defined. Based on this classification the types of variables are.

- Local variables
- Global variables

Local Variables :

The variables which are declared or defined inside a user defined function are called as local variables or **automatic variables.** They are always stored in the **Stack Segment** of the memory and they exist as long as that function is being executed. Once the function is exited the variables are deleted and the will be created again if that function is called again for execution. These variables can be accessed only inside the function they defined. If you try to print its value outside it function it will give an error.

Example—Local variables

```
#include<stdio.h>

void main(void)
{
     int a;                    // Local Variable (uninitialized)
     int b = 100;              //Local Variable (initialized)
     printf("a = %d",a);
     printf("b = %d",b);
}

Output : a = 4379 (some garbage value)
          b = 100
```

Global Variable :

Global variables are the ones which are declared or defined outside a function. They are never declared/defined inside any function like main or any other user defined function. These variables are always stored in the **Data Segment** of the memory and they exist as long as that program is being executed. They are called as global because unlike local variables they can be accessed from any part of the program.

Example—Global variables

```
#include<stdio.h>
int a;                    //Global variable (uninitialized)
int b =100;               //Global variable (initialized)
void main(void)
{
     Printf("a = %d  \n  b = %d",a,b);
}
          Output : a = 0    b = 100
```

Operators in C

C provides a rich set of operators spanning from arithmetic, logical, bitwise to combinational operators. If more than one operator appears in the same equation than we have a priority and precedence table to refer.

Priority	Operators
1	( ) [ ] -> .
2	! ~ - * & (sizeof) cast ++ --
3	/ %
4	+ -
5	< <= >= >
6	== !=
7	&
8	^
9	&&
10	||
11	?:
12	= += -=
13	,

The priority of the operator is given in the above table and the precedence of the operators are from left to right for all rows except for row number 3, 4, 11, 12 where the precedence is from left to right.

Control instructions

While loop

The ***while*** loop is called a control instruction because it changes the flow of execution of program depending upon certain conditions. The syntax of this loop is:

```
while (expression)
{
        statement(s);
}
```

If the expression inside the parentheses of the while instruction evaluates out to be **true** than the body of the while loop is executed otherwise the program execution directly jumps to the instruction after the while loop.

In C an expression is said to be true if it is a value greater than 0. Therefore

while (expression) *if expression = 0 then it is false and while is not executed.*
if expression >= 1 then it is true and while is executed.

For loop

It is one of the most widely used control instruction, it has a structure slightly complicated than the while loop. The syntax of this loop is:

```
for (initialize ;  test ;  increment )
{
        statement(s);
}
```

The structure is executed as follows:

- The program control first initializes a variable.
- Then it goes to the next step i.e. test, where it tests the expression. If that expression evaluates out to be true, the body of for loop is executed; otherwise the program control leaves the for loop and executes the next instruction.

- If the expression is true, then after executing the for body the program control goes to the increment section where it increments the variable.
- After this it again comes to the test section checks the same expression and executes the for loop only if result is true and it keeps on doing the same unless the test evaluates to be false.

Do While loop

The ***Do While*** loop is a variant of the While loop. It performs nearly the same function as the while loop. It differs from the while only in the aspect that it does not check the condition before it executes the loop for the first time. The syntax of this loop is:

```
do
{
        statement(s);
}
while(expression);
```

The loop is executed at least *once* even if the expression evaluates to be false. After the loop has executed once, then the expression is checked and if it is found to be false, then the loop terminates and the control terminates to the next instruction after while. If the expression is found to be true then the control again jumps to the instruction within the loop.

If– else statement

This is a very versatile feature of C which enables us to take decision depending on a situation.

```
if (expression)
{
        statement(s);
}
else
{
        statement(s);
}
```

Only if a particular condition is met the body of "if" will be executed otherwise the body of the "else" will be executed.

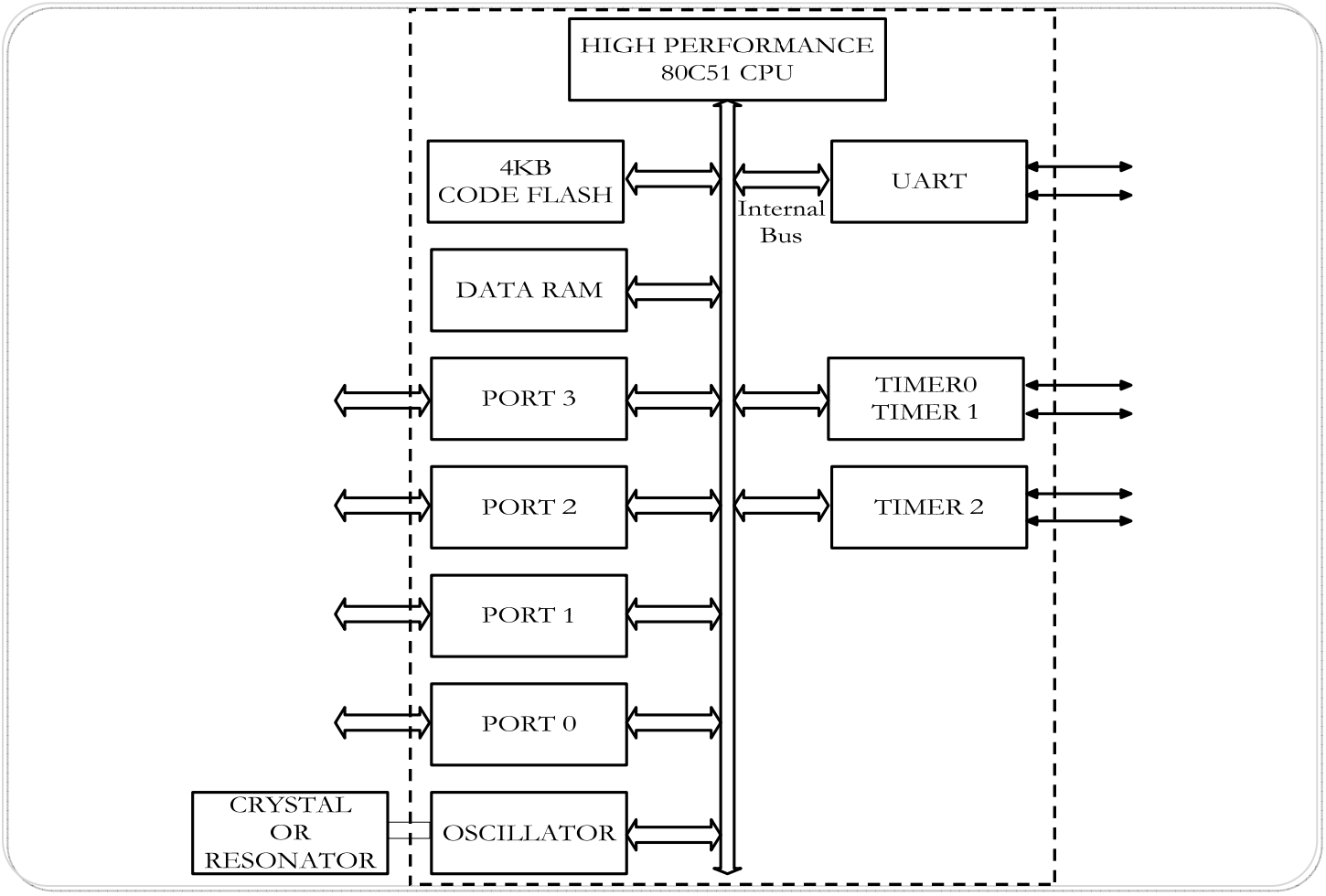
Switch statement

The ***switch*** statement is useful in eliminating the multiple if else statements. It comes handy in situations where we need to make a decision from an array of options available. The syntax of this statement is:

```
switch (expression)
{
        case 1:
                statement(s);
                break;
        case 2:
                statement(s);
                break;
        :
        :
        :
        default:
                statement(s);
                break;
}
```

The switch statement provides a way for multi branched conditions in a very neat manner. Imagine you have ten conditions to check and having to write ten if-else blocks. It is generally a good practice to write a default block which provides for default execution of statements when nothing else matches.

µController architecture



Inside the 8051µC

The above block gives an overview of the 8051, although there are other resources that the 8051 provides for the user's disposal most importantly the register banks. The array of register are.

1. Accumulator (A)
2. B register
3. 4 banks of 8 registers each (R0 to R7)
4. Data Pointer (DPTR)

The accumulator is the most important register it is 8 bits wide and used in many internal operations of the ALU (Arithmetic Logic Unit) as well as in external communication of the 8051.

Ports

The 8051 has 4 I/O ports(P0-P3). As the very name suggests these ports are used to bring and send the data IN and OUT of the 8051.

Port 0

P0 occupies the pins 32 through 39 on the 8051 chip. It can be configured as an I/P or O/P port through proper programming. In order to do so we need to connect an external pull up resistor at each port pin. This is due to the fact that P0 uses open drain technology unlike the other ports.

Dual role of P0 :

When we are interfacing external memory with the 8051 for data storage and then fetching it, we also need to take care of the addresses in which the data is stored i.e. the user also needs to specify the address, P0 is responsible to bring that data/address IN and OUT of the controller hence it is also designated as AD0-AD7. This saves our pins.

Port 1

P1 occupies the pins 1 to 8 in the chip. It can be configured as an I/P or O/P port with the help of programming. This port doesn't requires any pull up resistors as it has built in pull ups for all the 8 pins. Although for practical reasons you will still find external pull ups on an 8051 development board. P1 employs N-MOS technology.

Port 2

P2 occupies pins 21 to 28 in the 8051 chip, it can be used as I/P or O/P port. Just like P1, P2 doesn't require any pull up resistors as it has internal pull ups.

Dual role of P2 :

Since the 8051 can access 64KB of external memory, it needs a path for the 16 bits of the address. While P0 provides the lower 8 bits via A0-A7 it is the job of P2 to provide bits A8-A15. in other words when8051 is connected to external memory, P2 cannot be used for I/O.

Port 3

P3 occupies pins 10 to 17 in the 8051 chip, it is a bidirectional port and it doesn't requires external pull ups. Port 3 has the additional function of providing some extremely important signals such as interrupts. The table below provides these alternate functions.

P3 bits	Function
P3.0	RxD
P3.1	TxD
P3.2	INTO
P3.3	INT1
P3.4	T0
P3.5	T1
P3.6	WR
P3.7	RD

Each of the ports discussed above can be programmed as I/P or O/P port. A port can be configured as I/P port by writing a 1 to all its pins, and as an O/P port by writing a 0 to all its 8 pins.

Note: The pin numbers can vary depending on manufacturer. Please refer the appropriate datasheet for your device.

Example program

```
#include <reg51.h>
void main(void)
{
     P0 = 0xFF;     // Input port
     P1 = 0x00;     // Output port
     P1 =  P0;      //Taking value from I/P port and writing it to O/P
```

Note : On reset all the ports of 8051 are automatically at high voltage. However, in programming we need to configure them as I/P port by writing FF

Timers

The 8051 has 2 timers : Timer 0 and Timer 1. They can be used either as timers or as event counters. These are basically hard wired registers which are 16 bits wide. Now since the 8051 has an 8 bit architecture, each 16 bit timer is accessed as two separate registers of lower byte and higher byte. In the timer mode the registers are used to generate some time delays and in counter mode it is used to count external events .

Timer 0 registers

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

The lower byte register is called TL0 (timer 0 lower byte) and higher byte register is known as TH0. These can be accessed as any other register such as A,B, R0 etc.

Timer 1 registers

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

These 16 bit register is split into 2 bytes, referred to as TL1(Timer 1 lower byte) and TH1(timer 1 higher byte).

Other important registers

The registers shown above are mere registers which can do some counting of numbers, but there is a great deal of programming which goes behind running the timers according to our requirement. For this we use some registers namely TMOD, TCON.

TMOD register

GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
TIMER 1				TIMER 0			

Both timers 0 and 1 use the same register, called TMOD. To set the various timer operation modes. TMOD is an 8 bit register in which the lower 4 bits are set aside for timer 0 and the upper 4 bits for timer 1. In each case the lower 2 bits are used to set the timer mode and the upper 2 bits to specify the operation.

TCON register

TF1	TR1	TF0	TRO	IE1	IT1	IE0	IT0

As of now the most important bit in the TCON register is the TR0 or TR1 bits. These are the timer run control bits which initiates the timer count.

Interrupts

A micro controller can be thought of as a master which gets its work done from other devices connected as peripherals to it. For this purpose the µc needs to manage it communication with these peripherals efficiently. One of the methods of communicating with the peripherals is POLLING, wherein the µc continuously checks whether its peripherals wants its help, and whenever there is a request for communication the µc serves that device. However, this method is not efficient , due to the wastage of the µc's time in polling the devices. The other and more efficient choice is the interrupt method.

In the interrupt method, the µc instead of wasting its time in polling the devices, can do other important work. The peripheral devices need to tell the µc if they need its help. This is done in the form of an interrupt. Whenever the µc receives an interrupt it first takes a backup of its current work and than goes to help the interrupting device by executing its ISR.

Interrupt Service Routine

For every interrupt, there must be an ISR, or interrupt handler. For every interrupt there is a fixed location in memory that holds the address of its ISR. This area of the memory is called the Interrupt Vector Table (IVT). The 8051 has 6 different interrupts.

Interrupts	ROM Location
Reset	0000
External hardware interrupt 0 (INT0)	0003
Timer 0 interrupt (TF0)	000B
External hardware interrupt 1 (INT1)	0013
Timer 1 interrupt (TF1)	001B
Serial COM interrupt	0023

Enabling or disabling interrupts

Upon reset, all interrupts are disabled, meaning that the µc will not respond to them. The interrupts must be enabled by software for the µc to respond. The is a register called IE (interrupt Enable) that is used to enable/disable the interrupts.

EA	--	ET2	ES	ET1	EX1	ET0	EX0

Interrupt priority

What happens if two or more interrupts are activated at the same time? Which interrupt will be serviced first. When the 8051 is powered up, the following default priorities are assigned to the interrupts.

Priority Level	Interrupt
1	External Interrupt (INT0)
2	Timer Interrupt 0 (TF0)
3	External Interrupt 1 (INT1)
4	Timer Interrupt1 (TF1)
5	Serial Communication (RI, TI)

Now suppose if external hardware interrupt 0 and 1 are activated at the same time, than external interrupt 0 (INT0) is serviced first because it is having higher default priority. Only after INT0 has been serviced INT1 is serviced.

Embedded C

There is not much of a difference between C and Embedded C programming. Embedded C is designed to be more efficient than C when is comes utilisation of memory. Hence it is the language of choice for embedded system designer working on μc's , as these chips have limitations of on chip memory. For eg. the code space for 8051 is limited to 64KB. The programs written in C are given to special compilers which produces the hex file and the goal of an embedded programmer is to create smaller hex files, hence a good understanding of C data types is necessary.

Data types

The various data types in Embedded C are:

1. unsigned char
2. signed char
3. unsigned int
4. signed int
5. float
6. Sbit(single bit)
7. SFR(special function register)

While programming it will be a good practice to replace int data types with char where ever possible, significantly reducing memory consumption We can use the sbit data type wherever we require a flag (semaphore) instead of using a char or int. Signed variables should be used only in places where we are dealing with values which may be negative or positive like temperature.

Sbit:

The sbit keyword is widely used specifically designed to access single bits of SFR's. some of the SFR's in our controller are designed to be bit addressable. Some of these SFR's are P0 through P3, hence we can use sbit to use single bits of the ports P0– P3.

SFR:

The 8051 has many SFR's which are located just above the internal RAM starting from 80h-FFh.

Bitwise operators

The bitwise operator's are of great importance in embedded C programming, they help a great deal in reducing the size of the code. There are six of these operators.

1. & AND
2. | OR
3. ^ XOR
4. ~ NOT (Complement)
5. >> RIGHT SHIFT
6. << LEFT SHIFT

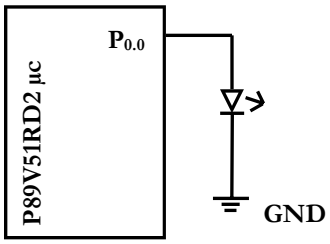
Example program

```
#include <reg51.h>
void main(void)
{
     P0 = 0x35 & 0x0F;        //ANDing
     P1 = 0x04 | 0x68;        //ORing
     P2 = 0x54 ^ 0x78;        //XORing
     P0 = ~0x55;              //Complementing
     P1 = 0x9A >> 3;          //right shift (divide by 2)
     P2 = 0x77 << 4;          //left shift (multiply by 2)
}
```

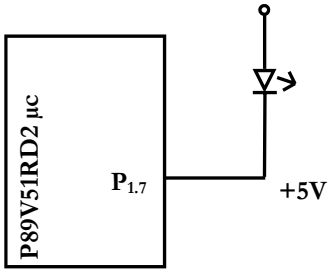
Note : We are using the header file #include <reg51.h> because it contains the prototype declarations of the 8051 μc resources, so we are using ports P0,P1,P2 directly in the code.

Example circuitry—Glow LED

To glow the LED's we first need to understand our hardware connection between the μc and the LED, and also study other ways to do it.



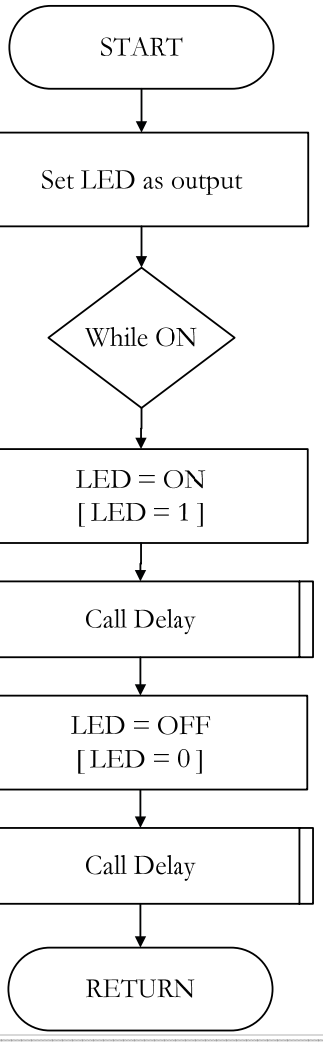
Active high control



Active low control

The above figure shows the two configurations in which an LED can be connected. In the first figure, the anode of LED is connected to the port pin and cathode is grounded. Hence we need to pull the pin high to glow the LED. In the second figure, anode of the LED is connected to +5V and cathode is connected to the port pin.

Hence we need to pull the pin low so as to glow the LED. Pulling the pin low will provide ground to the LED. As we all know that an LED is basically a PN junction diode, which emits light when it is forward biased, so we are meeting this condition in both the above cases by the different hardware connections.

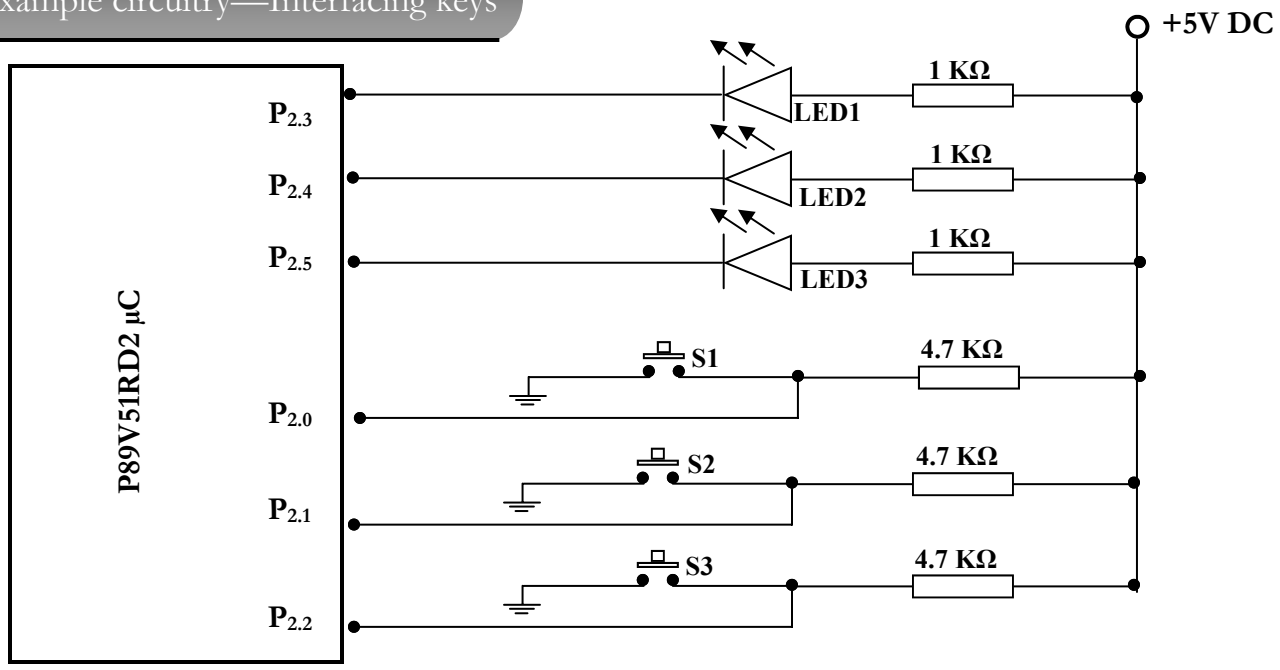


Example program– Glow LED's

```
#include <reg51.h>
sbit LED1 = P2^3;   //LED 1 is connected at port pin 2.3
sbit LED2 = P2^4;   //LED 1 is connected at port pin 2.4
sbit LED3 = P2^5;   //LED 1 is connected at port pin 2.5
void main(void)
{
     LED1 = 0;      // Glow LED1
     LED2 = 0;      // Glow LED2
     LED3 = 0;      // Glow LED2
     delay(100);    // wait for some time between LED ON and LED OFF
     LED1 = 1;      //LED1 OFF
     LED2 = 1;      //LED2 OFF
     LED3 = 1;      //LED3 OFF
     delay(100);
}
```

Vcc = 5V

Example circuitry—Interfacing keys



```
When Key is pressed its value will become zero i.e. Logic LOW .On key
hit detection we will change LED status.

#include <reg51.h>
sbit LED1 = P2^3;
sbit LED2 = P2^4;
sbit LED3 = P2^5;
sbit KEY1 = P2^0;
sbit KEY2 = P2^1;
sbit KEY3 = P2^2;

void main(void)
{
     while(1)
     {
     If(KEY1)  //if key not pressed
     LED1=1;   // turn LED Off
     else      //else
     LED1=0;   //turn on LED

     If(KEY2)  //if key not pressed
     LED2=1;   // turn LED Off
     else      //else
     LED2=0;   //turn on LED
     }
}
```

For More sample program refer resources CD

A-61, 1 st Floor, Raj Industrial Complex,
Military Road, Marol,
Andheri (East),
Mumbai – 400059
India

Contact Information
Office: (91-22) 66751507
Office: (91-22) 29207086
Mobile: (91) 9930776661

Email: info@robosoftsystems.co.in

Visit us at
www.robosoftsystems.co.in