Lecture Notes in Microprocessor Systems using 8051 Microcontroller

Richard Myrick T. Arellaga

Contents

1 Review of Computer Architecture

1.1 How it All Started

- 500 B.C. Babylonians made the Abacus, the first mechanical computer
- 1642 **Blaise Pascal** invented a mechanical calculator made of gears and wheels known as the Pascaline.
- 1800's **Charles Babbage** made the **Difference Engine**, a machine used to calculate astronomical and mathematical tables.
- **Lady Ada Augusta** Countess of Lovelace describe the workings of the Difference Engine. Known as the First Programmer.
- Babbage designed an improved engine, a steam powered engine called the **Analytical Engine**. It is a stored program computer that can store a thousand 20-digit decimal numbers and variables and used Punched Cards as inputs. It is said that modern day computers are based on the architecture of the Analytical Engine.
- In 1801 **Joseph Marie Jacquard** used punched card to select intricate weaving patterns in the cloth that it produced which was called today as *Jacquard's Loom*.
- In 1889, Herman Hollerith developed a machine used to count sort and collate information stored in punch cards and was used in the 1890 census in the US.
- 1896 when Herman Hollerith formed a company called the Tabulating Machine Company later became International Business Machines (IBM) Inc.
- A German inventor named **Konrad Zuse** invented the first modern electromechanical computer the Z3. The Z3 was a relay logic computer clocked at 5.33Hz.
- In 1943 **Alan Turing** designed a machine to break the secret message coming from the mechanical Enigma Machine. This machine is called **Colossus** probably because of its size and used vacuum tubes. It could not solve any other problems other than breaking the secret message. It was a fixed program computer often called a **special purpose computer or specific purpose computer**.
- The **ENIAC - Electronic Numerical Integrator and Computer** was the first general purpose programmable electronic computer system developed at the University of Pennsylvania in 1946 by John Presper Eckert and John W. Mauchly. It contains over 17,000 vacuum tubes over 500 miles of wires and weighed over 30 tons yet it performs 100,000 operations per second. The machine can be reprogrammed by changing its connections.

- **Rear Admiral Grace Hopper** coined the term "Bug" when a moth was stuck in a relay in the Harvard Mark II Computer impeding the operation of the relay.
- A breakthrough in **December 23, 1947** were the development of the transistor at the Bell Labs by **John Bardeen, William Shockley and Walter Brattain**.
- **1958** an engineer in Texas Instruments named **Jack Kilby** invented the Integrated Circuit. This invention led to the development of digital integrated circuits in the 1960's.
- With the development of integrated circuits Intel Engineers **Federico Faggin, Ted Hoff (Marcian Hoff) and Stan Mazor** developed the **4004** microprocessor (US Patent 3,821,715)originally designed for the Busicom calculator in 1969.
- In **1971** Intel bought the rights of the 4004 from Busicom and later that year the 4004 was released on the market. The First 4-bit microprocessor with the operations of 6000 operations per second much slower than the ENIAC but it was so small and requires lower power.
- Intel released the first 8 bit microprocessor called the **8008**. It was able to address 16kb of memory had 45 instructions and the speed of 300,000 operations per second. The 8008 is the predecessor of today's computer.
- 1974, the **8080** microprocessor was introduced by Intel with 64kb of memory and 74 instructions. Motorola also launched the 68000 microprocessor.
- 1976, **Federico Faggin** left Intel and started his own company Zilog. During this year the **Z80** microprocessor was announced and was compatible with the programs written for the 8080 plus added features making it the most powerful microprocessor at that time. The Z80 had 64kb direct addressable memory and 176 instructions.
- While Intel was busy inventing the first microprocessor, It was during 1970 and 1971 **Gary Boone** of Texas Instruments invented the **Microcontroller** the **TMS1000**.
- Intel also created many significant Microcontrollers beside producing the world's first ever microprocessor. The important ones produced by Intel are the **8048** and the **8051** microcontrollers. The 8048 was introduced in 1976 and was the first of Intel's microcontrollers.
- Zilog introduced the Zilog Z8 microcontroller in 1979 which includes Z8, Z8 Encore, eZ8 Encore, eZ8 Encore XP and eZ8 Encore MC.
- In 1981, the 8051 was introduced and one of the most popular microcontrollers. It is even used now and is considered to be the one of the most long-lived microcontrollers.

- It was during the 1990s that advanced microcontroller with electrically erasable and programmable ROM memories such as the flash memory.
- In 1993, with the introduction of EEPROM memory beginning with the Microchip PIC16C84 to be electrically erased quickly without expensive package as required for EPROM allowing both rapid prototyping and in-system programming.
- AVR family of microcontrollers developed in 1996 at Atmel by two PhD students at the Norwegian Institute of Technology Alf-Egil Bogen and Vegard Wollan. It is a modified Harvard Architecture machine with flash memory. By 2003, Atmel had shipped 500 million AVR flash microcontrollers. Atmel was Acquired by Microchip in 2016.

1.2 Computer Architecture

1.2.1 Internal Organization of a Computer

A computer can be broken down into three parts:

- Central Processing Unit (CPU)
- Memory
- Input/Output (I/O) Peripherals

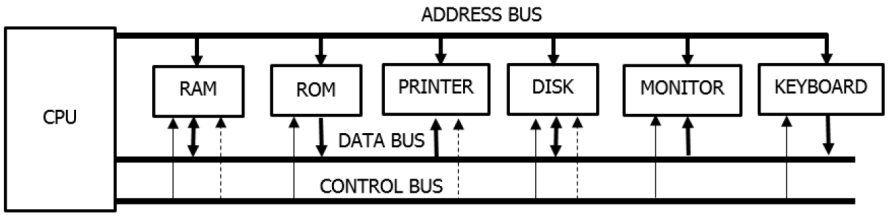

Figure 1: Computer Simplified Diagram

1.2.2 Functions of the CPU

- To execute (process) information stored in memory.
- The CPU is connected to memory and I/O devices through strips of wire called bus.
- Three types of busses: Data, Address and Control Bus.
- Each device has a unique address.
- Address is placed on the address bus, the data to be written or to be read is placed on the data bus and the proper signal of read or write is placed on the control bus.

1.2.3 Instruction Execution

We can think of a microprocessor execution of instructions as consisting of several basic stages:

- Fetch Instruction – the task of reading the next instruction from memory into the instruction register.
- Decode Instruction – the task of determining what operation the instruction in the instruction register represents (ex., add, move, etc.)
- Fetch Operands – the task of moving the instruction's operand data into appropriate registers.
- Execute Operation – the task of feeding the appropriate registers through the ALU and back into an appropriate register.
- Store Results – the task of writing into memory. If each stage takes one clock cycle, then we can see that a single instruction may take several cycles to complete.

1.2.4 Types of Memory

1. ROM (Read-only Memory)
    - **Types of ROM**
        - PROM (Programmable Read-Only Memory)
            - Programmable, once burned cannot be changed.
            - Also called One Time Programmable (OTP) memory.
        - EPROM (Erasable Read-Only Memory)
            - Erasable, can be erase by UV light, takes long time.
            - Also called, UV-EPROM
        - EEPROM (Electrically Erasable Read-Only Memory)
            - Electrically Erasable
            - A portion (event a byte) can be erased
            - Can be erased while it is on board
        - Flash EPROM
            - Same as EEPROM but a byte cannot be erased
            - Can be erased block by block
        - Mask ROM
            - It refers to the ROM in which the contents are programmed during IC fabrication stage.
            - Costy, only done in high volume production.
2. RAM (Random Access Memory)
    - **Types of RAM**

- SRAM (Static Random Access Memory)
  - Storage cells are made of flip-flops
  - Requires number of transistor to keep one bit of data
  - Use of CMOS technology allows producing high capacity SRAM, but still far below than DRAM.
- NV-RAM (Non-Volatile Random Access Memory)
  - Like other RAM, CPU can read/write on NV-RAM but unlike them, data does not loss on power-cut.
  - Use an internal lithium battery to store the energy.
  - Uses an intelligent control circuitry.
- DRAM (Dynamic Random Access Memory)
  - Uses capacitor to store bits of data
  - Requires continuous refreshing due to leakage
  - Smaller in size, so high in density

1.2.5 Hardware Architecture

Computer Hardware Architecture can be divided into two, Von Neumann and Harvard Architecture.

- **Von Neumann Architecture**
  This describes a design architecture for an electronic digital computer with subdivisions of a processing unit consisting of an arithmetic logic unit and processor registers, a control unit containing an instruction register and program counter, a memory to store both data and instructions, external mass storage, and input and output mechanisms. The meaning of the term has evolved to mean a stored-program computer in which an instruction fetch and a data operation cannot occur at the same time because they share a common bus. This is referred to as the Von Neumann bottleneck and often limits the performance of the system.

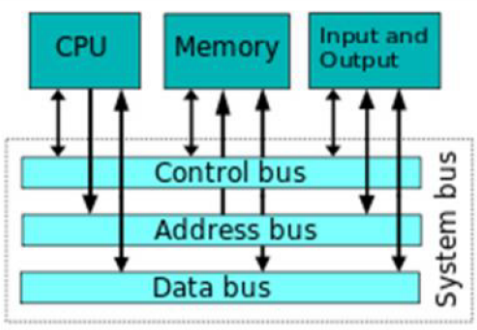

Figure 2: Von Neumann Architecture

- **Harvard Architecture**
  is a computer architecture with physically separate storage and signal pathways for instructions and data. The term originated from the Harvard Mark I relay-based computer, which stored instructions on punched tape (24 bits wide) and data in electro-mechanical counters. These early machines had data storage entirely contained within the central processing unit, and provided no access to the instruction storage as data. Programs needed to be loaded by an operator; the processor could not boot itself.

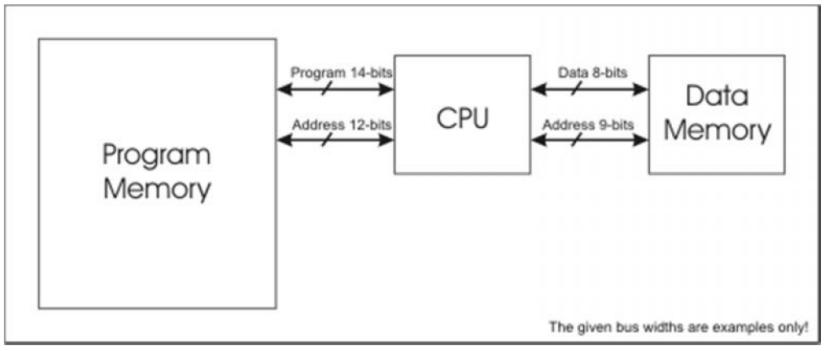

Figure 3: Harvard Architecture

1.2.6 Software Architecture

- **CISC** - is a computer where single instructions can execute several low-level operations (such as a load from memory, an arithmetic operation, and a memory store) and/or are capable of multi-step operations or addressing modes within single instructions.
- **RISC** - is a CPU design strategy based on the insight that simplified (as opposed to complex) instructions can provide higher performance if this simplicity enables much faster execution of each instruction.

1.3 Microprocessor Vs Microcontroller

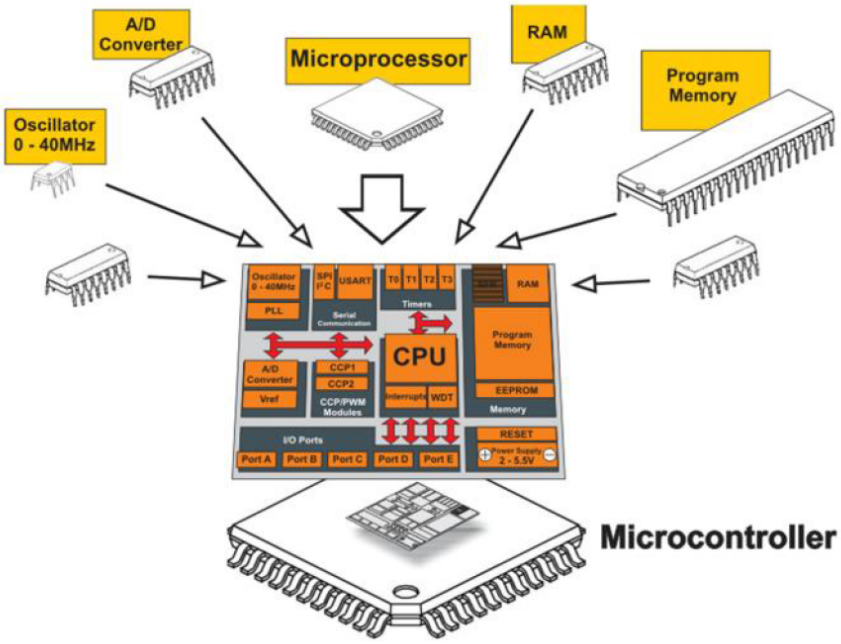

Figure 4: Microcontroller Vs Microprocessor

A microcontroller differs from a microprocessor in many ways. The first and most important difference is the functionality. In order that microprocessor may be used, other components such as memory and I/O peripherals must be added to it. Even though microprocessors are considered powerful computing machines they cannot directly communicate with peripheral devices. Simply, in order to communicate with peripheral environment, the microprocessor must use specialized circuits added as external chips. In short microprocessors are the pure heart of the computers. On the other hand a Microcontroller is a tiny computer (CPU including peripherals) on a single silicon chip. Microcontrollers are usually used in controlled products such as automobile control systems, remote controls, appliances, power tools or any other embedded system.

1.3.1 General Purpose Microprocessor System and Microcontroller System

- A designer using a general purpose microprocessor such as Pentium or PowerPC must add RAM, ROM, I/O ports to make them functional

- Although it becomes bulkier and costlier still they have the advantage of versatility
- But for the microcontroller all are integrated in a single chip, so it is sometimes called "Single Chip Computer"
- In many applications, the space used, the power consumption and the price per unit is much more critical consideration than the computing power, in those applications microcontrollers are used.

1.3.2 Application of Microcontrollers

- Consumer electronics – mobile phones, digital cameras, camcorders, calculators;
- Home Appliances – microwave ovens, answering machines, home security, thermostat;
- Office Automation – fax machines, copiers, printers and scanners;
- Business equipment – cash registers, alarm systems, card readers, product scanners;
- Automobiles – transmission control, cruise control, fuel injection, anti-lock brakes, active suspension.

1.3.3 Characteristics of Embedded Systems

- Single function – an embedded system usually executes only one program repeatedly until power is cut off.
- Tightly constrained – All computing systems have constraints on design metrics, but those on embedded systems can be especially tight.
- Reactive and Real-time – many embedded systems must continually react to changes in the system's environment and must computer certain results in real time without delay.

1.3.4 Design Challenges

- Non-recurring engineering cost
- Unit cost
- Size
- Performance
- Power
- Flexibility

- Time-to-market
- Time-to-prototype
- Correctness
- Safety

1.3.5 Microcontroller for Embedded System

- Microprocessors and microcontrollers are widely used in embedded system products.
- An embedded system is controlled by its own internal microprocessor (or microcontroller) as opposed to external controller.
- Printer is an example of embedded system, it does only one task, taking data from the PC and print it.
- In a mouse, there is a microcontroller which tracks the mouse's position and sends the information to the PC.
- In an embedded system, typically only one application software is burned into its ROM.

1.3.6 High End Embedded System

- Sometimes microcontroller is inadequate for a certain embedded system.
- So, some manufacturers like Intel, AMD, and Freescale targeted their microprocessor for the high end embedded market.
- Another chip called ARM (Advance RISC Machine) is used for high end embedded system. ARM is developed by Acorn Computers for the BBC Microcomputer in the 80's.

2 8051 Microcontroller Architecture

2.1 Introduction

The 8051 microcontroller was designed by Intel in 1981. It is an 8-bit microcontroller. It features 4Kb of ROM, 128 bytes of RAM, Two (2) 16-bit Timers in a 40 Pin DIP Package. It also features 32 bi-directional I/O Lines and full duplex serial port. It has five vector interrupt structure, 8 bit stack pointer, 16-bit program counter, and a data pointer. A simplified block diagram is shown in 5

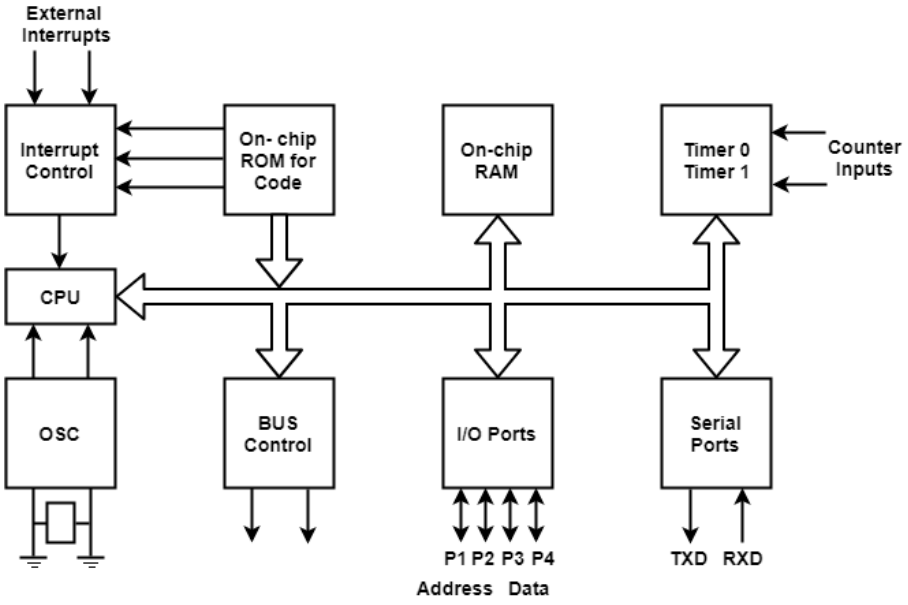

Figure 5: 8051 Architecture

- 8051 has 4KBytes of internal ROM. The address space is from 0000 to 0FFFh. If the program size is more than 4KBytes , the 8051 will fetch the code automatically from external memory.
- Accumulator is an 8-bit register widely used for all arithmetic and logical operations. It is also used to transfer data between external memory. B register is used along with Accumulator for multiplication and division.
- PSW (Program Status Word). This is an 8 bit register which contains the arithmetic status of ALU and the bank select bits of register banks.
- Stack Pointer (SP) – it contains the address of the data item on the top of the stack. Stack may reside anywhere on the internal RAM. On reset, SP is initialized to 07 so that the default stack will start from address 08 onwards.

- Data Pointer (DPTR) – DPH (Data pointer higher byte), DPL (Data pointer lower byte). This is a 16 bit register which is used to furnish address information for internal and external program memory and for external data memory.
- Program Counter (PC) – 16 bit PC contains the address of next instruction to be executed. On reset PC will set to 0000. After fetching every instruction PC will increment by one.

2.2 Pin Diagram

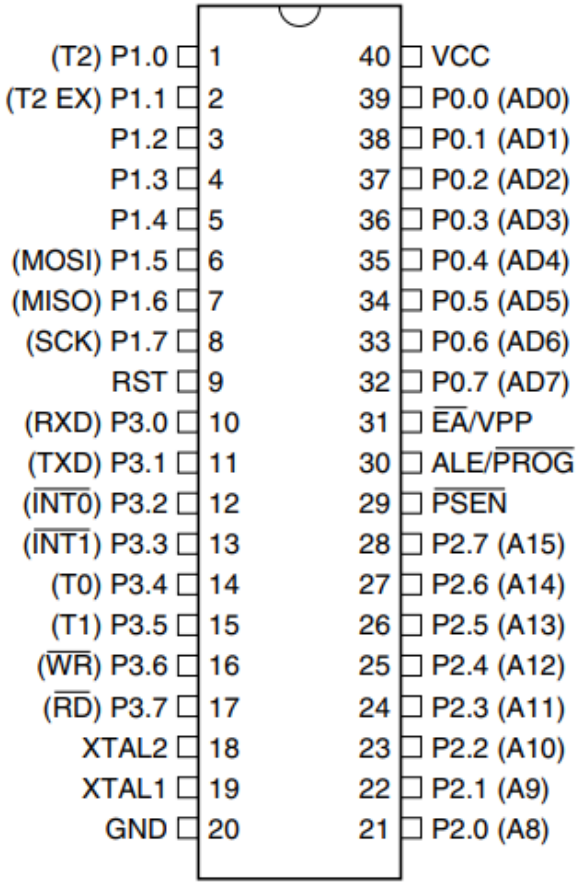

Figure 6: 8051 Pin Diagram

Pinout Description	
Pins 1-8	**PORT 1.** Each of these pins can be configured as an input or an output.
Pin 9	**RESET.** A logic one on this pin disables the microcontroller and clears the contents of most registers. In other words, the positive voltage on this pin resets the microcontroller. By applying logic zero to this pin, the program starts execution from the beginning.
Pins 10-17	**PORT 3.** Similar to port 1, each of these pins can serve as general input or output. Besides, all of them have alternative functions.
Pin 10	**RXD.** Serial asynchronous communication input or Serial synchronous communication output.
Pin 11	**TXD.** Serial asynchronous communication output or Serial synchronous communication clock output.
Pin 12	**INT0.** External Interrupt 0 input
Pin 13	**INT1.** External Interrupt 1 input
Pin 14	**T0.** Counter 0 clock input
Pin 15	**T1.** Counter 1 clock input
Pin 16	**WR.** Write to external (additional) RAM
Pin 17	**RD.** Read from external RAM
Pin 18-19	**XTAL2, XTAL1.** Internal oscillator input and output. A quartz crystal which specifies operating frequency is usually connected to these pins.
Pin 20	**GND.** Ground.
Pin 21-28	**Port 2.** If there is no intention to use external memory then these port pins are configured as general inputs/outputs. In case external memory is used, the higher address byte, i.e. addresses A8-A15 will appear on this port. Even though memory with capacity of 64Kb is not used, which means that not all eight port bits are used for its addressing, the rest of them are not available as inputs/outputs.
Pin 29	**PSEN.** If external ROM is used for storing program then a logic zero (0) appears on it every time the microcontroller reads a byte from memory.
Pin 30	**ALE.** Prior to reading from external memory, the microcontroller puts the lower address byte (A0-A7) on P0 and activates the ALE output. After receiving signal from the ALE pin, the external latch latches the state of P0 and uses it as a memory chip address. Immediately after that, the ALE pin is returned its previous logic state and P0 is now used as a Data Bus.
Pin 31	**EA.** By applying logic zero to this pin, P2 and P3 are used for data and address transmission with no regard to whether there is internal memory or not. It means that even there is a program written to the microcontroller, it will not be executed. Instead, the program written to external ROM will be executed. By applying logic one to the EA pin, the microcontroller will use both memories, first internal then external (if exists).
Pin 32-39	**PORT 0.** Similar to P2, if external memory is not used, these pins can be used as general inputs/outputs. Otherwise, P0 is configured as address output (A0-A7) when the ALE pin is driven high (1) or as data output (Data Bus) when the ALE pin is driven low (0).
Pin 40	**VCC.** +5V power supply.

2.3 Port Structure

2.3.1 Port 0

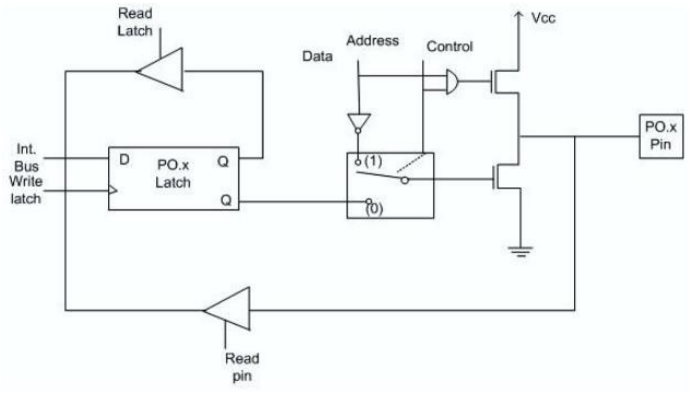

Figure 7: PORT0 Structure

Port 0 is called true bidirectional port as it floats (tristated) when configured as input.
Let us assume that control is '0'. When the port is used as an input port, '1' is written to the latch. In this situation both the output MOSFETs are 'off'. Hence the output pin floats. This high impedance pin can be pulled up or low by an external source. When the port is used as an output port, a '1' written to the latch again turns 'off' both the output MOSFETs and causes the output pin to float. An external pull-up is required to output a '1'. But when '0' is written to the latch, the pin is pulled down by the lower MOSFET. Hence the output becomes zero.
When the control is '1', address/data bus controls the output driver MOSFETs. If the address/data bus (internal) is '0', the upper MOSFET is 'off' and the lower MOSFET is 'on'. The output becomes '0'. If the address/data bus is '1', the upper transistor is 'on' and the lower transistor is 'off'. Hence the output is '1'. Hence for normal address/data interfacing (for external memory access) no pull-up resistors are required.

Port-0 latch is written to with 1's when used for external memory access.

2.3.2 Port 1

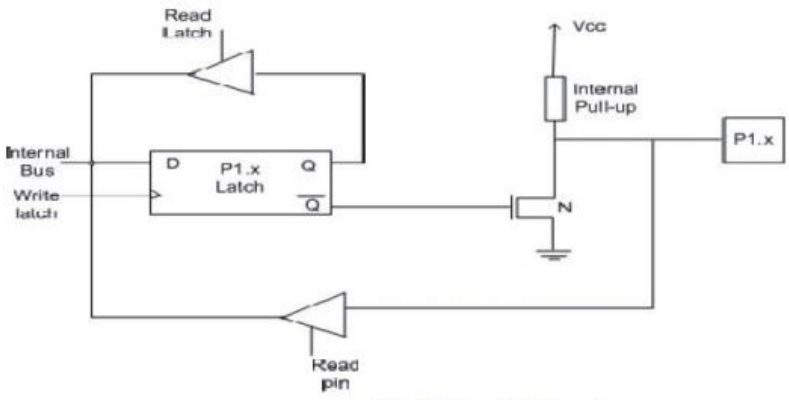

Figure 8: PORT1 Structure

Port-1 does not have any alternate function i.e. it is dedicated solely for I/O interfacing. When used as output port, the pin is pulled up or down through internal pull-up. To use port-1 as input port, '1' has to be written to the latch. In this input mode when '1' is written to the pin by the external device then it read fine. But when '0' is written to the pin by the external device then the external source must sink current due to internal pull-up. If the external device is not able to sink the current the pin voltage may rise, leading to a possible wrong reading.

2.3.3 Port 2

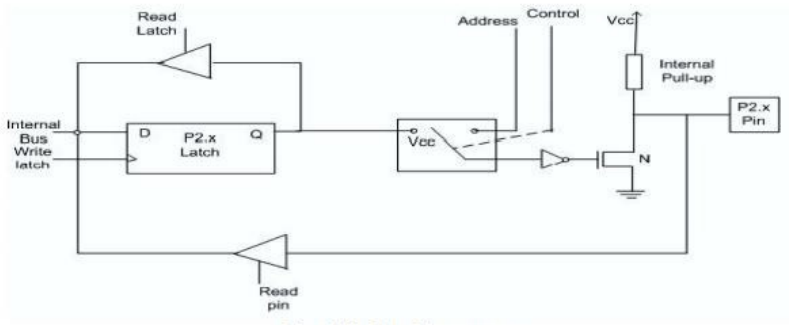

Figure 9: PORT2 Structure

Port-2 is used for higher external address byte or a normal input/output port. The I/O operation is similar to Port-1. Port-2 latch remains stable when Port-2

pin are used for external memory access. Here again due to internal pull-up there is limited current driving capability.

2.3.4 Port 3

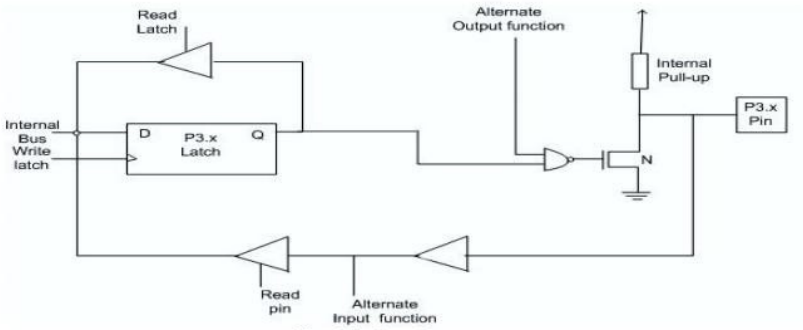

Figure 10: PORT3 Structure

Each pin of Port-3 can be individually programmed for I/O operation or for alternate function. The alternate function can be activated only if the corresponding latch has been written to '1'. To use the port as input port, '1' should be written to the latch. This port also has internal pull-up and limited current driving capability.

3 Basic Input/Output Interfacing

3.1 Output Interfacing - LEDs

Light Emitting Diodes or LEDs are widely used components in many applications. They are made of semiconductor material such as Gallium Arsenide (GaAs) and Gallium Phosphide (GaP). Until the mid 90s Blue and White LEDs do not exist. the development on the Gallium Nitride (GaN) material system completed the pallete of colors.

Main LED Materials

- **Indium Gallium Nitride (InGaN)** - blue, green and ultraviolet high-brightness LEDs
- **Aluminum Gallium Indium Phosphide (AlGaInP)** - yellow, orange and red high-brightness LEDs
- **Aluminum Gallium Arsenide (AlGaAs)** - red and infrared LEDs
- **Gallium Phosphide (GaP)** - yellow and green LEDs

Typical characteristic of an LED is shown below

Absolute Maximum Ratings: (Ta=25℃) .

ITEMS	Symbol	Absolute Maximum Rating	Unit
Forward Current	I_F	20	mA
Peak Forward Current	I_{FP}	30	mA
Suggestion Using Current	I_{su}	16-18	mA
Reverse Voltage (V_R=5V)	I_R	10	uA
Power Dissipation	P_D	105	mW
Operation Temperature	T_{OPR}	-40 ~ 85	℃
Storage Temperature	T_{STG}	-40 ~ 100	℃
Lead Soldering Temperature	T_{SOL}	Max. 260℃ for 3 Sec. Max. (3mm from the base of the expoxy bulb)	

Absolute Maximum Ratings: (Ta=25℃)

ITEMS	Symbol	Test condition	Min.	Typ.	Max.	Unit
Forward Voltage	V_F	I_F=20mA	1.8	---	2.2	V
Wavelenength (nm) or TC(k)	$\Delta\lambda$	I_F=20mA	620	---	625	nm
*Luminous intensity	I_V	I_F=20mA	150	---	200	mcd
50% Viewing Angle	$2\theta 1/2$	I_F=20mA	40	---	60	deg

Figure 11: Typical LED Datasheet

3.1.1 Connecting an LED to the Microcontroller

There are two ways which we can interface LED to the microcontroller.

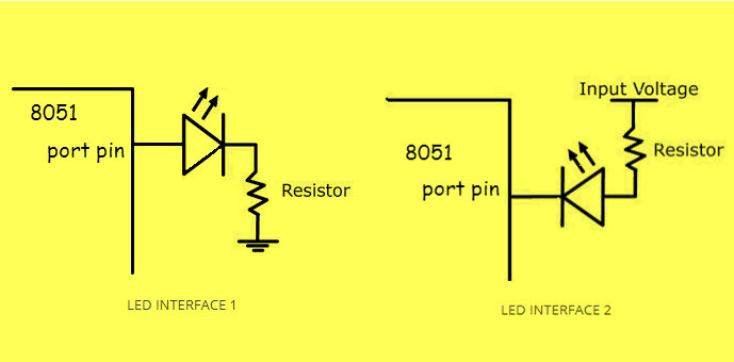

Figure 12: LED Typical Connection

With the LED Interface 1, the LED is connected in current source mode, thus the microcontroller is providing current to the LED, with this connection the microcontroller port must be HIGH to turn the LED on. LED Interface 2 is current sink mode, the Input Voltage is the source of current thus the flow of current exits thru the microcontroller pin. In current sink, the microcontroller must be LOW to make the LED glow.

Notice that a resistor is used with the LED. The resistor is an integral part to limit the current flow to protect the LED as well as the microcontroller. **The AT89S52 can provide a maximum of 26mA at PORT0 and 71mA for Ports 1,2 and 3**. However it is recommended that current in **each port pin must be limited to 10mA under steady state condition**. With these we can compute the value of the resistor by looping from the microcontroller to the LED resulting to the following equation:

$$R = \left(\frac{VCC - V_F}{I_F} \right)$$

Example:
Interface a typical red LED with VF=1.2V and IF=8mA

$$R = \left(\frac{5V - 1.2V}{8mA} \right); R = 475ohms$$

if the computed value is not within commercial values available, round off to the nearest commercial values say 470 ohms. Re-calculate if the available resistor is within the maximum electrical characteristics of the LED and the microcontroller. Compute for the power dissipated, for our application a typical 0.25W

resistor is more than enough. Because of the presence of pull-up resistors, it is recommended that LEDs are connected in current sink mode as shown in figure 13.

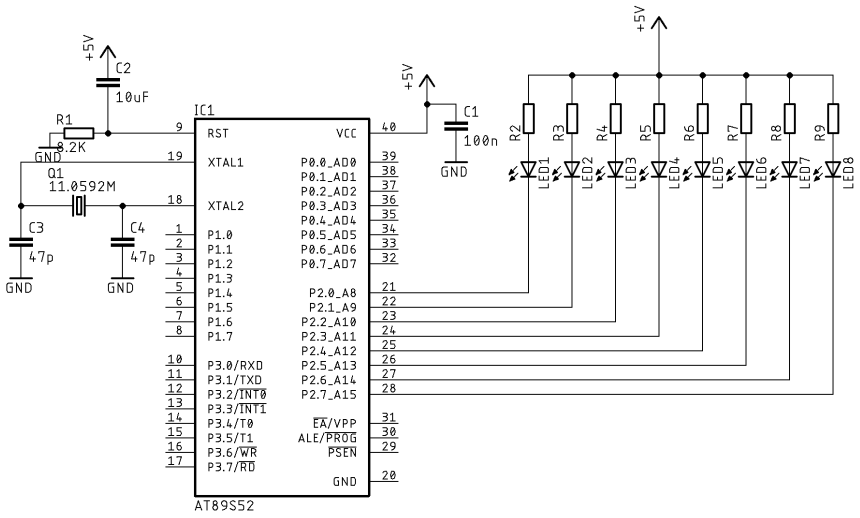

Figure 13: LED Interface at Port 2

Interfacing with microcontroller is incomplete without the firmware. The code to turn on all the LEDs is shown below. It is compiled using Keil uVision C51 Integrated Development Environment.

```
1
2  include <reg52.h>
3
4  void main (void)
5  {
6     P2 = 0x00;  //turn on all leds at port 2
7
8     while(1);
9
10 }
```

To turn on a particular LED for example LED at Port2 bit 5, we will be using the sbit directive to access an individual bit in a port.

```
1
2  include <reg52.h>
3
4  sbit LED5 = P2^5;
5
6  void main (void)
7  {
8    LED5 = 0; //turn on LED 5
9
10   while(1);
11
12 }
```

To make the LEDs blink we need some sort of delay since the microcontroller runs fast that the human eye could not distinguish. Using a crystal of 11.0592MHz and 1 machine cycle needs 12 clock cycles to complete we get 11.0592M/12 = 926.6KHz. Getting the period T=1/F, we get 1.078uS. To create an approximate 1ms delay we need 1275 machine cycles (1.079x1275 = 1375 approx 1.3ms) .

```
1
2  /****
3  LED Blink
4  ****/
5  include <reg52.h>
6
7  /*milliseconds delay for AT89S52 with 11.0592MHz Clock using Keil Compiler*/
8  void delay_ms (unsigned int ms)
9  {
10    unsigned int i,j;
11
12       for(i=0; i<ms; i++){
13        for(j=0; j<1275; j++);
14       }
15 }
16
17 void main (void)
18 {
19
20    while(1)
21       {
22         P2 = 0x00;
23         delay_ms(500);
24         P2 = 0xFF;
25         delay_ms(500);
26       }
27 }
```

3.2 Input Interfacing - Switches

The switch is a basic input device used to control the operation of any output device connected to the microcontroller. It basically breaks the electrical circuit and interrupts the flow of current. It can be connected in active high or active low configurations.

- **Active High** - A pull-down resistor is connected to ground. When pressed it asserts logic high and when disconnected it asserts logic low.
- **Active Low** - A pull-up resistor is connected to VCC. When pressed it asserrts logic low and when disconnected it asserts logic high.

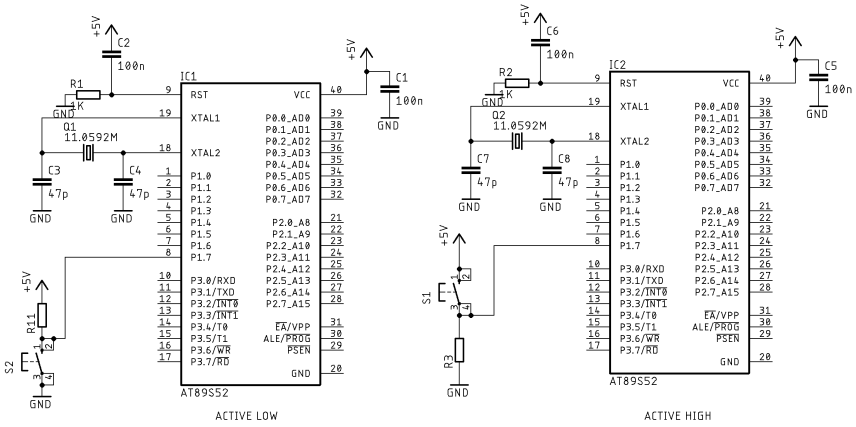

Figure 14: Typical Switch Connection

In using switches in 8051, **active low** configuration is recommended because of the presence of the internal pull-up resistors at Port 1,2 and 3. When using Port 0 an external 10Kohm pull-up resistor is required.

3.2.1 Using push button switches

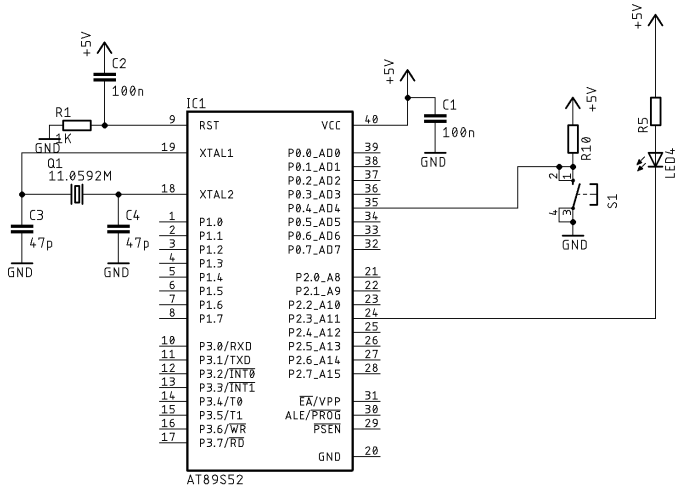

Figure 15: LED and Switch Interface

A switch is connected at Port 0 bit 4 and a LED is connected at Port2 bit 3. We would like to turn on the LED when the switch is pressed and turn off when the switch is not engaged. The bit where the switch is connected must be set to '1' to tell the microcontroller that it is in input mode. To determine the state of the switch we make use of the if statement.

```
1
2  include <reg52.h>
3
4  sbit SW = P0^4;
5  sbit LED = P2^3;
6
7  void main (void)
8  {
9    SW = 1; // Tell the MCU to make P0.4 as input
10
11    while(1)
12       {
13          if(SW == 0){
14            LED = 0;
15          }else{
16            LED = 1;
17          }
18       }
19 }
```

3.2.2 Switch Bounce

When we press a button or toggle a switch, two metal parts come in contact but they dont connect instantly but the metal parts connect and disconnect several times before the actual connection is made. The same thing happens when the button is release. This results to false triggering also known as **switch bounce**.

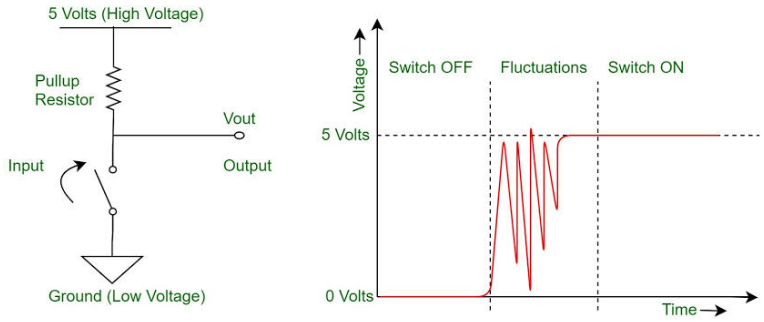

Figure 16: Switch Bounce

Switch bouncing is the non-ideal behavior of any switch which generates multiple transition of a single input. Switch bouncing is not a major problem when dealing with power circuits, but is a big deal in digital circuits. This however can be solved by using hardware debouncing circuit or a software debouncing routine. In software, addition of a little delay can usually solve this problem. Shown below is an example code to debounce a switch.

```
#include<reg52.h>

//Value of Delay
#define DEBOUNCE_VALUE  240

//Switch Status
#define SWITCH_PRESSED    1
#define SWITCH_BOUNCE     0

//LED STATUS
#define LED_ON           1
#define LED_OFF          0

// Connection
sbit Led = P2^3;    //pin connected to toggle Led
sbit Switch =P0^4;  //Pin connected to toggle led

//Function provides a delay to prevent from switch bouncing
void DebounceDelay(void)
{
   int iTimeDelay = 0;
      for(iTimeDelay=0; iTimeDelay < DEBOUNCE_VALUE; iTimeDelay++){
         }
}

//Function to check the status of Switch
int CheckSwitchDebounce(void)
{
   int iRetValue =  SWITCH_BOUNCE;
   if(Switch == 0)
   {
      DebounceDelay(); //Wait time more then bouncing period
      if(Switch == 0)
         {
            iRetValue = SWITCH_PRESSED;
         }
   }
return iRetValue ;
}


void main(void)
{
   Led = 0; //configuring as output pin
   Switch = 1; //Configuring as input pin

   while(1)  //Super loop to continuously monitor the status of the switch
      {
          //Check the switch status
         if(SWITCH_PRESSED == CheckSwitchDebounce())
         {
            Led = LED_ON;     //Led On
         }
         else
         {
            Led = LED_OFF;    //Led off
         }
      }
}
```