**MICROCONTROLLER BASED DIGITAL IC TESTER**

**BY:-**

**DIVYA ORAON : 160606009**

**POONAM KUMARI : 160606019**

**SADHANA KUMARI : 160606025**

**SHUBHAM ANAND : 160606039**

Under the guidance of:

**Priyanka saha**

**DOCUMENTATION ON**

**MICROCONTROLLER BASED DIGITAL IC TESTER**

CONTENTS:

1. INTRODUCTION
   1. GENERAL INTRODUCTION
   2. INTRODUCTION TO MICROCONTROLLER
      1. WHAT IS MICROCONTROLLER
      2. MICROCONTROLLER CHIP DETAILS WHICH WE USE
2. BRIEF DESCRIPTION OF 8051 MICROCONTROLLER
   1. GENERAL DETAILS
   2. COMPARISONS BETWEEN MICROPROCESSORS & MICROCONTROLLERS
   3. ADVANTAGES OF HARVARD ARCHITECTURE
   4. MEMORY ORGANISATION (INTERNAL & EXTERNAL)
   5. 8051 ARCHITECTURE DIAGRAM
   6. FEATURES OF 8051
   7. 8051 BUSES &CPU FUNCTIONING IN DATA TRANSFER
   8. MICROCONTROLLER 89S52 DETAILS
      1. BLOCK DIAGRAM
      2. IMPORTANT FEATURES
      3. PIN CONFIGURATION
      4. PIN DESCRIPTION
   9. 8051 INTERRUPTS
   10. ADDRESSING MODES
   11. TYPES OF INSTRUCTIONS
   12. APPLICATION OF MICROCONTROLLER
       1. LED ON/OFF WITH THE HELP OF REMOTE CONTROL
          1. LED INTERFACING
       2. INTERFACING LCD DISPLAY WITH MICROCONTROLLER
          1. SPECIAL CHARACTERISTICS OF LCD
          2. ADVANTAGES OF LCD
          3. DRAWBACKS
          4. INTERFACING LCD WITH 8051
3. PROJECT DETAILS
   1. ASSIGNED PROJECT
   2. OBJECTIVE
   3. THEORY
      1. WHAT IS DIGITAL IC TESTER & ITS FUNCTIONING
      2. FEATURES OF IC TESTER
      3. INTRODUCTION TO LCD
      4. INTRODUCTION TO 4x3 KEYPAD
   4. COMPONENTS
   5. BLOCK DIAGRAM
   6. CIRCUIT DIAGRAM
   7. TECHNICAL SPECIFICATION
   8. IC DETAILS
      1. IC 7408
      2. IC 7432
      3. IC 7400
      4. IC 7486
      5. IC74266
      6. IC7402
      7. IC 7404
   9. SOFTWARE PROGRAM
   10. FLOW CHARTS
   11. SNAPSHOTS
   12. SOFTWARE USED
   13. WORK DONE
   14. CONCLUSION
4. FUTURE ASPECTS
5. REFERENCES
6. ***INTRODUCTION***
   1. ***General Introduction***

In any manufacturing industry there are continuous efforts in cost reductions, upgrade quality and improve overall efficiencies. In electronic industry, with dramatic increase in circuit complexity and the need for the higher levels of reliability, testing is an important aspect. However we should recognize in the real world that no product is perfect, so that testing and in particular automatic testing will be an essential part of production in the foreseeable future.

In industries, research centers and college, some common IC's are frequently used; many times people face problems due to some fault in these integrated circuits. So it is very essential to test them before actually using them in any of the applications. Microcontroller based digital IC tester is best solution for these problems.

This project has the capability of testing any available digital IC of the TTL or CMOS family of 20 pins. The main advantage over the industry standard for the project is its low cost and eases of updating to any new IC design which may be inducted in the market by any company only through software updating.

***1.02 Introduction To Microcontroller***

Microcontroller is an electronic device which is used as a computing unit or arithmetic unit like a computer

On the other hand, it can be said that it is a data processing unit. Sometimes it may be used to control a device, i.e. turn on or turn off a device or control a device. It is a 40pin IC having HARVARD architecture with 16 bit address bus and 8 bit data bus and 64KByte programming memory and 64Kbyte data memory

* + 1. **What is microcontroller?**

A microcontroller is also known as a true computer on a single chip. It has a CPU,PC ,SP registers, in addition to a fixed amount of RAM,ROM and I/O ports, including serial communication & timer, all embedded together on a single chip. In some microcontrollers an ADC & DAC are also embedded in a single chip.

* + 1. ***Microcontroller Chip Details Which We Use***

The next part of the proposed work is the temperature monitoring. The 8051 microcontroller is used here as the main controller. The microcontroller used here is known as AT89C52 which has 8KB on-chip flash memory, 256 bytes of Ram, 4 ports each of 8 bit P0-P3, 3 timers, serial communication module and interrupt module makes the system more versatile for our present project work.

1. ***BRIEF DESCRIPTION OF 8051 MICROCONTROLLER***
   1. ***General Details:***

A microcontroller is also known as a true computer on a single chip. It has a CPU, PC,SP registers, in addition to a fixed amount of RAM,ROM and I/O ports, including serial communication system and timer, all embedded together on a single chip. In some microcontrollers an ADC & DAC are also embedded on a single chip

**2.02 Comparisons between microprocessor and microcontroller:**

**MICROCONTROLLER**R

**MICROPROCESSOR**

|  |  |
| --- | --- |
| It is a 40 pin IC | It is a 40 pin IC |
| It has a 16 bit address bus | It has a 16 bit address bus |
| It has a 8 bit data bus | It has a 8 bit data bus |
| Developed by VON NEUMANN architecture | Developed by Harvard architecture. |
| Slower than 8051 controller | Faster than 8085 processor |
| Available memory-64kbyte | Available memory-64kbyte program memory and 64kbyte data memory |
| Operating frequency-3 to 5 MHz | Operating frequency-12 MHz |
| RAM,ROM Interrupt Controller, bus controller, I/O ports etc are connected externally. | RAM,ROM Interrupt Controller, bus controller, I/O ports etc are connected internally. |
| Multi-purpose(single processor Can be used for many jobs) | Used for dedicated purpose (single controller can be used for single purpose) |
| Costlier. | Cheaper. |
| It is much bulkier due to attached circuitry | Lighter as on chip circuitry. |

Table.1: Comparisons between microprocessor and microcontroller

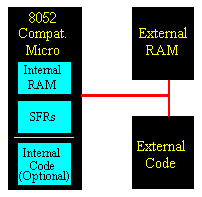
**2.03 Advantages of Harvard Architecture:**

* It has two types of memories separated.
* Data can be accessed even when program is being executed i.e. instruction fetches can occur in parallel with data transfers.
* Much faster due to presence of separate data and program memory.
* Data much secured.

**2.04 Memory Organization**:

The 8051 has three very general types of memory. To effectively program the 8051 it is necessary to have a basic understanding of these memory types.

The memory types are illustrated in the following graphic. They are: On-Chip Memory, External Code Memory, and External RAM.



**External Code Memory:**

External Code Memory is code (or program) memory that resides off-chip. This is often in the form of an external EPROM.Code memory is the memory that holds the actual 8051 program that is to be run. This memory is limited to 64K and comes in many shapes and sizes: Code memory may be found *on-chip*, either burned into the microcontroller as ROM or EPROM. Code may also be stored completely *off-chip* in an external ROM or, more commonly, an external EPROM. Flash RAM is also another popular method of storing a program. Various combinations of these memory types may also be used--that is to say, it is possible to have 4K of code memory *on-chip* and 64k of code memory *off-chip* in an EPROM.

When the program is stored on-chip the 64K maximum is often reduced to 4k, 8k, or 16k. This varies depending on the version of the chip that is being used. Each version offers specific capabilities and one of the distinguishing factors from chip to chip is how much ROM/EPROM space the chip has.

However, code memory is most commonly implemented as off-chip EPROM. This is especially true in low-cost development systems and in systems developed by students.

**External RAM:**

External RAM is RAM memory that resides off-chip. This is often in the form of standard static RAM or flash RAM.As an obvious opposite of *Internal RAM*, the 8051 also supports what is called *External RAM*.

As the name suggests, External RAM is any random access memory which is found *off-chip*. Since the memory is off-chip it is not as flexible in terms of accessing, and is also slower. For example, to increment an Internal RAM location by 1 requires only 1 instruction and 1 instruction cycle. To increment a 1-byte value stored in External RAM requires 4 instructions and 7 instruction cycles. In this case, external memory is 7 times slower!

What External RAM loses in speed and flexibility it gains in quantity; While Internal RAM is limited to 128 bytes (256 bytes with an 8052), the 8051 supports External RAM up to 64K.

**On-Chip Memory:**

On-Chip Memory refers to any memory (Code, RAM, or other) that physically exists on the microcontroller itself.

The 8051 includes a certain amount of on-chip memory. On-chip memory is really one of two types: *Internal RAM* and *Special Function Register (*SFR) memory. The layout of the 8051's internal memory is presented in the following memory map:

The **layout of the 8051's internal memory** is presented in the following memory map:

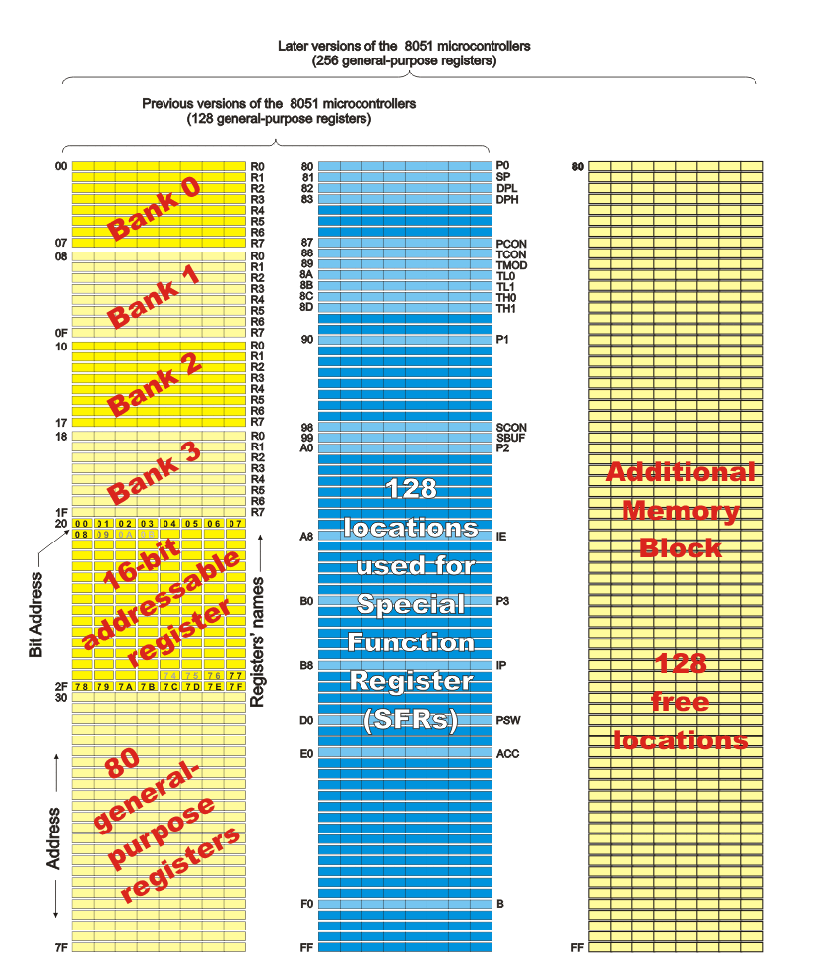
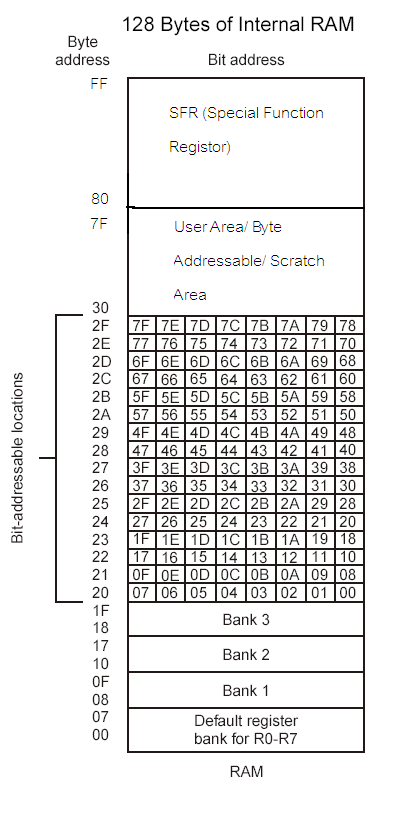
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Fig.8 layout of the 8051's internal memory



**RAM memory space allocation in the 8051**

**REGISTER BANKS:**

**Memory Location: 00- IF H**

This part of RAM is divided into four register banks.

The four register banks are divided into 8 registers R0-R7.

The default register bank is zero but we can also choose the other register banks.

The register bank one is not used (generally) as it also serves purpose of stack**.**

BANK SELECTION:

|  |  |  |
| --- | --- | --- |
| **RS1(PSW.4)** | **RS0(PSW.3)** | **BANK SELECTED** |
| 0 | 0 | Bank0 |
| 0 | 1 | Bank1 |
| 1 | 0 | Bank2 |
| 1 | 1 | Bank3 |

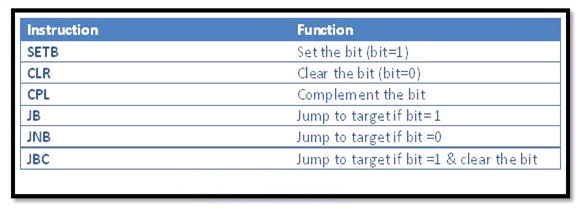
Table.2: Bank Selection

**BIT ADDRESSABLE RAM:**

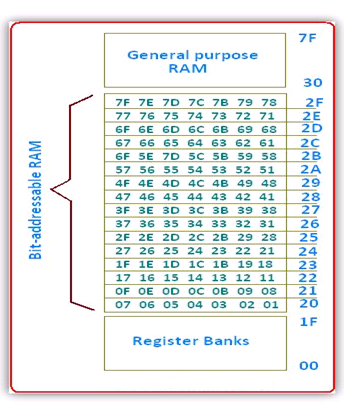
**Memory Locations: 20- 2F H**

* This part of RAM is bit addressable means we can use the bits address for directly using them.
* A bit can be set with SETB & cleared with CLR.
* We can only access this directly by address. There are no special names for this part of RAM.

**Operations we can perform on a bit are shown below in the table.**

****

**Below we can see how the bits are numbered**:

****

**SCRATCH PAD RAM:**

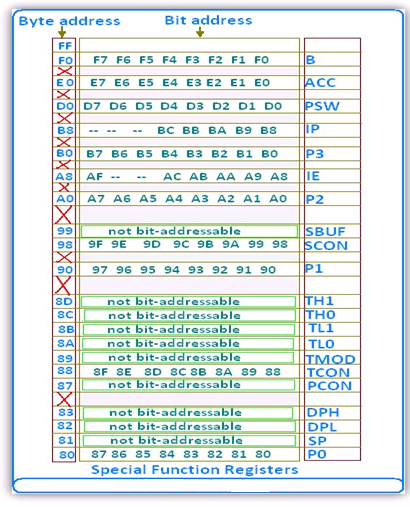
**Memory Address: 30 -7F H**

* This part of RAM doesn’t have any specific function and is known as general purpose RAM. We can use it in whatever way we like.
* This can be accessed only by its address.

**SPECIAL FUNCTION REGISTERS(SFRs):**

**Memory Location: 80- FF H**

* All the ports are also designated as special function register
* Some of the SFRs are bit addressable.
* The SFR’s can be accessed through names as well as address.
* The bit address is independent of the byte address.



**Figure (a): Special Function Register**

**What Are SFRs?**

The 8051 is a flexible microcontroller with a relatively large number of modes of operations. Your program may inspect and/or change the operating mode of the 8051 by manipulating the values of the 8051's Special Function Registers (SFRs).

SFRs are accessed as if they were normal Internal RAM. The only difference is that internal RAM is from address 00h through 7Fh whereas SFR registers exist in the address range of 80h through FFH.

Each SFR has an address (80h through FFH) and a name as shown in the above figure (a). As you can see, although the address range of 80h through FFH offers 128 possible addresses, there are only 21 SFRs in a standard 8051. All other addresses in the SFR range (80h through FFH) are considered invalid. Writing to or reading from these registers may produce undefined values or behavior.

**SFR Types**

As we can see in the figure (a) itself, SFRs related to the I/O ports. The 8051 has four I/O ports of 8 bits, for a total of 32 I/O lines. Whether a given I/O line is high or low and the value read from the line are controlled by the other SFRs in green.

Some SFRs which in some way control the operation or the configuration of some aspect of the 8051; For example, **TCON** controls the timers, **SCON** controls the serial port.

The remaining SFRs, with green backgrounds, are "other SFRs." These SFRs can be thought of as auxiliary SFRs in the sense that they don't directly configure the 8051 but obviously the 8051 cannot operate without them. For example, once the serial port has been configured using **SCON**, the program may read or write to the serial port using the **SBUF** register.

**SFR Descriptions**

In This section we will discuss each of the standard SFRs found in the above SFR figure (a)

**P0 (Port 0, Address 80h, and Bit-Addressable):** This is input/output port 0. Each bit of this SFR corresponds to one of the pins on the microcontroller. For example, bit 0 of port 0 is pin P0.0, bit 7 is pin P0.7. Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin whereas a value of 0 will bring it to a low level.

**SP (Stack Pointer, Address 81h):** This is the stack pointer of the microcontroller. This SFR indicates where the next value to be taken from the stack will be read from in Internal RAM. If you push a value onto the stack, the value will be written to the address of SP + 1. That is to say, if SP holds the value 07h, a PUSH instruction will push the value onto the stack at address 08h. This SFR is modified by all instructions which modify the stack, such as PUSH, POP, and LCALL, RET, RETI, and whenever interrupts are provoked by the microcontroller.

**DPL/DPH (Data Pointer Low/High, Addresses 82h/83h):** The SFRs DPL and DPH work together to represent a 16-bit value called the Data Pointer. The data pointer is used in operations regarding external RAM and some instructions involving code memory. Since it is an unsigned two-byte integer value, it can represent values from 0000h to FFFFH (0 through 65,535 decimal).

**PCON (Power Control, Addresses 87h):** The Power Control SFR is used to control the 8051's power control modes. Certain operation modes of the 8051 allow the 8051 to go into a type of "sleep" mode which requires much less power. These modes of operation are controlled through PCON. Additionally, one of the bits in PCON is used to double the effective baud rate of the 8051's serial port.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| SMOD | x | x | x | GF1 | GF0 | PD | IDL |

* SMOD – Serial mode bit used to determine the baud rate with Timer 1.

Baud rate= Oscillator frequency in Hz ***/*** N [256-(TH1)]

* If SMOD = 0 then N = 384. If SMOD = 1 then N = 192. TH1 is the high byte of timer 1 when it is in 8-bit auto reload mode.
* GF1 and GF0 are General purpose flags not implemented on the standard device
* PD is the power down bit. Not implemented on the standard device
* IDL activate the idle mode to save power. Not implemented on the standard device

**TCON (Timer Control, Addresses 88h, and Bit-Addressable):** The Timer Control SFR is used to configure and modify the way in which the 8051's two timers operate. This SFR controls whether each of the two timers is running or stopped and contains a flag to indicate that each timer has overflowed. Additionally, some non-timer related bits are located in the TCON SFR. These bits are used to configure the way in which the external interrupts are activated and also contain the external interrupt flags which are set when an external interrupt has occurred.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| TF1 | TR1 | TF0 | TR0 | IE1 | IT1 | IE0 | IT0 |

* TF1 – Timer 1 overflow flag
* TR1 – Timer 1 run control bit
* TF0 – Timer 0 overflow flag
* TR0 – Timer 0 run control bit
* IE1 – External interrupt 1 edge flag. Set to 1 when edge detected.
* IT1 – Edge control bit for external interrupt 1. 1 = edge, 0 = level
* IE0 – External interrupt 0 edge flag. Set to 1 when edge detected
* IT0 – Edge control bit for external interrupt 0. 1 = edge, 0 = level

**TMOD (Timer Mode, Addresses 89h):** The Timer Mode SFR is used to configure the mode of operation of each of the two timers. Using this SFR your program may configure each timer to be a 16-bit timer, an 8-bit auto reload timer, a 13-bit timer, or two separate timers. Additionally, you may configure the timers to only count when an external pin is activated or to count "events" that are indicated on an external pin.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| Gate | C/T | M1 | M0 | Gate | C/ T | M1 | M0 |
| **Timer 1** | | | | **Timer 0** | | | |

* Gate – if 1 timer x is enabled when INTx is high and TRx is high. if 0 timer x is enabled when TRx is high.
* C/T- if 1 timer x is clocked from Tx pin. if 0 timer x is clocked from oscillator/12

|  |  |  |
| --- | --- | --- |
| M1 | M0 | MODE |
| 0 | 0 | 13- bit mode for compatibility to 8084 family |
| 0 | 1 | 16-bit timer/counter. user must reload in software |
| 1 | 0 | 8-bit auto reload. TLx is automatically reloaded from THx |
| 1 | 1 | TL0 is 8-bit counter controlled by Timer0 control bits.TH0 is 8-bit counter controlled by Timer1 control bits.Timer1 is stopped |

**TL0/TH0 (Timer 0 Low/High, Addresses 8Ah/8Ch):** These two SFRs, taken together, represent timer 0. Their exact behavior depends on how the timer is configured in the TMOD SFR; however, these timers always count up. What is configurable is how and when they increment in value.

**TL1/TH1 (Timer 1 Low/High, Addresses 8Bh/8Dh):** These two SFRs, taken together, represent timer 1. Their exact behavior depends on how the timer is configured in the TMOD SFR; however, these timers always count up. What is configurable is how and when they increment in value.

**P1 (Port 1, Address 90h, and Bit-Addressable):** This is input/output port 1. Each bit of this SFR corresponds to one of the pins on the microcontroller. For example, bit 0 of port 1 is pin P1.0, bit 7 is pin P1.7. Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin whereas a value of 0 will bring it to a low level.

**SCON (Serial Control, Addresses 98h, Bit-Addressable):** The Serial Control SFR is used to configure the behavior of the 8051's on-board serial port. This SFR controls the baud rate of the serial port, whether the serial port is activated to receive data, and also contains flags that are set when a byte is successfully sent or received.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| SM0 | SM1 | SM2 | REN | TB8 | RB8 | TI | RI |

* SM2 – Enables multiprocessor communication in modes 2 and 3.
* REN – Receiver enable
* TB8 – Transmit bit 8. This is the 9th bit transmitted in modes 2 and 3.
* RB8 – Receive bit 8. This is the 9th bit received in modes 2 and 3.
* TI – Transmit interrupt flag. Set at end of character transmission. Cleared in software.
* RI – Receive interrupt flag. Set at end of character reception. Cleared in software.

|  |  |  |  |
| --- | --- | --- | --- |
| SM0 | SM1 | Operation | Baud rate |
| 0 | 0 | Shift register | Osc/12 |
| 0 | 1 | 8-bit UART | Set by timer |
| 1 | 0 | 9-bit UART | Osc/12 or Osc/64 |
| 1 | 1 | 9-bit UART | Set by timer |

**SBUF (Serial Control, Addresses 99h):** The Serial Buffer SFR is used to send and receive data via the on-board serial port. Any value written to SBUF will be sent out the serial port's TXD pin. Likewise, any value which the 8051 receives via the serial port's RXD pin will be delivered to the user program via SBUF. In other words, SBUF serves as the output port when written to and as an input port when read from.

**P2 (Port 2, Address A0h, and Bit-Addressable):** This is input/output port 2. Each bit of this SFR corresponds to one of the pins on the microcontroller. For example, bit 0 of port 2 is pin P2.0, bit 7 is pin P2.7. Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin whereas a value of 0 will bring it to a low level.

**IE (Interrupt Enable, Addresses A8h):** The Interrupt Enable SFR is used to enable and disable specific interrupts. The low 7 bits of the SFR are used to enable/disable the specific interrupts, where as the highest bit is used to enable or disable ALL interrupts. Thus, if the high bit of IE is 0 all interrupts are disabled regardless of whether an individual interrupt is enabled by setting a lower bit.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| EA | x | ET2 | ES | ET1 | EX1 | ET0 | E |

* EA – Global interrupt enable
* x – not defined
* ET2 – Timer 2 interrupt enable
* ES – Serial port interrupt enable
* ET1 – Timer 1 interrupt enable
* EX1 – External interrupt 1 enable
* ET0 – Timer 0 interrupt enable
* EX0 – External interrupt 0 enable

**P3 (Port 3, Address B0h, and Bit-Addressable):** This is input/output port 3. Each bit of this SFR corresponds to one of the pins on the microcontroller. For example, bit 0 of port 3 is pin P3.0, bit 7 is pin P3.7. Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin whereas a value of 0 will bring it to a low level.

**IP (Interrupt Priority, Addresses B8h, and Bit-Addressable):** The Interrupt Priority SFR is used to specify the relative priority of each interrupt. On the 8051, an interrupt may either be of low (0) priority or high (1) priority. An interrupt may only interrupt interrupts of lower priority. For example, if we configure the 8051 so that all interrupts are of low priority except the serial interrupt, the serial interrupt will always be able to interrupt the system, even if another interrupt is currently executing. However, if a serial interrupt is executing no other interrupt will be able to interrupt the serial interrupt routine since the serial interrupt routine has the highest priority. 0

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| x | x | PT2 | PS | PT1 | PX1 | PT0 | PX |

* x – not defined
* PT2 – Priority for timer 2 interrupt
* PS – Priority for serial port interrupt
* PT1 – Priority for timer 1 interrupt
* PX1 – Priority for external interrupt 1
* PT0 – Priority for timer 0 interrupt
* PX0 – Priority for external interrupt 0

**PSW (Program Status Word, Addresses D0h, and Bit-Addressable):** The Program Status Word is used to store a number of important bits that are set and cleared by 8051 instructions. The PSW SFR contains the carry flag, the auxiliary carry flag, the overflow flag, and the parity flag. Additionally, the PSW register contains the register bank select flags which are used to select which of the "R" register banks are currently selected.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| CY | AC | F0 | RS1 | RS0 | OV | -- | P |

* CY PSW.7 Carry flag.
* AC PSW.6 Auxiliary carry flag.
* F0 PSW.5 Available to the user for general purpose.
* RS1 PSW.4 Register Bank selector bit 1.
* RS0 PSW.3 Register Bank selector bit 0.
* OV PSW.2 Overflow flag.
* -- PSW.1 User definable bit.
  + - P PSW.0 Parity flag. Set/cleared by hardware each instruction cycle to indicate an Odd / even number of 1 bit in the accumulator.

|  |  |  |  |
| --- | --- | --- | --- |
| RS1 | RS0 | Register bank | Address |
| 0 | 0 | 0 | 00H-07H |
| 0 | 1 | 1 | 08H-0FH |
| 1 | 0 | 2 | 10H-17H |
| 1 | 1 | 3 | 18H-1FH |

**ACC (Accumulator, Addresses E0h, and Bit-Addressable):** The Accumulator is one of the most-used SFRs on the 8051 since it is involved in so many instructions. The Accumulator resides as an SFR at E0h, which means the instruction **MOV A, #20h** is really the same as **MOV E0h,#20h**. However, it is a good idea to use the first method since it only requires two bytes whereas the second option requires three bytes.

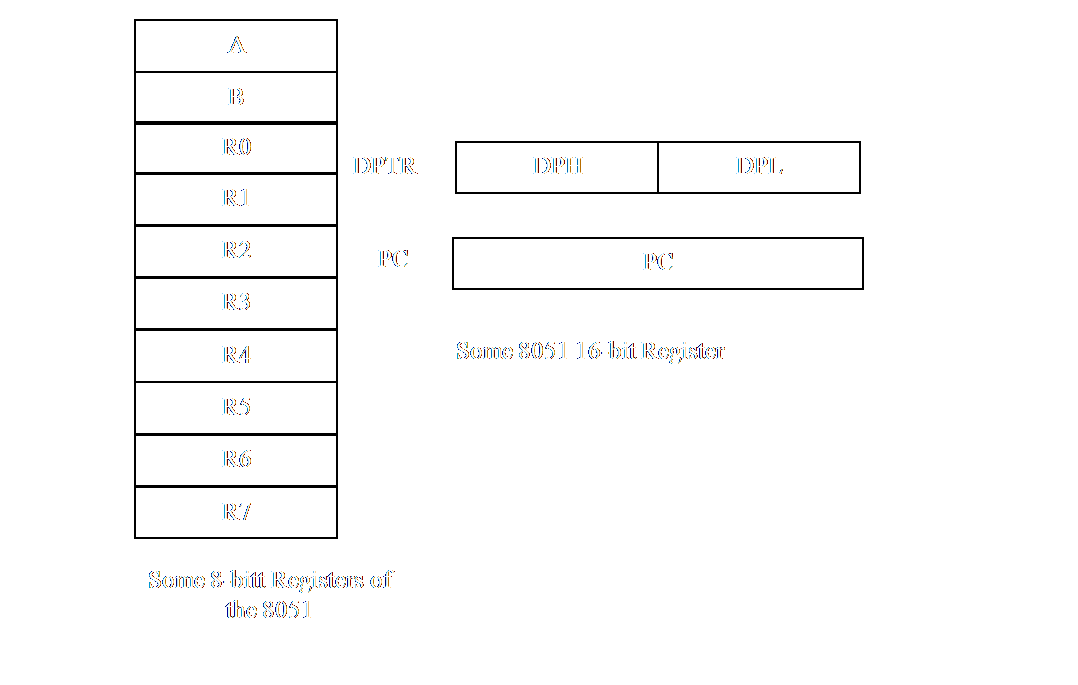
**B (B Register, Addresses F0h, Bit-Addressable):** The "B" register is used in two instructions: the multiply and divide operations. The B register is also commonly used by programmers as an auxiliary register to temporarily store values.

**Basic Registers**

**The Accumulator:**

The Accumulator, as its name suggests, is used as a general register to accumulate the results of a large number of instructions. It can hold an 8-bit (1-byte) value and is the most versatile register the 8051 has due to the shear number of instructions that make use of the accumulator. More than half of the 8051’s 255 instructions manipulate or use the accumulator in some way.

For example, if you want to add the number 10 and 20, the resulting 30 will be stored in the Accumulator. Once you have a value in the Accumulator you may continue processing the value or you may store it in another register or in memory.



**The "R" registers:**

The "R" registers are a set of eight registers that are named R0, R1, etc. Up to and including R7. These registers are used as auxiliary registers in many operations.

Perhaps you are adding 10 and 20. The original number 10 may be stored in the Accumulator whereas the value 20 may be stored in, say, register R4. To process the addition you would execute the command:

**ADD A, R4**

After executing this instruction the Accumulator will contain the value 30.

You may think of the "R" registers as very important auxiliary, or "helper", registers.The Accumulator alone would not be very useful if it were not for these "R" registers. The "R" registers are also used to temporarily store values.

**The "B" Register:**

The "B" register is very similar to the Accumulator in the sense that it may hold an 8-bit (1-byte) value.

The "B" register is only used by two 8051 instructions: MUL AB and DIV AB. Thus, if you want to quickly and easily multiply or divide A by another number, you may store the other number in "B" and make use of these two instructions.

Aside from the MUL and DIV instructions; The “B” register is often used as yet another temporary storage register much like a ninth "R" register.

**The Data Pointer (DPTR):**

The Data Pointer (DPTR) is the 8051’s only user-accessible 16-bit (2-byte) register. The Accumulator, "R" registers, and "B" register are all 1-byte values.

DPTR, as the name suggests, is used to point to data. It is used by a number of commands which allow the 8051 to access external memory. When the 8051 accesses external memory; it will access external memory at the address indicated by DPTR.

While DPTR is most often used to point to data in external memory, many programmers often take advantage of the fact that it’s the only true 16-bit register available. It is often used to store 2-byte values which have nothing to do with memory locations.

**The Program Counter (PC):**

The Program Counter (PC) is a 2-byte address which tells the 8051 where the next instruction to execute is found in memory. When the 8051 is initialized PC always starts at 0000h and is incremented each time an instruction is executed. It is important to note that PC isn’t always incremented by one. Since some instructions require 2 or 3 bytes the PC will be incremented by 2 or 3 in these cases.

The Program Counter is special in that there is no way to directly modify it’s value. That is to say, you can’t do something like PC=2430h. On the other hand, if you execute LJMP 2430h you’ve effectively accomplished the same thing.

It is also interesting to note that while you may change the value of PC (by executing a jump instruction, etc.) there is no way to read the value of PC.

**The Stack Pointer (SP)**

The Stack Pointer, like all registers except DPTR and PC, may hold an 8-bit (1-byte) value. The Stack Pointer is used to indicate where the next value to be removed from the stack should be taken from.

When you push a value onto the stack, the 8051 first increments the value of SP and then stores the value at the resulting memory location.

When you pop a value off the stack, the 8051 returns the value from the memory location indicated by SP and then decrements the value of SP.

This order of operation is important. When the 8051 is initialized SP will be initialized to 07h. If you immediately push a value onto the stack, the value will be stored in Internal

RAM address 08H. This makes sense taking into account what was mentioned two paragraphs above: First the 8051 will increment the value of SP (from 07h to 08h) and then will store the pushed value at that memory address (08h).

SP is modified directly by the 8051 by six instructions: PUSH, POP, ACALL, LCALL, RET, and RETI. It is also used intrinsically whenever an interrupt is triggered.

**2.05 8051 Architecture Diagram:**

Internal Interrupts

Interrupt Control

Timer (0,1)

128Byte RAM

4K ROM

External Interrupts

Data bus

CPU

Oscillator(12 MHz)

UART

I/O port control

Bus Control

TxD RxD

P0 P2 P1 P3

* 1. **Features Of 8051:**
* It has internally 4K ROM.
* It has 128 Bytes on chip RAM.
* Two 16bit TIMER or COUNTER to count the internal or external pulses.
* One FULL DUPLEX UART (Universal Asynchronous Receiver Transmit) for SERIAL communication where TxD stands for TRANSMIT DATA and RxD received data.
* Four ON-CHIP I/O port (P0,P1,P2,P3),each are 8bitwise.So microcontroller have 32bit I/O lines, where P0,P2 are used for address and data bus as well as I/O port.P1 port is used only as I/O port and P3 is used for various job purpose.
* INTERRUPTs**:** interrupts are six-two external, three internal and RESET is another interrupt.
* On chip bus controller which controls all the buses.
* On chip oscillator circuit.
* 8bit CPU optimization for process control. It may be used as an ALU or data processing unit.
* On chip interrupt controller which have three internal interrupts and two external interrupt. From these five only one interrupts goes to CPU.
* The operating frequency 12MHZ.
* The maximum memory capacity of a 8051 microcontroller is 64KByte programming memory and 64Kbyte data memory.

**2.07 8051 Buses & CPU Functioning In Data Transfer:**

**Microcontroller buses:**

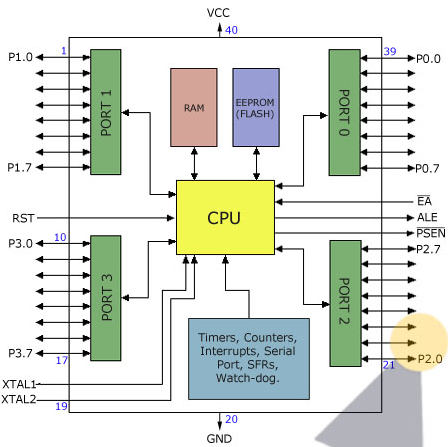
1. DATA BUS-Instructions and data are transferred through this bus.
2. ADDRESS BUS-Used to point the memory or I/O locations that is to be read from or written to.
3. CONTROL BUS-controls the system that determine what kind of information is on the data bus and determines where the data will go,along with the address bus.

**CPU controlled data transfers:**

* CPU reads data/instructions from memory(memory read)
* CPU writes data to memory(memory write)
* CPU reads data from input device(I/O read)
* CPU writes data to an output device(I/O write)

**2.08 MICROCONTROLLER 89S52 DETAILS:**

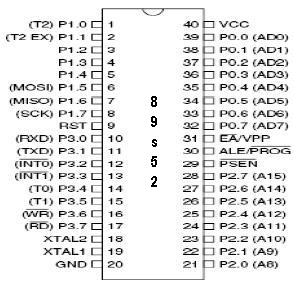
**2.08.1 BLOCK DIAGRAM:**

****

**2.08.2 IMPORTANT FEATURES:**

* It provides many functions ([CPU](http://en.wikipedia.org/wiki/Central_processing_unit), [RAM](http://en.wikipedia.org/wiki/Random_access_memory), [ROM](http://en.wikipedia.org/wiki/Read-only_memory), [I/O](http://en.wikipedia.org/wiki/Input/output), [interrupt](http://en.wikipedia.org/wiki/Interrupt) logic, [timer](http://en.wikipedia.org/wiki/Timer), etc.) in a single [package](http://en.wikipedia.org/wiki/Integrated_circuit_packaging)
* 8-bit ALU, Accumulator and Registers; hence it is an [8-bit](http://en.wikipedia.org/wiki/8-bit) [microcontroller](http://en.wikipedia.org/wiki/Microcontroller)
* 8-bit [data bus](http://en.wikipedia.org/wiki/Data_bus) - It can access 8 bits of data in one operation
* 16-bit [address bus](http://en.wikipedia.org/wiki/Address_bus) - It can access 216 memory locations - 64 [kiB](http://en.wikipedia.org/wiki/Kibibyte) (65536 locations) each of RAM and ROM
* On-chip RAM - 128 [bytes](http://en.wikipedia.org/wiki/Bytes) (data memory)
* On-chip ROM - 4 KB (program memory)
* Four [byte](http://en.wikipedia.org/wiki/Byte) bi-directional [input/output](http://en.wikipedia.org/wiki/Input/output) port
* UART ([serial port](http://en.wikipedia.org/wiki/Serial_port))
* Two 16-bit Counter/timers
* Two-level [interrupt](http://en.wikipedia.org/wiki/Interrupt) priority
* [Power saving](http://en.wikipedia.org/wiki/Power_management) mode

**2.08.3 PIN CONFIGURATION:**

****

**2.08.4 PIN DESCRIPTION:**

* P1.0-P1.7(1 to 8**):** stands for PORT 1, used in simple I/O operation**.**
* P0.0-P0.7(39 to 32):stands for PORT 0, used in simple I/O operation as well as ADDRESS and DATA bus multiplexing.
* P2.0-P2.7(21 to 28):stands for PORT 2,used inI/O operation as well as higher order ADDRESS bus(A8-A15)
* P3.0-P3.7(10-17):stands for PORT 3used in I/O operationand other functions as follows:
* RxD(10): used for serial data reception. The data is serially received through this pin into SBUF register.
* TxD(11): used for serial data transmission. The data is serially transmitted through this pin from SBUF register.
* INT0 bar(12): External interrupt zero, active low signal. Interrupt zero comes through this pin into the microcontroller.
* INT1 bar(13): External interrupt one, active low signal. Interrupt one comes through this pin into the microcontroller.
* T0(14): Timer zero, external pulse comes through this pin.
* T1(15): Timer one, external pulse comes through this pin.
* WR’(16): when this signal goes low, MEMORY WRITE or I/O WRITE operation takes place.
* RD’(17): when this signal goes low, MEMORY READ or I/O READ operation takes place.
* Vcc(40): this pin is connected with +5V DC
* /EA/Vpp(31): external access: GND if programmed, is accessed externally.
* ALE//PROG(30): Address Latch Enable (ALE) is used to demultiplex address and data bus. At the time of EPROM programming, 50ms pulse is connected to this pin.
* /PSEN(29): Program Store Enable: distinguishes between programming memory and data memory. When signal goes LOW, programming ROM is activated.

**2.09 8051 Interrupt:**

There are 5 interrupt sources for the 8051. Since the main RESET input can be also considered as an interrupt, there can be 6 interrupts listed.

1. RESET: When the reset pin is activated, the 8051 jumps to the address location 0000H

2. Two interrupts are set aside for the timers. One for timer 0 and another for timer

1. Memory locations 000BH and 001BH in the interrupt vector table belong to timer 0 and

timer 1 respectively

3. Two interrupts are set aside for hardwire interrupts. Pin no 12(p3.2) and pin no. 13

(p3.3) in the port 3 are for the external hardwire interrupts INT0 and INT1 respectively.

This external interrupts are also referred as an EX1 and EX2.

4. Serial communications has a single interrupt that belongs to both receive and

transmit.

INTERRUPT

PIN

LOCATION

|  |  |  |
| --- | --- | --- |
| System RESET | RST | 0000H |
| External interrupt 0 | IE0 | 0003H |
| Timer/counter 0 | TF0 | 000BH |
| External interrupt 1 | IE1 | 0013H |
| Timer/counter 1 | TF1 | 001BH |
| Serial interrupt  Table.6: Interrupt Details | RI or TI | 0023H |

* 1. 2222

**2.10 ADDRESSING MODES:**

Each instruction requires certain data on which it has to operate. There are various techniques to specify data for instruction. These techniques are called addressing modes.8051 microcontroller uses the following addressing modes-

|  |  |  |
| --- | --- | --- |
| **ADDRESSING MODES** | **DESCRIPTION** | **EXAMPLE** |
| *Immediate addressing* | Operand (which immediately follows the instruction op. code) is the data value to be used. | MOV A,#25h  (25h are immediately stored into the accumulator) |
| *Register addressing* | One of the 8 general purpose registers,R0-R7,can be specified as the instruction operand. Register is generally referred as Rn. | MOV A,R0  (Copy the content of R0 into accumulator. If any data is previously stored in the accumulator, that will be deleted completely) |
| *Direct addressing* | Data value is directly obtained from the memory location specified in the operand. | MOV A,22h  (22h acts as an address and content of that particular address are copied into accumulator) |
| *Indirect addressing* | The address of the operand is specified by a register.  It supports only R0 & R1. | MOV A,@R0  (The content of R0 acts as an address and the content of that particular address will be copied into the accumulator.) |
| *Absolute addressing* | Used only with AJMP (Absolute Jump) and ACALL(Absolute Call) instructions. | ACALL SADD (a11)  (jump to short range address.the destination address will be within 2K.) |
| *Long addressing* | Used with instructions LJMP and CALL. It is full 16 bit address is specified in operand so that the jump can be made to a location within 64Kbyte code memory space | LJMP 5000H  (After executing, it jumps to memory location 5000H.) |

|  |  |  |
| --- | --- | --- |
| **ADDRESSING MODES** | **DESCRIPTION** | **EXAMPLE** |
| *Indexed addressing* | A separate register, either PC or DPTR, is used as base address and accumulator is used as an offset address.  Effective address= base address value + offset address value | MOVC A,@A+DPTR  (MOVC moves data from the external code memory space. Address operand= base address(DPTR) + index address(accumulator) |
| *Relative addressing* | Relative address is a 8-bit signed number (-128 to 127), which is added to PC to determine the address of the next instruction. Before addition, PC is incremented. Therefore the new address is relative to the next instruction and then jump to the new address instruction. | SJMP LABEL\_X  (after executing,  new address= PC+ relative offset needed to reach LABEL\_X) |

* 1. **Types Of Instructions:**

1. ARITHMATIC INSTRUCTIONS
2. LOGICAL INSTRUCTIONS
3. DATA TRANSFER INSTRUCTIONS
4. BIT-ORIENTED INSTRUCTIONS
5. BRANCH INSTRUCTIONS

**Arithmetic Operations**

|  |  |  |  |
| --- | --- | --- | --- |
| **Mnemonic** | **Description** | **Bytes** | **Cycles** |
| ADD A,Rn | Add register to A | 1 | 1 |
| ADD A,direct | Add direct byte to A | 2 | 1 |
| ADD A,@Ri | Add indirect RAM to A | 1 | 1 |
| ADD A,#data | Add immediate data to A | 2 | 1 |
| ADDC A,Rn | Add register to A with Carry | 1 | 1 |
| ADDC A,direct | Add direct byte to A with Carry | 2 | 1 |
| ADDC A,@Ri | Add indirect RAM to A with Carry | 1 | 1 |
| ADDC A,#data | Add immediate data to A with Carry | 2 | 1 |
| SUBB A,Rn | Subtract register from A with Borrow | 1 | 1 |
| SUBB A,direct | Subtract direct byte from A with Borrow | 2 | 1 |
| SUBB A,@Ri | Subtract indirect RAM from A with Borrow | 1 | 1 |
| SUBB A,#data | Subtract immediate data from A with Borrow | 2 | 1 |
| INC A | Increment A | 1 | 1 |
| INC Rn | Increment register | 1 | 1 |
| INC direct | Increment direct byte | 2 | 1 |
| INC @Ri | Increment indirect RAM | 1 | 1 |
| DEC A | Decrement A | 1 | 1 |
| DEC Rn | Decrement register | 1 | 1 |
| DEC direct | Decrement direct byte | 2 | 1 |
| DEC @Ri | Decrement indirect RAM | 1 | 1 |
| INC DPTR | Increment Data Pointer | 1 | 2 |
| MUL AB | Multiply A and B (A x B => BA) | 1 | 4 |
| DIV AB | Divide A by B (A/B => A + B) | 1 | 4 |
| DAA | Decimal Adjust A | 1 | 1 |

**Logical Operations**

|  |  |  |  |
| --- | --- | --- | --- |
| **Mnemonic** | **Description** | **Bytes** | **Cycles** |
| ANL A,Rn | AND register to A | 1 | 1 |
| ANL A,direct | AND direct byte to A | 2 | 1 |
| ANL A,@Ri | AND indirect RAM to A | 1 | 1 |
| ANL A,#data | AND immediate data to A | 2 | 1 |
| ANL direct,A | AND A to direct byte | 2 | 1 |
| ANL direct,#data | AND immediate data to direct byte | 3 | 2 |
| ORL A,Rn | OR register to A | 1 | 1 |
| ORL A,direct | OR direct byte to A | 2 | 1 |
| ORL A,@Ri | OR indirect RAM to A | 1 | 1 |
| ORL A,#data | OR immediate data to A | 2 | 1 |
| ORL direct,A | OR A to direct byte | 2 | 1 |
| ORL direct,#data | OR immediate data to direct byte | 3 | 2 |
| XRL A,Rn | Exclusive-OR register to A | 1 | 1 |
| XRL A,direct | Exclusive-OR direct byte to A | 2 | 1 |
| XRL A,@Ri | Exclusive-OR indirect RAM to A | 1 | 1 |
| XRL A,#data | Exclusive-OR immediate data to A | 2 | 1 |
| XRL direct A | Exclusive-OR A to direct byte | 2 | 1 |
| XRL direct,#data | Exclusive-OR immediate data to direct byte | 3 | 2 |
| CLR A | Clear A | 1 | 1 |
| CPL A | Complement A | 1 | 1 |
| RL A | Rotate A Left | 1 | 1 |
| RLC A | Rotate A Left through Carry | 1 | 1 |
| RR A | Rotate A Right | 1 | 1 |
| RRC A | Rotate A Right through Carry | 1 | 1 |
| SWAP A | Swap nibbles within A | 1 | 1 |

**Data Transfer Operations**

|  |  |  |  |
| --- | --- | --- | --- |
| **Mnemonic** | **Description** | **Bytes** | **Cycles** |
| MOV A,Rn | Move register to A | 1 | 1 |
| MOV A,direct | Move direct byte to A | 2 | 1 |
| MOV A,@Ri | Move indirect RAM to A | 1 | 1 |
| MOV A,#data | Move immediate data to A | 2 | 1 |
| MOV Rn,A | Move A to register | 1 | 1 |
| MOV Rn,direct | Move direct byte to register | 2 | 2 |
| MOV Rn,#data | Move immediate data to register | 2 | 1 |
| MOV direct,A | Move A to direct byte | 2 | 1 |
| MOV direct,Rn | Move register to direct byte | 2 | 2 |
| MOV direct,direct | Move direct byte to direct byte | 3 | 2 |
| MOV direct,@Ri | Move indirect RAM to direct byte | 2 | 2 |
| MOV direct,#data | Move immediate data to direct byte | 3 | 2 |
| MOV @Ri,A | Move A to indirect RAM | 1 | 1 |
| MOV @Ri,direct | Move direct byte to indirect RAM | 2 | 2 |
| MOV @Ri,#data | Move immediate data to indirect RAM | 2 | 1 |
| MOV DPTR,#data16 | Load Data Pointer with 16-bit constant | 3 | 1 |
| MOVC A,@A+DPTR | Move Code byte relative to DPTR to A | 1 | 2 |
| MOVC A,@A+PC | Move Code byte relative to PC to A | 1 | 2 |
| MOVX A,@Ri | Move External RAM (8-bit addr) to A | 1 | 2 |
| MOVX A,@DPTR | Move External RAM (16-bit addr) to A | 1 | 2 |
| MOVX @Ri,A | Move A to External RAM (8-bit addr) | 1 | 2 |
| MOVX @DPTR | A Move A to External RAM (16-bit addr) | 1 | 2 |
| PUSH direct | Push direct byte onto stack | 2 | 2 |
| POP direct | Pop direct byte from stack | 2 | 2 |
| XCH A,Rn | Exchange register with A | 1 | 1 |
| XCH A,direct | Exchange direct byte with A | 2 | 1 |
| XCH A,@Ri | Exchange indirect RAM with A | 1 | 1 |
| XCHD A,@Ri | Exchange low-order Digit indirect RAM with A | 1 | 1 |

**Single Bit (Boolean Variable) Operations**

|  |  |  |  |
| --- | --- | --- | --- |
| **Mnemonic** | **Description** | **Bytes** | **Cycles** |
| CLR C | Clear Carry flag | 1 | 1 |
| CLR bit | Clear direct bit | 2 | 1 |
| SETB C | Set Carry flag | 1 | 1 |
| SETB bit | Set direct bit | 2 | 1 |
| CPL C | Complement Carry flag | 1 | 1 |
| CPL bit | Complement direct bit | 2 | 1 |
| ANL C,bit | AND direct bit to Carry flag | 2 | 2 |
| ANL C,/bit | AND complement of direct bit to Carry flag | 2 | 2 |
| ORL C,bit | OR direct bit to Carry flag | 2 | 2 |
| ORL C,/bit | OR complement of direct bit to Carry flag | 2 | 2 |
| MOV C,bit | Move direct bit to Carry flag | 2 | 1 |
| MOV bit,C | Move Carry flag to direct bit | 2 | 2 |

**Program Flow Control**

|  |  |  |  |
| --- | --- | --- | --- |
| **Mnemonic** | **Description** | **Bytes** | **Cycles** |
| ACALL addr11 | Absolute subroutine call | 2 | 2 |
| LCALL addr16 | Long subroutine call | 3 | 2 |
| RET | Return from subroutine | 1 | 2 |
| RETI | Return from interrupt | 1 | 2 |
| AJMP addr11 | Absolute Jump | 2 | 2 |
| LJMP addr16 | Long Jump | 3 | 2 |
| SJMP rel | Short Jump at relative address | 2 | 2 |
| JMP @A+DPTR | Jump indirect relative to DPTR | 1 | 2 |
| JZ rel | Jump if A is Zero | 2 | 2 |
| JNZ rel | Jump if A is Not Zero | 2 | 2 |
| JC rel | Jump if Carry flag is set | 2 | 2 |
| JNC rel | Jump if No Carry flag | 2 | 2 |
| JB bit,rel | Jump if direct Bit is set | 3 | 2 |
| JNB bit,rel | Jump if direct Bit is Not set | 3 | 2 |
| JBC bit,rel | Jump if direct Bit is set and Clear bit | 3 | 2 |
| CJNE A,direct,rel | Compare direct to A and Jump if Not Equal | 3 | 2 |
| CJNE A,#data,rel | Compare immediate to A and Jump if Not Equal | 3 | 2 |
| CJNE Rn,#data,rel | Compare immediate to register and Jump if Not Equal | 3 | 2 |
| CJNE @Ri,#data,rel | Compare immediate to indirect and Jump if Not Equal | 3 | 2 |
| DJNZ Rn,rel | Decrement register and Jump if Not Zero | 2 | 2 |
| DJNZ direct,rel | Decrement direct byte and Jump if Not Zero | 3 | 2 |
| NOP | No operation | 1 | 1 |

**Notes:**

* Rn is Working register R0-R7.
* Direct 128 internal RAM locations, any I/O port, control or status register
* @Ri Indirect internal RAM location addressed by register R0 or R1
* data 8-bit constant included in instruction
* data16 16-bit constant included in instruction
* Bit 128 software flags, any I/O pin, control or status bit.
* addr16 Destination address may be anywhere in 64-kByte program address space
* addr11 Destination address will be within same 2-kByte page of program address space as first byte of the following instruction

rel 8-bit offset relative to first byte of following instruction (+127, -128)

**2.12 Application Of Microcontroller:**

Here we apply the concept of microcontroller to make an LED blink with the help of a remote control system wherein the microcontroller is interfaced with the LED. For this dedicated work we will burn an assembly language programming in 8051 with the help of burner.

**2.12.1 LED on/off with the help of remote control**

It is the simplest application of Microcontroller

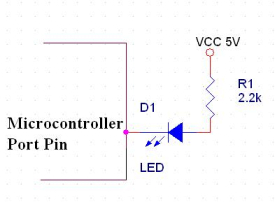
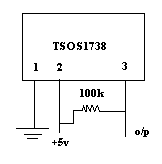
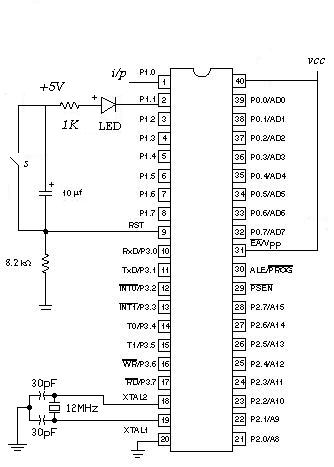
Fig:- 1

Figure 1 shows basic principal of interfacing LED to microcontroller. As we can see the Anode is connected through a resistor to Vcc & the Cathode is the Microcontroller pin. So when the Port Pin is **HIGH the LED is OFF** & when the Port Pin is **LOW the LED is turned ON.**

**2.12.1.1 LED interfacing (basic circuit diagram)**

****  **Figure 2**

The first step is to build the circuit. At this point as we are familiar with the parts used. (2 resistors, 3 capacitors, 1 LED) we can either put these parts together using a veroboard or wire wrap. This design is intended for use with an Atmel 89C51. Most microcontrollers (such as a normal 8051 or 8751) cannot handle the current required to turn an LED on and off but the ATMEL part has this capability.

Vcc = 5V and Gnd = 0V

We need only three main components

* IR sensor TSOP 1738
* 89C51 micro-controller
* IC 78L05.

**2.12.2 Interfacing LCD display with Microcontroller:**

Liquid Crystal Display (LCD) consists of rod-shaped tiny molecules sandwiched between a flat piece of glass and an opaque substrate. These rod-shaped molecules in between the plates align into two different physical positions based on the electric charge applied to them. When electric charge is applied they align to block the light entering through them, where as when no-charge is applied they become transparent.

Light passing through makes the desired images appear. This is the basic concept behind LCDdisplays.     
  
LCDs are most commonly used because of their advantages over other display technologies. They are thin and flat and consume very small amount of power compared to LED displays and cathode ray tubes (CRTs)

**2.12.2.1 Special characteristics of LCDs:**

Liquid Crystals are very sensitive to constant electric fields. Only AC-voltages should be applied as DC voltages can cause an electrochemical reaction, which destroys the liquid crystals irreversibly.

Temperature dependent and in a very cold or hot environment LCD may not work correctly. This is a reversible effect. Some displays need temperature compensation circuits to automatically adjust the applied LC voltage.

**2.12.2.2 Advantages of LCDs:**

* Consumes less power and generates less heat
* Saves lot of space compared picture tubes due to LCD's flatness
* Due to less weight and flatness LCDs are highly portable.
* No flicker and lesser screen glare in LCDs to reduce eyestrain.

**2.12.2.3 Drawbacks:**

The disadvantages of LCD displays are,

* LCDs cannot form multiple resolution images.
* The contrast ratio for LCD images is lesser than CRT and plasma displays.
* Due to their longer response time, LCDs show ghost images and mixing when images change   rapidly

**2.12.2.4 Interfacing LCD With 8051:**

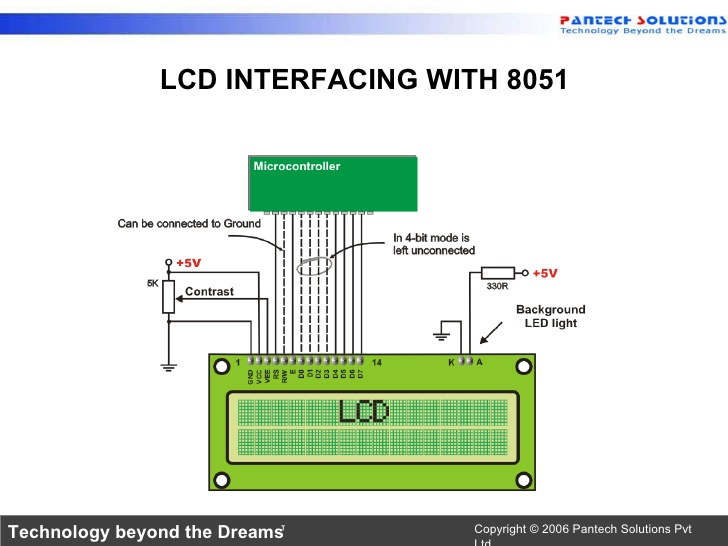


Fig.42: Different parts of an LCD display:

RS: register select-if RS instruction is 0, instruction command code register is selected allowing the user to send a command such as clear display, cursor at home etc. If RS is 1, the data register is selected, allowing the user to send data to be displayed on the LCD.

R/W: read/write: if R/W is 1 then reading, If R/W is 0 then writing

E-enable: when data is supplied to data pins, a high-to-low pulse must be applied to this pin in order for the two LCD latch in the data present at the data pins. This pulse must be a minimum of 450ns wide.

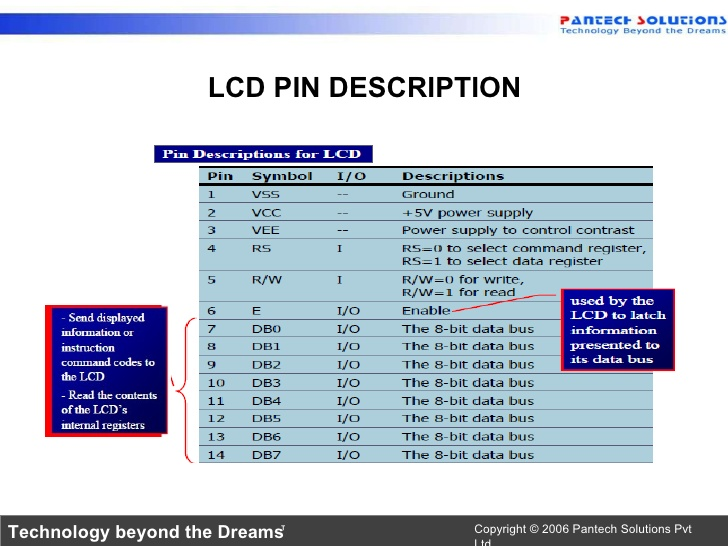


Fig.43: LCD Pin Description

To display the letters and numbers ,we send ASCII code for the letters A-Z and a-z and numbers 0-9 to these pins while making RS=1.we also use RS=0 check the busy flag to see if the LCD is ready to receive information.it is recommended to writing any data to the LCD.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **RS** | **R/W** | **D7** | **D6** | **D5** | **D4** | **D3** | **D2** | **D1** | **D0** | **FUNCTION** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Clear LCD and memory home counter |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | Clear home cursor only |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | I/O | S | S=1/0,shift screen/cursor  I/O=1/0,cursor R/L,screen L/R |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | D | C | B | D=1/0,screen ON/OFF  C=1/0,cursor ON/OFF  B=1/0,blink/not blink |
| 0 | 0 | 0 | 0 | 0 | 1 | S/C | R/L | 0 | 0 | S/C=1/0,screen/cursor  R/L=1/0,shift one space R/L |
| 0 | 0 | 0 | 0 | 1 | DL | N | F | 0 | 0 | DL=1/0,8/4 bit per character  N=1/0,2/1 rows of character  F=1/0,5\*10/5\*7 dots per character |
| 0 | 1 | BF | CURRENT  ADDRESS |  |  |  |  |  |  | BF=1/0,busy/not busy |

Table.13:LCD REGISTER description and function

1. **PROJECT DETAILS:**
   1. **Assigned Project:**

“MICROCONTROLLER BASED DIGITAL IC TESTER”

* 1. **Objective:**

To Display IC Tested ‘OK’ Or ‘NOT OK’

* 1. **Theory:**

**3.03.1 What is digital IC tester & its functioning:**

An Integrated Circuit tester (IC tester) is used to test Integrated Circuits (ICs). We can easily test any digital IC using this kind of an IC tester. For testing an IC, we need to use different hardware circuits for different ICs; like we need a particular kind of tester for testing a logic gate and another for testing flip flops or shift registers which involves more complication and time involved will also be more. So here’s an IC tester to overcome this problem. Unlike other IC testers, this is more reliable and easier since we don’t need to rig up different kind of circuits for different kind of ICs, each time we need to test them.

Unlike the IC testers available in the market today which are usually expensive, this IC tester is affordable and user-friendly. This IC tester is constructed using 8951 microcontroller along with a keyboard and a display unit. It can test digital ICs having a maximum of 24 pins. Since it is programmable, any number of ICs can be tested within the constraint of the memory available. This IC tester can be used to test a wide variety of ICs which includes simple logic gates and also sequential and combinational ICs like flip-flops, counters, shift registers etc. It is portable and easy to use.

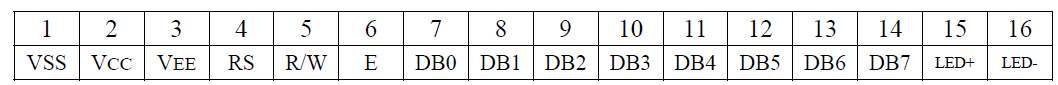
**3.03.2 Features of IC Tester:**

* Display - 16 x 2Backlit alphanumeric LCD
* Keyboard - 24 keys keyboard
* Supply Input Voltage:9V DC Adaptor
* Can test more than 300+ ICs
* Tests wide range of Digital IC's such as 74 Series, 40/45 Series
* 20 pin DIP ZIF sockets.
* Auto Search facility for listed IC's.
* Tests 7- Segment display of common cathode & common anode type
* Test by: Truth table/sequence table comparison
* Package Digital IC - Digital ICs 14, 16, 20 Pin DIP / Linear ICs 6, 8, 14, 16 & 20 Pin DIP

**3.03.3 INTRODUCTION TO LCD:**

The LCD in the board is 16X2 characters display.

**PIN DIAGRAM OF 16X2 LCD:**



DB0-DB7: Data pins

RS: Register select. If RS =0, the instruction command code register is selected, allowing the user to send a command such as clear display, cursor at home etc. If RS=1, the data register is selected allowing the user to send data to be displayed on the LCD.

R/W: Read or Write. R/W=1 when reading;

R/W=0 when writing.

E: Enable. The Enable pin is used by LCD to latch information presented to its data pins. When data is supplied to data pins, a high to low pulse must be applied to this pin in order for the LCD to latch in the data present at the data pins.

LCD COMMAND CODES:

08 h-Displays off, cursor off

01h-Clear Display screen

06-Increment cursor (shift cursor to right)

0Ch- Display on, cursor off.

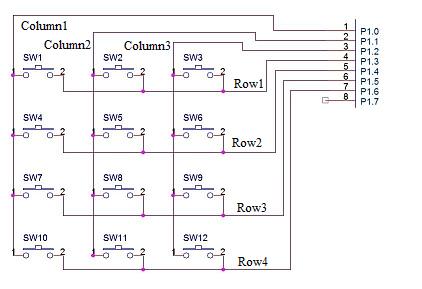
80H – Force cursor to beginning of first line

0C0h - Force cursor to beginning of first line

**3.03.4 Introduction To 4X3 KEYPAD:**

Row pins are connected to lower 4 pins of port 1. And column pins are connected to lower 3 pins of port 3.

The keyboard section is as shown in the figure below. The keyboard is designed as a matrix and is interfaced to port 1 of the master IC. Here P1.0, P1.1, P1.2, P1.3 are configured as input ports, and P3.0, P3.1, P3.2 are configured as output ports. The keyboard consists of 10 digits as well as an ‘ENTER’ button and ‘RESET’ button.

In case of matrix Keypad both the ends of switches are connected to the port Pin. Over here we have considered a 4x3 matrix keypad i.e. four rows and three columns. So in all twelve switches have been interfaced using just seven lines. The adjoining figure shows the diagram of a matrix keypad and how it is interfaced with the controller.

As you can see no pin is connected to ground, over here the controller pin itself provides the ground. We pull one of the Column Pins low & check the row pins if any of the Pin is low then we come to know which switch is pressed.

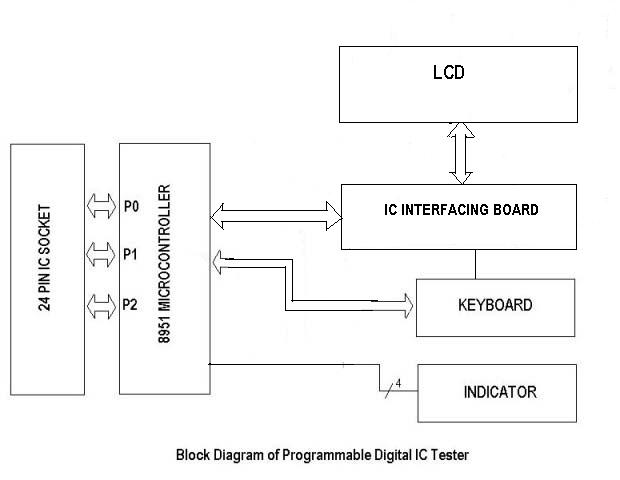
Suppose we make column 1 pin low and while checking the rows we get Row 3 is low then we come to know switch 7 has been pressed.

* 1. **Components TABLE:**

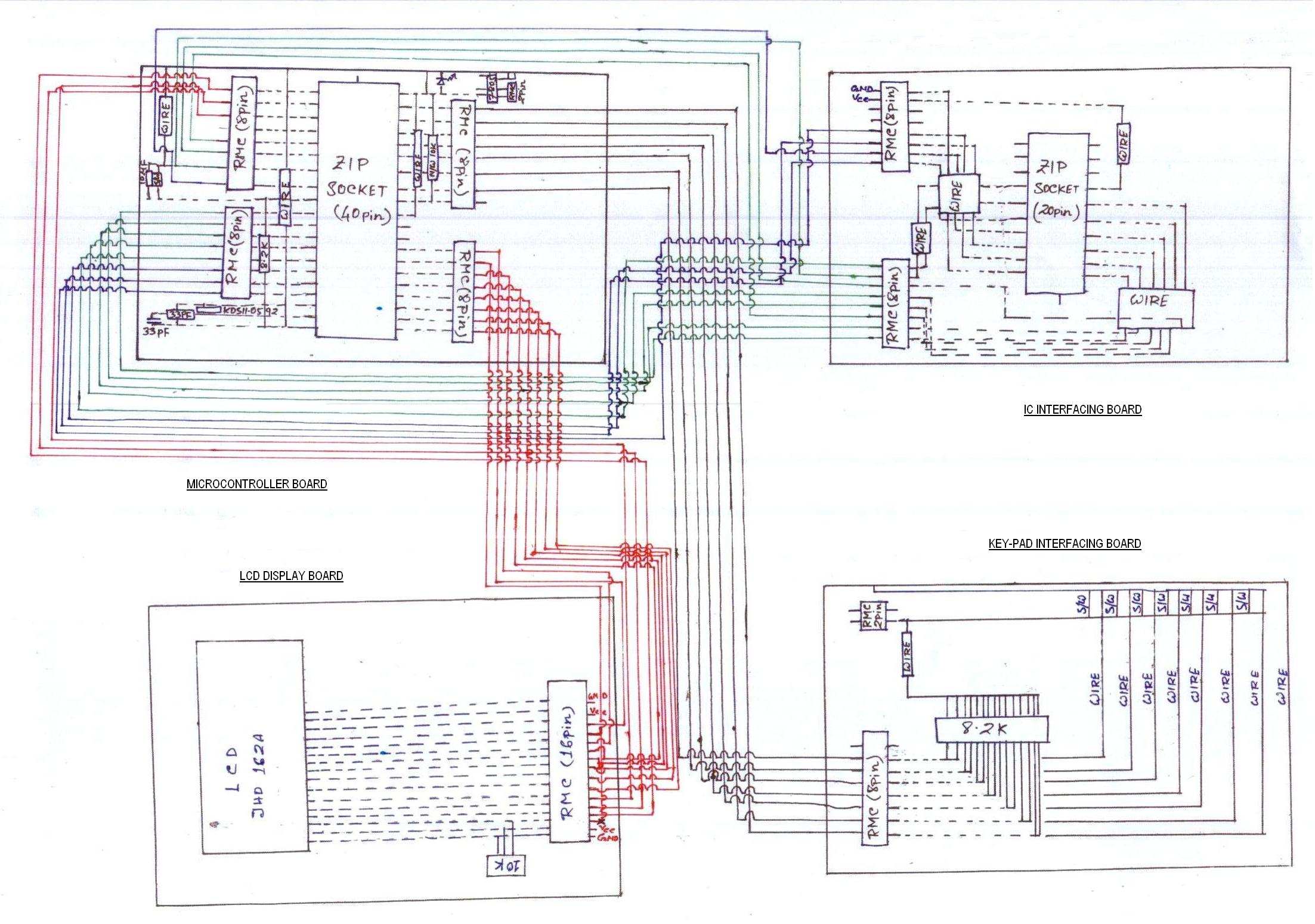
|  |  |  |
| --- | --- | --- |
| **Name** | **Specification** | **Quantity** |
| IC Vero Board | KS 110 | 4 |
| ZIP Socket | 40 pin,20pin | 1,1 |
| Microcontroller IC | AT89S52 | 1 |
| Power Supply IC(5V) | 7805 | 1 |
| Crystal Oscillator | 11.0592 MHz | 1 |
| Capacitor | 10μF(Polarized),33 pF-paper | 1,2 |
| Resistor | 8.2K,1K | 15,1 |
| POT | 10 K | 1 |
| LED | Red(colour) | 2 |
| RMC Connector | 8pin,2pin,16pin | 7,3,1 |
| Tact S/W | 2pin,4pin | 5,3 |
| N/W resistance | 10 K | 1 |
| Battery | 9V | 1 |
| Battery Connector | - | 1 |
| LCD display | JHD 162A | 1 |
| IC | 7408,7432,7400,7486,74266,7404,7402 | 1,1,1,1,1,1,1 |

Table.18: Components of the project work

* 1. **Block Diagram:**

****

* 1. **Circuit diagram:**

****

* 1. **Technical Specification:**

**FAMILY :** TTL, CMOS,

**RANGE :** Logic Gates, Shift Register, Adder and Multiplexer can be tested.

**TEST SOCKETS :** A single 24 pin ZIF sockets for IC Testing.

**PACKAGE :** DIP14, 16 & 20 pins.

**DISPLAY :** 16X2 LCD Display.

**INDICATOR LEDs :** 4 bright LEDs of 3 mm each.

**KEY PAD :** 4X3 MATRIX TYPE**.**

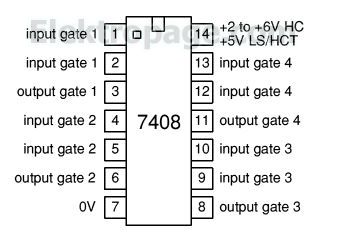
**ELECTRICAL :** 230 V (+/- 10 %), 1 phase, 50 Hz (+/- 2 %)

**BATTERY :** 9V

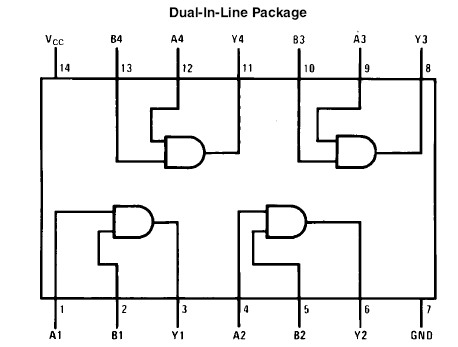
* 1. **IC Details:**
     1. **IC 7408:**

The AND gate is a digital logic gate that implements logical disjunction - it behaves according to the truth table. A HIGH output (1) results if both the inputs to the gate are HIGH (1). If either input is LOW, a LOW output (0) results.

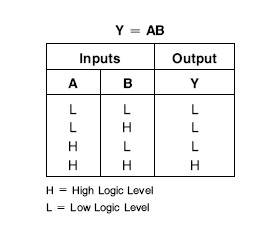
**Pin Diagram:**



**Circuit Diagram:**



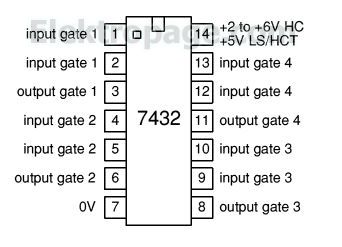
**Truth Table:**

****

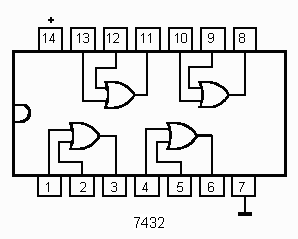
* + 1. **IC 7432**

The OR gate is a digital logic gate that implements logical disjunction - it behaves according to the truth table shown below. A HIGH output (1) results if one or both the inputs to the gate are HIGH (1). If neither input is HIGH, a LOW output (0) results.

**Pin Diagram:**



**Circuit Diagram:**



**Truth Table:**

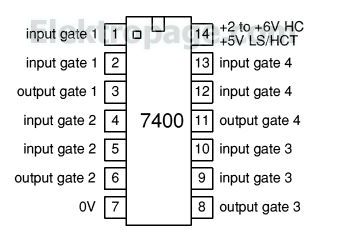
Y=A+B

|  |  |  |
| --- | --- | --- |
| Input | | Output |
| A | B | Y |
| 0  0  1  1 | 0  1  0  1 | 0  1  1  1 |

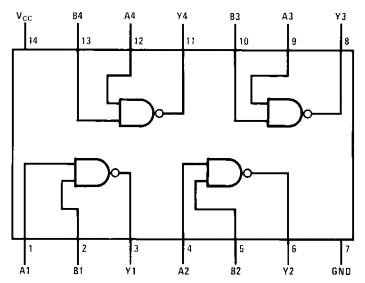
**3.08.3 IC 7400:**

The NAND gate is a digital logic gate that implements logical disjunction – it behaves according to the truth table. A HIGH O/P (1) results when both the I/P’s are LOW (0) & HIGH (1) LOW (0) I/P combination. And only if both the I/P’s are HIGH (1), a LOW O/P (0) results.

**Pin Diagram:**

****

**Circuit Diagram:**

****

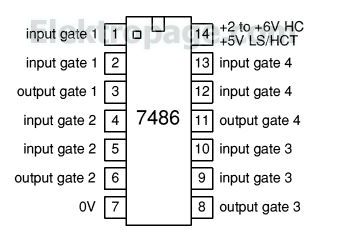
**Truth Table:**

|  |  |  |
| --- | --- | --- |
| Input | | Output |
| A | B | Y |
| 0  0  1  1 | 0  1  0  1 | 1  1  1  0 |

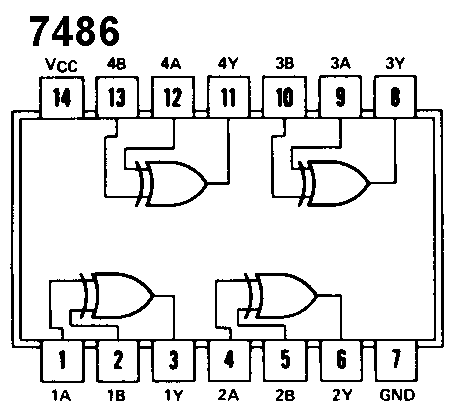
**3.08.4 IC 7486:**

The XOR gate is a digital logic gate that implements logical disjunction - it behaves according to the truth table. A HIGH (1) O/P results if the I/P combination are HIGH (1) & LOW (0). And when both the I/P are HIGH / LOW, the results are HIGH (1).

**Pin Diagram:**

****

**Circuit Diagram:**

****

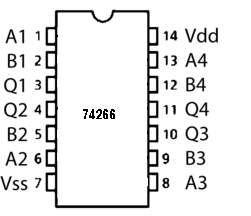
**Truth Table:**

|  |  |  |
| --- | --- | --- |
| Input | | Output |
| A | B | Y |
| 0  0  1  1 | 0  1  0  1 | 0  1  1  0 |

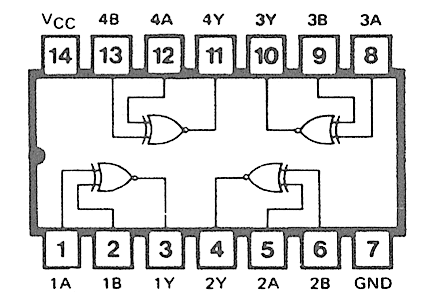
**3.08.5 IC 74266:**

The XNOR gate is a digital logic gate that implements logical disjunction - it behaves according to the truth table. A HIGH O/P (1) results if both the I/P’s HIGH & LOW. A LOW O/P (0) results if the I/P combination are HIGH & LOW.

**Pin Diagram:**

****A, B – Input , Q – Output

**Circuit Diagram:**

****

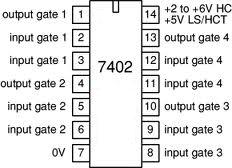
**Truth Table:**

|  |  |  |
| --- | --- | --- |
| Input | | Output |
| A | B | Y |
| 0  0  1  1 | 0  1  0  1 | 1  0  0  1 |

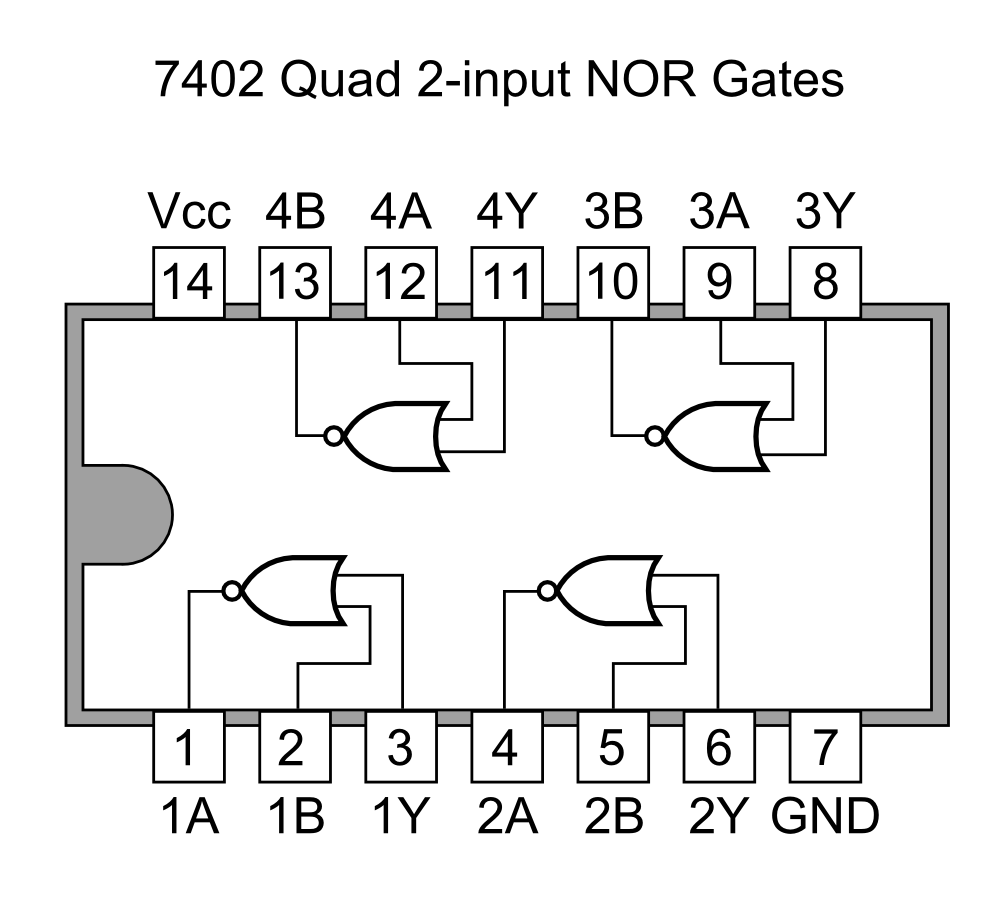
**3.08.6 IC 7402:**

The NOR gate is a digital logic gate that implements logical disjunction - it behaves according to the truth table. A HIGH O/P (1) results if both the I/P’s are LOW. When both the I/P’s are HIGH (1) & combination are I/P’s are HIGH & LOW, the results are only LOW (0).

**Pin diagram:**



**Circuit Diagram:**

****

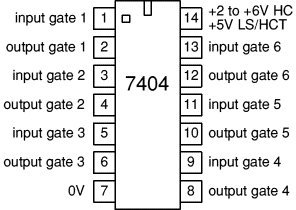
**Truth Table:**

|  |  |  |
| --- | --- | --- |
| Input | | Output |
| A | B | Y |
| 0  0  1  1 | 0  1  0  1 | 1  0  0  0 |

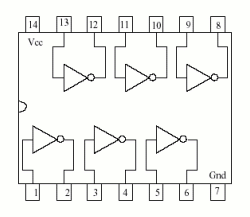
**3.08.7 IC 7404:**

The NOT gate is a digital logic gate that implements logical disjunction - it behaves according to the truth table. A HIGH O/P (1) results if I/P to gate is LOW (0). And LOW O/P (0) results if I/P to gate is HIGH (1).

**Pin Diagram:**

****

**Circuit Diagram:**

****

**Truth Table:**

|  |  |
| --- | --- |
| A | Y |
| 0  1 | 1  0 |

* 1. **Software Program:**

ORG 0000H

MOV P0,#0FFH

MOV DPTR,#CONF

ACALL CONF\_DISP

MOV DPTR,#MSG1

ACALL DISPLAY

MOV DPTR,#CONF2

ACALL CONF\_DISP

MOV DPTR,#MSG11

ACALL DISPLAY

MOV DPTR,#CONF

ACALL CONF\_DISP

MOV DPTR,#MSG2

ACALL DISPLAY

MOV DPTR,#CONF1

ACALL CONF\_DISP

MOV DPTR,#MSG21

ACALL DISPLAY

MOV DPTR,#CONF1

ACALL CONF\_DISP

MOV DPTR,#MSG22

ACALL DISPLAY

MOV DPTR,#CONF

ACALL CONF\_DISP

MOV DPTR,#MSG3

ACALL DISPLAY

MOV DPTR,#CONF1

ACALL CONF\_DISP

MOV DPTR,#MSG31

ACALL DISPLAY

BACK1: MOV DPTR,#CONF

ACALL CONF\_DISP

MOV DPTR,#MSG32

ACALL DISPLAY

BACK11: ACALL DELAY1

JNB PO.O,P\_AND

JNB PO.1,P\_OR

JNB PO.2,P\_NAND

JNB PO.3,P\_XOR1

JNB PO.4,P\_XNOR

JNB PO.5,P\_NOR

JNB PO.6,P\_NOT

AJMP BACK11

P\_AND: MOV DPTR,#CONF

ACALL CONF\_DISP

MOV DPTR,#MSG40

ACALL DISPLAY

MOV P3,#00H

SETB P1.4

SETB P1.5

SETB P1.6

SETB P1.7

MOV P3,#00H

ACALL DELAY1

MOV A,P1

ANL A,#0F0H

JNZ BACKA

MOV P3,#55H

ACALL DELAY1

MOV A,P1

ANL A,#0F0H

JNZ BACKA

MOV P3,#0AAH

ACALL DELAY1

MOV A,P1

ANL A,#0F0H

JNZ BACKA

MOV P3,#0FFH

ACALL DELAY1

MOV A,P1

ANL A,#0F0H

XRL A,#0F0H

JNZ BACKA

MOV DPTR,#CONF1

ACALL CONF\_DISP

MOV DPTR,#MSG41

ACALL DISPLAY

AJMP BACK1

BACKA: MOV DPTR,#CONF1

ACALL CONF\_DISP

MOV DPTR,#MSG42

ACALL DISPLAY

AJMP BACK1

P\_OR: MOV DPTR,#CONF

ACALL CONF\_DISP

MOV DPTR,#MSG43

ACALL DISPLAY

MOV P3,#00H

SETB P1.4

SETB P1.5

SETB P1.6

SETB P1.7

MOV P3,#00H

ACALL DELAY1

MOV A,P1

ANL A,#0F0H

JNZ BACKO

MOV P3,#55H

ACALL DELAY1

MOV A,P1

ANL A,#0F0H

XRL A,#0F0H

JNZ BACKO

MOV P3,#AAH

ACALL DELAY1

MOV A,P1

ANL A,#0F0H

XRL A,#0F0H

JNZ BACKO

MOV P3,#0FFH

ACALL DELAY1

MOV A,P1

ANL A,#0F0H

XRL A,#0F0H

JNZ BACKO

MOV DPTR,#CONF1

ACALL CONF\_DISP

MOV DPTR,#MSG41

ACALL DISPLAY

AJMP BACK1

BACKO: MOV DPTR,#CONF1

ACALL CONF\_DISP

MOV DPTR,#MSG42

ACALL DISPLAY

AJMP BACK1

P\_XOR: MOV DPTR,#CONF

ACALL CONF\_DISP

MOV DPTR,#MSG46

ACALL DISPLAY

MOV P3,#00H

SETB P1.4

SETB P1.5

SETB P1.6

SETB P1.7

MOV P3,#00H

ACALL DELAY1

ANL A,#0F0H

JNZ BACKXO

MOV P3,#55H

ACALL DELAY1

MOV A,P1

ANL A,#0F0H

XRL A,#0F0H

JNZ BACKXO

MOV P3,#AAH

ACALL DELAY1

MOV A,P1

ANL A,#0F0H

XRL A,#0F0H

JNZ BACKXO

MOV P3,#0FFH

ACALL DELAY1

ANL A,#0F0H

JNZ BACKXO

MOV DPTR,#CONF1

ACALL CONF\_DISP

MOV DPTR,#MSG41

ACALL DISPLAY

AJMP BACK1

BACKXO: MOV DPTR,#CONF1

ACALL CONF\_DISP

MOV DPTR,#MSG42

ACALL DISPLAY

AJMP BACK1

P\_NAND: MOV DPTR,#CONF

ACALL CONF\_DISP

MOV DPTR,#MSG45

ACALL DISPLAY

MOV P3,#00H

SETB P1.4

SETB P1.5

SETB P1.6

SETB P1.7

MOV P3,#00H

MOV P3,#00H

ACALL DELAY1

MOV A,P1

ANL A,#0F0H

XRL A,#0F0H

JNZ BACKNA

MOV P3,#55H

ACALL DELAY1

MOV A,P1

ANL A,#0F0H

XRL A,#0F0H

JNZ BACKNA

MOV P3,#AAH

ACALL DELAY1

MOV A,P1

ANL A,#0F0H

XRL A,#0F0H

JNZ BACKNA

MOV P3,#0FFH

ACALL DELAY1

MOV A,P1

ANL A,#0F0H

JNZ BACKNA

MOV DPTR,#CONF1

ACALL CONF\_DISP

MOV DPTR,#MSG41

ACALL DISPLAY

AJMP BACK1

BACKNA: MOV DPTR,#CONF1

ACALL CONF\_DISP

MOV DPTR,#MSG42

ACALL DISPLAY

AJMP BACK1

P\_NOT: MOV DPTR,#CONF

ACALL CONF\_DISP

MOV DPTR,#MSG48

ACALL DISPLAY

MOV P1,#60H

MOV P3,#66H

MOV P1,#00H

MOV P3,#00H

ACALL DELAY1

MOV A,P1

MOV R1,P3

ANL A,#60H

XRL A,#60H

JNZ BACKN

MOV A,R1

ANL A,#66H

XRL A,#66H

JNZ BACKN

MOV P3,#99H

MOV P1,#90H

ACALL DELAY1

MOV A,P1

MOV R1,P3

ANL A,#60H

JNZ BACKN

MOV A,R1

ANL A,#66H

JNZ BACKN

MOV DPTR,#CONF1

ACALL CONF\_DISP

MOV DPTR,#MSG41

ACALL DISPLAY

AJMP BACK1

BACKN: MOV DPTR,#CONF1

ACALL CONF\_DISP

MOV DPTR,#MSG42

ACALL DISPLAY

AJMP BACK1

P\_NOR: MOV DPTR,#CONF

ACALL CONF\_DISP

MOV DPTR,#MSG44

ACALL DISPLAY

CLR P3.1

CLR P1.4

CLR P3.3

CLR P1.5

CLR P3.4

CLR P1.6

CLR P1.7

CLR P3.6

SETB P3.0

SETB P3.2

SETB P3.5

SETB P3.7

MOV P3,#00H

MOV P1,#00H

ACALL DELAY

MOV A,P3

ANL A,#0A5H

XRL A,#0A5H

JNZ BACKNO

MOV P3,#50H

MOV P1,#30H

ACALL DELAY

MOV A,P3

ANL A,#0A5H

JNZ BACKNO

MOV P3,#0AH

MOV P1,#C0H

ACALL DELAY

MOV A,P3

ANL A,#0A5H

JNZ BACKNO

MOV P3,#5AH

MOV P1,#0F0H

ACALL DELAY

MOV A,P3

ANL A,#0A5H

JNZ BACKNO

MOV DPTR,#CONF1

ACALL CONF\_DISP

MOV DPTR,#MSG41

ACALL DISPLAY

AJMP BACK1

BACKNO: MOV DPTR,#CONF1

ACALL CONF\_DISP

MOV DPTR,#MSG42

ACALL DISPLAY

AJMP BACK1

P\_XNOR: MOV P1,#90H

MOV P3,#24H

MOV P1,#00H

MOV P3,#00H

ACALL DELAY1

MOV A,P1

MOV R1,P3

ANL A,#90H

XRL A,#90H

JNZ BACKXN

MOV A,R1

ANL A,#24H

XRL A,#24H

JNZ BACKXN

MOV P1,#20H

MOV P3,#92H

ACALL DELAY1

MOV A,P1

MOV R1,P3

ANL A,#90H

JNZ BACKXN

MOV A,R1

ANL A,#24H

JNZ BACKXN

MOV P1,#40H

MOV P3,#49H

ACALL DELAY1

MOV A,P1

MOV R1,P3

ANL A,#90H

JNZ BACKXN

MOV A,R1

ANL A,#24H

JNZ BACKXN

MOV P1,#60H

MOV P3,#0DBH

ACALL DELAY1

MOV A,P1

MOV R1,P3

ANL A,#90H

XRL A,#90H

JNZ BACKXN

MOV A,R1

ANL A,#24H

XRL A,#24H

JNZ BACKXN

MOV DPTR,#CONF1

ACALL CONF\_DISP

MOV DPTR,#MSG41

ACALL DISPLAY

AJMP BACK1

BACKXN: MOV DPTR,#CONF1

ACALL CONF\_DISP

MOV DPTR,#MSG42

ACALL DISPLAY

AJMP BACK1

CONF\_DISP:

CLR A

MOVC A,@A+DPTR

INC DPTR

CJNE A,#0FFH,COMD

RET

COMD:

MOV R3,#01H

ACALL DELAY

SJMP CONF\_DISP

COMNWRT:

MOV P2,A

CLR P1.5

CLR P1.8

SETB P1.4

NOP

NOP

CLR P1.3

RET

DISPLAY:

CLR A

MOVC A,@A+DPTR

INC DPTR

CJNE A,#0FFH,DATA

RET

DATA:

MOV R3,#03H

ACALL DELAY

SJMP DISPLAY

DATAWRT:

MOV P2,A

SETB P1.5

CLR P1.8

SETB P1.5

NOP

NOP

CLR P1.5

RET

DELAY:

PUSH 02H

PUSH 01H

HERE3: MOV R1,#0FFH

HERE2: MOV R2,#0FFH

HERE1: DJNZ R2,HERE1

DJNZ R1,HERE2

DJNZ R3,HERE3

POP 01H

POP 02H

RET

DELAY1:

PUSH 02H

PUSH 01H

HERE4: MOV R1,#12H

HERE5: MOV R2,#0FFH

HERE6: DJNZ R2,HERE6

DJNZ R1,HERE5

DJNZ R3,HERE4

POP 01H

POP 02H

RET

CONF: DB 01H,06H,12H,ABH,CDH,0CH,38H,80H,FFH

CONF1: DB C0H,3AH,CDH,0FFH

CONF2: DB 07H,0CH,FFH

MSG1: DB 'WELCOME TO',FFH

MSG11: DB 'DSCSDEC',FFH

MSG2: DB 'DESIGNED BY',FFH

MSG21: DB 'PRIYANKA SAHA',FFH

MSG22: DB 'RAJANYA GUHA ',FFH

MSG3: DB 'GUIDED BY',FFH

MSG31: DB 'MR.ANIRBAN NEOGI',FFH

MSG32: DB 'PRESS ANY SWITCH',FFH

MSG40: DB 'IC TESTED AND',FFH

MSG43: DB 'IC TESTED OR',FFH

MSG44: DB 'IC TESTED NOR',FFH

MSG45: DB 'IC TESTED NAND',FFH

MSG46: DB 'IC TESTED XOR',FFH

MSG47: DB 'IC TESTED XNOR',FFH

MSG48: DB 'IC TESTED NOT',FFH

MSG41: DB 'OK',FFH

MSG42: DB 'NOT OK',FFH

**Display NOT OK**

**Configuration How to Display**

**Checking O/P pin 1**

**Given all I/P pins 11**

**Display NOT OK**

**Configuration How to Display**

**Checking O/P pin 0**

**Given all I/P pins 10**

**Display NOT OK**

**Configuration How to Display**

**Checking O/P pin 0**

**Given all I/P pins 01**

**Display NOT OK**

**Configuration How to Display**

**Checking O/P pin 0**

**Given all I/P pins 00**

**Whether AND is Selected**

**Display Mr. Anirban Neogi**

**Configuration How to Display**

**Configuration How to Display**

**Display Rajanya Guha**

**Start**

**Display Priyanka Saha**

**Display DSCSDEC**

**Configuration How to Display**

**Configuration How to Display**

Y N

Y

N

N

Y

Y N

**Configuration How to Display**

**Stop**

**Display OK**

**Stop**

**Display NOT OK**

**Display NOT OK**

**Display NOT OK**

**Display NOT OK**

**Configuration How to Display**

**Checking O/P pin 1**

**Given all I/P pins 11**

**Configuration How to Display**

**Checking O/P pin 1**

**Given all I/P pins 10**

**Configuration How to Display**

**Checking O/P pin 1**

**Given all I/P pins 01**

**Checking O/P pin 0**

**Configuration How to Display**

**Given all I/P pins 00**

**Whether OR is Selected**

**Display OK**

**Configuration How to Display**

Y Y N

**Stop**

**Display OK**

**Configuration How to Display**

**Display NOT OK**

**Configuration How to Display**

**Checking O/P pin 0**

**Given all I/P pins 11**

**Display NOT OK**

**Configuration How to Display**

**Checking O/P pin 1**

**Given all I/P pins 10**

**Display NOT OK**

**Configuration How to Display**

**Checking O/P pin 1**

**Given all I/P pins 01**

**Display NOT OK**

**Configuration How to Display**

**Given all I/P pins 00**

**Checking O/P pin 1**

**Whether NAND is Selected**

**Stop**

**Display OK**

**Configuration How to Display**

**Display NOT OK**

**Configuration How to Display**

**Checking O/P pin 0**

**Given all I/P pins 11**

**Display NOT OK**

**Configuration How to Display**

**Checking O/P pin 1**

**Given all I/P pins 10**

**Display NOT OK**

**Configuration How to Display**

**Checking O/P pin 1**

**Given all I/P pins 01**

**Display NOT OK**

**Configuration How to Display**

**Checking O/P pin 0**

**Given all I/P pins 00**

**Whether XOR is Selected**

**Stop**

**Display OK**

**Configuration How to Display**

**Display NOT OK**

**Checking O/P pin 1**

**Configuration How to Display**

**Given all I/P pins 11**

**Display NOT OK**

**Configuration How to Display**

**Checking O/P pin 0**

**Given all I/P pins 10**

**Checking O/P pin 0**

**Display NOT OK**

**Configuration How to Display**

**Given all I/P pins 01**

**Display NOT OK**

**Configuration How to Display**

**Checking O/P pin 1**

**Given all I/P pins 00**

**Whether XNOR is Selected**

**Whether NOR is Selected**

**Display NOT OK**

**Configuration How to Display**

**Checking O/P pin 1**

**Given all I/P pins 00**

**Display NOT OK**

**Configuration How to Display**

**Checking O/P pin 0**

**Given all I/P pins 01**

**Given all I/P pins 10**

**Display NOT OK**

**Configuration How to Display**

**Checking O/P pin 0**

**Stop**

**Display OK**

**Configuration How to Display**

**Display NOT OK**

**Configuration How to Display**

**Checking O/P pin 0**

**Given all I/P pins 11**

**Stop**

**Display OK**

**Configuration How to Display**

**Display NOT OK**

**Configuration How to Display**

**Checking O/P pin 0**

**Given I/P pin 1**

**Display NOT OK**

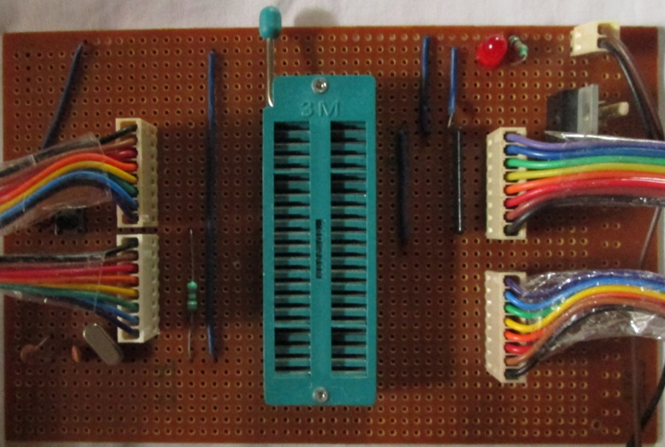
**Configuration How to Display**

**Checking O/P pin 1**

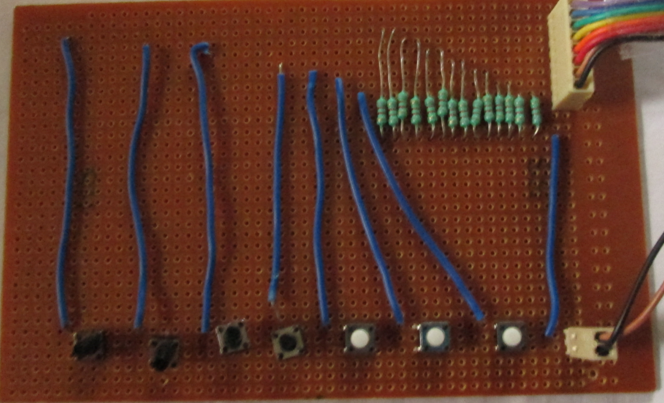
**Given I/P pin 0**

**Whether NOT is Selected**

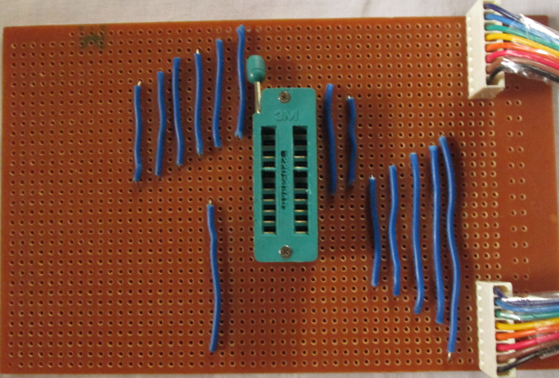
**3.11 Snapshots:**

****

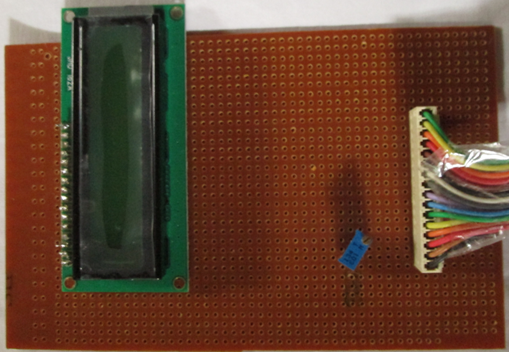
**Microcontroller Board**

****

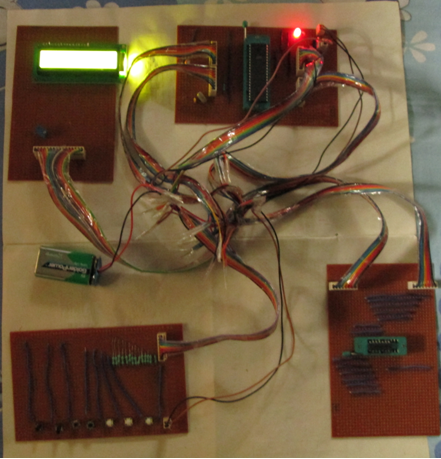
**Keyboard Interfacing Board**

****

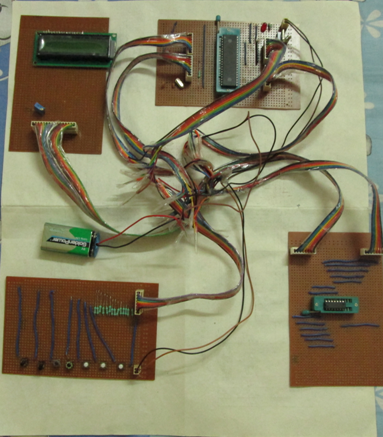
**IC Interfacing Board**

****

**LCD Board**

****

**Full Circuit With Power**

****

**Full Circuit without power**

* 1. **Software Used:**

We used the Software TOPWIN6 to burn the program on the 89s52 chip.

* 1. **Work Done:**
* After deciding on the topic of the project we gathered information about it.
* **The first job was to collect the required components.**
* We started with making the microcontroller board followed by the LCD board, IC interfacing board and the keyboard interfacing board.
* We made the necessary connections and finally interconnected all the boards.
* We connected a 9v battery with the microcontroller board.
* We wrote the program and with the help of TOPWIN6 software we burned it on 89s52 chip.
* We tested the digital ICs one by one to check whether they were OK or NOTOK.
  1. **Conclusion:**

The project report has been successfully completed and the main objective of emulating an IC tester on 89s52 micro controller has been achieved. We have completed the designing of the hardware and have to submit the hardware implementation.

1. **FUTURE ASPECTS:**

The project can be extended as following:

1) It can be extended for more than 16 pin ICs by changing some hardware and coding.

2) It can be extended to analog IC’s

1. **REFERENCES:**

[www.wikipedia.com](http://www.wikipedia.com)

[www.google.com](http://www.google.com)

electrofriends.com

Digital Electronic Circuits by Salivahanan