INTERNSHIP REPORT

*A report submitted in partial fulfilment of the requirements for the Award of Degree of*

## BACHELOR OF TECHNOLOGY

**in**

## ELECTRONICS AND COMMUNICATION ENGINEERING

**by**

**ADITYA KUMAR**

## Regd.NO.: 21781A04L8

**Under Supervision of**

**INTRAINZ INNOVATION(EDUTECH)**

VISHNU P NAIR

WITHEmbedded system

INTRAINZ, BENGALURU

 **(Duration: 01/08/2023 to 01/10/2023)**

SRI VENKATESWARA COLLEGE OF ENGINEERING& TECHNOLOGY (AUTONOMOUS)

R.V.S NAGAR, CHITTOOR – 517127. (A.P)

(Approved by AICTE, New Delhi, Affiliated to JNTUA, Ananta Puram) (Accredited by NBA, New Delhi & NAAC, Bengaluru)

(An ISO 9001:2000 Certified Institution)

2022-2023

SRI VENKATESWARA COLLEGE OF ENGINEERING& TECHNOLOGY (AUTONOMOUS)

R.V.S NAGAR, CHITTOOR – 517127. (A.P)

(Approved by AICTE, New Delhi, Affiliated to JNTUA, Ananta Puram) (Accredited by NBA, New Delhi & NAAC, Bengaluru)

(An ISO 9001:2000 Certified Institution)



**CERTIFICATE**

This is to certify that the “**Internship report”** submitted by **ADITYA KUMAR (Regd.No.:21781A04L8)** is work done by him/her and submitted during 2022- 2023 academic year, in partial fulfillment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY in ELECTRONICS AND COMMUNICATION ENGINEERING,** at **INTRAINZ INNOVATION, BENGALURU.**

# Head of the Department

**CERTIFICATE FO INTERNSHIP**

**(Issued by the company letter scanned copy)**



**ACKNOWLEDGEMENT**

A grateful thanks to **Dr. R. Venkataswamy,Chairman** of Sri Venkateswara College of Engineering & Technology for providing education in their esteemed institution.

I wish to record my deep sense of gratitude and profound thanks to our beloved

**Vice Chairman**, **Sri R.V.Srinivas** for his valuable support throughout the course.

I express our sincere thanks to **Dr.M.MOHAN BABU,** our beloved principal for his encouragement and suggestion during the course of study.

With the deep sense of gratefulness, I acknowledge **Dr. T.SOMASSOUNDARAM,** Head of the Department, Electronics and Communication Engineering, for giving us inspiring guidance in undertaking internship.

I express our sincere thanks to the internship coordinator, for her keen interest, stimulating guidance, constant encouragement with our work during all stages, to bring this report into fruition.

I wish to convey my gratitude and sincere thanks to all members for their support and cooperation rendered for successful submission of report.

Finally, I would like to express my sincere thanks to all teaching, non-teaching faculty members, our parents, friends and for all those who havesupported us to complete the internship successfully.

# NAME OF THE STUDENT: ADITYA KUMAR

# ROLL. NO. : 21781A04L8

**WEEKLY OVER VIEW OF INTERNSHIP ACTIVITIES**

|  |  |  |  |
| --- | --- | --- | --- |
| **1stWEEK** | **DATE** | **DAY** | **NAME OF THE TOPIC /MODULE COMPLETED** |
|  | Monday |  |
|  | Tuesday |  |
| 04/08/23 | Wednesday | Introduction to Embedded system(history,Evolution,Application) |
| 05/08/23 | Thursday | Overview of Architecture of 8051-Processor core and Functional Block Diagram |
|  | Friday |  |
|  | Saturday |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **2ndWEEK** | **DATE** | **DAY** | **NAME OF THE TOPIC/MODULE COMPLETED** |
|  | Monday |  |
|  | Tuesday |  |
| 11/08/23 | Wednesday | Memory organisation and Special Function registers(SFRs) |
| 12/08/23 | Thursday | Addressing Modes and Instruction set |
|  | Friday |  |
|  | Saturday |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **3rd WEEK** | **DATE** | **DAY** | **NAME OF THE TOPIC/MODULE COMPLETED** |
|  | Monday |  |
|  | Tuesday |  |
| 18/08/23 | Wednesday | Developing, Building and Debugging ALP's and (Assembly level programming) |
| 19/08/23 | Thursday | Embedded C and I/O functionalities -Interfacing LED'S and SWITCHE |
|  | Friday |  |
|  | Saturday |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **4th WEEK** | **DATE** | **DAY** | **NAME OF THE TOPIC/MODULE COMPLETED** |
|  | Monday |  |
|  | Tuesday |  |
| 25/08/23 | Wednesday | Interfacing LCD with 8051 |
| 26/08/23 | Thursday | Interfacing keypad matrix with 8051 |
|  | Friday |  |
|  | Saturday |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **5th WEEK** | **DATE** | **DAY** | **NAME OF THE TOPIC/MODULE COMPLETED** |
|  | Monday |  |
|  | Tuesday |  |
| 01/09/23 | Wednesday | Interfacing keypad matrix with 8051 |
| 02/09/23 | Thursday | Serial Communication with UART Protocal |
|  | Friday |  |
|  | Saturday |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **6th WEEK** | **DATE** | **DAY** | **NAME OF THE TOPIC/MODULE COMPLETED** |
|  | Monday |  |
|  | Tuesday |  |
| 08/09/23 | Wednesday | Interrupt programming and Priority management |
| 09/09/23 | Thursday | Communcation Protocols (I²C, Spi) |
|  | Friday |  |
|  | Saturday |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **7th WEEK** | **DATE** | **DAY** | **NAME OF THE TOPIC/MODULE COMPLETED** |
|  | Monday |  |
|  | Tuesday |  |
| 15/09/23 | Wednesday | Introduction to Advanced controllers - ARM cortex |
| 16/09/23 | Thursday | Minor project(counter based on IR sensor) |
|  | Friday |  |
|  | Saturday |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **8th WEEK** | **DATE** | **DAY** | **NAME OF THE TOPIC/MODULECOMPLETED** |
|  | Monday |  |
|  | Tuesday |  |
| 22/09/23 | Wednesday | I/O functionalities -Interfacing Leds and Switches, LCD with ARM controller |
| 23/09/23 | Thursday | Major project(Counter programming in keil and simulation on Proteus) |
|  | Friday |  |
|  | Saturday |  |

# ABSTRACT

**INDEX**

# 

|  |  |  |
| --- | --- | --- |
| **S.no** | **CONTENTS** |  |
| 1. | Introduction |  |
| 2. | Chapter-1 Introduction to Embedded systems |  |
| 1.1.History of Embedded systems  1.2.Evolution  1.3.Applications |  |
| 3 | Chapter-2 Architecture of 8051 micro controller |  |
| 2.1.Processor core  2.2.Functional block diagram |  |
| 4 | Chapter-3 Memory organisation and Special Function registers |  |
| 3.1. Introduction to Memory organisation  3.2.Registers |  |
| 5  6  7  8  9  10  11  12 | Chapter-4 Assembly level programming  4.1.Developing  4.2.Building and debugging  Chapter-5 Interfacing keypad matrix with 8051  Chapter-6 Timers and Counters  Chapter-7 Serial communication with UART Protocol  Chapter-8 Interrupt  8.1.Programming  8.2.Priority management  Chapter-9 Communication protocols  9.1.I2C  9.2.Spi  Chapter-10 Advance controllers  10.1.Introduction  10.2.ARM cortex  Chapter-11 Programming with Embedde C  As per the Internship content include the number of chapters |  |
|  | Conclusion |  |
|  | Bibliography |  |

1. **Introduction**

**Embedded System**

    Whenever I hear the term “**Embedded System**”, what comes to mind is “A combination of hardware and software” as instructed at the colleges. Well, instead of calling it as merely a combination of hardware and software, it would be apt to define it as application specific, organized hardware, controlled by specific software in which the hardware and software are the components of the embedded system. And there are many versions of the definition of an embedded system which ultimately culminate as said above.

**“Parts” of an embedded system-**

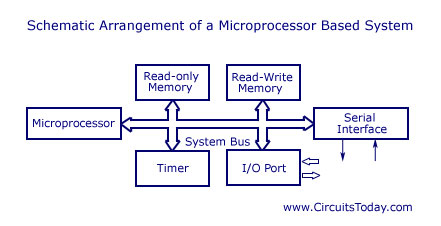
    Mainly, the hardware components constitute power source, microcontroller/microprocessor, timers, memory, and whatever needed for running the specific task. And the software components constitute programs such as compilers, integrated development environments (IDE), assemblers, simulators etc., which are used to create codes that “instruct” the hardware to do the assigned job in an efficient manner.

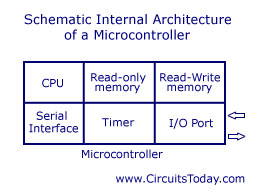
Notable computer languages that are used for programming in embedded systems are embedded C, embedded C++, embedded JAVA and assembly. Here embedded C and others contain a specific library for a microcontroller to work. (Like the specific header files such as math.h, conio.h). Mostly, for simple applications, an assembly is used, which produces more efficient, compact codes.

Some of the open-source operating systems used in embedded systems are Android, Microcontroller operating system (µCOS), VS works.

**The secret of Processor/Controller:**

    Microcontroller/microprocessor is analogous to the brain of the embedded system. It performs all the calculation and decision part of the process. You would be surprised to know that the only arithmetic operation the processor/controller is capable of doing is, addition! (and the modern computer too). Multiplication is repeated addition, subtraction is the addition of negative numbers and division is repeated addition of negative numbers.

[](https://www.circuitstoday.com/wp-content/uploads/2011/12/microprocessor-based-system.jpg)

[](https://www.circuitstoday.com/wp-content/uploads/2011/12/microcontroller-system.jpg)

   In most of the embedded systems, a microcontroller is chosen. Because, as you all know, we can call the microcontroller as a mini-computer. The microcontroller integrates many useful components such as memory, timers, counters, ADC, DAC etc onto the same package along with the controller. In contrast, for a microprocessor, timers, counters are to be provided separately and memory should also be interfaced separately which involves some additional circuitry and again which eats more space. One of the major goals of embedded system devices is compactness. Hence, embedded system vendors mostly adopt for microcontrollers.

This doesn’t mean that microprocessor is less applicable! It is as important as the microcontroller which has its own applications. In general, the microcontroller is designed for a specific purpose. [For example, an automatic washing machine, a cell phone etc. Of course, you can implement the same cell phone with a microprocessor, but it takes too much space and also as much circuitry is involved, more power is consumed.] But microprocessor is designed for a general purpose. The same microprocessor can be used for designing an automatic washing machine and also a cell phone, but that’s not the case with microcontrollers, its specific for a specific device. In summarizing, we can roughly refer the microcontroller as an enhanced microprocessor, enhanced for a specific task.

**Some Examples:**

[](https://www.circuitstoday.com/wp-content/uploads/2011/12/embedded-systems.jpg)

Many companies such as Integrated electronics, Microchip, Atmel, Philips, Hitachi manufactures microcontrollers. Most notable among them are, 8051 from Intel, PIC series from Microchip, AVR series from Atmel, 68HC11 and HD44780 LCD controller from Hitachi. The PIC controllers are mostly used by hobbyists.  Some examples of the embedded system worth mentioning are cell phones, air conditioner, car dashboard control, PMPs, robots, scientific calculators etc.

**Applications of embedded systems**

Embedded systems are used in a wide range of industries ranging from calculators to drones. Below are certain applications of embedded systems;

**Traffic control system**

Embedded system integrated traffic lights can detect which directions have the highest density of traffic and change the traffic lights and their timings based on this info. This will be a great boon to cities that are controlled by fixed timer traffic lights.

**Smart homes**

IOT (Internet Of Things) is closely knitted with embedded systems. All the appliances of a smart home which can be controlled via the internet are integrated with embedded systems.

**Automotive industry**

Vehicles are equipped with embedded systems. Major functions like temperature control (AC), ABS, airbags, automatic rain sensor wipers etc are controlled by these systems.

**Healthcare**

A lot of instruments used in healthcare like blood pressure monitor, scanners, pacemakers etc works with the help of embedded systems.

# Microcontroller Memory Organization and Types – Explained with Memory Segments

In microcontroller based bare metal embedded systems, memory is a resource constrained key component. Because microcontrollers come with a limited amount of memory. Hence, it is a limited resource and we should use it efficiently.

## Memory Introduction

Computer memory stores every information in the form of bits either zero or one. Hence, the main basic component of memory is a bit. But memory is usually organized in bytes. One byte consists of 8 bits. Therefore, one byte is the minimum information that microcontrollers can read and write. In other words, every memory location is byte addressable. That means, each memory location consists of one byte and each location has a unique address. Hence, memory is organized in the order of hundreds and thousands of bytes.

### **Power control (PCON)**

The PCON register is responsible for managing the power modes of the 8051 microcontrollers. It is placed at the address 87H and is not bit addressable.

In most embedded system devices, power management is the biggest constraint for any developer. The 8051 offers a power-down mode and an idle mode to decrease its power requirements. In the power-down mode, the clock of the microcontroller is frozen, stopping all the operations of the microcontroller. This drops the voltage requirement for the working of the microcontroller from 5v to 2v, and the current demand reduces below 60 microamperes.

**Power down mode**

During the power-down mode, the status of all the SFRs and input/output ports remains the same, and the only way to wake up this sleepy microcontroller is by using a hardware reset. When the microcontroller comes back to its normal mode after the reset, the values of all the SFRs are lost, but the data in the general-purpose RAM remains the same.

**Idle mode**

In the case of the idle mode, the most power-hungry unit, the CPU of the microcontroller, is turned off. All the other peripherals like the timer, serial communication, and interrupts keep functioning normally. Also, the statuses of the CPU, Accumulator, and the stack pointer remain as it is.

There are two ways to pull the 8051 microcontroller out from the idle mode. An interrupt that changes the value at PCON.0. Or a hardware reset.

Here’s what the innards of the PCON register contain.

* **SMOD:** The baud rate in serial communication doubles when this bit is set to 1.
* **GF1 and GF0:** are general-purpose flag bits.
* **Power Down(PD)**: When this bit is set to 1, the microcontroller goes into power-down mode.
* **Idle(IDL):** This bit puts the microcontroller in idle mode when the bit is 1.

### **DPTR**

DPTR or the data pointer register, as the name suggests, is a register that points to some data. In any computer system, data is kept at a particular address, and the DPTR is responsible for storing the address. Due to the 16-bit address bus of the 8051 microcontroller, it has a 16-bit DPTR so that it can access all the data locations. It is divided into two parts, namely DPH(4 upper bits ) and DPL(4 lower bits). It is not a bit addressable and is placed at locations 82H(DPL) and 83H(DPH)

In 8051 there are 1-byte, 2-byte instructions and very few 3-byte instructions are present. The opcodes are 8-bit long. As the opcodes are 8-bit data, there are 256 possibilities. Among 256, 255 opcodes are implemented.

The clock frequency is12MHz, so 64 instruction types are executed in just 1 µs, and rest are just 2 µs. The Multiplication and Division operations take 4 µsto to execute.

In 8051 There are six types of addressing modes.

* Immediate AddressingMode
* Register AddressingMode
* Direct AddressingMode
* Register IndirectAddressing Mode
* Indexed AddressingMode
* Implied AddressingMode

## Immediate addressing mode

In this Immediate Addressing Mode, the data is provided in the instruction itself. The data is provided immediately after the opcode. These are some examples of Immediate Addressing Mode.

In these instructions, the # symbol is used for immediate data. In the last instruction, there is DPTR. The DPTR stands for Data Pointer. Using this, it points the external data memory location. In the first instruction, the immediate data is AFH, but one 0 is added at the beginning. So when the data is starting with A to F, the data should be preceded by 0.

## Register addressing mode

In the register addressing mode the source or destination data should be present in a register (R0 to R7). These are some examples of RegisterAddressing Mode.

In 8051, there is no instruction like **MOVR5, R7**. But we can get the same result by using this instruction **MOV R5, 07H**, or by using **MOV 05H, R7**. But this two instruction will work when the selected register bank is **RB0**. To use another register bank and to get the same effect, we have to add the starting address of that register bank with the register number. For an example, if the RB2 is selected, and we want to access R5, then the address will be (10H + 05H = 15H), so the instruction will look like this **MOV 15H, R7**. Here 10H is the starting address of Register Bank 2.

## Direct Addressing Mode

In the Direct Addressing Mode, the source or destination address is specified by using 8-bit data in the instruction. Only the internal data memory can be used in this mode. Here some of the examples of direct Addressing Mode.

The first instruction will send the content of registerR6 to port P0 (Address of Port 0 is 80H). The second one is forgetting content from 45H to R2. The third one is used to get data from Register R5 (When register bank RB0 is selected) to register R5.

## Register indirect addressing Mode

In this mode, the source or destination address is given in the register. By using register indirect addressing mode, the internal or external addresses can be accessed. The R0 and R1 are used for 8-bit addresses, and DPTR is used for 16-bit addresses, no other registers can be used for addressing purposes. Let us see some examples of this mode.

## Indexed addressing mode

In the indexed addressing mode, the source memory can only be accessed from program memory only. The destination operand is always the register A. These are some examples of Indexed addressing mode.

The C in MOVC instruction refers to code byte. For the first instruction, let us consider A holds 30H. And the PC value is1125H. The contents of program memory location 1155H (30H + 1125H) are moved to register A.

## Implied Addressing Mode

In the implied addressing mode, there will be a single operand. These types of instruction can work on specific registers only. These types of instructions are also known as register specific instruction. Here are some examples of Implied Addressing Mode.

These are 1- byte instruction. The first one is used to rotate the A register content to the Left. The second one is used to swap the nibbles in A.

# Assembly Programming

Top of Form

Bottom of Form

Assembly language is a low-level programming language for a computer or other programmable device specific to a particular computer architecture in contrast to most high-level programming languages, which are generally portable across multiple systems. Assembly language is converted into executable machine code by a utility program referred to as an assembler like NASM, MASM, etc.

**Debugger**: A debugger executes a program according to the controlling of the user. A debugger allows the location and correction of errors if present in any program, and this is known as **debugging**.

Efficient LED Blinking for Embedded Systems

**Circuit and working**  
 The circuit for efficient LED blinking for embedded systems. It uses different arrangements to make five different LEDs (LED1 through LED5) blink at individual rates. The circuit is built around 74HC14 (IC1) containing six inverters with Schmitt triggers at the inputs. The frequency of blinking is usually selected in the 0.2-20Hz range.

The combination of inverter N1, resistor R1 and capacitor C1 is used as a square wave generator to blink LED1. LED1 blinks when output OUT0 from the MCU is high and goes off when output OUT0 is low.

The combination of inverter N2, resistor R2 and capacitor C2 works as a second square wave generator that makes LED2 blink. LED2 blinks when output OUT1 of the MCU is low and glows constantly when the output is high due to reverse orientation of D2.

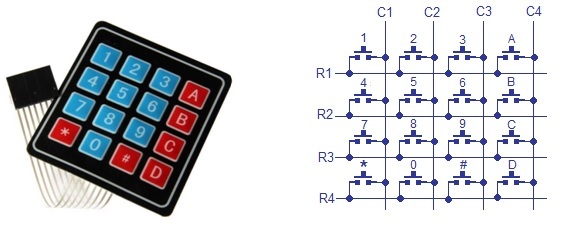
Sometimes an LED needs more current than that provided by a single inverter. In that case, two or more inverters can be connected in parallel. Such an arrangement is made using inverters N3 and N4, resistor R3 and capacitor C3. The direction of diode D3 determines the logic of control of LED3 as indicated in the previous arrangements.

If you want two LEDs blinking alternately, the setup is made using inverters N5 and N6 together with resistor R4 and capacitor C4.

When OUT3 is low, LED4 is ‘off’ and LED5 is ‘on.’ When OUT3 is high, LED4 and LED5 blink alternately. The frequency of blinking in all the cases depends on the value of the adjoining resistor and capacitor.

User will give input through keypad and then that input will be displayed on LCD.

**KEYPAD STRUCTURE:**

In a keypad, push button switches are arranged in rows and columns. For a 4×4 keypad 16 switches are used and to connect to microcontroller we need 16 inputs pins. But the arrangement is changed by connecting switches in a special way. Now we need only 8 pins of microcontroller to connect keypad to it.              [](https://microcontrollerslab.com/wp-content/uploads/2017/02/keypad-interfacing-with-8051.jpg)

The status of each key/switch is determined by Scanning the rows or columns. The column pins (Col 1–Col4) are connected to the microcontroller as the inputs pinsand the rows pins (Row 1–Row 4) are connected to the output pins of the microcontroller. Normally, all the column pins are pulled high by internal or external pull up resistors. Now we can read the status of each switch through scanning.

**READING DATA:**

Scanning is done in a different way. Columns pins are used as input pins, and rows pins are used as output. If a low logic is given to all the Rows and high logic is given to each Column.

**For finding Column number:**

* When a switch/key is pressed, the corresponding row and column will get short.
* Output of the corresponding column goes to go low.
* Since we have made all the rows zero so this gives the column number of the pressed key.

**For Finding Row number:**

* After the detection of column number, the controller set’s all the rows to high.
* Each row is one by one set to zero by the microcontroller and the earlier detected column is checked and obviously it becomes zero.
* The row due to which the column gets zero is the row number of the pressed key.

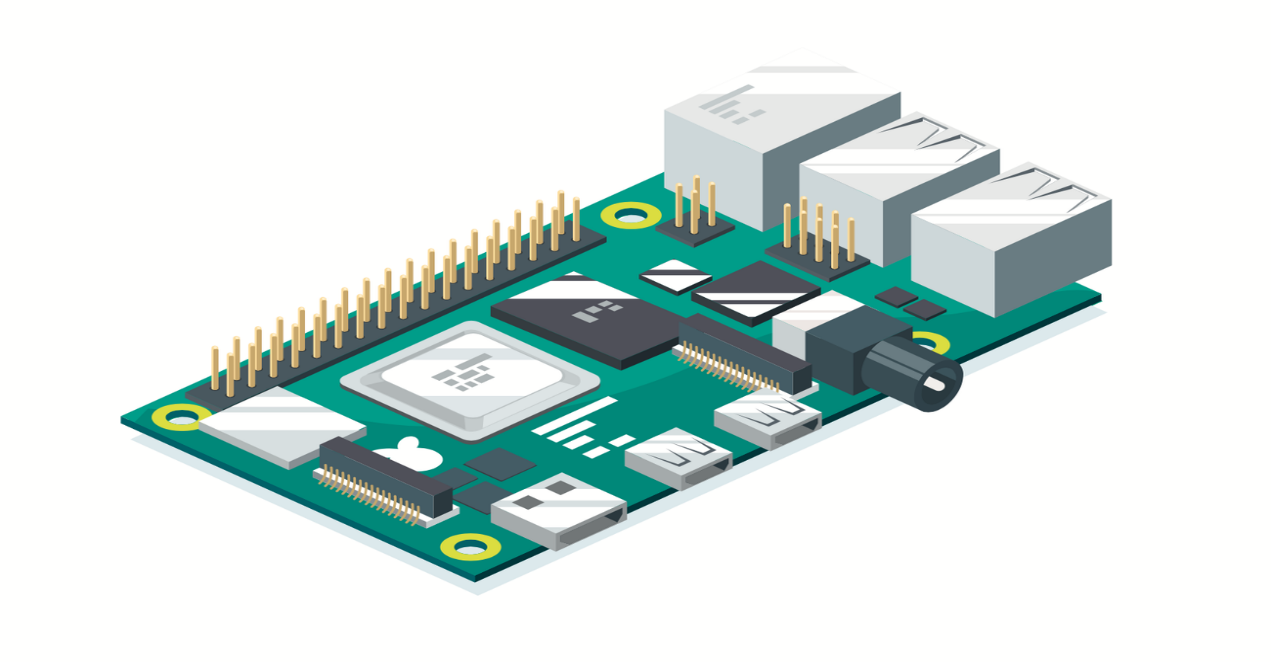
### **keypad interfacing 8051 microcontroller8**

For the interfacing of keypad with the microcontroller, it is good to connect LCD also, so that we can observe specific changes if the keypad is pressed. Keypad 4×3 (having 4 rows and 3 columns) is connected to Port 3 of 8051 microcontroller to scan input. LCD is connected to Port 2 of the microcontroller for displaying output. Port1 pin1and pin2 of microcontroller is connected to RS and EN pins of LCD respectively. LED is connected to port 1 pin 3.

### **WORKING of keypad interfacing 8051 microcontroller**

In the start, LCD will display “WELCOME” and then display “Enter No.”. In above screenshot, I pressed 2. It is then written on LCD. When button is pressed, Led will get on and then then off after some delay. The number will appear on the screen as long as other number is not pressed. This display arrangement can be changed by arranging the code in different way.

Serial Communications Protocols - Part Two: UART



we will be looking at the popular UART device. It’s not a communication protocol but rather the physical circuit used in microcontrollers or stand-alone ICs to tx and rx serial data.

Serial communications are the perfect mechanism for transmitting information between devices such as microcontroller data-producing peripherals, and other smart devices. We hope you will find this resource invaluable next time your design needs to incorporate a serial communication bus. The UART is commonly also referred to as the Serial Port on a device. However, this can often be confused with another kind of serial port, which uses the DB9 style connector and uses RS-232 as a hardware protocol

**UART** stands for Universal Asynchronous Receiver Transmitter. This communication system requires two pins, except for the ground. One is labeled TX, the transmitter, and the other labeled RX, the receiver. The appearance of the word asynchronous in the name means that it does not require a clock signal. As it does not have a clock, the transmitting and the receiving devices must use the same baud rate.

**Baud rate** (Bd) is a unit of measurement of the transmission rate. This parameter determines the communication speed over the data channel of the transmitter and receiver. Baud rate can be represented as bits per second to make it simpler. So, a rate of 1000 Bd means that the communication speed is 1000 bits per second, or the duration of one bit is 1/1000 second or 1 millisecond. Baud rates have standard metric prefixes such as kBd (kilobaud), MBd (megabaud), or GBd (Gigabaud). The baud rate includes non-data parts of a message such as a start and stop bits, which means that the transmission rate of useful information sent from the transmitter to the receiver will be slightly less.

Some of the more popular predefined UART baud rates include:

* 110 Baud
* 300 Baud
* 1200 Baud
* 4800 Baud
* 9600 Baud
* 19200 Baud
* 38400 Baud
* 115200 Baud

Sometimes, both communicating devices’ baud rate can be adjusted, while sometimes, one of the devices will have a fixed baud rate requiring that the other one change match it.

So, to connect the two devices using the UARTs, the transmitter TX pin should be connected to the receiver RX pin:

Basic UART wiring with TX, RX, and ground

You will have **RX** and **TX** pins on both devices in some configurations, allowing you to send commands from a microcontroller to a dedicated IC and receive orders or information back in the other direction. That is called a duplex connection. There are also applications when feedback from the second device is not required. Hence, communications are only necessary for one direction; therefore, only one TX to RX transmission line is needed. This is called a simplex connection.

**UART** device sends a start bit (beginning communications)

The data bits are then sent as an eight-bit stream, representing the real data, where the most significant bit (MSB) is labeled D7, and the least essential bit (LSB) is labeled D0.

# Interrupts

An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. Whenever an interrupt occurs, the controller completes the execution of the current instruction and starts the execution of an **Interrupt Service Routine** (ISR) or **Interrupt Handler**. ISR tells the processor or controller what to do when the interrupt occurs. The interrupts can be either hardware interrupts or software interrupts.

## Hardware Interrupt

A hardware interrupt is an electronic alerting signal sent to the processor from an external device, like a disk controller or an external peripheral. For example, when we press a key on the keyboard or move the mouse, they trigger hardware interrupts which cause the processor to read the keystroke or mouse position.

## Software Interrupt

A software interrupt is caused either by an exceptional condition or a special instruction in the instruction set which causes an interrupt when it is executed by the processor. For example, if the processor's arithmetic logic unit runs a command to divide a number by zero, to cause a divide-by-zero exception, thus causing the computer to abandon the calculation or display an error message. Software interrupt instructions work similar to subroutine calls.

## What is Polling?

The state of continuous monitoring is known as **polling**. The microcontroller keeps checking the status of other devices; and while doing so, it does no other operation and consumes all its processing time for monitoring. This problem can be addressed by using interrupts.

In the interrupt method, the controller responds only when an interruption occurs. Thus, the controller is not required to regularly monitor the status (flags, signals etc.) of interfaced and inbuilt devices.

## Enabling and Disabling an Interrupt

Upon Reset, all the interrupts are disabled even if they are activated. The interrupts must be enabled using software in order for the microcontroller to respond to those interrupts.

IE (interrupt enable) register is responsible for enabling and disabling the interrupt. IE is a bitaddressable register.

## Interrupt inside Interrupt

What happens if the 8051 is executing an ISR that belongs to an interrupt and another one gets active? In such cases, a high-priority interrupt can interrupt a low-priority interrupt. This is known as **interrupt inside interrupt**. In 8051, a low-priority interrupt can be interrupted by a high-priority interrupt, but not by any another low-priority interrupt.

**Introduction to I²C and SPI protocols**

Today, at the low end of the communication protocols, we find I²C (for ‘Inter-Integrated Circuit’, protocol) and SPI (for ‘Serial Peripheral Interface’). Both protocols are well-suited for communications between integrated circuits, for slow communication with on-board peripherals. At the roots of these two popular protocols we find two major companies – Philips for I²C and Motorola for SPI – and two different histories about why, when and how the protocols were created.

**Minor project**

Report on “architecture of 8051 microcontroller”.

A **microcontroller** is a small and low-cost microcomputer, which is designed to perform the specific tasks of embedded systems like displaying microwave’s information, receiving remote signals, etc.

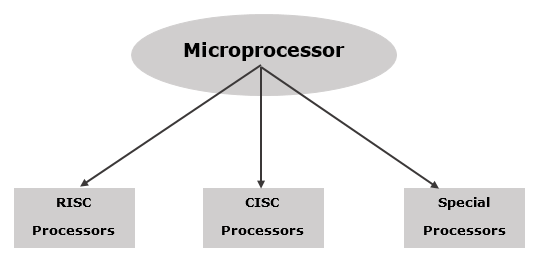
The general microcontroller consists of the processor, the memory (RAM, ROM, EPROM), Serial ports, peripherals (timers, counters), etc

8051 microcontroller is designed by Intel in 1981. It is an 8-bit microcontroller. It is built with 40 pins DIP (dual inline package), 4kb of ROM storage and 128 bytes of RAM storage, 2 16-bit timers. It consists of are four parallel 8-bit ports, which are programmable as well as addressable as per the requirement. An on-chip crystal oscillator is integrated in the microcontroller having crystal frequency of 12 MHz.

Let us now discuss the architecture of 8051 Microcontroller.

In the following diagram, the system bus connects all the support devices to the CPU. The system bus consists of an 8-bit data bus, a 16-bit address bus and bus control signals. All other devices like program memory, ports, data memory, serial interface, interrupt control, timers, and the CPU are all interfaced together through the system bus.

A microprocessor can be classified into three categories −



Some of the RISC processors are −

* Power PC: 601, 604, 615, 620
* DEC Alpha: 210642, 211066, 21068, 21164
* MIPS: TS (R10000) RISC Processor
* PA-RISC: HP 7100LC

Some of the CISC Processors are −

* IBM 370/168
* VAX 11/780
* Intel 80486

Special processor

A coprocessor is a specially designed microprocessor, which can handle its particular function many times faster than the ordinary microprocessor.

**For example** − Math Coprocessor.

Some Intel math-coprocessors are:

* 8087-used with 8086
* 80287-used with 80286
* 80387-used with 80386

**MAJOR PROJECT**

Developing a “digital clock using a microcontroller”.

**8**components needed **:**

**1. Microcontroller**(I have used AT89S52-8051 family), any programmable microcontroller can be used.

**2.7 segment display**

**3.Crystal oscillator (12MHz)**

**4.Capacitor (10uF, 33pF/22pF)**

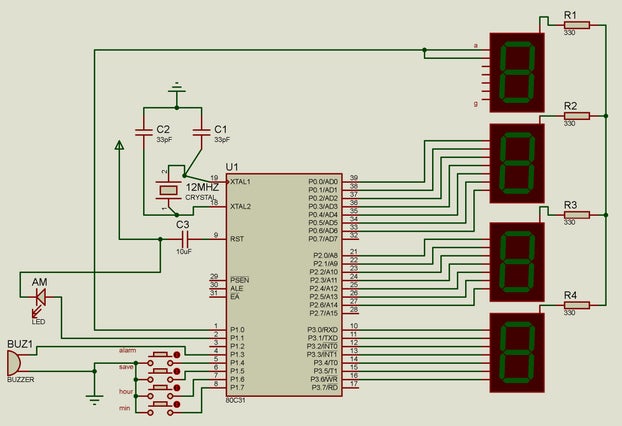
**5.LEDs**

**6.resistances (330 Ohm)**

**7.buzzer (piezo)**

**8.push switches**

**Circuit diagram**



This is the circuit diagram of the digital clock using 8051 microcontroller.

As we can see the microcontroller is connected to **three 7 segment display with distinct ports not multiplexed** and the last hour digit is only connected to a pin as it only shows **1.**

**LED and buzzer**are self explanatory according to the code.

1 of the LED is for AM and I have connected another **LED not shown in the figure for alarm.**

Crystal Oscillator of 12MHz is connected to clock speed and attaining the **exact 1second counting using the interrupt property of the microcontroller.**

**THE MIDDLE LEDS DENOTING SECOND ARE CONNECTED TO "28TH AND 32ND" PIN.**

**Please pardon me, 3 LEDs aren't shown in the circuit diagram for my laziness.**

28th pin LED: first 30 second blink

32nd pin LED: rest 30 second blink

**CONCLUSION:**

An embedded system is designed to do a specific job only.

Example: a washing machine can only wash clothes, an air conditioner can control the temperature in the room in which it is placed.

* The hardware & mechanical components will consist all the physical visible things that are used for input, output, etc.
* An embedded system will always have a chip(either microprocessor or micro controller) that has the code or software which drives the system.