UNIT 4

APPLICATION OF I/O DEVICES, TIMERS AND INTERRUPTS IN MICROCONTROLLER-BASED SYSTEMS

4.1 Introduction

Microcontrollers are designed to interact with the outside world through **Input/Output (I/O) ports**, **timers**, **counters**, **interrupts**, and **serial communication interfaces**. These features allow a microcontroller to monitor sensors, control devices, measure time, count events, and communicate with other systems.

The 8051 microcontroller contains:

- Four I/O ports (P0, P1, P2, P3)
- Two timers/counters
- Serial communication interface
- Interrupt handling system

What Each Part Does

Component	What It Receives	What It Does
I/O Port	Digital signals	Controls LEDs, motors, sensors
Timer	Internal clock pulses	Measures time
Counter	External pulses	Counts events
Interrupt	Urgent external signals	Gets CPU attention immediately
Serial Communication	Data bits	Exchanges information with other devices

These features make the 8051 suitable for automation, robotics, traffic control, security systems, and industrial control applications.

***4.2 Addressing of I/O and RAM ***

NOTE: revison

1. Address Bus = Carrying the Direction

You learned:

- Address Bus
- Data Bus

Think of a post office.

Address Bus

Carries the destination:

```
House No. 25
Street A
```

In a microcontroller:

```
Address = 30H
```

The CPU says:

"I want to access location 30H."

This address travels on the address bus.

Data Bus

Carries the actual information.

Example:

```
Address = 30H
Data = 55H
```

The address bus points to location 30H.

The data bus carries 55H.

---

2. Addressing Modes = How the Address is Specified

An **addressing mode is not the address itself**.

It is the **method used to find the address or data**.

Example

Immediate addressing.

```
MOV A,#25H
```

When this program is stored in ROM/program memory, the instruction and the data `25H` are stored together. The data came directly from the instruction itself. That's why it is called **Immediate Addressing**.

Eg. Write the number 25 on the board. You already know the number. No need to look for it.

The data is given directly. No memory location is needed.

Direct addressing.

```
MOV A,30H
```

`Eg.` Go to locker 30 and bring me what's inside. Now you must go to location 30 first.

The CPU is told exactly where the data is:

```
Address = 30H
```

Indirect addressing - The address is stored somewhere else.

```
MOV A,@R0
```

The CPU first looks inside R0.

Suppose:

```
R0 = 30H
```

Then it goes to address 30H.

Direct vs Indirect

Direct

```
MOV A,30H
```

CPU already knows address.

```
Address = 30H
```

Indirect

```
MOV A,@R0
```

CPU must first find address.

```
R0 → 30H
       ↓
Address
```

Indexed Addressing

This one is mainly used when accessing tables stored in program memory.

```
MOVC A,@A+DPTR
```

Suppose:

```
DPTR = 1000H
A    = 05H
```

CPU calculates: `1000H + 05H = 1005H`

Then reads data from: `Address 1005H`

Think of a library.

```
Shelf starts at 1000H
Book number = 5
```

To find the book:

```
Shelf + Offset
1000H + 5
```

This is why it is called **indexed**.

You start from a base address and add an index.

So:

```
Addressing mode = How to find data
Address = Actual location
```

Summary of Common Addressing Modes

Addressing Mode	Example	Where is the data?
Immediate	`MOV A,#55H`	Inside the instruction itself
Direct	`MOV A,30H`	At memory address 30H
Register	`MOV A,R1`	Inside register R1
Indirect	`MOV A,@R0`	Address stored in R0
Indexed	`MOVC A,@A+DPTR`	Base address + index

The key thing to remember is:

Every piece of data ultimately comes from some memory location somewhere. The addressing mode simply tells the CPU **how to find it**.

That's why addressing modes are often described as:

Methods used by the CPU to locate operands (data).

---

3. Addressing of RAM

Now let's come to today's topic.

Addressing of RAM means:

How does the CPU access RAM locations?

Example:

```
MOV A,40H
```

The CPU accesses RAM address: `40H`

Here the address refers to a RAM location.

RAM example:

```
Address      Data
30H          15H
31H          25H
32H          55H
```

If CPU executes:

```
MOV A,31H
```

Result:

```
A = 25H
```

because 31H is the address being accessed.

---

4. Addressing of I/O

This is the same idea, but now the destination is not RAM.

The destination is an I/O port.

Example:

```
MOV P1,#55H
```

Port 1 has an address: `P1 = 90H`

Internally, the CPU knows: "I am accessing location 90H."

But location 90H is not RAM.

It is the Port 1 register.

Think of it like this:

```
House 30 = RAM
House 90 = Port 1
House A0 = Port 2
House B0 = Port 3
```

The CPU uses addresses for all of them.

Big Picture

All these topics are connected.

```
Address
   ↓
A location to be accessed
```

Address Bus

```
Carries the address
```

Addressing Mode

```
Method used to find the address/data
```

RAM Addressing

```
Accessing memory locations
```

I/O Addressing

```
Accessing input/output ports
```

So when you see **"Addressing of I/O and RAM"**, you can think:

"This topic is about how the microcontroller identifies and accesses RAM locations and I/O port locations using their addresses."

4.2 Addressing of I/O and RAM

Definition

Addressing is the method used by a microcontroller to access data stored in memory locations or communicate with I/O devices.

The 8051 uses separate areas for:

- Internal RAM
- Special Function Registers (SFRs)
- External RAM
- I/O Ports

Internal RAM Organization

The 8051 contains **128 bytes of internal RAM**.

Memory Allocation

Address Range	Function
00H – 1FH	Register Banks
20H – 2FH	Bit-addressable RAM
30H – 7FH	General Purpose RAM

Example

```
MOV A,30H
```

Meaning:

- Copy data stored in RAM location 30H into accumulator A.

I/O Port Addressing

The 8051 has four ports:

Port	Address
P0	80H
P1	90H
P2	A0H
P3	B0H

Example

```
MOV P1,#55H
```

Operation:

- Sends binary value 01010101 to Port 1.

Importance of Addressing

Addressing enables the microcontroller to:

- Access memory locations
- Read sensors
- Control output devices
- Exchange information efficiently

4.3 I/O Programming

Definition

Input/Output programming is the process of transferring data between a microcontroller and external devices.

Examples of external devices include:

Input Devices

- Push buttons
- Keypads
- Sensors
- Switches

Output Devices

- LEDs
- LCD displays
- Relays
- Motors

Input Programming Example

Reading a switch connected to Port 1.

```
MOV A,P1
```

Operation:

- Reads the current status of Port 1 into the accumulator.

If switch is ON:

```
P1 = 1
```

If switch is OFF:

```
P1 = 0
```

Output Programming Example

Turning ON LEDs connected to Port 2.

```
MOV P2,#FFH
```

Result:

- All LEDs connected to Port 2 turn ON.

Practical Applications

- Traffic light systems
- Home automation
- Industrial machine control
- Security systems

4.4 Bit Manipulation

Definition

Bit manipulation means controlling or testing individual bits rather than an entire byte.

The 8051 supports direct bit operations.

Common Bit Instructions

SETB

Remember that Port 1 contains 8 bits:

```
P1.7
P1.6
P1.5
P1.4
P1.3
P1.2
P1.1
P1.0
```

Or:

Bit: 7 6 5 4 3 2 1 0

Example:

Sets a bit to 1.

```
SETB P1.0
```

Think of Port 1 as an 8-bit register:

```
P1 = 0 0 0 0 0 0 0 0
     7 6 5 4 3 2 1 0
```

After:

```
SETB P1.0
```

it becomes:

```
P1 = 0 0 0 0 0 0 0 1
```

The last bit (bit 0) is now 1.
Result:

- Pin P1.0 becomes HIGH.

CLR

Clears a bit to 0.

Example:

```
CLR P1.0
```

Result:

- Pin P1.0 becomes LOW.

CPL

Complements a bit.

CPL (Complement) is a bit manipulation instruction used to invert the value of a bit. If the bit is 1, it becomes 0; if the bit is 0, it becomes 1. It is commonly used for toggling outputs such as LEDs. i.e Change the bit to its opposite value.

Example:

```
CPL P1.0
```

Result:

- If bit was 1 it becomes 0.
- If bit was 0 it becomes 1.

JB (Jump if Bit Set)

Example:

```
JB P1.0,LABEL
```

JB stands for: **Jump if Bit is Set**

where:

- **Jump** = move program execution to another location (label).
- **Bit Set** = bit value is **1**.

So the instruction:

```
JB P1.0,LED_ON
```

means:

"Check bit P1.0. If it is 1, jump to the label LED_ON."

Example 1: Bit is 1

```
JB P1.0,LED_ON
MOV P2,#00H

LED_ON:
MOV P2,#FFH
```

Suppose:

```
P1.0 = 1
```

CPU executes:

1. Check P1.0
2. Finds it is 1
3. Jumps directly to `LED_ON`

Result:

```
P2 = FFH
```

The instruction:

```
MOV P2,#00H
```

is skipped.

Example 2: Bit is 0

Suppose:

```
P1.0 = 0
```

CPU executes:

1. Check P1.0
2. Finds it is 0
3. Does NOT jump
4. Continues with next instruction

```
MOV P2,#00H
```

What is a Label?

A label is simply a name marking a location in the program.

Example:

```
START:
MOV A,#55H

CHECK:
JB P1.0,LED_ON

LED_ON:
SETB P2.0
```

Here:

- START
- CHECK
- LED_ON

are labels. The CPU can jump to them.

Related Instructions

JNB (Jump if Not Bit Set)

```
JNB P1.0,LABEL
```

Means:

Jump if bit = 0.

JB

```
JB P1.0,LABEL
```

Means:

Jump if bit = 1.

Definition

JB (Jump if Bit Set) is a conditional branch instruction in the 8051 microcontroller. It tests a specified bit, and if that bit is equal to 1 (set), program execution jumps to the specified label; otherwise, execution continues with the next instruction.

Advantages of Bit Manipulation

- Faster execution
- Efficient memory use
- Precise control of hardware devices

4.5 Programming Timers and Counters

Definition of Timer

A timer is a hardware circuit used to measure time intervals.

Applications include:

- Delay generation
- Clock generation
- Event timing
- Traffic light control

Definition of Counter

A counter counts external events.

Examples:

- Counting products on a conveyor belt
- Counting vehicles
- Counting pulses from sensors

Timers in 8051

The 8051 contains:

- Timer 0
- Timer 1

Each timer is 16-bit.

Timer Registers

A timer register is a special register in a microcontroller used to store timer values and control timer operations. In the 8051, the main timer registers are TH0, TL0, TH1, TL1, TMOD, and TCON, which are used for counting, timer configuration, and timer control.

Think of timer as a digital stopwatch. A stop watch needs:

- A place to store the current count
- Buttons to start and stop it.
- Setting to choose how it works.

The 8051 uses special registers for these purposes.

Main Timer Registers in 8051 – TIMER 1& TIMER 0

1. TH0 and TL0 (Timer 0 Registers)

Timer 0 is 16 bits long, but the 8051 is an 8-bit microcontroller.

Therefore Timer 0 is divided into:

Register	Meaning
TH0	Timer 0 High Byte
TL0	Timer 0 Low Byte

Example:

```
TH0 = FCH
TL0 = 18H
```

Together they form:

```
FC18H
```

This is the current timer value.

---

2. TH1 and TL1 (Timer 1 Registers)

Used exactly the same way for Timer 1.

Register	Meaning
TH1	Timer 1 High Byte
TL1	Timer 1 Low Byte

Example:

```
TH1 = FFH
TL1 = 00H
```

Together:

```
FF00H
```

---

3. TMOD Register (Timer Mode Register)

This register selects:

- Timer 0 or Timer 1
- Timer mode
- Timer or Counter operation

Example:

```
MOV TMOD,#10H
```

Meaning:

```
Timer 0
Mode 1 (16-bit timer)
```

Think of TMOD as the **settings menu** of the timer.

01H → Timer 0 Mode 1
02H → Timer 0 Mode 2
03H → Timer 0 Mode 3

10H → Timer 1 Mode 1
20H → Timer 1 Mode 2
30H → Timer 1 Mode 3

---

4. TCON Register (Timer Control Register)

This register starts, stops, and monitors timers.

Important bits:

Bit	Function
TR0	Start/Stop Timer 0
TF0	Timer 0 Overflow Flag
TR1	Start/Stop Timer 1
TF1	Timer 1 Overflow Flag

Example:

```
SETB TR0
```

Means:

```
Start Timer 0
```

Example:

```
CLR TR0
```

Means:

```
Stop Timer 0
```

What Does the Timer Actually Count?

Suppose:

```
TH0 = 00H
TL0 = 00H
```

Timer starts.

The value changes:

```
0000H
0001H
0002H
0003H
...
```

just like a stopwatch counting:

```
0
1
2
3
4
...
```

When it reaches:

```
FFFFH
```

and tries to count again:

```
FFFFH + 1
```

it overflows and becomes:

```
0000H
```

The overflow flag:

```
TF0 = 1
```

is automatically set.

This tells the CPU: "The timer has completed its count."

Overflow happens when a timer has reached its maximum count and tries to count one more time.

Example

A 16-bit timer can count up to:

```
FFFFH
```

When the next pulse arrives:

```
FFFFH + 1
```

the timer cannot store a larger value, so it wraps around to:

```
0000H
```

This is called **overflow**.

```
FFFDH
FFFEH
FFFFH
  ↓
0000H  ← Overflow
```

At that moment, the 8051 automatically sets an overflow flag:

```
TF0 = 1   (for Timer 0)
TF1 = 1   (for Timer 1)
```

The CPU can check this flag and know:

"The timer has completed its count."

The ON/OFF switch and warning indicators.

TMOD Register

Used to select:

- Timer mode
- Timer or counter operation

TCON Register

Controls:

- Starting timer
- Stopping timer
- Overflow status

Timer Modes

Mode	Description
Mode 0	13-bit timer
Mode 1	16-bit timer
Mode 2	8-bit auto reload
Mode 3	Split timer mode

Timer Programming Example

```
MOV TMOD,#01H
MOV TH0,#0FCH
MOV TL0,#18H
SETB TR0
```

Explanation:

- Timer 0 selected.
- Timer started.
- Generates a time delay.

Applications of Timers

- Digital clocks
- Traffic lights
- Motor control
- Pulse generation

4.6 Interrupts

Definition

An interrupt is a signal that temporarily stops the current program and forces the microcontroller to execute another important task.

After completing the interrupt task, the CPU returns to the original program.

Why Interrupts Are Needed

Without interrupts:

- CPU continuously checks devices.

With interrupts:

- CPU responds only when needed.

This improves efficiency.

Interrupt Sources in 8051

Interrupt	Source
INT0	External Interrupt 0
Timer 0	Timer Overflow 0
INT1	External Interrupt 1
Timer 1	Timer Overflow 1
Serial Port	Serial Communication

Interrupt Enable Register (IE)

Used to enable or disable interrupts.

Example:

```
SETB EA
```

Operation:

- Enables all interrupts.

Advantages of Interrupts

- Faster response
- Better CPU utilization
- Supports real-time applications

Applications

- Alarm systems
- Industrial automation
- Traffic control systems
- Communication systems

4.7 Basics of Serial Programming

Definition

Serial communication transfers data one bit at a time through a single communication line.

It is used when two devices need to exchange information.

Examples of Serial Devices

- Computer
- Microcontroller
- GSM module - Global System for Mobile Communications, it allows a microcontroller to communicate through a mobile phone network.
- GPS module - Global Positioning System, it is receiving signals from satellites and determines location

Information provided

- Latitude
- Longitude
- Speed

- Time
- Altitude

Example

Vehicle tracking system:

```
GPS Satellite
      ↓
GPS Module
      ↓
Microcontroller
      ↓
Display/Internet
```

Result:

```
Vehicle Location:
Latitude: -6.7924
Longitude: 39.2083
```

- Bluetooth module - allows wireless communication over short distances.

Why Are These Called Serial Devices?

Because they communicate with the microcontroller **one bit at a time** through serial communication.

Serial Communication Types

Simplex

One-way communication.

Example:

- Television broadcast

Half Duplex

Communication in both directions but not simultaneously.

Example:

- Walkie-talkie

Full Duplex

Communication in both directions simultaneously.

Example:

- Mobile phone communication

Serial Port of 8051

The 8051 uses:

- TXD (Transmit Data)
- RXD (Receive Data)

Located on Port 3:

Pin	Function
P3.0	RXD
P3.1	TXD

Serial Communication Registers

SBUF

Stores transmitted or received data.

SCON

Controls serial communication modes.

Example of Data Transmission

```
MOV SBUF,#'A'
```

Operation:

- Character A is transmitted through TXD.

Applications of Serial Communication

- Computer interfacing
- Wireless communication
- Embedded systems networking
- Industrial monitoring systems

Summary

After studying this unit, you should be able to:

✓ Explain addressing of RAM and I/O devices

✓ Program input and output ports

✓ Perform bit manipulation operations

✓ Explain timers and counters in the 8051

✓ Describe interrupt operation and applications

✓ Explain serial communication and its importance

✓ Write simple programs using I/O ports, timers, interrupts, and serial communication