Programmable Logic Controllers and Microcontrollers (5th Sem)

ज्ञानम परमं बलम

Electrical Engineering Department,
Govt. Polytechnic Panchkula

Table of Contents

CHAPTER –1

Introduction to PLC

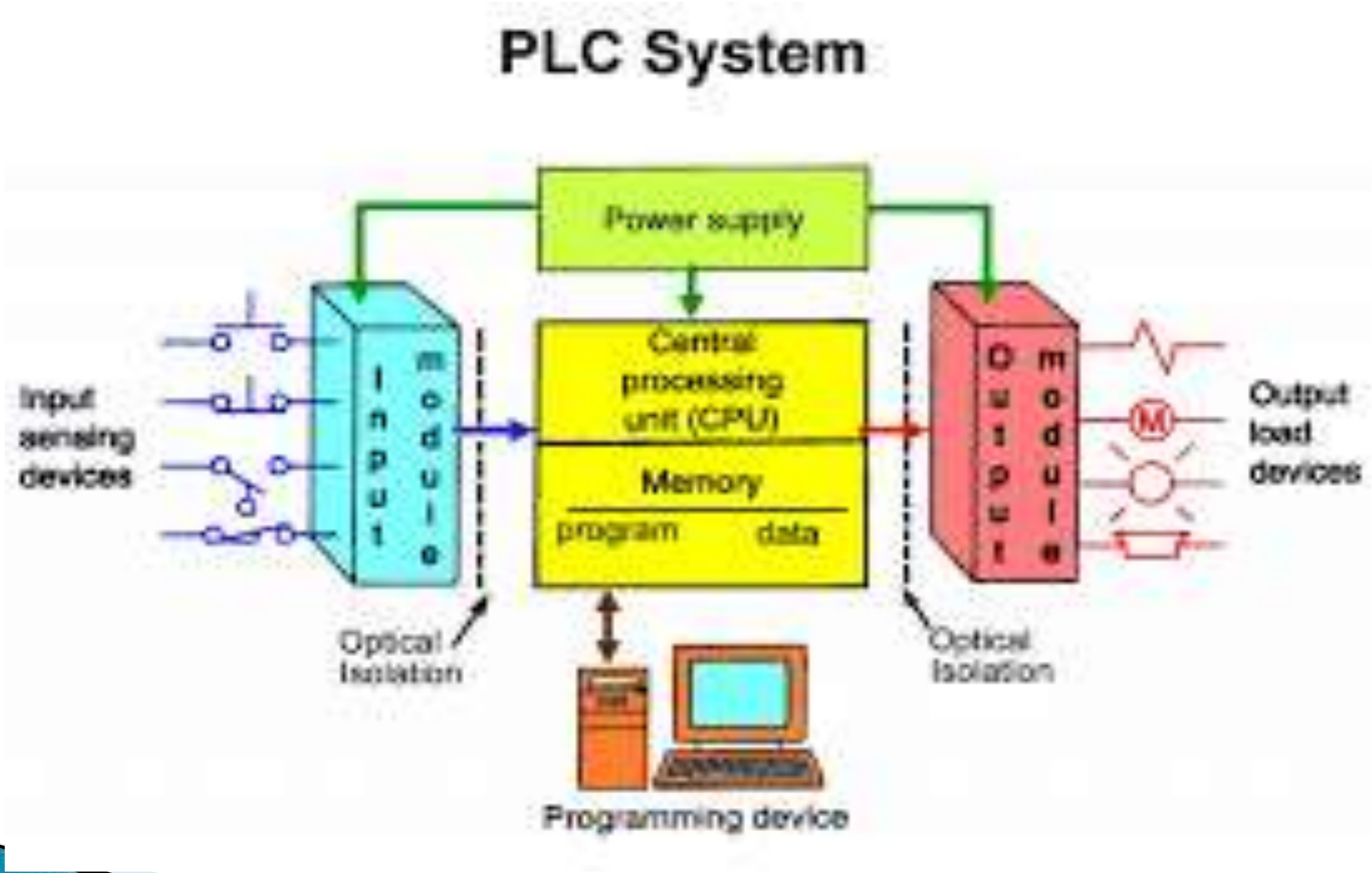
Concept of PLC

- A PROGRAMMABLE LOGIC CONTROLLER (**PLC**) is an industrial computer **control** system that continuously monitors the state of input devices and makes decisions based upon a custom **program** to **control** the state of output devices.

  Their primary goal of PLC is

- **To eliminate the high costs associated with inflexible, relay-controlled systems**

Building Blocks of PLC



Basic PLC Elements

- The basic elements of a PLC include **input modules or points**, a **Central Processing Unit (CPU)**, **output modules or points**, and a **programming device.**
- The type of input modules or points used by a PLC depend upon the types of input devices used. Some input modules or points respond to digital inputs, also called discrete inputs, which are either on or off. Other modules or inputs respond to analog signals.

PLC Input Circuit

- The primary function of a PLC's input circuitry is to convert the signals provided by these various switches and sensors into logic signals that can be used by the CPU.
- The CPU evaluates the status of inputs, outputs, and other variables as it executes a stored program.
- The CPU then sends signals to update the status of outputs.

Programming Device

- The programming device is used to enter or change the PLC’s program or to monitor or change stored values.
- Once entered, the program and associated variables are stored in the CPU.
- In addition to these basic elements, a PLC system may also incorporate an operator interface device of some sort to simplify monitoring of the machine or process.

Difference between relay and PLC

- The **difference between** a **PLC** and **relay** logic is that a **PLC** is a programmable device where as **relay** logic is a network of hardwired electrical devices.
- Both a **PLC** and **relay** logic can perform logical computation, but a **PLC** does it using a microprocessor and **relay** logic does it using electric circuits

Limitations of Relays

- Prior to PLCs, many control tasks were performed by contactors, control relays and other electromechanical devices. This is often referred to as **hard-wired control.**

- Circuit diagrams had to be designed, electrical components specified and installed, and wiring lists created. If an error was made, the wires had to be reconnected correctly. A change in function or system expansion required extensive component changes and rewiring.
- Requires periodic maintenance and testing.
- Relay operation can be affected due to ageing of the components and dust, pollution resulting in spurious trips
- Operation speed for an electromagnetic relays is limited by the mechanical inertia of the component

Advantages of PLC over Electromagnetic Relays

- Smaller physical size than hard-wire solutions.
- Easier and faster to make changes.
- PLCs have integrated diagnostics and override functions.
- Diagnostics are centrally available.
- Applications can be immediately documented.
- Applications can be duplicated faster and less expensively.

PLC Size

- 1. SMALL
- it covers units with up to 128 I/O's and memories up to 2 Kbytes.
- 2. MEDIUM
- They have up to 2048 I/O's and memories up to 32 Kbytes.
- 3. LARGE
- They have up to 8192 I/O's and memories up to 750 Kbytes.

PLC Programming Languages

There are only 5 languages that are considered to be standard languages for use on PLCs, according to IEC section 61131-3.

- Ladder Diagram (LD)
- A sequential function chart
- Function Block Diagram
- Instruction List
- Structured Text

Ladder Diagram

- Ladder Diagram is the oldest PLC language. This graphical programming language was modeled from relay logic to allow engineers and electricians to transition smoothly into programming PLCs.
- Within Ladder, rungs and rails represent the real world electrical connections. Specifically, the vertical “rails” represent the supply power of the device while the rungs that are connected to the rails are equal to the amount of control circuits.

Sequential Function Charts

- A sequential function chart is a graphical programming language that mimics a flow chart. You use steps and transitions to get output.
- Steps are functions within the program and house events that are activated based on state and other specified conditions.
- Transitions are instructions based on true/false values that move you from one step to another.
- Branches are used to initiate multiple steps at a time. The branches act like threads where functions can run concurrently.
- All of these steps, transitions, and branches are housed in a series of scripts that execute in a procedural manner. The visual nature of the language allows users to monitor processes that both heavily use conditional logic and run parallel instructions. PLCs that are prone to suffering from bottlenecks can be more intuitively maintained and troubleshooted using the chart to follow the logic of the program.

Function Block Diagram

- Block based programming languages are a type of graphical language that minimizes code into blocks, which allows for a simple way to create executable commands.
- FBD in particular describes a function between inputs and outputs that are connected by connection lines. The logic of the inputs and outputs are stored in blocks. The blocks are programmed onto sheets and the PLC scans these sheets in order or by specified connections between blocks, much like procedural languages.
- The I/O focus mirrors that of ladder logic. Yet, the code that the blocks contain allow engineers to develop more complex batch control tasks among other repeatable tasks.

Instruction List

- This is the PLC’s equivalent to assembly language. This gives you immediate access to the machine itself, which allows you to write code that is compressed and fast. The code is represented in the manner that the language’s name suggests: in a list of commands.
- Structured Text is a high level language designed to program PLCs. This is essentially the C++ of the PLC world. Any PLC that requires complex data handling will most likely use ST.

Structured Text

- Structured Text is a high level language designed to program PLCs. This is essentially the C++ of the PLC world. Any PLC that requires complex data handling will most likely use ST.

Advantages of PLCs

- Less wiring.
- Wiring between devices and relay contacts are done in the PLC program.
- Easier and faster to make changes.
- Trouble shooting aids make programming easier and reduce downtime.
- Reliable components make these likely to operate for years before failure.

PLC Manufacturer/Brands

AMERICAN

1. Allen Bradley
2. Gould Modicon
3. Texas Instruments
4. General Electric
5. Westinghouse
6. Cutter Hammer
7. Square D

EUROPEAN

1. Siemens
2. Klockner & Mouller
3. Festo
4. Telemechanique

JAPANESE

1. Toshiba
2. Omron
3. Fanuc
4. Mitsubishi

Applications of PLC

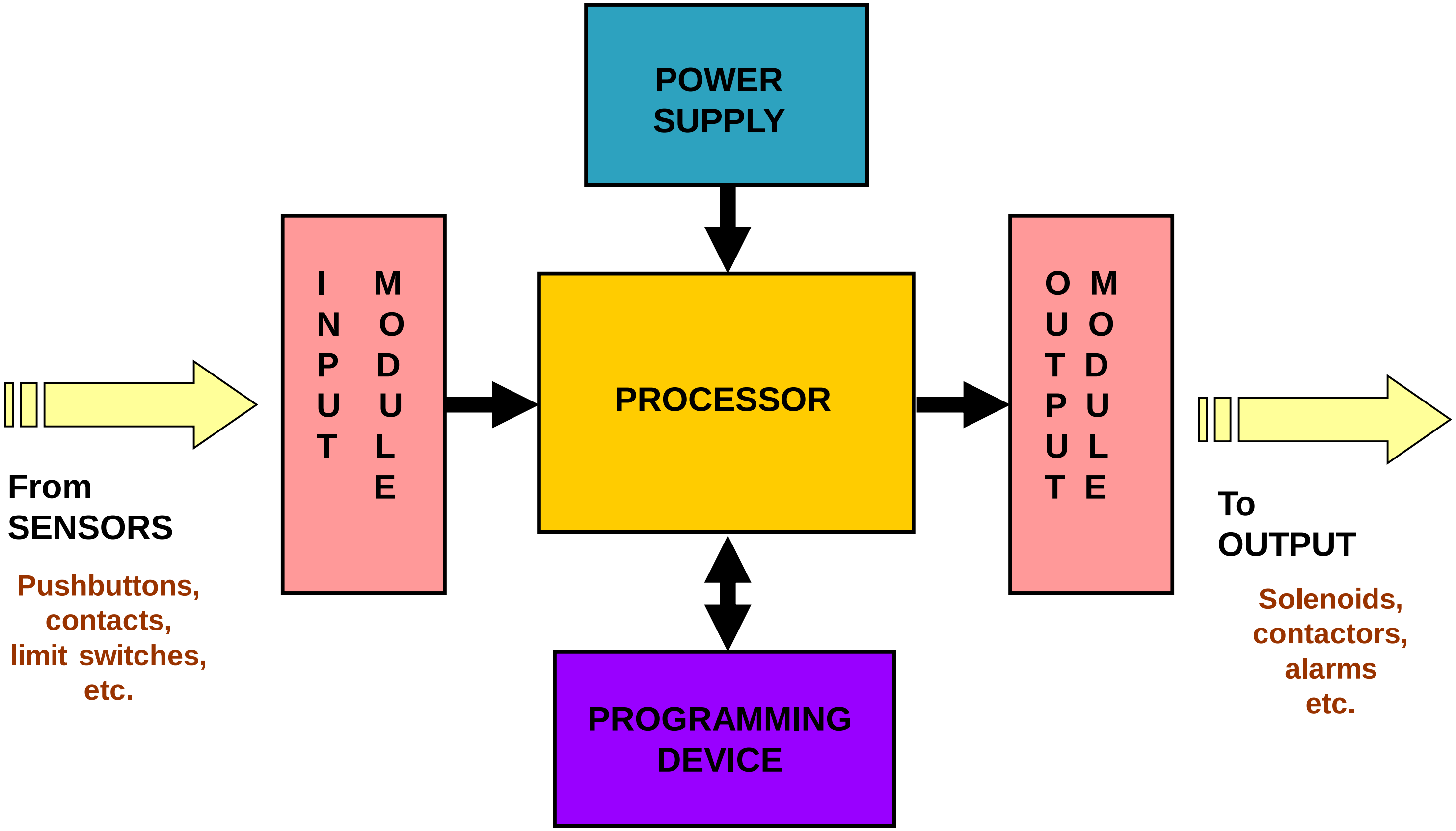
- Manufacturing / Machining
- Food / Beverage
- Metals
- Power
- Mining
- Petrochemical / Chemical

CHAPTER-2

Working of PLC

Prepared by Mrs. Alka Kalra

Major Components of a Common PLC



POWER SUPPLY

INPUT MODULE

PROCESSOR

OUTPUT MODULE

PROGRAMMING DEVICE

From SENSORS

Pushbuttons, contacts, limit switches, etc.

To OUTPUT

Solenoids, contactors, alarms etc.

PLC Principle/Operation

- Read all field input devices via the input interfaces, execute the user program stored in application memory, then, based on whatever control scheme has been programmed by the user, turn the field output devices on or off, or perform whatever control is necessary for the process application.

- This process of sequentially reading the inputs, executing the program in memory, and updating the outputs is known as scanning.

PLC Operation

PHASE 1 – Input Status scan

- A PLC scan cycle begins with the CPU reading the status of its inputs.

PHASE 2– Logic Solve/Program Execution

- The application program is executed using the status of the inputs

PHASE 3– Logic Solve/Program Execution

- Once the program is executed, the CPU performs diagnostics and communication tasks

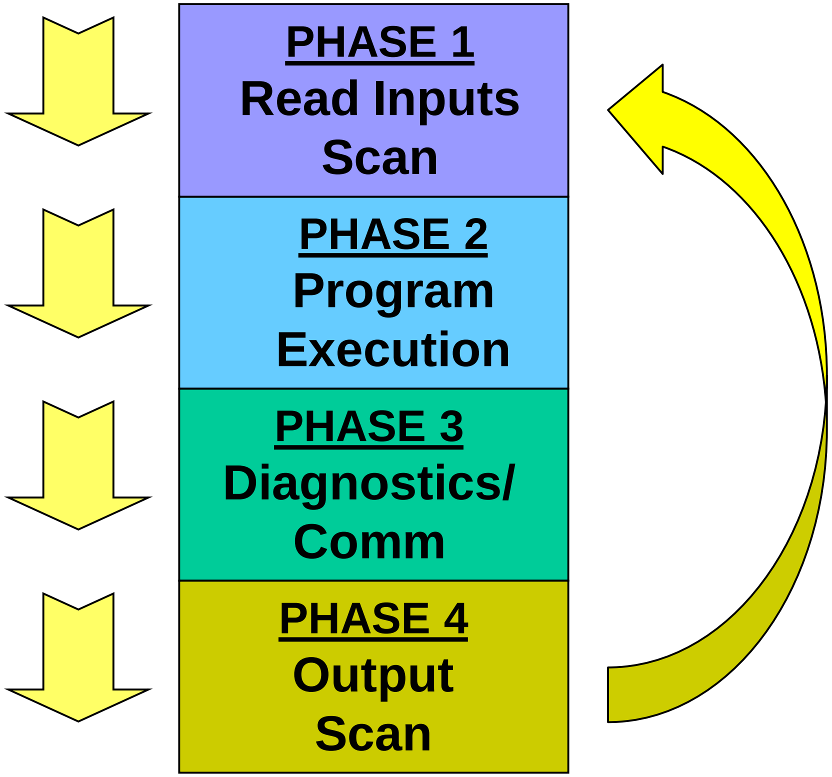
PHASE 4 - Output Status Scan

- An output status scan is then performed, whereby the stored output values are sent to actuators and other field output devices. The cycle ends by updating the outputs.

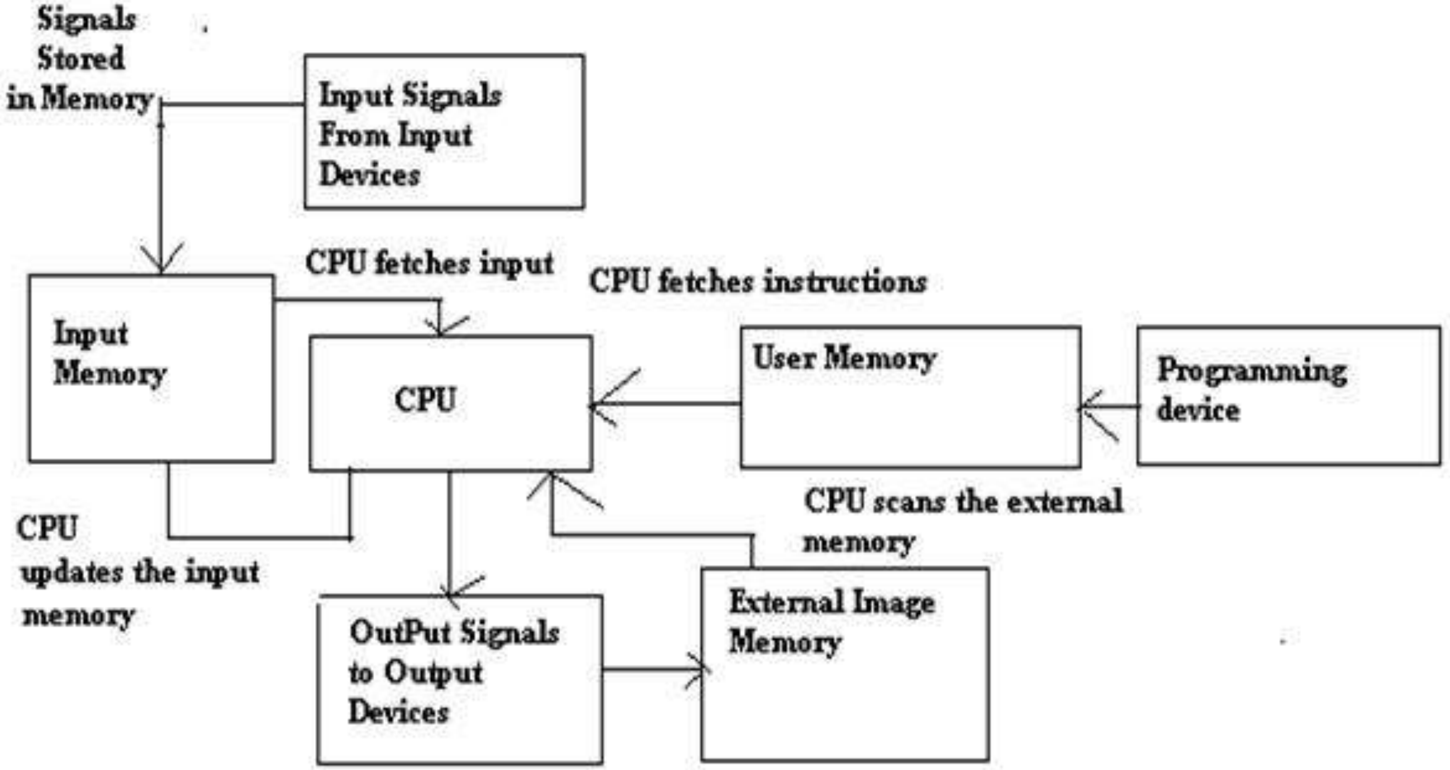
PLC Operation

- As soon as Phase 4 are completed, the entire cycle begins again with Phase 1 input scan.
- The time it takes to implement a scan cycle is called SCAN TIME.
  - The scan time composed of the program_scan time, which is the time required for solving the control program, and the I/O update time, or time required to read inputs and update outputs.
- The program scan time generally depends on the amount of memory taken by the control program and type of instructions used in the program. The time to make a single scan can vary from 1 ms to 100 ms

While the PLC is running, the scanning process includes the following four phases, which are repeated continuously as individual cycles of operation:



PLC WORKING


Signals
Stored
in Memory
Input Signals
From Input
Devices
CPU fetches input
CPU fetches instructions
Input
Memory
CPU
User Memory
Programming
device
CPU scans the external
memory
CPU
updates the input
memory
OutPut Signals
to Output
Devices
External Image
Memory

PLC WORKING

- The input sources convert the real-time analog electric signals to suitable digital electric signals and these signals are applied to the PLC through the connector rails.
- These input signals are stored in the PLC external image memory in locations known as bits. This is done by the CPU
- The control logic or the program instructions are written onto the programming device through symbols or through mnemonics and stored in the user memory.

PLC WORKING

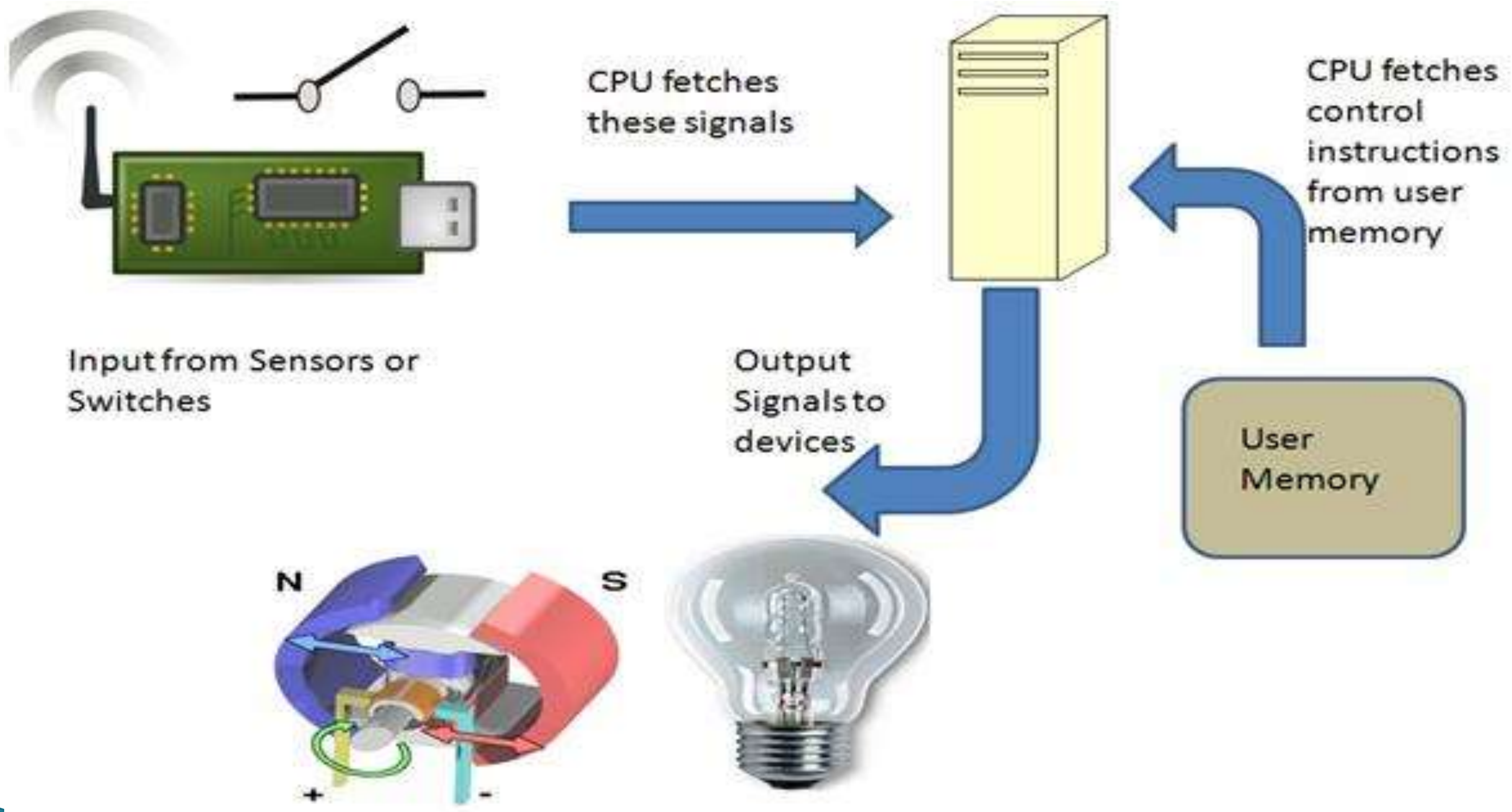
- The CPU fetches these instructions from the user memory and executes the input signals by manipulating, computing, processing them to control the output devices.

- The execution results are then stored in the external image memory which controls the output drives.

- The CPU also keeps a check on the output signals and keeps updating the contents of the input image memory according to the changes in the output memory.

- The CPU also performs internal programming functions like setting and resetting of the timer, checking the user memory.

PLC Working



Input from Sensors or Switches

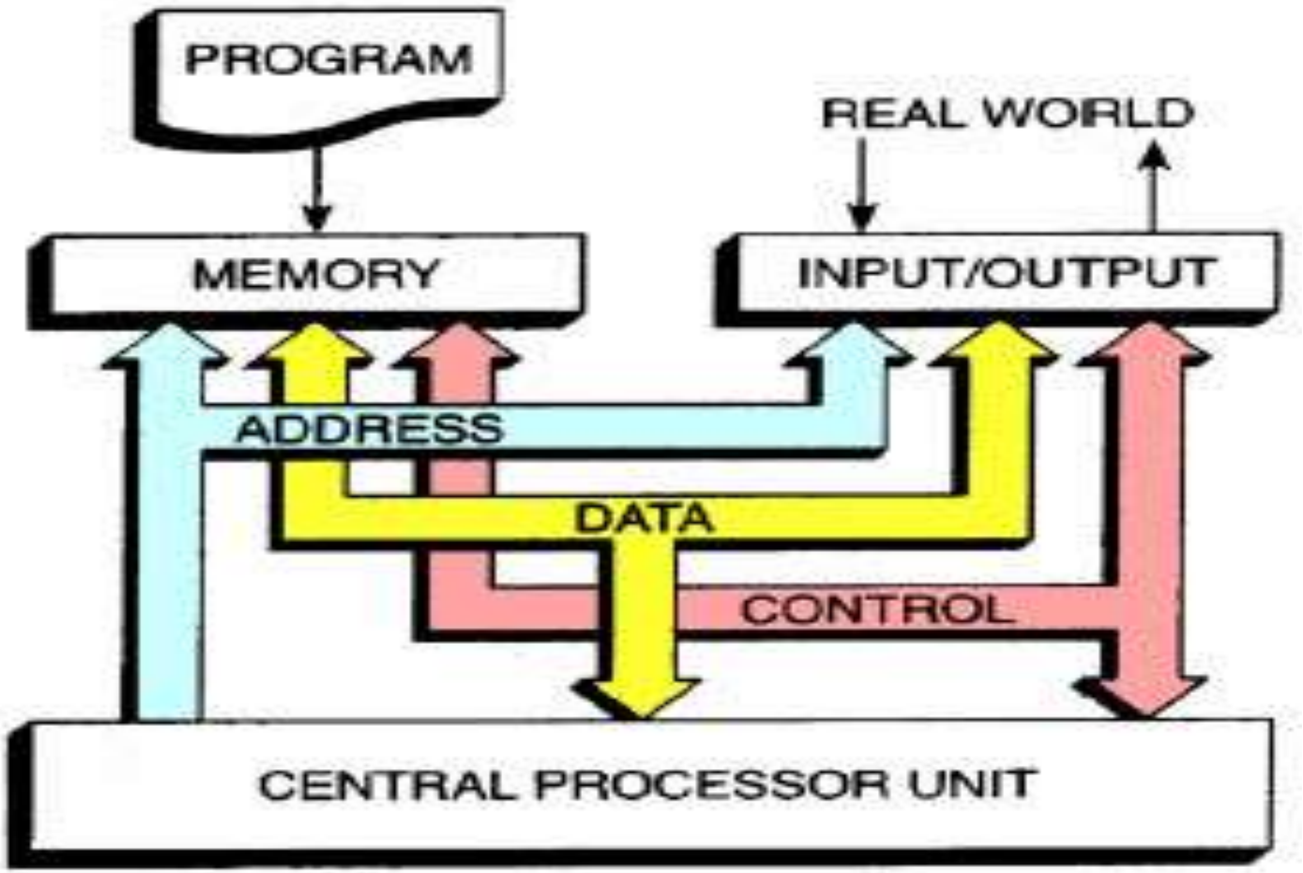
CPU fetches these signals

CPU fetches control instructions from user memory

Output Signals to devices

User Memory

PLC Architecture



PLC Modules

- PLCs are capable of monitoring the inputs continuously from sensors and producing the output decisions to operate the actuators based on the program. Every PLC system needs at least these three modules:
- CPU Module
- Power Supply Module
- One or more I/O Module

CPU/Procssor module

- CPU module consists of a central processor and its memory. The Processor is responsible for doing all the necessary computations and data processing by accepting the inputs and producing appropriate outputs.

Function of CPU/Processor

- The main function of the microprocessor is to analyze data coming from field sensors through input modules, make decisions based on the user's defined control program and return signal back through output modules to the field devices.
- Field sensors(Input): switches, flow, level, pressure, temp. transmitters, etc.
- Field output devices: motors, valves, solenoids, lamps, or audible devices.

Memory Structures

- These PLCs use retentive memory to save user programs and data when the power supply breaks or fails and to resume the execution of a user program ones the power is restored.
- Thus, these PLCs do not need any use of a keyboard or monitor for re programming the processor each time.
- The retentive memory can be implemented with the use of long-life batteries, EEPROM modules and flash memory methods.

Memory Structures/Designs

- **VOLATILE**.
- A volatile memory is one that loses its stored information when power is removed.
- Even momentary losses of power will erase any information stored or programmed on a volatile memory chip.
- Common Type of Volatile Memory
- **RAM. R**andom **A**ccess **M**emory(Read/Write)

RAM

- Read/write indicates that the information stored in the memory can be retrieved or read, while write indicates that the user can program or write information into the memory.
- The words **random access** refer to the ability of any location (address) in the memory to be accessed or used. Ram memory is used for both the user memory (ladder diagrams) and storage memory in many PLC’s.

CMOS-RAM Memory

- The CMOS-RAM (**C**omplimentary **M**etal **O**xide **S**emiconductor) is probably one of the most popular. CMOS-RAM is popular because it has a very low current drain when not being accessed (15microamps), and the information stored in memory can be retained by as little as 2Vdc.

- RAM memory must have battery backup to retain or protect the stored program.

Memory Design

- **NON-VOLATILE**
- Has the ability to retain stored information when power is removed, accidentally or intentionally. These memories do not require battery back-up.
- Common Type of Non-Volatile Memory
- **ROM, R**ead **O**nly **M**emory
- Read only indicates that the information stored in memory can be read only and cannot be changed. Information in ROM is placed there by the manufacturer for the internal use and operation of the PLC.

Non-Volatile Memory

- **PROM, P**rogrammable **R**ead **O**nly **M**emory
- Allows initial and/or additional information to be written into the chip.
- PROM may be written into only once after being received from the PLC manufacturer; programming is accomplish by pulses of current.
- The current melts the fusible links in the device, preventing it from being reprogrammed. This type of memory is used to prevent unauthorized program changes.

Non Volatile Memory

- **EPROM, E**rasable **P**rogrammable **R**ead **O**nly **M**emory
- Ideally suited when program storage is to be semi-permanent or additional security is needed to prevent unauthorized program changes.
- The EPROM chip has a quartz window over a silicon material that contains the electronic integrated circuits. This window normally is covered by an opaque material, but when the opaque material is removed and the circuitry exposed to ultra violet light, the memory content can be erased.
- The EPROM chip is also referred to as **UVPROM.**

EEPROM Memory

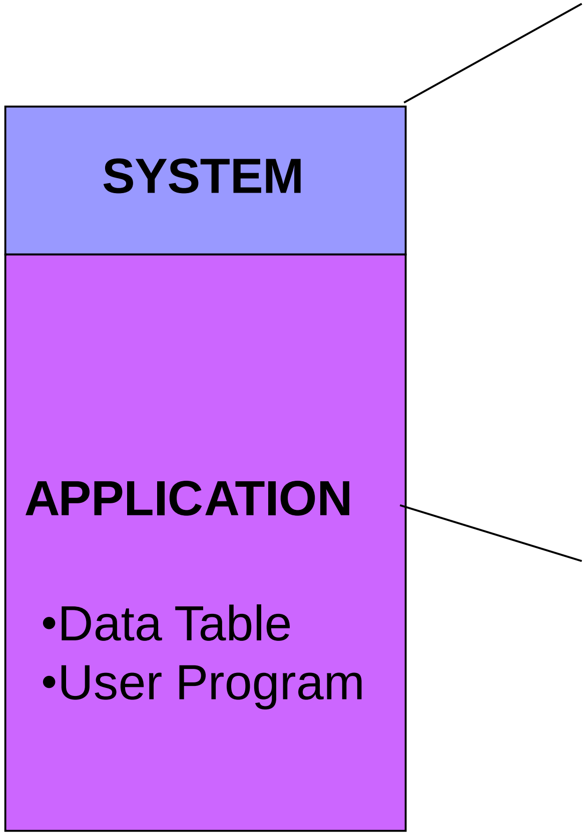
Electrically **E**rasable **P**rogrammable **R**ead **O**nly **M**emory

- Also referred to as E^2PROM, is a chip that can be programmed using a standard programming device and can be erased by the proper signal being applied to the erase pin.
- EEPROM is used primarily as a non-volatile backup for the normal RAM memory. If the program in RAM is lost or erased, a copy of the program stored on an EEPROM chip can be down loaded into the RAM.

Memory

- Processor module includes both ROM and RAM memories.
  - ROM (Program Memory/System Memory) contains the operating system, driver and application programs,
  - RAM (Data Memory/Application Memory)stores user-written programs and working data.
- The program information or the control logic is stored in the user memory or the program memory from where the CPU fetches the program instructions.
- The input and output signals and the timer and counter signals are stored in the input and output external image memory respectively.

Memory Map Organization



•System memory includes an area called the EXECUTIVE, composed of permanently-stored programs that direct all system activities, such as execution of the users control program, communication with peripheral devices, and other system activities.
•The system memory also contains the routines that implement the PLC’s instruction set, which is composed of specific control functions such as logic, sequencing, timing, counting, and arithmetic.
•System memory is generally built from read-only memory devices.

•The application memory is divided into the data table area and user program area.
•The data table stores any data associated with the user’s control program, such as system input and output status data, and any stored constants, variables, or preset values. The data table is where data is monitored, manipulated, and changed for control purposes.
•The user program area is where the programmed instructions entered by the user are stored as an application control program.

BUS or Rack

- In some modular PLCs bus or rack is provided in the backplane of the circuit into which all the modules like CPU and other I/O modules are plugged to the corresponding slots.
- This bus enables the communication between CPU and I/O modules to send or receive the data.
- This communication is established by addressing the I/O modules according to the location from CPU module along the bus.

BUS or Rack

- Suppose, if the input module is located in the second slot,
  - the address must be I2:1.0 (second slot first channel only as an example).
- Some buses provide necessary power to I/O module circuitry, but they do not provide any power to sensors and actuators connected to I/O modules

I/O Structures/Modules

- The I/O interface section of a PLC connects it to external field devices.
- The input and out modules of the programmable logic controller are used to connect the sensors and actuators to the system to sense the various parameters such as temperature, pressure and flow, etc.
- These I/O modules are of two types: digital or analog.

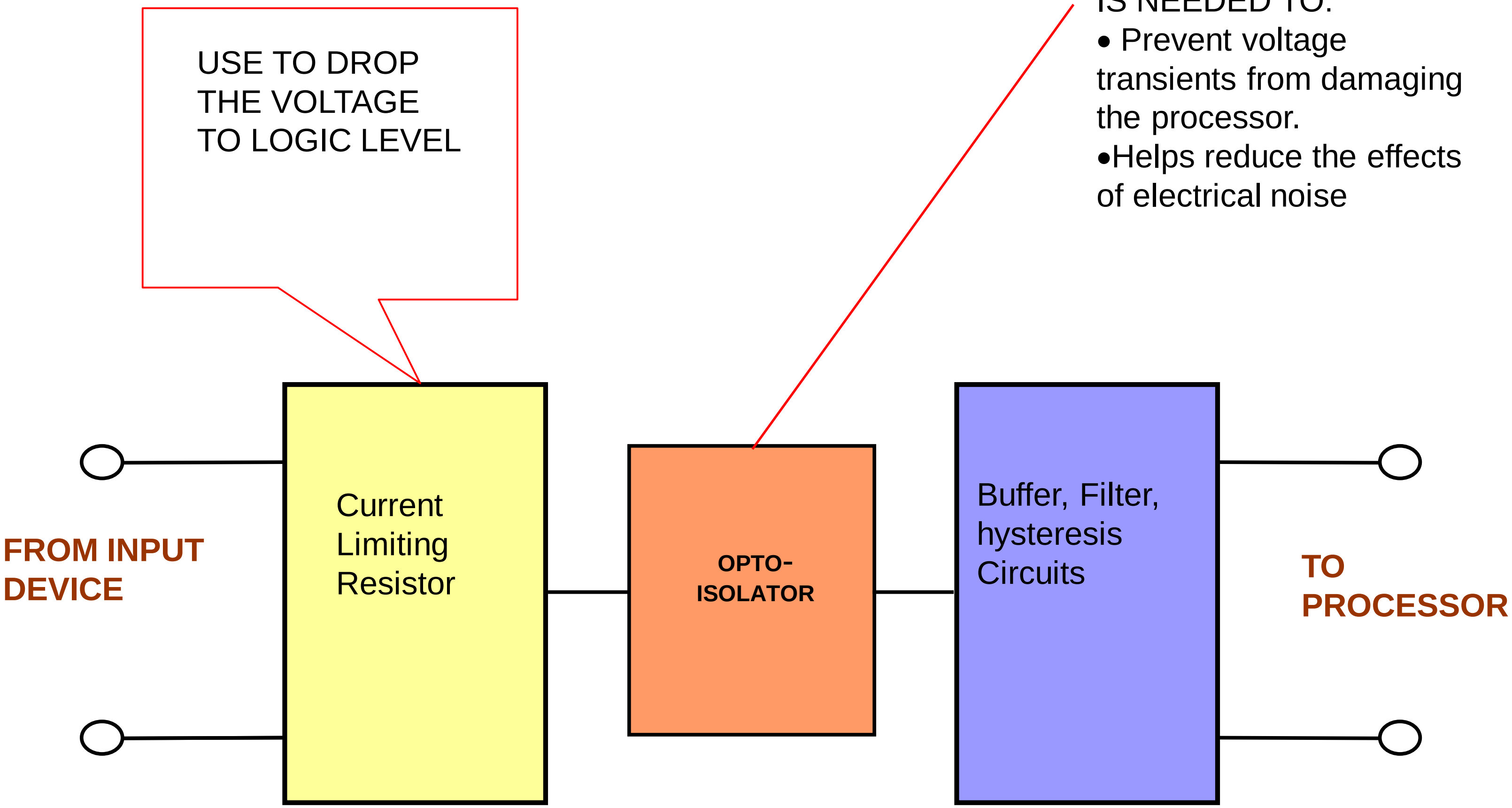
I/O interface

- The main purpose of the I/O interface is to condition the various signals received from or sent to the external input and output devices.

- Input modules converts signals from discrete or analog input devices to logic levels acceptable to PLC’s processor.

- Output modules converts signal from the processor to levels capable of driving the connected discrete or analog output devices.

I/O Structure/MODULE


USE TO DROP
THE VOLTAGE
TO LOGIC LEVEL

IS NEEDED TO:

- Prevent voltage transients from damaging the processor.
- Helps reduce the effects of electrical noise

FROM INPUT DEVICE

Current Limiting Resistor

OPTO-ISOLATOR

Buffer, Filter, hysteresis Circuits

TO PROCESSOR

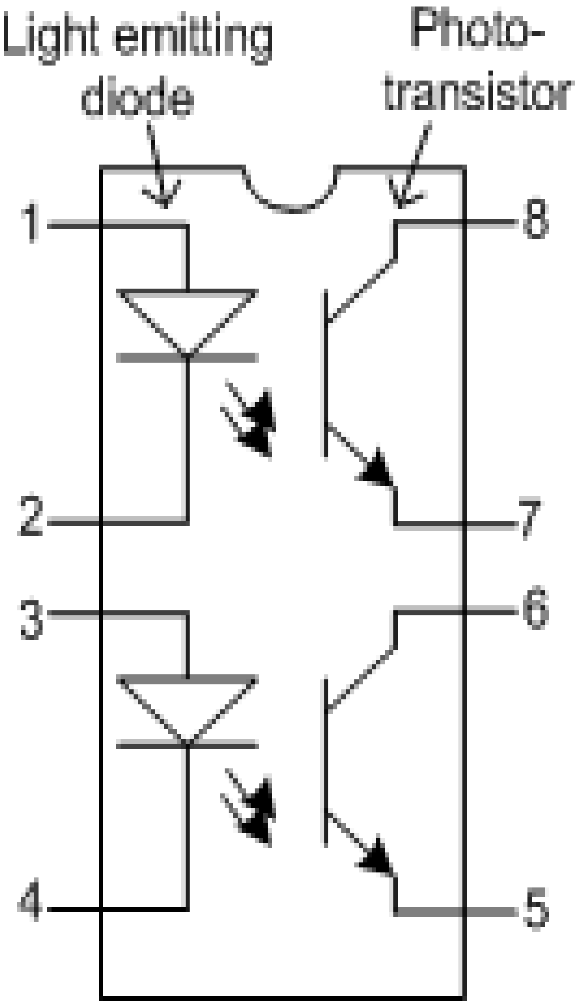


Figure 1.6a Dual-optocoupler IC in 8-pin DIP

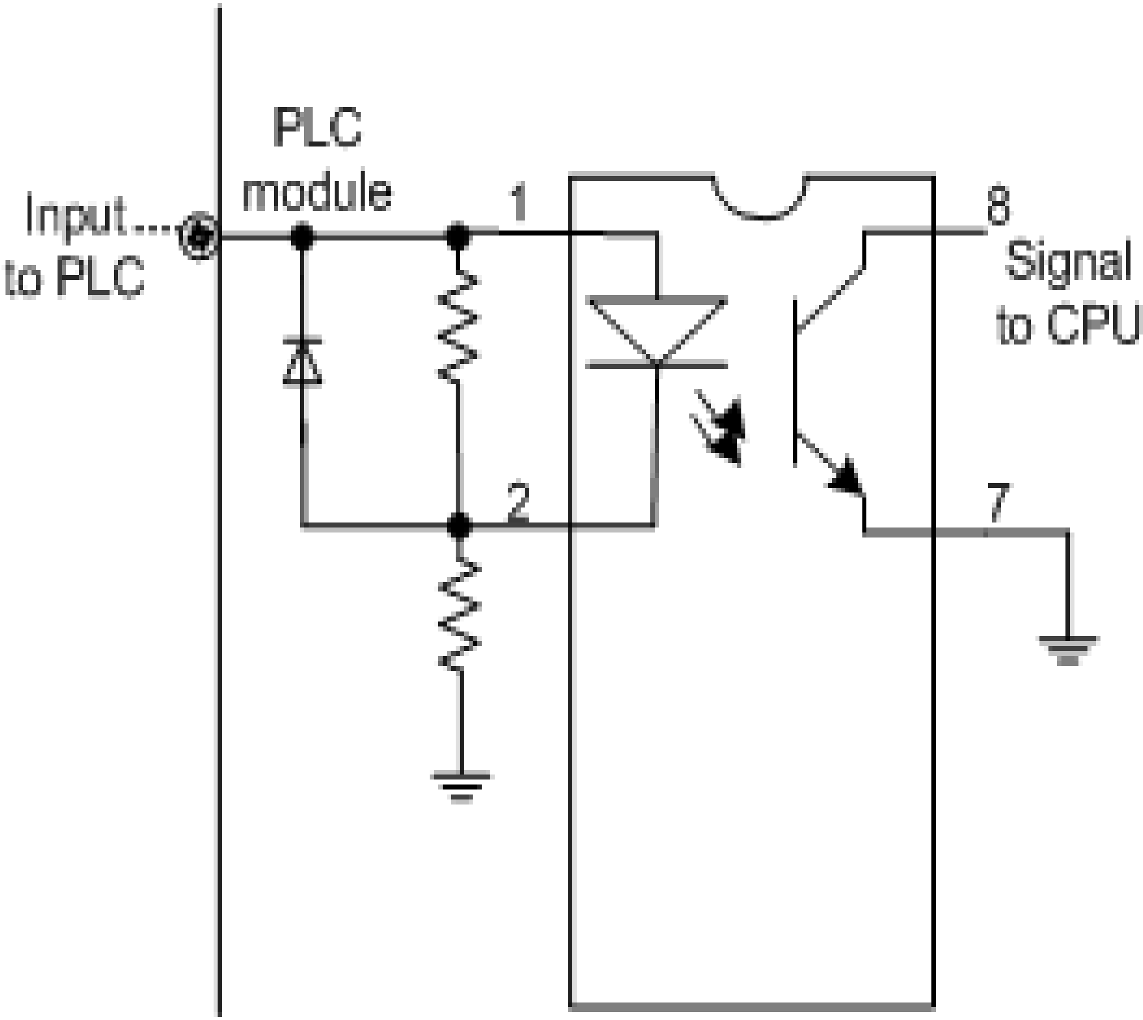
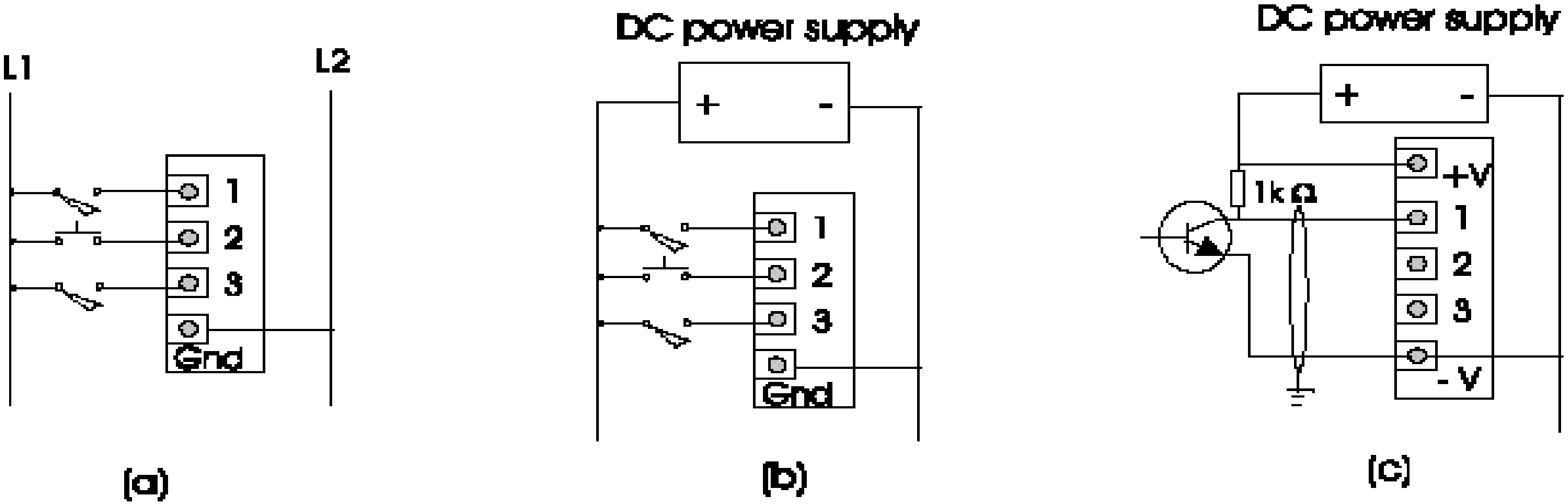


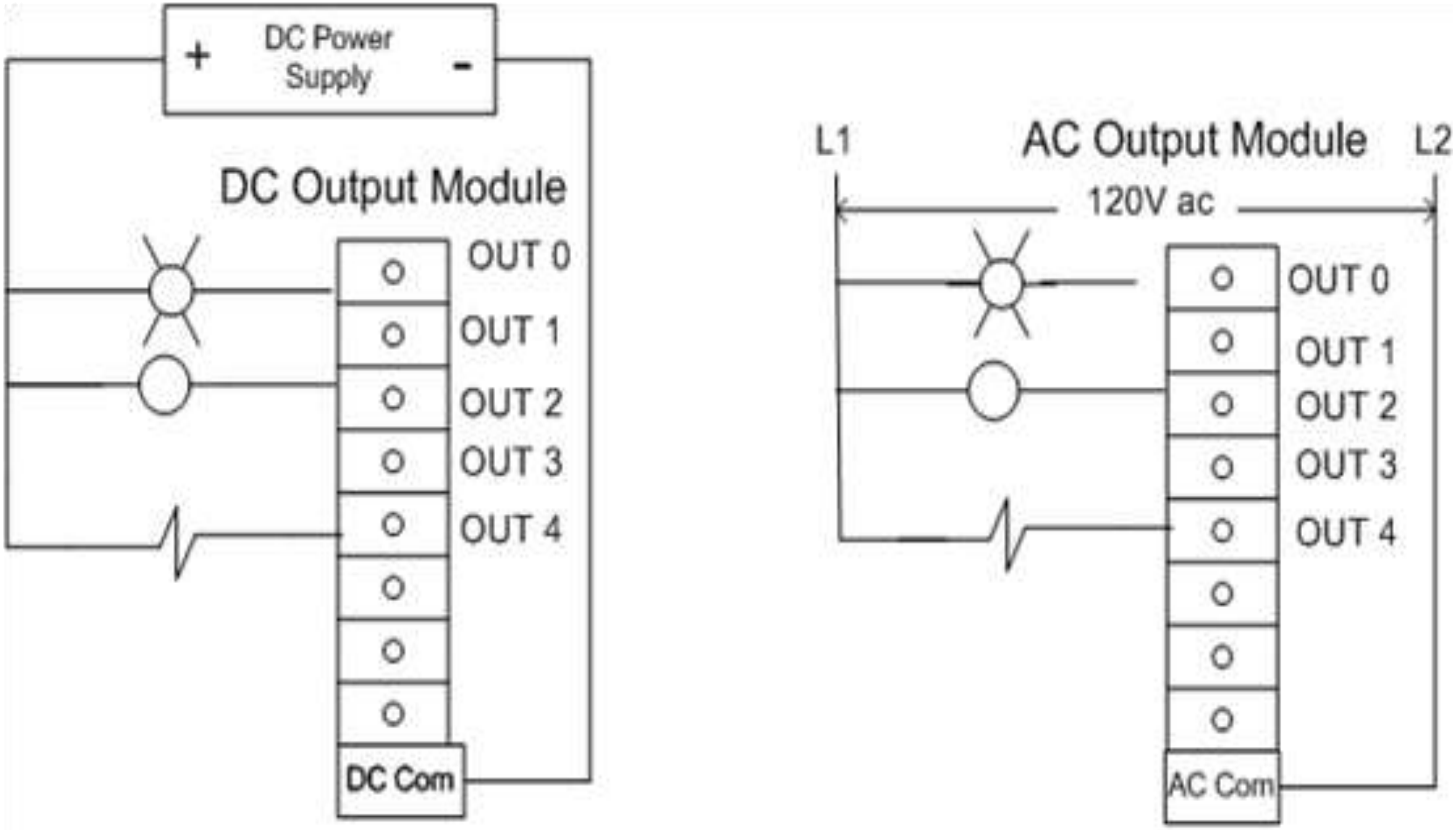
Figure 1.6b Basic DC input circuit

Input Connections



AC DC TTL

DC/AC Output Wiring Connections



DIFFERENT TYPES OF I/O Structures

Pilot Duty Outputs

- Outputs of this type typically are used to drive high-current electromagnetic loads such as solenoids, relays, valves, and motor starters.

These loads are highly inductive and exhibit a large current.

General - Purpose Outputs

- These are usually low- voltage and low-current and are used to drive indicating lights and other non-inductive loads.
- Noise suppression may or may not be included on this types of modules.

Discrete Inputs

- Circuits of this type are used to sense the status of limit switches, push buttons, and other discrete sensors.
- Noise suppression is of great importance in preventing false indication of inputs turning on or off because of noise.

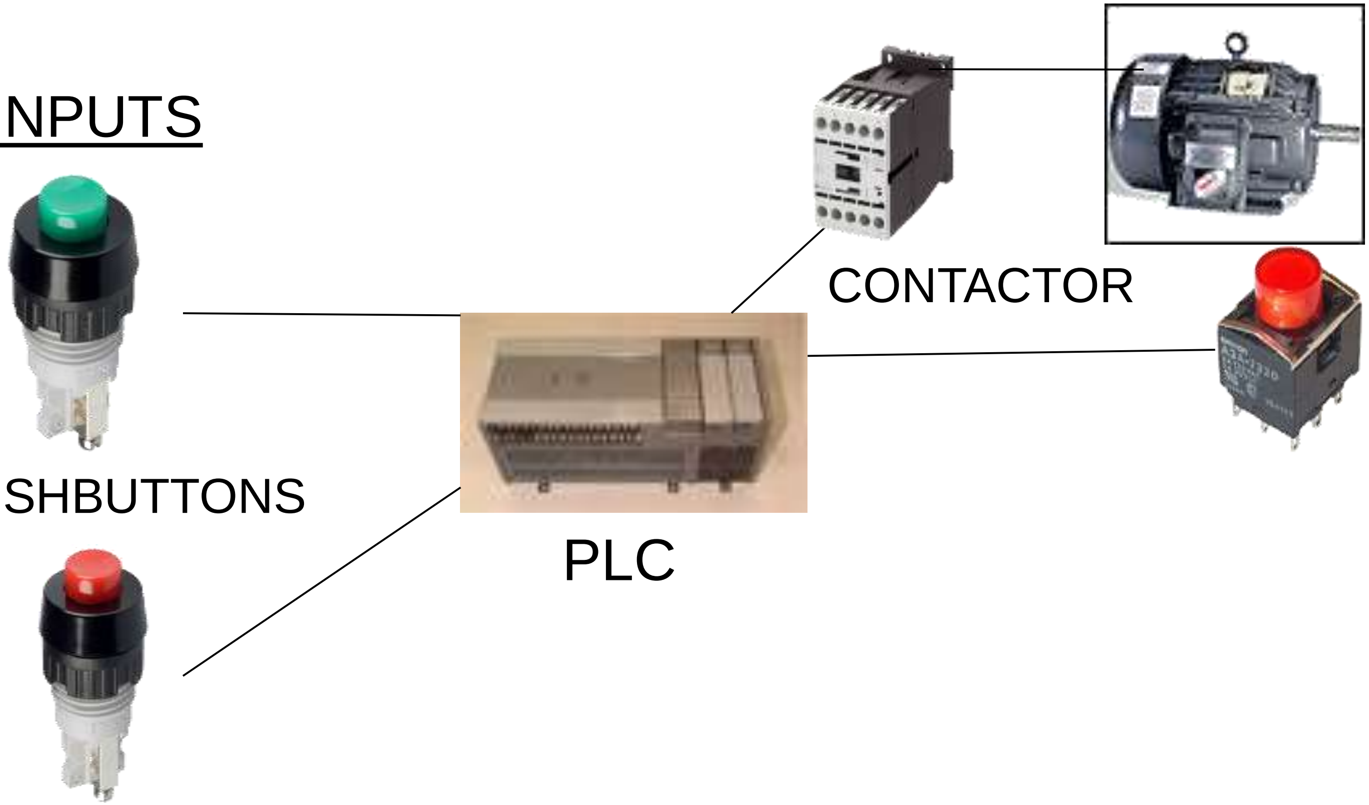
Analog I/O Circuits

- Circuits of this type sense or drive analog signals.
- Analog inputs come from devices, such as thermocouples, strain gages, or pressure sensors, that provide a signal voltage or current that is derived from the process variable.
- Standard Analog Input signals: 4-20mA; 0-10V
- Analog outputs can be used to drive devices such as voltmeters, X-Y recorders, servomotor drives, and valves through the use of transducers.
- Standard Analog Output signals: 4-20mA; 0-5V; 0-10V

Special - Purpose I/O

- Circuits of this type are used to interface PLCs to very specific types of circuits such as servomotors, stepping motors PID (proportional plus integral plus derivative) loops, high-speed pulse counting, resolver and decoder inputs, multiplexed displays, and keyboards.

- This module allows for limited access to timer and counter presets and other PLC variables without requiring a program loader.



INPUTS

PUSHBUTTONS

PLC

OUTPUTS

CONTACTOR

MOTOR

LAMP

PROGRAMMING DEVICE

- It is used to enter the desired program that will determine the sequence of operation and control of process equipment or driven machine.
- Also known as:
- Industrial Terminal (Allen Bradley)
- Program Development Terminal (General Electric)
- Programming Panel (Gould Modicon)
- Programmer (Square D)
- Program Loader (Idec-Izumi)
- Programming Console (Keyence / Omron)

PROGRAMMING DEVICE TYPES

- Various types of programming devices are used to enter, modify and troubleshoot a PLC program.
- Hand held unit with LED / LCD display
- Personal Computer (PC)
- Desktop type with a CRT display /Desktop Console
-

Hand held unit with LED / LCD display

- In the handheld programming device method, a proprietary device is connected to PLC through a connecting cable.
- This device consists of a set of keys that allows to enter, edit and dump the code into the PLC.
- These handheld devices consist of small display to make the instruction that has been programmed visible.
- These are compact and easy to use devices, but these handheld devices have limited capabilities.

Desktop consoles

- Desktop consoles are likely to have a visual display unit with a full keyboard and screen display.
- Keyboard and monitor are used for programming.
- Programming Unit communicated with PLC through serial or Parallel port.

Personal Computer(PC)

- PC is used for programming the PLC in conjunction with the software given by the manufacturer.
- By using this PC we can run the program in either online or offline mode, and can also edit, monitor, diagnose and troubleshoot the program of the PLC.
- The way of transferring the program to the PLC is shown in the above figure wherein the PC consists of program code corresponding to control application which is transferred to the PLC CPU via programming cable.
- . A major advantage of using a computer is that the program can be stored on the hard disk or a CD and copies can be easily made.

PLC Communications Ports Application/Use

Changing resident PLC programs - uploading/downloading from a supervisory controller (Laptop or desktop computer).

- Forcing I/O points and memory elements from a remote terminal.
- Linking a PLC into a control hierarchy containing several sizes of PLC and computer.
- Monitoring data and alarms, etc. via printers or Operator Interface Units (OIUs).

PLC Communications Standards

RS 232

Used in short-distance computer communications, with the majority of computer hardware and peripherals. Has a maximum effective distance of approx. 30 m at 9600 baud.

RS 422 / RS 485

Used for longer-distance links, often between several PCs in a distributed system.

RS 485 can have a maximum distance of about 1000 meters.

Local Area Network(LAN)

Local Area Network provides a physical link between all devices plus providing overall data exchange management or protocol, ensuring that each device can “talk” to other machines and understand data received from them.

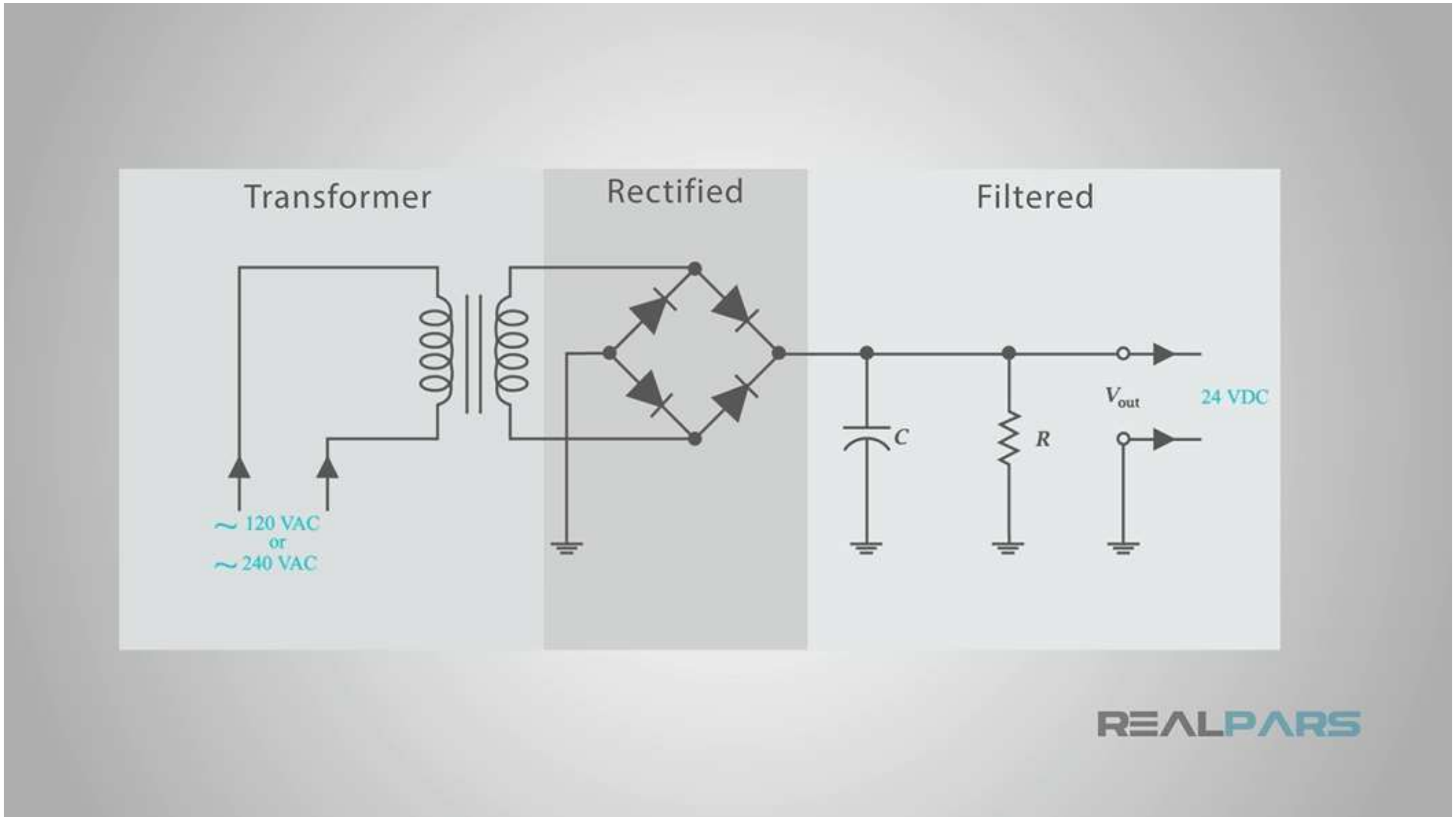
LANs provide the common, high-speed data communications bus which interconnects any or all devices within the local area.

LANs are commonly used in business applications to allow several users to share costly software packages and peripheral equipment such as printers and hard disk storage.

Power Supply Module

- A **PLC power supply** is the workhorse of the **PLC** system. It converts your line voltage, 120 or 240 volts AC, to a lower DC voltage, commonly 24 volts DC. This DC voltage is then sent into the rack to **power** the rest of the **PLC** components.
- The output 5V DC drives the computer circuitry, and in some PLCs 24DC on the bus rack drives few sensors and actuators.

PLC Power Supply



PLC Power Supply

- Line voltage is stepped down with a transformer, rectified to convert it to DC, filtered with capacitors, and protected during this process. All of this is packed into that small looking power supply.
- This DC voltage is used to power the rest of the PLC and components.
- The common current ratings for PLC’s are anywhere from 2 to 10 amps for smaller systems and up to 50 amps for larger, more powerful controllers.

Criteria for Selection a PLC/Specifications

Number of logical inputs and outputs

- This specifies the number of I/O devices that can be connected to the controller.

- There should be sufficient I/O ports to meet present requirements with enough spares to provide for moderate future expansion.

Specifications

- MEMORY CAPACITY
  The amount of memory required for a particular application is related to the length of the program and the complexity of the control system.
- Simple applications having just a few relays do not require significant amount of memory.
- Program length tend to expand after the system have been used for a while.
- It is advantageous to a acquire a controller that has more memory than is presently needed.

Specifications

OUTPUT-PORT POWER RATINGS

- Each output port should be capable of supplying sufficient voltage and current to drive the output peripheral connected to it.

Scan Time

- This is the speed at which the controller executes the relay ladder logic program. This variable is usually specified as the scan time per 1000 logic nodes and typically ranges from 1 to 200 milliseconds.

PLC Communications

Programmable Controllers and Networks

Dedicated Network System of Different Manufacturers

Manufacturer	**Network**
Allen-Bradley	Data Highway
Gould Modicon	Modbus
General Electric	GE Net Factory LAN
Mitsubishi	Melsec-NET
Square D	SY/NET
Texas Instruments	TIWAY

Specifications

- Communications Port
  - RS-232, **RS 422 / RS 485,LAN**
- Software
  - 1. Allen-Bradley – Rockwell Software RSLogix500
  - 2. Modicon - Modsoft
  - 3. Omron - Syswin
  - 4. GE-Fanuc Series 6 – LogicMaster6
  - 5. Square D- PowerLogic
  - 6. Texas Instruments – Simatic
  - 6. Telemecanique – Modicon TSX Micro

Practical Design Approach

A Detailed Design Process

1. Understand the process
2. Hardware/software selection
3. Develop ladder logic
4. Determine scan times and memory requirements

PLC Status Indicators

1. Power On
2. Run Mode
3. Programming Mode
4. Fault

Troubleshooting PLC System

1. Look at the process
2. PLC status lights
   - HALT - something has stopped the CPU
   - RUN - the PLC thinks it is OK (and probably is)
   - ERROR - a physical problem has occurred with the PLC
3. Indicator lights on I/O cards and sensors
4. Consult the manuals, or use software if available.
5. Use programming terminal / laptop.

Chapter 3

Ladder Diagram Programming

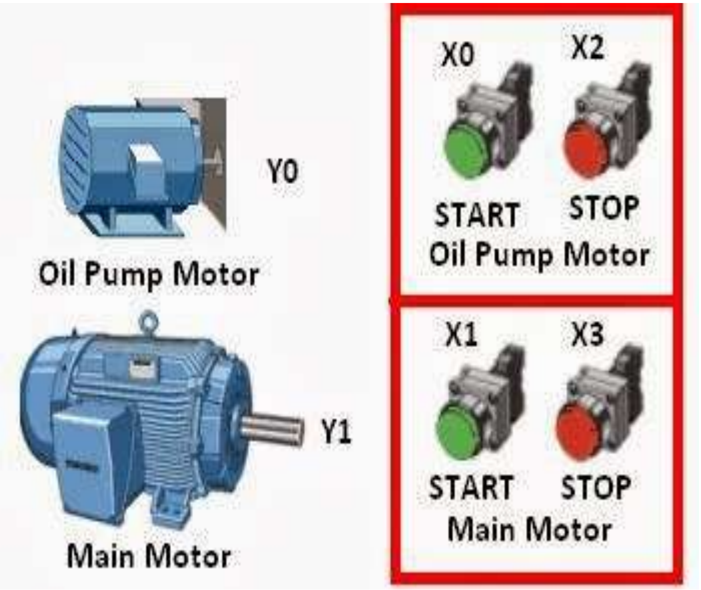
Basic Instruction Example-1

- <u>Problem Statement :</u>
- **Providing lubricant for the gear box before the lathe spindle starts to run which aims to ensure that the oil pump motor starts first and the main motor starts subsequently.**

- <u>Number of PLC Inputs Required</u>

- X0 – START pushbutton to Start Oil Pump Motor
- X1 – START pushbutton to Stop Main Motor
- X2 – STOP pushbutton to Stop Oil Pump Motor
- X3 – STOP pushbutton to Stop Main Motor

- <u>Number of PLC Outputs Required</u>
- Y0 – Oil Pump Motor
- Y1 – Main Motor

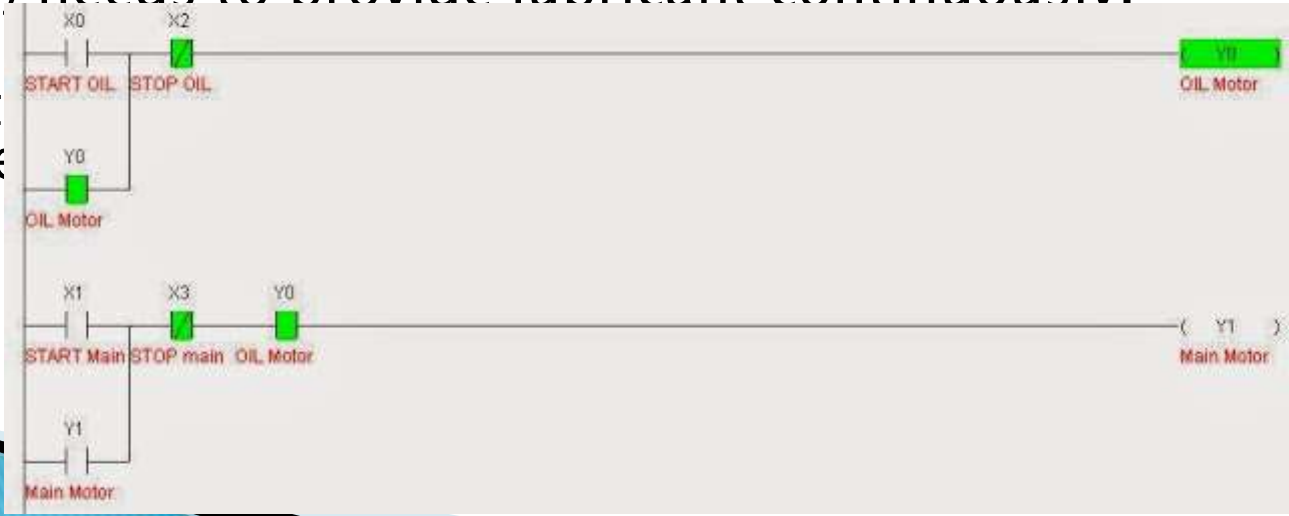


PLC Ladder Programming Description

- This program is a typical application of the conditional control circuit.

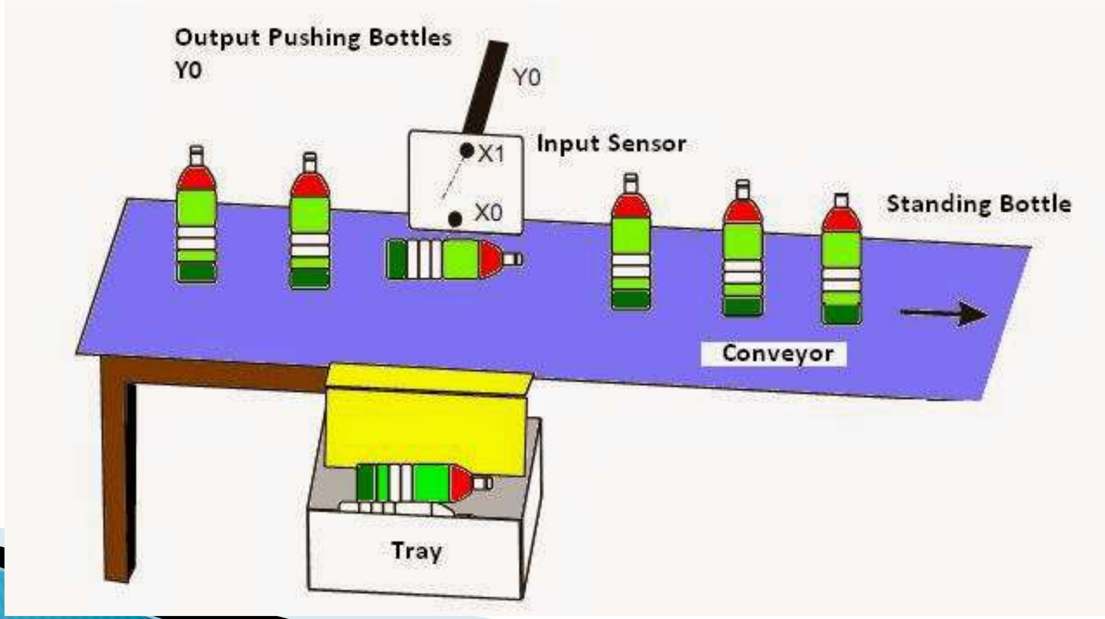
  Y0 = ON when Oil Pump START button is pressed.

  Therefore, the oil pump will start to provide lubricant for the gear box of main motor(Y1)
- Under the precondition of the operating state of the Oil pump, the main motor (Y1) will be ON when the Main motor START button is pressed.
- During the operation of main motor (Y1), oil pump (Y0) needs to provide lubricant continuously.
- STOP
  but stopped
  whe



Basic Instruction Example– 2

- Problem Statement: Detecting the standing bottles on the conveyor and pushing falling bottles in tray.
-



Ladder Program

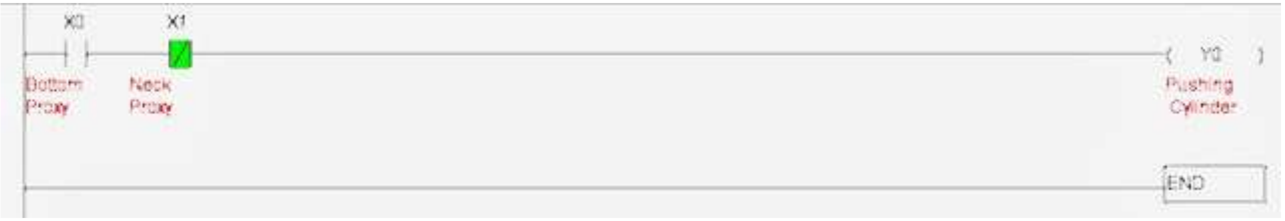
Number of PLC Inputs Required

X0 – Proximity Sensor to sense bottom of the Bottle i.e. X0 = ON when the detected input signal from the bottle-bottom is sheltered.

X1 – Proximity Sensor to sense upper part of the Bottle i.e. X1 = ON when the detected input signal from the bottle-neck is sheltered.

Number of PLC Outputs Required

Y0 – To operate Pushing Cylinder/Rod



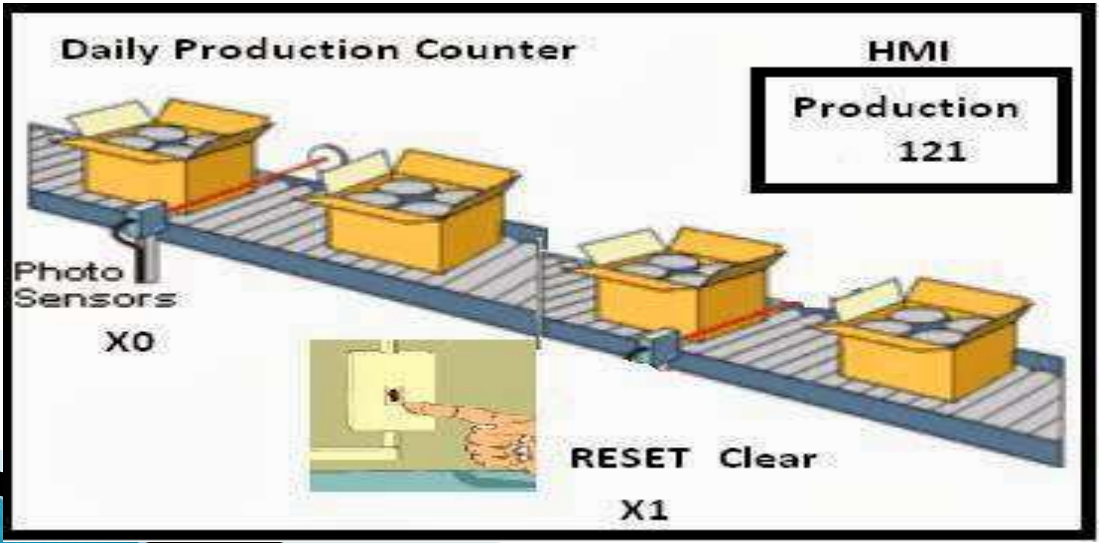
Program Description

- If the bottle on the conveyor belt is upstanding, the input signal from monitoring photocell at both bottle-bottom and bottle-neck will be detected. In this case, X0 = ON, and X1 = ON. The normally open (NO) contact X0 will be activated as well as the normally closed (NC) contact X1. Y0 remains OFF and pneumatic pushing pole will not perform any action.

- If the bottle from the conveyor belt is down, only the input signal from monitoring photocell at the bottle-bottom will be detected. In this case, X0 = ON, X1 = OFF. The state of output YO will be ON because the NO contact X0 activates and the NC contact X1 remains OFF. The pneumatic pushing pole will push the fallen bottle out of the conveyor belt.

Counter Programming Example

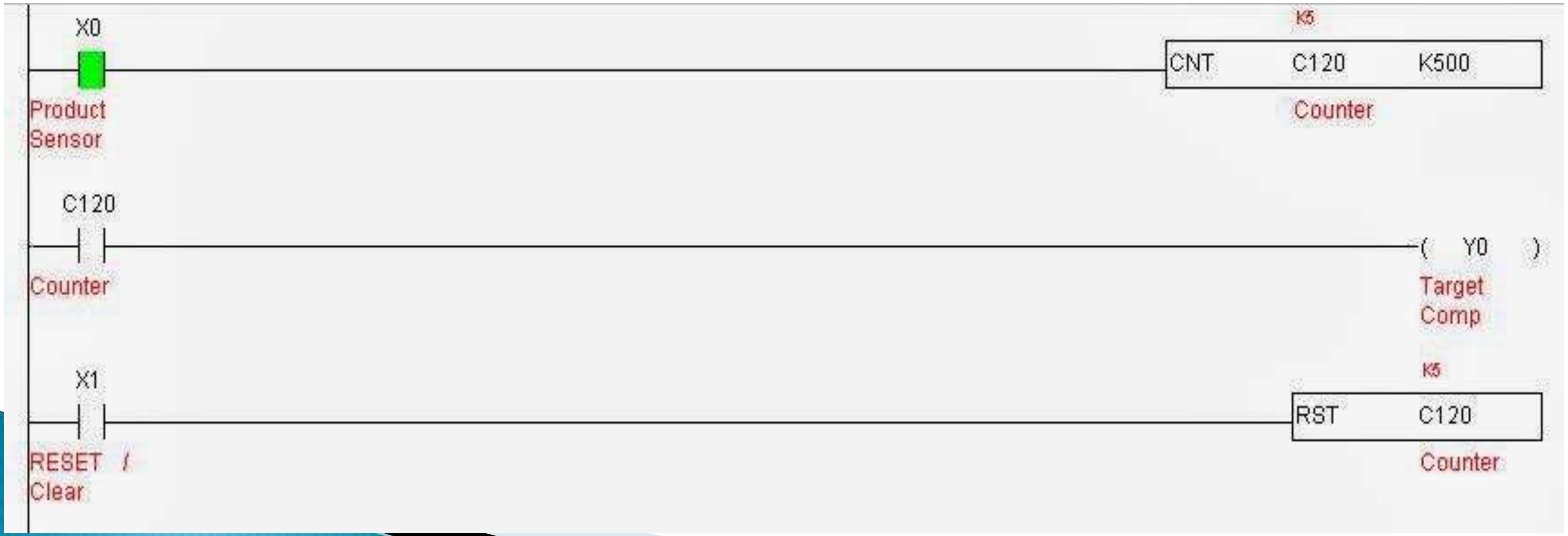
- **PLC Ladder Practice Problem:**
- The production line may be powered off accidentally or turned off for noon break. The program is to control the counter to retain the counted number and resume counting after the power is turned ON again. When the daily production reaches 500, the target completed indicator will be ON to remind the operator for keeping a record. Press the Clear button to clear the history records. The counter will start counting from 0 again.



UP counter Example

Latched 16 bit UP counter

- **Number of PLC Inputs Required**
- X0 – Product Detecting Sensor. X1 – Production Counter RESET/Clear
- **Number of PLC Outputs Required**
  Y0 – Production Counter Target Completed.
- **Number of PLC Counter Required:**
  C120 – 16 Bit Latched Counter. (Max Count =32,768)

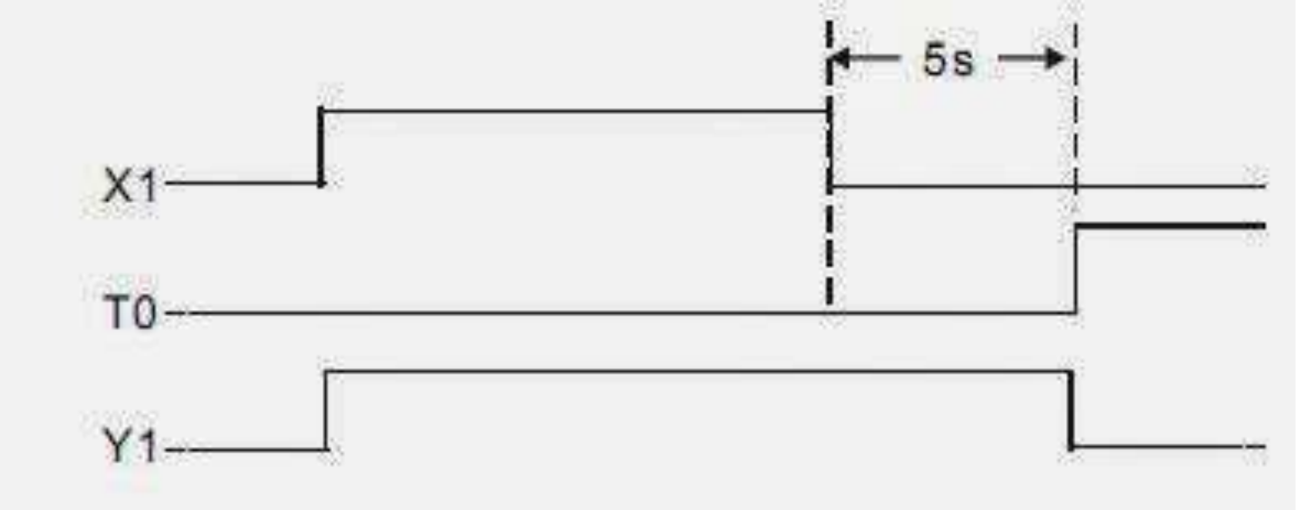


Program Description

- The latching counter is demanded for the situation of retaining data when power-off.
- When a product is completed, C120 will count for one time. When the number reaches 500, target completed indicator Y0 will be ON.
- For different series of PLC, the setup range of 16-bit latching counter is different.

Timer Programming Example

- Enabling the indicator to be ON immediately when switch pressed and OFF after a 5 sec delay by the switch.



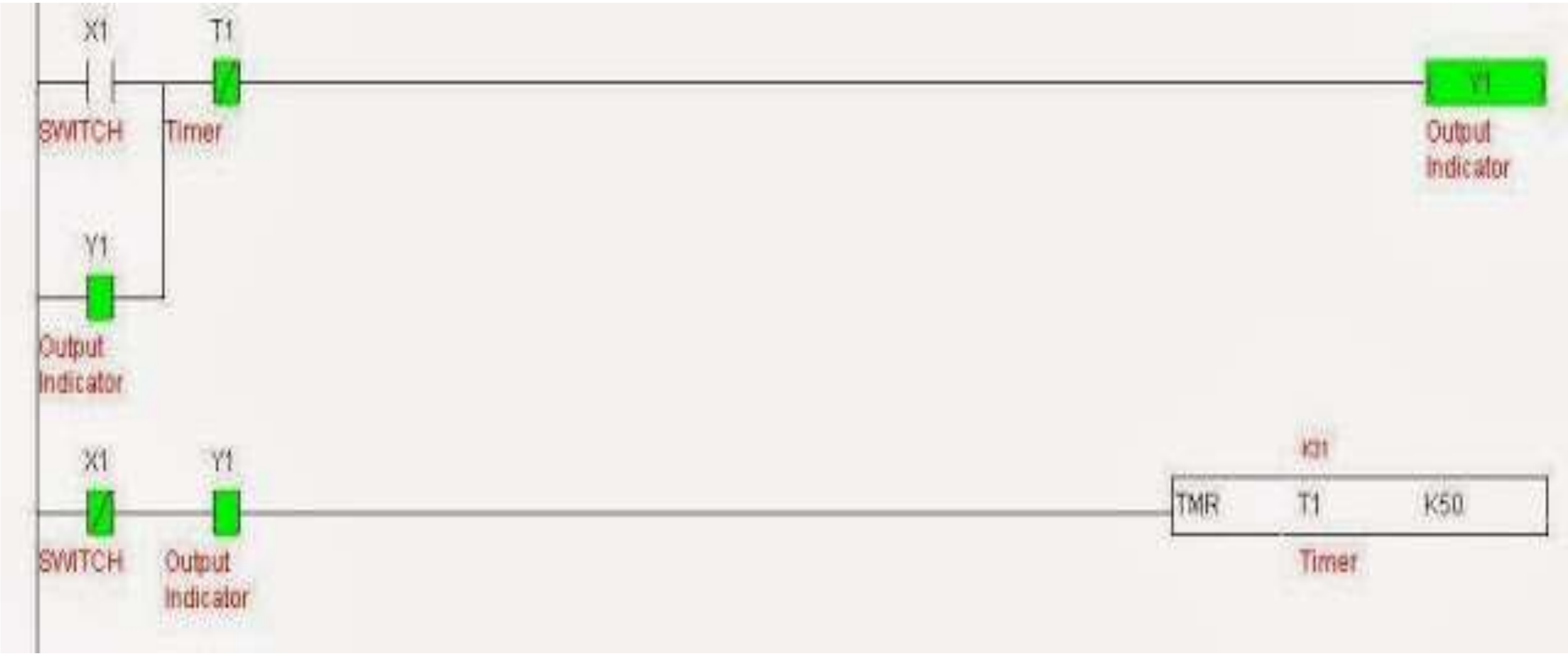
PLC Ladder Program Description

- **Number of PLC Inputs Required**
- X1 – Start Switch.
- **Number of PLC Outputs Required**
- Y1 – Output Indicator
- **Number of PLC Timer Required**
- T0 – 5 second Timer, 100 ms Time Base. (See K50 Preset Value for Timer)

When X1 = ON, TMR instruction will be executed. Timer T1 will be ON and start counting for 3 sec. When T1 reaches its set value, the NO (Normally Open) contact T1 will be activated and indicator YI will be ON.

When X1 = OFF, TMR instruction will not be executed. Timer T1 will be OFF and so will NO contact T1. Therefore, the indicator Y1 will be OFF.

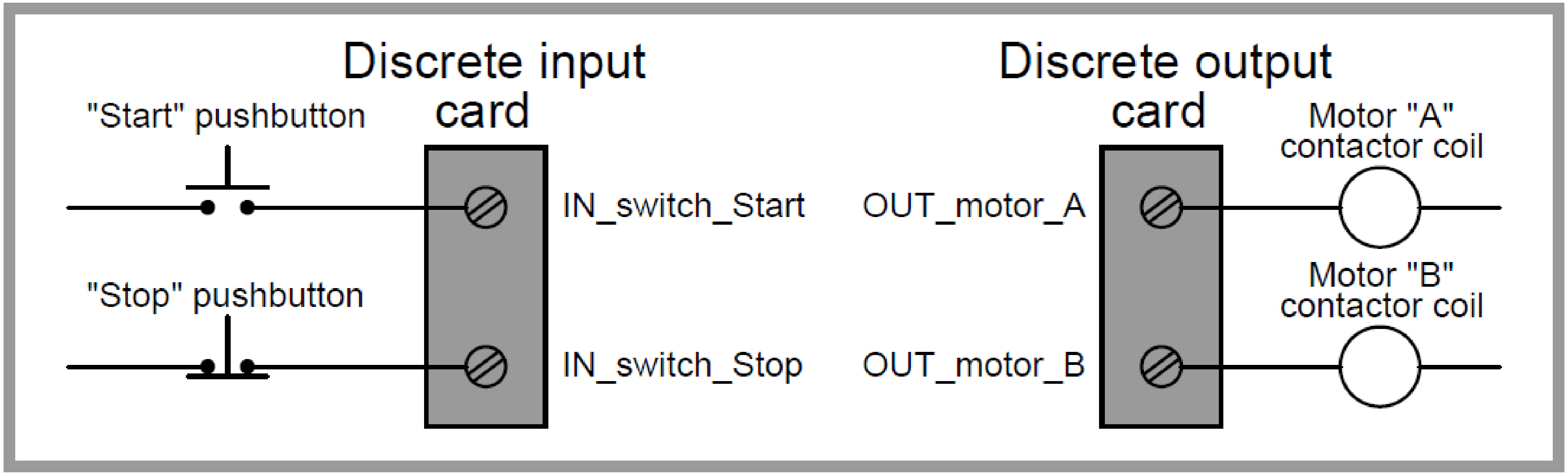
Ladder Program



Comparison Instruction Example

- A practical application for a comparative function is something called alternating motor control, where the run-times of two redundant electric motors are monitored, with the PLC determining which motor to turn on next based on which motor has run the least

Real-world I/O wiring

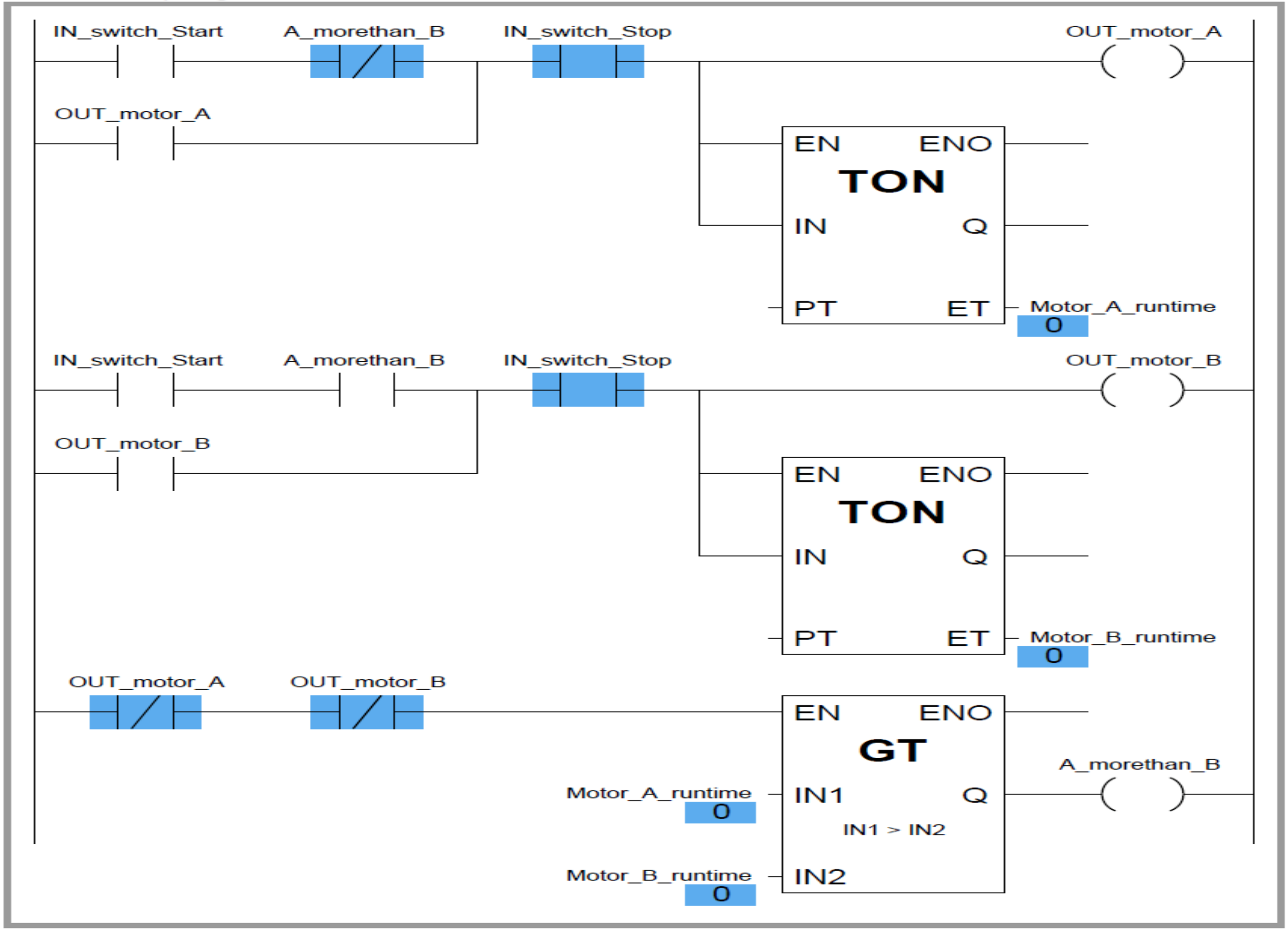


Description

- In this program, two retentive on-delay timers keep track of each electric motor's total run time, storing the run time values in two registers in the PLC's memory:
- Motor A runtime and Motor B runtime. These two integer values are input to the "greater than" instruction box for comparison.
- If motor A has run longer than motor B, motor B will be the one enabled to start up next time the "start" switch is pressed.
- If motor A has run less time or the same amount of time as motor B (the scenario shown by the blue-highlighted status indications), motor A will be the one enabled to start.
- The two series-connected virtual contacts OUT motor A and OUT motor B ensure the comparison between motor run times is not made until both motors are stopped.
- If the comparison were continually made, a situation might arise where both motors would start if someone happened to press the Start pushbutton with one motor is already running.

Ladder Program

PLC program



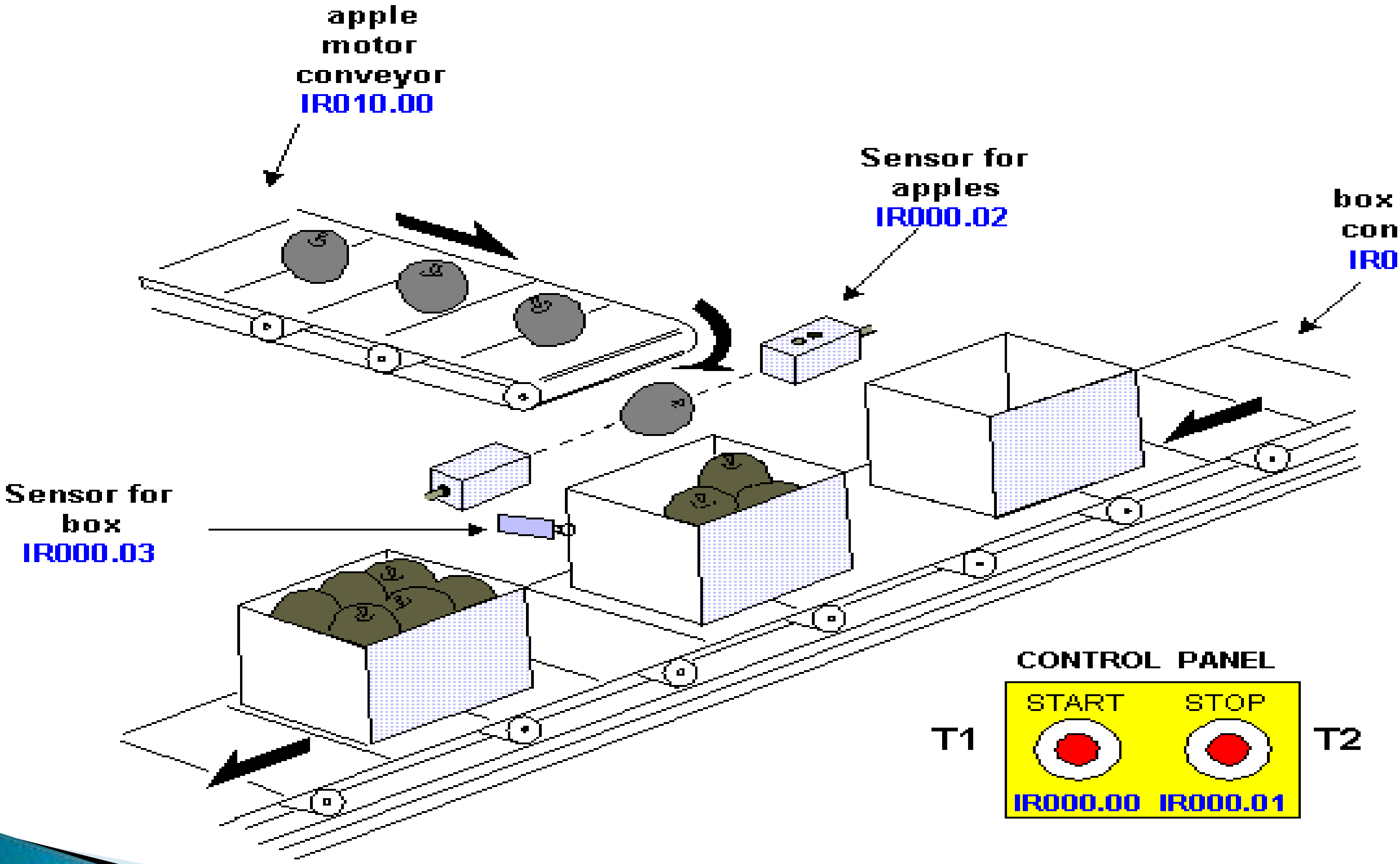
CHAPTER 4

Applications of PLCs

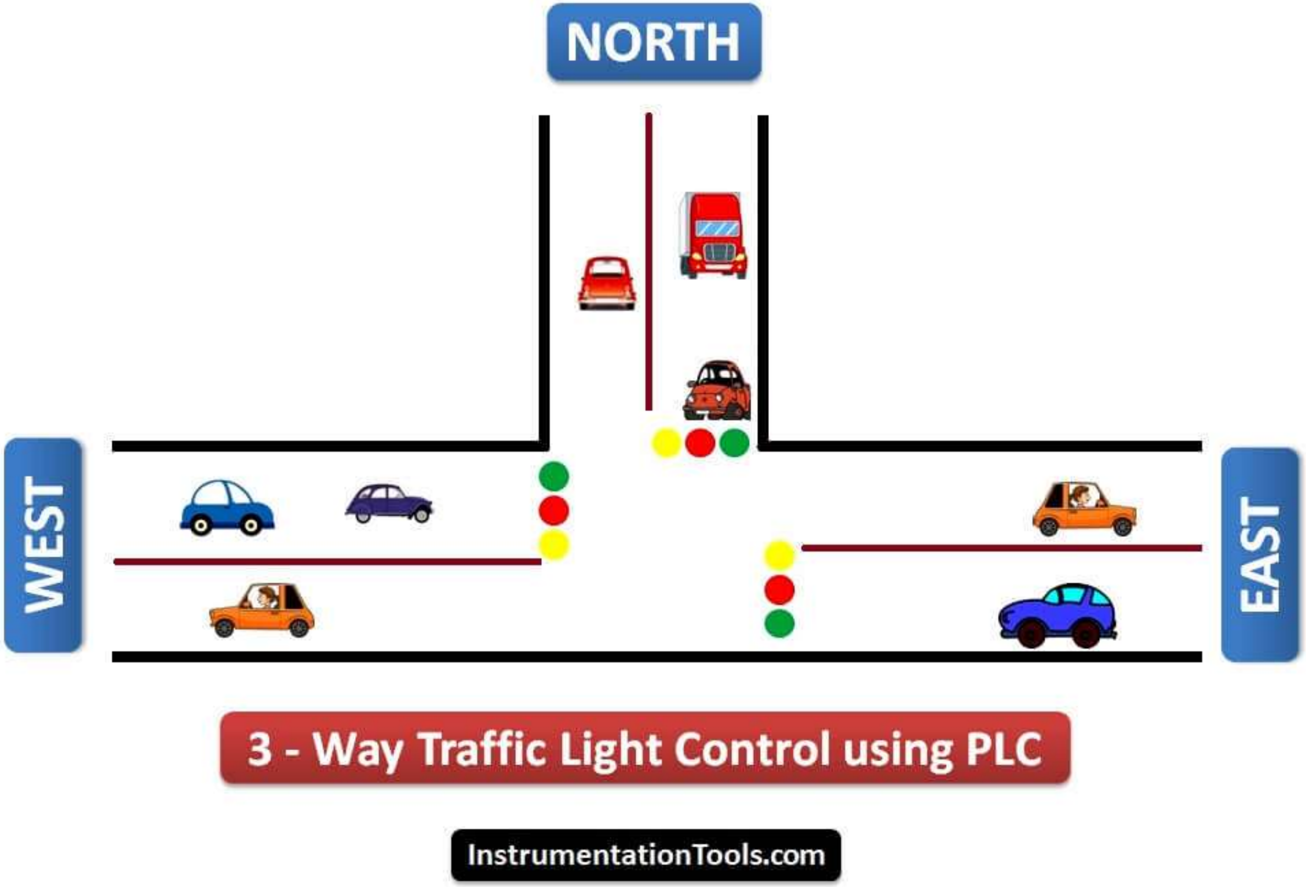
Automation of product packaging

- Product packaging is one of the most frequent cases for automation in industry.
- It can be encountered with small machines (ex. packaging grain like food products) and large systems such as machines for packaging medications.
- Example we are showing here solves the classic packaging problem with few elements of automation.

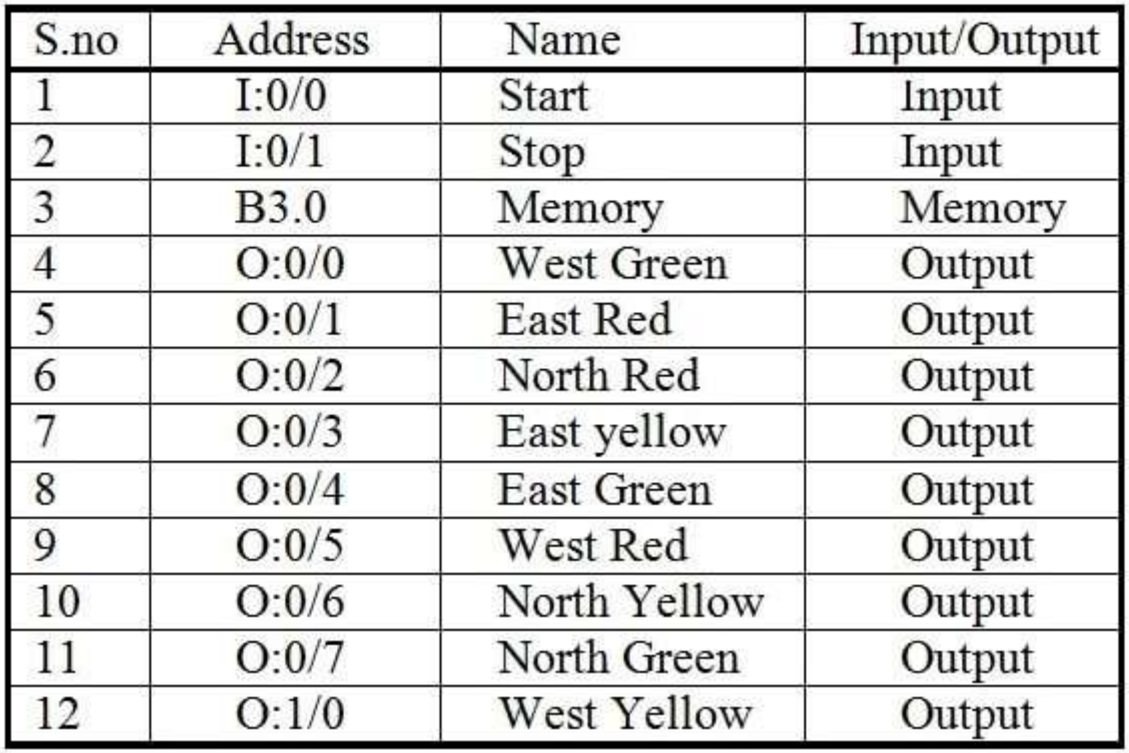
Automation of product packaging


apple motor conveyor IR010.00
Sensor for apples IR000.02
box motor conveyor IR010.01
Sensor for box IR000.03
CONTROL PANEL
START
STOP
T1
T2
IR000.00
IR000.01

Three WayTraffic Light Control using PLC



Traffic Light Control using PLC



S.no	Address	Name	Input/Output
1	I:0/0	Start	Input
2	I:0/1	Stop	Input
3	B3.0	Memory	Memory
4	O:0/0	West Green	Output
5	O:0/1	East Red	Output
6	O:0/2	North Red	Output
7	O:0/3	East yellow	Output
8	O:0/4	East Green	Output
9	O:0/5	West Red	Output
10	O:0/6	North Yellow	Output
11	O:0/7	North Green	Output
12	O:1/0	West Yellow	Output

Traffic Light Control using PLC

- They are so many ways to write a program for traffic light control ex: sequencer output method but in this normal input, outputs and timers are used.
- Timers are used to give time delay for output to turn ON and OFF.
- Reset coil is used at the end to run the program continuously.
- Comparator blocks are used to reduce the number of timers used.

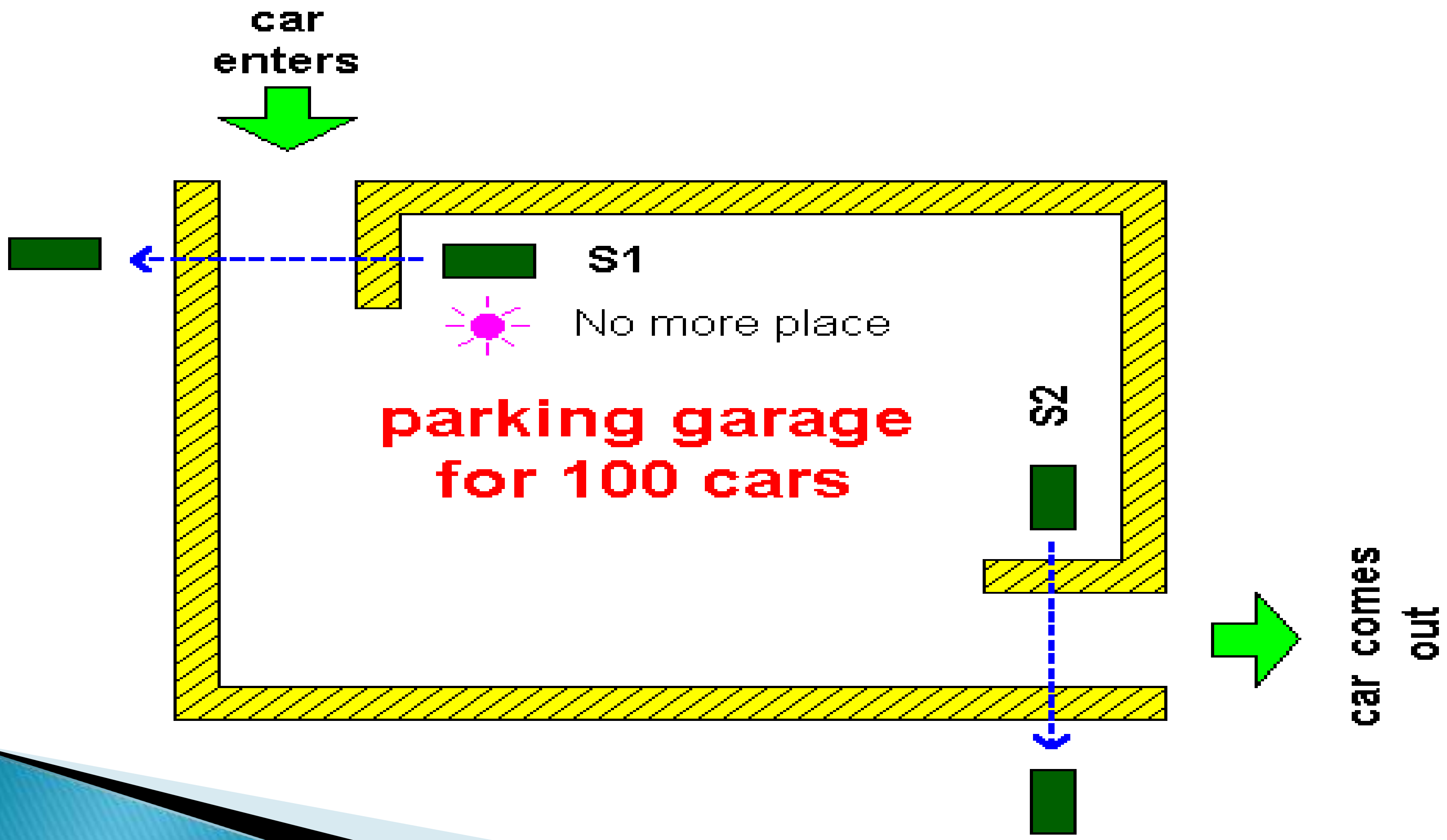
Steps or sequence of outputs to turn ON

S.NO	EAST	WEST	NORTH
1	R	G	R
2	Y	G	R
3	G	R	R
4	G	R	Y
5	R	R	G
6	R	Y	G

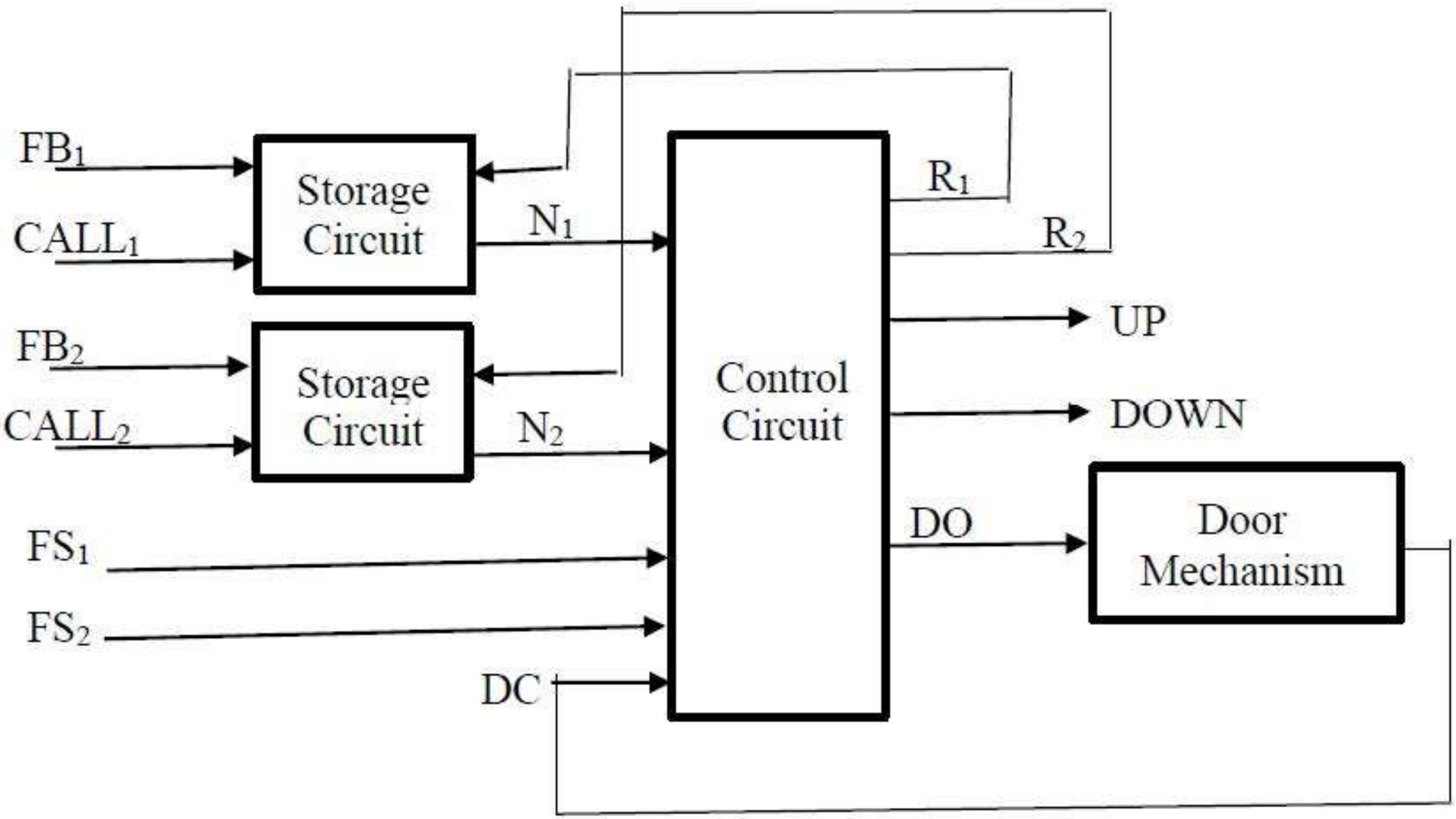
Car Parking System

- We are dealing with a simple system that can control 100 car at the maximum. Each time a car enters, PLC automatically adds it to a total sum of other cars found in the garage. Each car that comes out will automatically be taken off. When 100 cars park, a signal will turn on signalizing that a garage is full and notifying other drivers not to enter because there is no space available.

Automation of parking garage



PLC ladder logic for washing machine

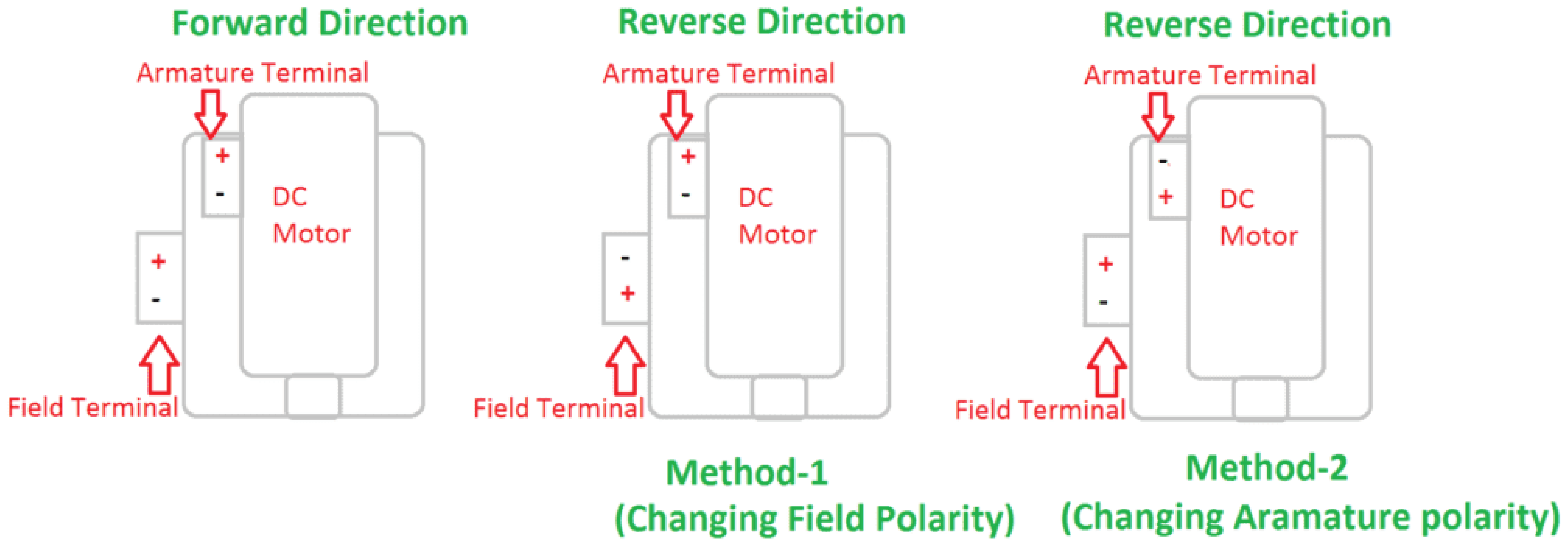


Forward & Reverse of DC motor:

- DC motors are totally different from AC motors. They have commutator, field winding and armature winding. The DC supply will be given to the field winding and armature winding. You can reverse the direction for DC motor in two ways.

Forward & Reverse of DC motor

DC Motor Forward and Reverse Direction



Forward Direction

Reverse Direction

Reverse Direction

Method-1
(Changing Field Polarity)

Method-2
(Changing Aramature polarity)

Forward & Reverse of DC motor

- By changing the supply Polarity in field winding or filed supply. Field terminal consist of F1 and F2. Normally, in forward direction the DC supply will be given such as F1 – Positive and F2 – Negative, to change the direction the polarity should be F1– Negative and F2 – Positive.
- Also in same way we can change the direction of DC Motor by changing the polarity of the armature winding. Armature terminal consist of A1 and A2. Normally, for forward direction the DC supply will be given such as A1 – Positive and A2 – Negative, to change the direction the polarity should be A1– Negative and A2 – Positive.

Direct-On-Line (DOL) starter

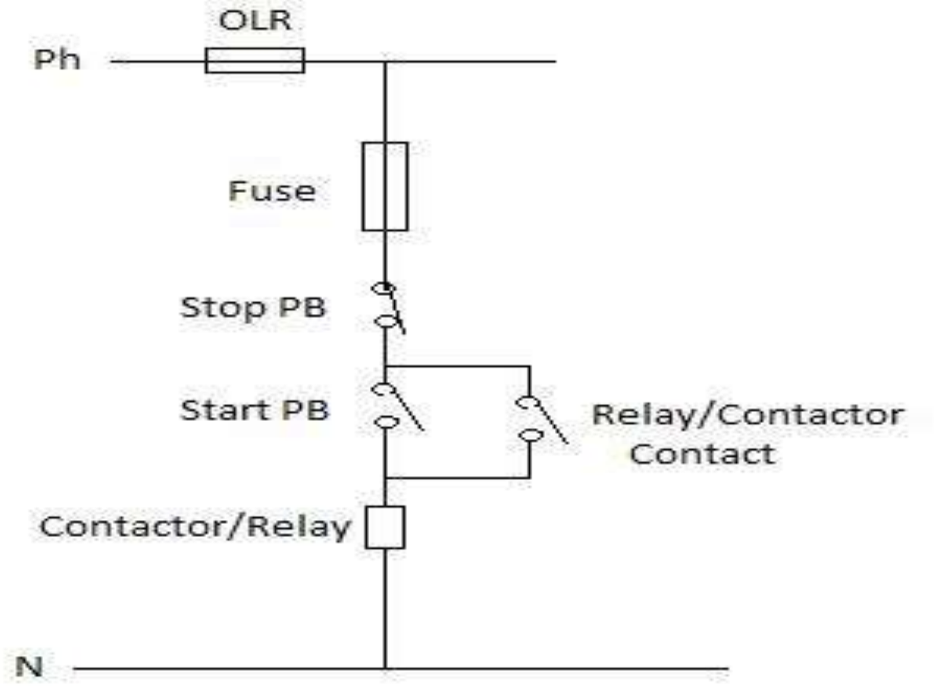
- One method of starting electric motors is using direct on line (DOL) or across the line starter. In this method full line voltage is applied to the motor terminals. This is simplest type of motor starter.

DOL Starter

- DOL motor starter contains fuse and over load relay (OLR) for protection purpose. The starter can be contain momentary contact or maintained contact push buttons. The example considered here is momentary contact push buttons. For starting purpose normally open (NO) push button is preferred whereas normally closed (NC) push button is used to stop the motor.

  The excessive supply voltage drop causing high inrush current is the criteria to limit the use of DOL starter. Conveyor motors, water pumps are the applications where DOL starters are used.

DOL Starter



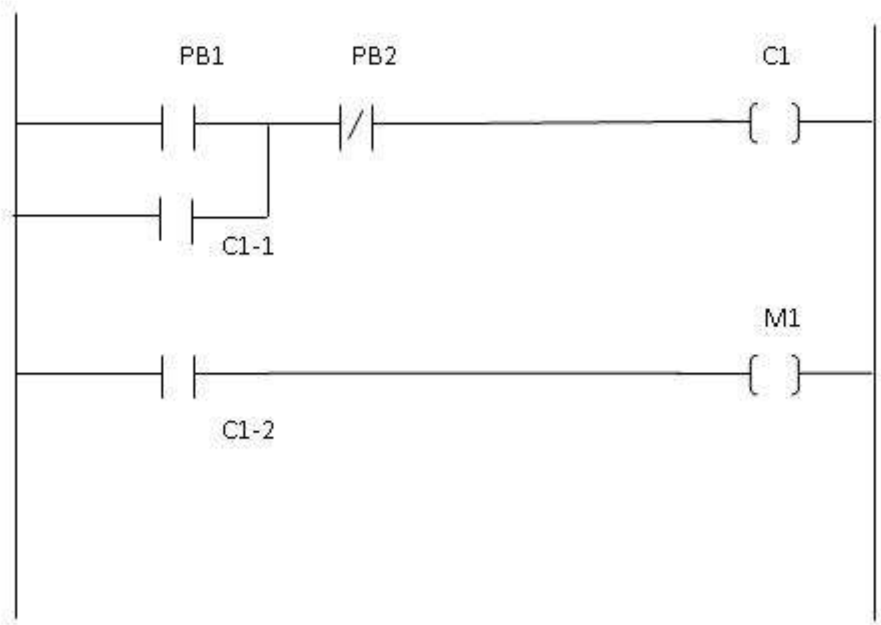
Control Circuit for DOL Starter

Sequence of Events

- Listing of Input and Output devices:
  Inputs: PB1- To start the motor
  PB2- To stop the motor
  Output: M1- Motor

  1. When Start push button (PB1) is pressed, Motor (M1) has to start.
  2. If Start pushbutton (PB1) is released and Stop pushbutton (PB2) is not pressed, Motor (M1) should remain on.
  3. When Stop push button (PB2 is pressed, Motor (M1) has tol stop.
  4. If stop push button is released and start is not pressed (released) motor shouldl remain off.

DOL Starter



CHAPTER 5

Introduction to SCADA

SCADA

- **SCADA** is the **acronym for** Supervisory Control And Data Acquisition. It is generally used for industrial control systems.
- Thus, it is not a comprehensive control system but it rather operates as supervisory software superior to PLCs and other devices.

SCADA system

- A SCADA system is a common process automation system which is used to gather data from sensors and instruments located at remote sites and to transmit and display this data at a central site for control or monitoring purposes.
- Common analog signals that SCADA systems monitor and control are levels, temperatures, pressures, flow rate and motor speed. Typical digital signals to monitor and control are level switches, pressure switches, generator status, relays & motors.
- The collected data is usually viewed on one or more SCADA Host computers located at the central or master site.

Features of SCADA

1. Analog inputs for live monitoring

- Analog inputs allow you to monitor real-time data across your network.
- Many SCADA systems will only have discrete inputs - which are digital and can only tell you if something is "on" or "off."
- An analog input will be able to tell you precise values, meaning you can have accurate data about whatever you're monitoring.
- Graphical web interface is used for easy-to-use interface system of SCADA

Control relays for remote access and control

- By having control relays on your SCADA system, you'll be able to remotely control any device in your network that is normally operated by a button or a switch.
- You'll be able to start equipment, open or close doors, or turn on lights.
- Instead of driving long distances to perform these simple tasks, you can do them right from your desk – without wasting time or money behind the windshield of a truck.

Functions of SCADA

- All the equipment connected through **machine control** are operated through instruction sent over the web. The processing unit analyses the data and supervises the healthy functioning of the signals transmitted in the entire unit. The data is stored for valuation in a distributed database. The acquired data also has reference metadata stored for in the different database such as a programmable logic controller (**PLC**).

How does it work?

- The complete **SCADA systems** has four significant units namely – data communication, data acquisition, information or data presentation and monitoring and control system. With complete symphony in the above four mentioned units, the entire operation of the automation system can be monitored.

How does it work?

- The data acquisition system fetches real-time data from all the connected machine units. The data reports about the status of all the components and sensors, where data communication network comes into play. The system ensures accuracy in data being transferred through the internet protocol. Once the data is collected, the processing unit analyses the data and presents it to the operator through **Human-Machine Interface (HMI)**.

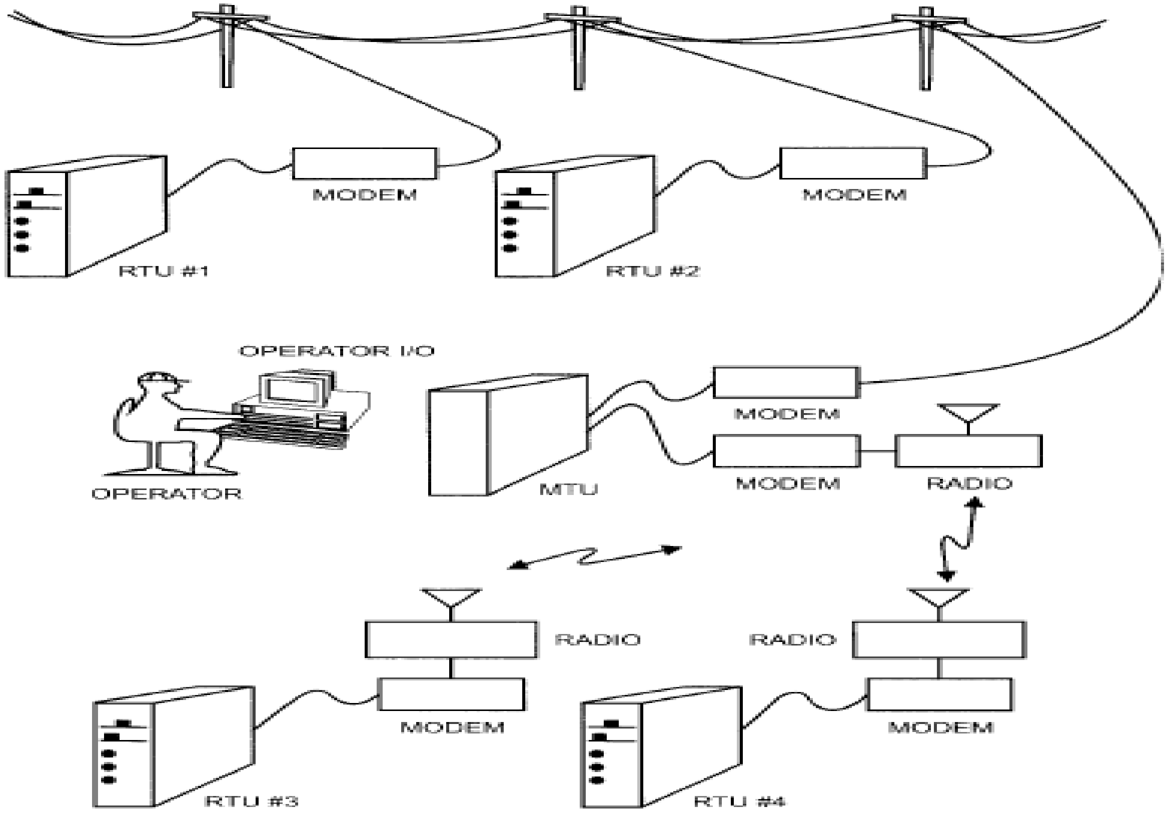
Scope of SCADA

- This is the reason why many more enterprises are looking for automating their industrial processes.
- With the **Industrial Internet of Things** taking the front seat, it is essential to look for companies helping businesses step towards automation. Schneider
- Electric India is one of the significant names in the research and development of the automation industry. It has not only brought these systems to common units but also helps the employees to up skill their knowledge by training them how to operate the machinery.

Elements of a SCADA system

- HMI
- RTU
- MTU
- Data transmission

Elements of a SCADA system



HMI unit

- A Human – Machine Interface (HMI) is the apparatus which presents process data to a human operator, and through this, the human operator monitor and control the process. HMI offers real–time monitor of data about the process and through which an operator can send commands to the controller. Input devices such as the keyboard, mouse, trackballs are available in this section.

MTU:

- MTU (master control unit), which is the system controller. Some industries use the term “host computer” instead of MTU.
- MTU communicates with the RTU that is located away from the central location. There can be many RTUs in the field, MTU can monitor and control the field using the scheduled program even when the operator is not present. Changes can be done in the process from the MTU end, can read some process parameter.

RTU

- Remote terminal unit (RTU) connection to sensors in the process and converting sensors signals to digital data and sending digital data to the supervisory system.
- RTU communicate with the MTU using modulated signal. RTU receives the modulated data from MTU and connection can be through cable or radio.
- RTU field device connection is through cables. RTU supplies both electrical power and actuator signal to the field device.

Advantages of SCADA

- Long distance monitoring
- Long distance training
- Protection against terrorism/vandalism–alarm
- Data management (engineering and operations)
- Automated operations with real–time control
- Reliability and Robustness (very large installed base, mission–critical processes)

Application 0f SCADA

- **SCADA** systems are **used** to control and monitor physical processes, examples of which are transmission of electricity, transportation of gas and oil in pipelines, water distribution, traffic lights, and other systems **used** as the basis of modern society.

Application of the System

- Although this is a new technology, businesses have already leveraged the technology in both the public and private sector. It has brought about immense satisfaction in the operation managers because the system makes the production line foolproof.
- This also streamlines packaging and delivery lines following the processes. The overall automation and process control has helped to save a significant amount of money and time.

SCADA Application Example

- A real world SCADA system can monitor and control hundreds to hundreds of thousands of I/O points. A typical **Water SCADA application** would be to monitor water levels at various water sources like reservoirs and tanks and when the water level exceeds a preset threshold, activate the system of pumps to move water to tanks with low tank levels.

CHAPTER 6

Micro Controller Series (MCS)-51 Over View

Why do we need to learn Microprocessors/controllers?

- The microprocessor is the core of computer systems.
- Nowadays many communication, digital entertainment, portable devices, are controlled by them.
- A designer should know what types of components he needs, ways to reduce production costs and product reliable.
- **Hardware :Interface to the real world**
- **Software :order how to deal with inputs**

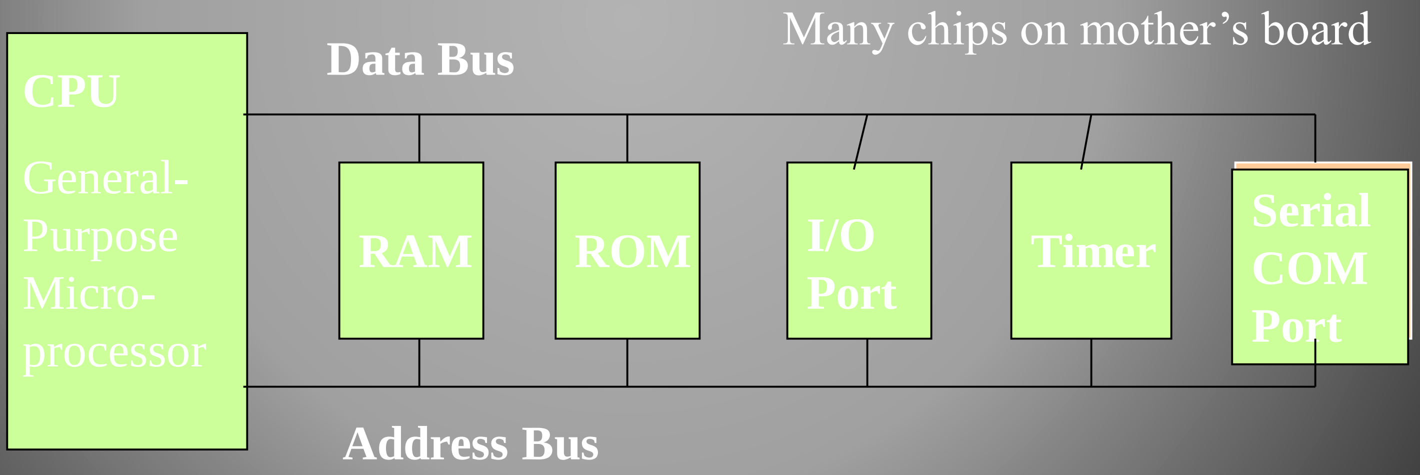
Elements of microprocessor/Microcontroller

- CPU: Central Processing Unit
- I/O: Input /Output
- Bus: Address bus & Data bus
- Memory: RAM & ROM
- Timer
- Interrupt
- Serial Port
- Parallel Port

Microprocessors

General-purpose microprocessor

- CPU for Computers
- No RAM, ROM, I/O on CPU chip itself
- Example：Intel's 8085,8086, Motorola's 680x0



Microprocessor vs. Microcontroller

Microprocessor

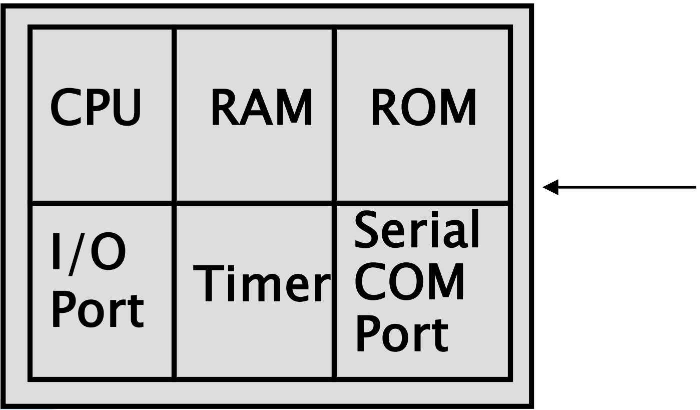
- CPU is stand-alone, RAM, ROM, I/O, timer are separate
- Designer can decide on the amount of ROM, RAM and I/O ports.
- expansive
- versatility
- general-purpose

Microcontroller

- CPU, RAM, ROM, I/O and timer are all on a single chip
- fix amount of on-chip ROM, RAM, I/O ports
- for applications in which cost, power and space are critical
- single-purpose

8051 Basic Component

- 4K bytes internal ROM
- 128 bytes internal RAM
- Four 8-bit I/O ports (P0 - P3).
- Two 16-bit timers/counters
- One serial interface



CPU

RAM

ROM

I/O Port

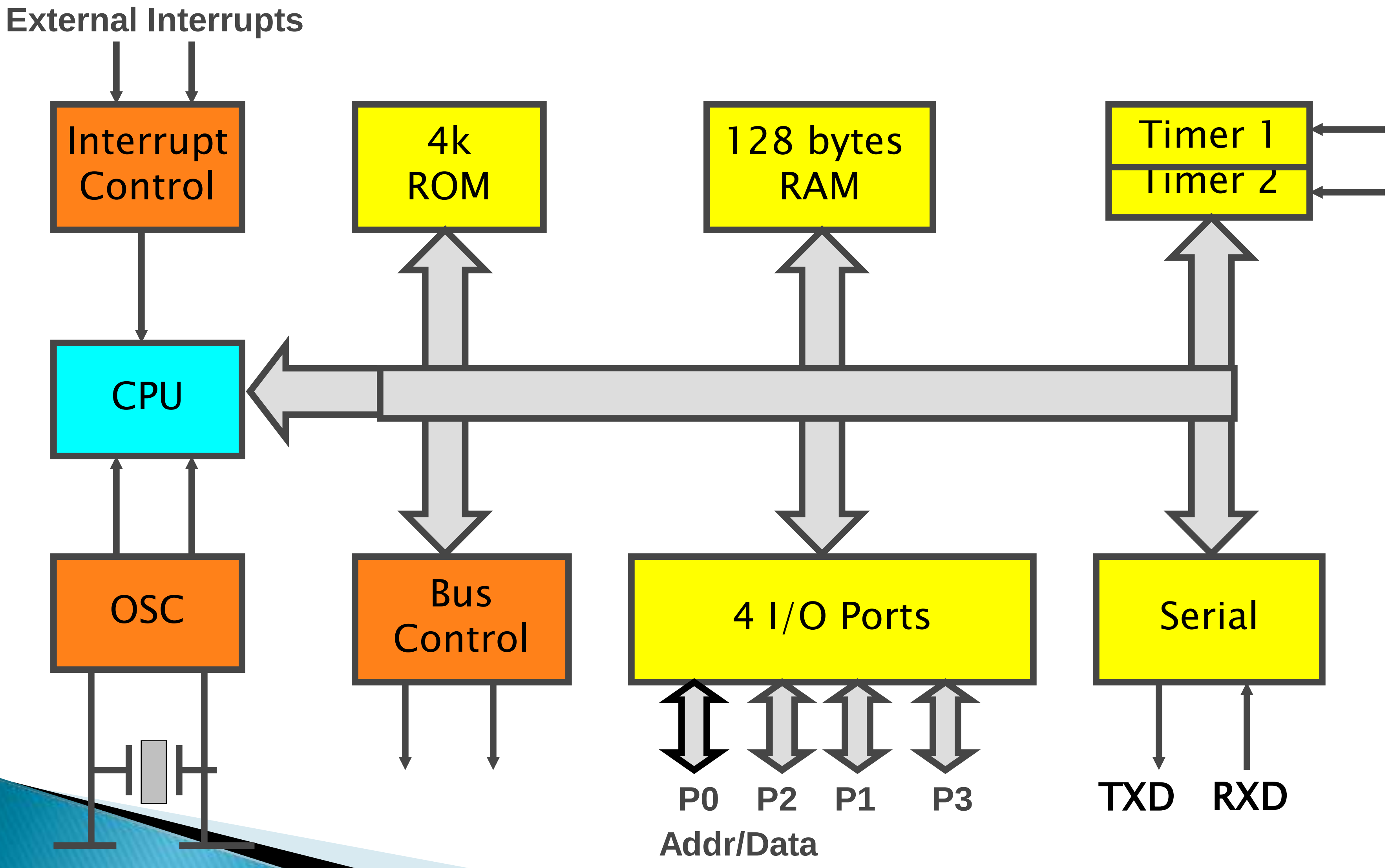
Timer

Serial COM Port

A single chip

Microcontroller

Block Diagram



Other 8051 featurs

- only 1 On chip oscillator (external crystal)
- 6 interrupt sources (2 external , 3 internal, Reset)
- 64K external code (program) memory(only read)PSEN
- 64K external data memory(can be read and write) by RD,WR
- Code memory is selectable by EA (internal or external)
- We may have External memory as data and code

Comparison of the 8051 Family Members

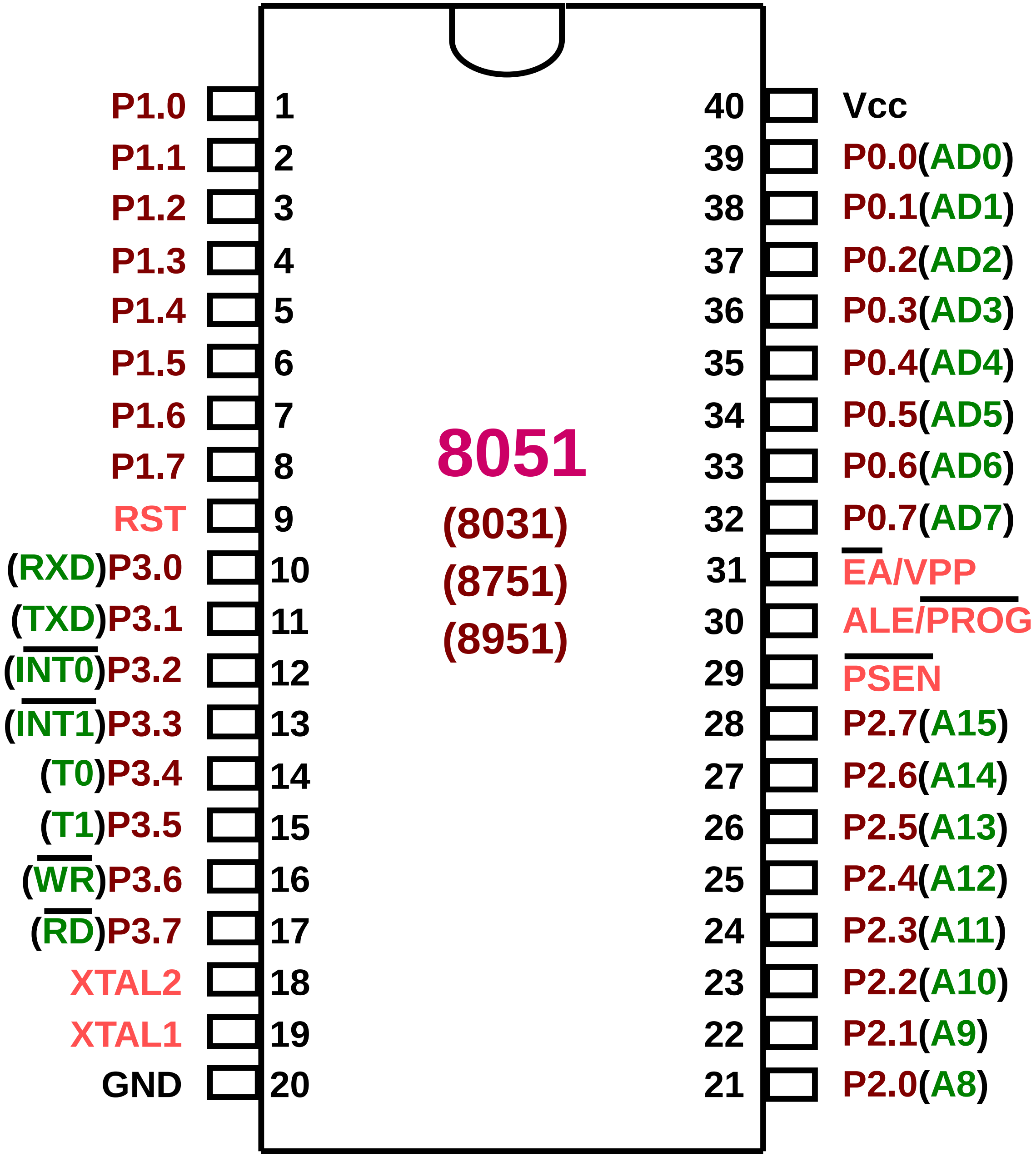
89XX	ROM	RAM	Timer	Int Source	IO pin	Other
8951	4k	128	2	6	32	-
8952	8k	256	3	8	32	-
8953	12k	256	3	9	32	WD
8955	20k	256	3	8	32	WD
898252	8k	256	3	9	32	ISP
891051	1k	64	1	3	16	AC
892051	2k	128	2	6	16	AC

WD: Watch Dog Timer
AC: Analog Comparator
ISP: In System Programable

Three criteria in Choosing a Microcontroller

- meeting the computing needs of the task efficiently and cost effectively
  - speed, the amount of ROM and RAM, the number of I/O ports and timers, size, packaging, power consumption
  - easy to upgrade
  - cost per unit
- availability of software development tools
  - assemblers, debuggers, C compilers, emulator, simulator, technical support
- wide availability and reliable sources of the microcontrollers

8051
Foot Print



IMPORTANT PINS (IO Ports)

- **One of the most useful features of the 8051 is that it contains four I/O ports (P0 - P3)**

- Port 0 （pins 32-39）：P0（P0.0～P0.7）
  - 8-bit R/W - General Purpose I/O
  - Or acts as a multiplexed low byte address and data bus for external memory design

- Port 1 （pins 1-8）　：P1（P1.0～P1.7）
  - Only 8-bit R/W - General Purpose I/O

- Port 2 （pins 21-28）：P2（P2.0～P2.7）
  - 8-bit R/W - General Purpose I/O
  - Or high byte of the address bus for external memory design

- Port 3 （pins 10-17）：P3（P3.0～P3.7）
  - General Purpose I/O
  - if not using any of the internal peripherals (timers) or external interrupts.
- **Each port can be used as input or output (bi-direction)**

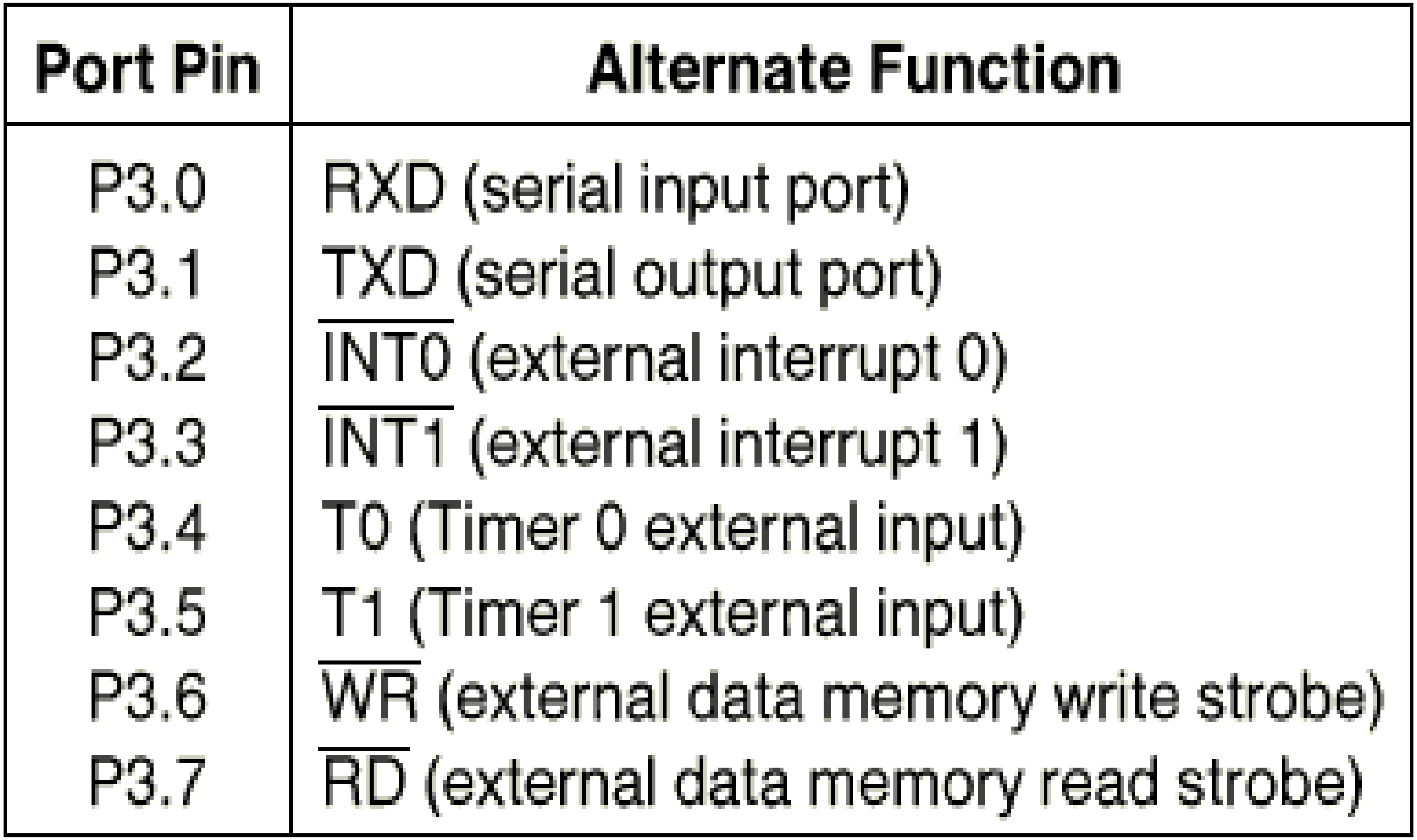
Pins of 8051

- **PSEN** (out): Program Store Enable, the read signal for external program memory (active low).
- **ALE** (out): Address Latch Enable, to latch address outputs at Port0 and Port2
- **EA** (in): External Access Enable, active low to access external program memory locations 0 to 4K
- **RXD,TXD**: UART pins for serial I/O on Port 3
- **XTAL1 & XTAL2**: Crystal inputs for internal oscillator.
- **Vcc（pin 40）：**
  - **Vcc provides supply voltage to the chip.**
  - **The voltage source is +5V.**
- **GND（pin 20）：ground**

Pins of 8051

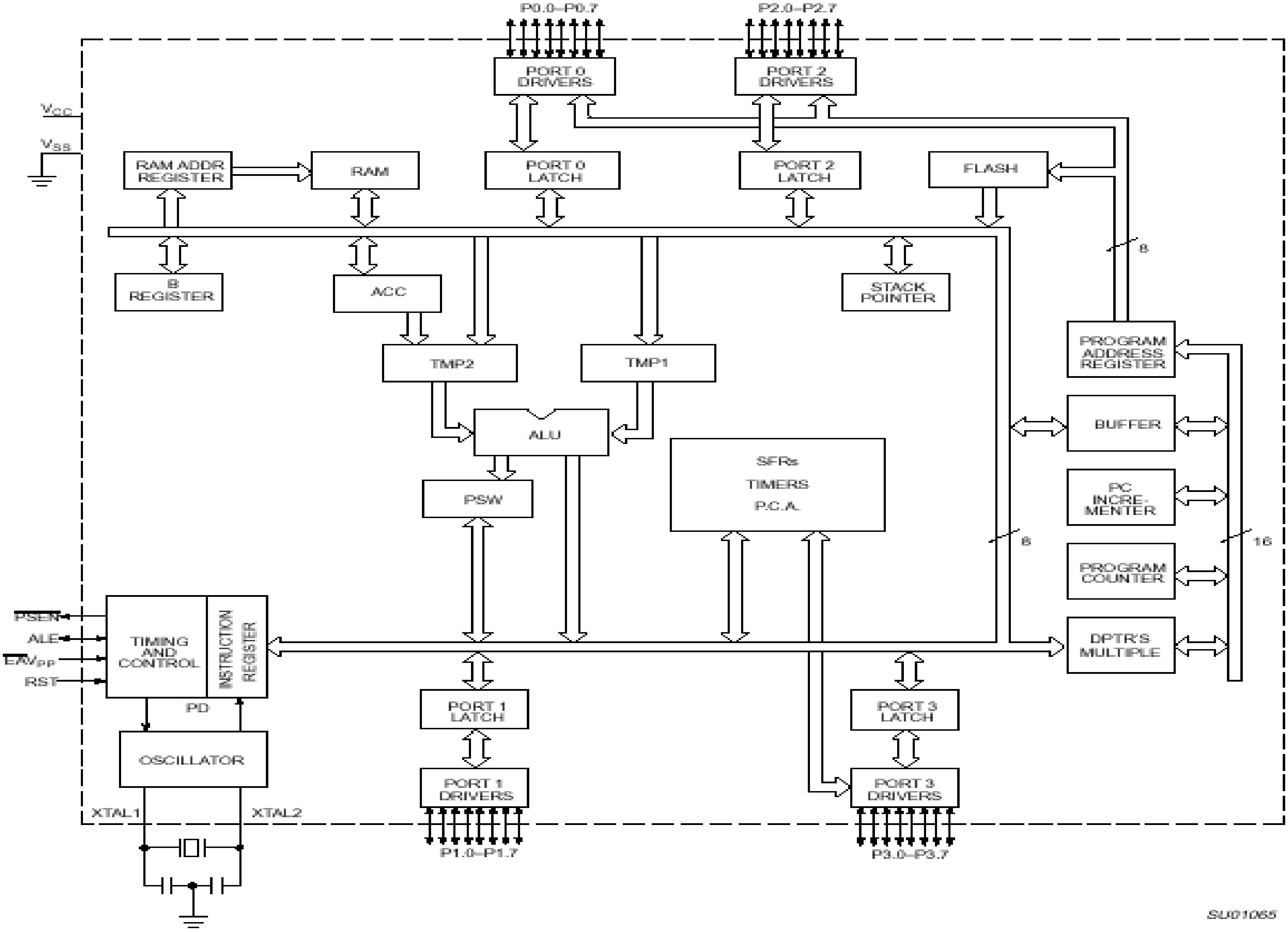
- RST（pin 9: Reset
  - input pin and active high（normally low）.
    - The high pulse must be high at least 2 machine cycles.
  - Power-on reset
    - Upon applying a high pulse to RST, the microcontroller will reset and all values in registers will be lost.
    - Reset values of some 8051 registers

Port 3 Alternate Functions



Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{\text{INT0}}$ (external interrupt 0)
P3.3	$\overline{\text{INT1}}$ (external interrupt 1)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input)
P3.6	$\overline{\text{WR}}$ (external data memory write strobe)
P3.7	$\overline{\text{RD}}$ (external data memory read strobe)

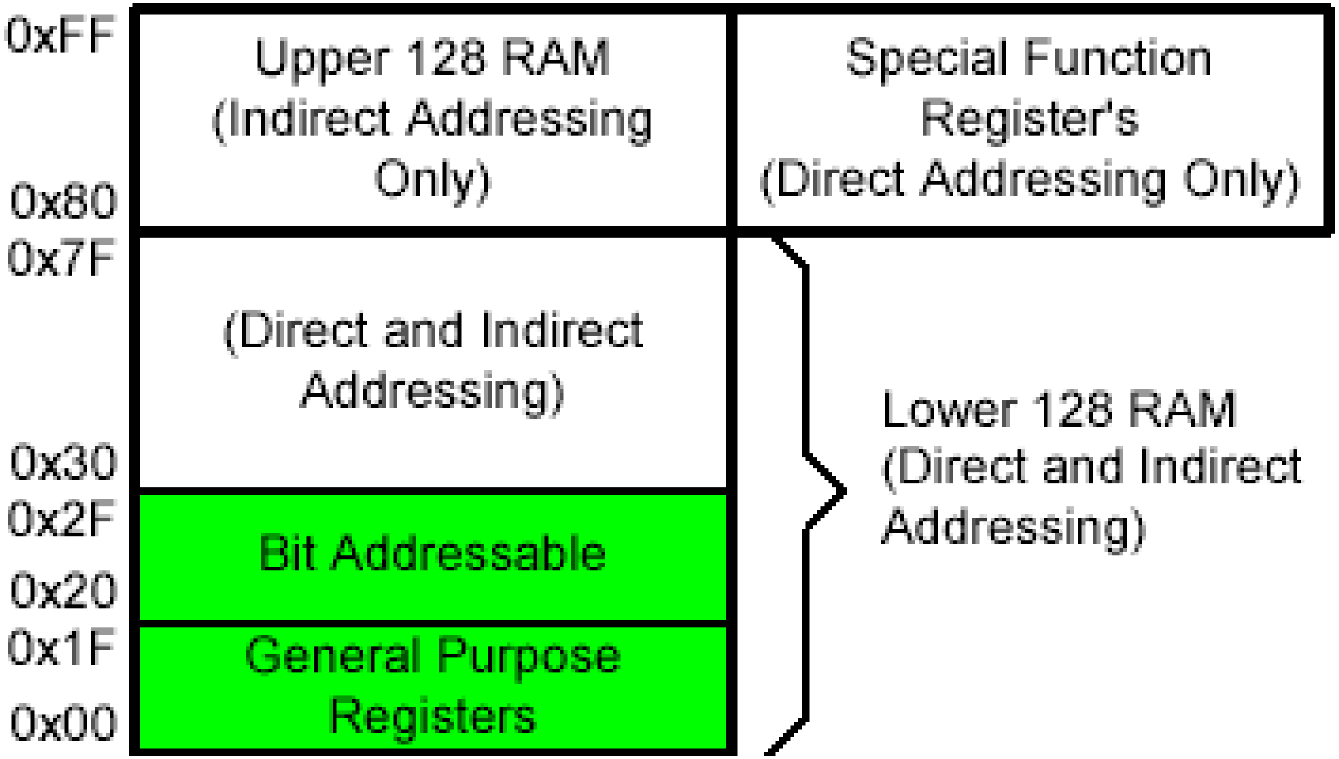
8051 Internal Block Diagram



Specific Features OF 8051

- 8 bit CPU with registers A and B
- 16 bit PC and DPTR(data pointer).
- 8 bit program status word(PSW)
- 8 bit Stack Pointer
- 32 I/O pins arranged as 4 8 bit ports:P0 to P3
- Two 16 bit timer/counters:T0 and T1
- Full duplex serial data receiver/transmitter
- Control registers : TCON,TMOD,SCON,PCON,IP and IE
- Two external and Three internal interrupt sources.
- Oscillator and Clock Circuits.
- 4K Internal ROM
- 128bytes Internal RAM
  - →4 register banks each having 8 registers
    →16 bytes, which may be addressed at the bit level.
    →80 bytes of general purpose data memory

On-Chip Memory
Internal RAM



0xFF

0x80

Upper 128 RAM (Indirect Addressing Only)

Special Function Register's (Direct Addressing Only)

0x7F

0x30

(Direct and Indirect Addressing)

0x2F

0x20

Bit Addressable

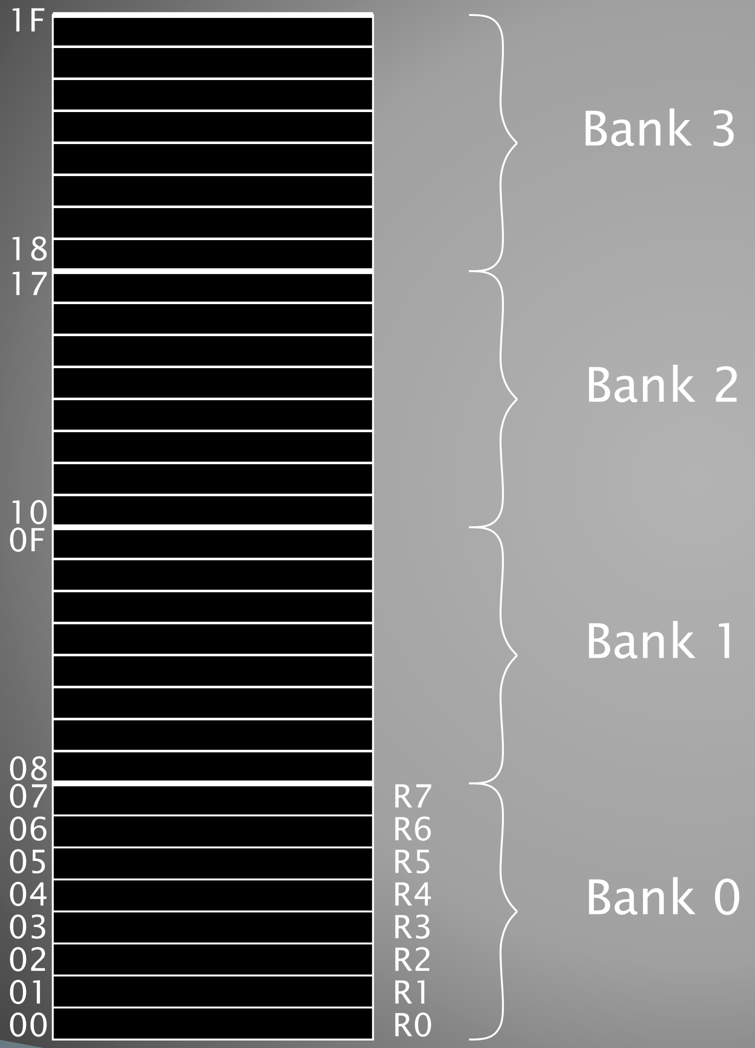
0x1F

0x00

General Purpose Registers

Lower 128 RAM (Direct and Indirect Addressing)

Registers Bank



1F
18
17
10
0F
08
07 R7
06 R6
05 R5
04 R4
03 R3
02 R2
01 R1
00 R0

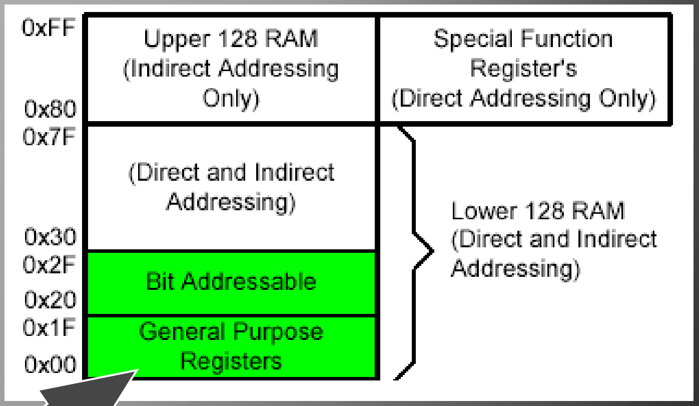
Bank 3

Bank 2

Bank 1

Bank 0

Four Register Banks
Each bank has R0-R7
Selectable by psw.2,3



0xFF
0x80
0x7F
0x30
0x2F
0x20
0x1F
0x00

Upper 128 RAM (Indirect Addressing Only)

Special Function Register's (Direct Addressing Only)

(Direct and Indirect Addressing)

Bit Addressable

General Purpose Registers

Lower 128 RAM (Direct and Indirect Addressing)

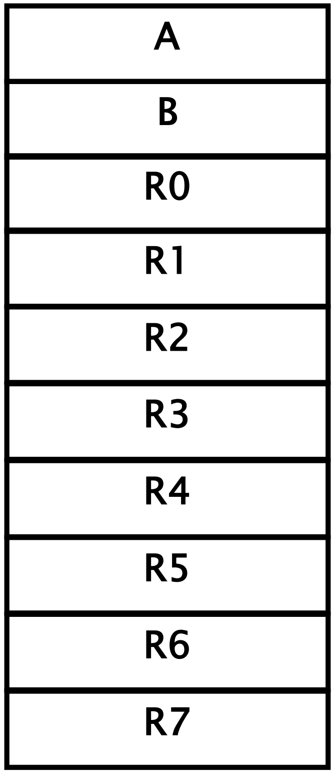
Register Banks

- Four banks of 8 byte-sized registers, **R0** to **R7**
- Addresses are :

18 – 1F	for bank 3
10 – 17	for bank 2
08 – 0F	for bank 1
00 – 07	for bank 0 (default)

- Active bank selected by bits [**RS1, RS0**] in PSW.
- Permits fast "context switching" in interrupt service routines (ISR).

Registers



A

B

R0

R1

R2

R3

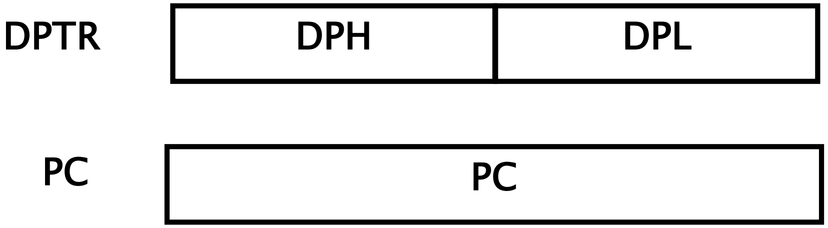
R4

R5

R6

R7

Some 8-bit Registers of the 8051



DPTR

DPH

DPL

PC

PC

Some 8051 16-bit Register

The Accumulator

- It 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.

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
- The "R" registers are also used to temporarily store values.
- For example, let’s say you want to add the values in R1 and R2 together and then subtract the values of R3 and R4. One way to do this would be:

```
MOV A,R3 ;Move the value of R3 into the accumulator
 ADD A,R4 ;Add the value of R4
 MOV R5,A ;Store the resulting value temporarily in R5
 MOV A,R1 ;Move the value of R1 into the accumulator
 ADD A,R2 ;Add the value of R2
 SUBB A,R5 ;Subtract the value of R5 (which now contains R3 +
 R4)
```

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.

The Data Pointer (DPTR)

- The Data Pointer (DPTR) is the 8051's only user-accessable 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.
- DPTR is most often used to point to data in external memory.

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

RESET Value of Some 8051 Registers:

Register	Reset Value
PC	0000
ACC	0000
B	0000
PSW	0000
SP	0007
DPTR	0000

RAM are all zero

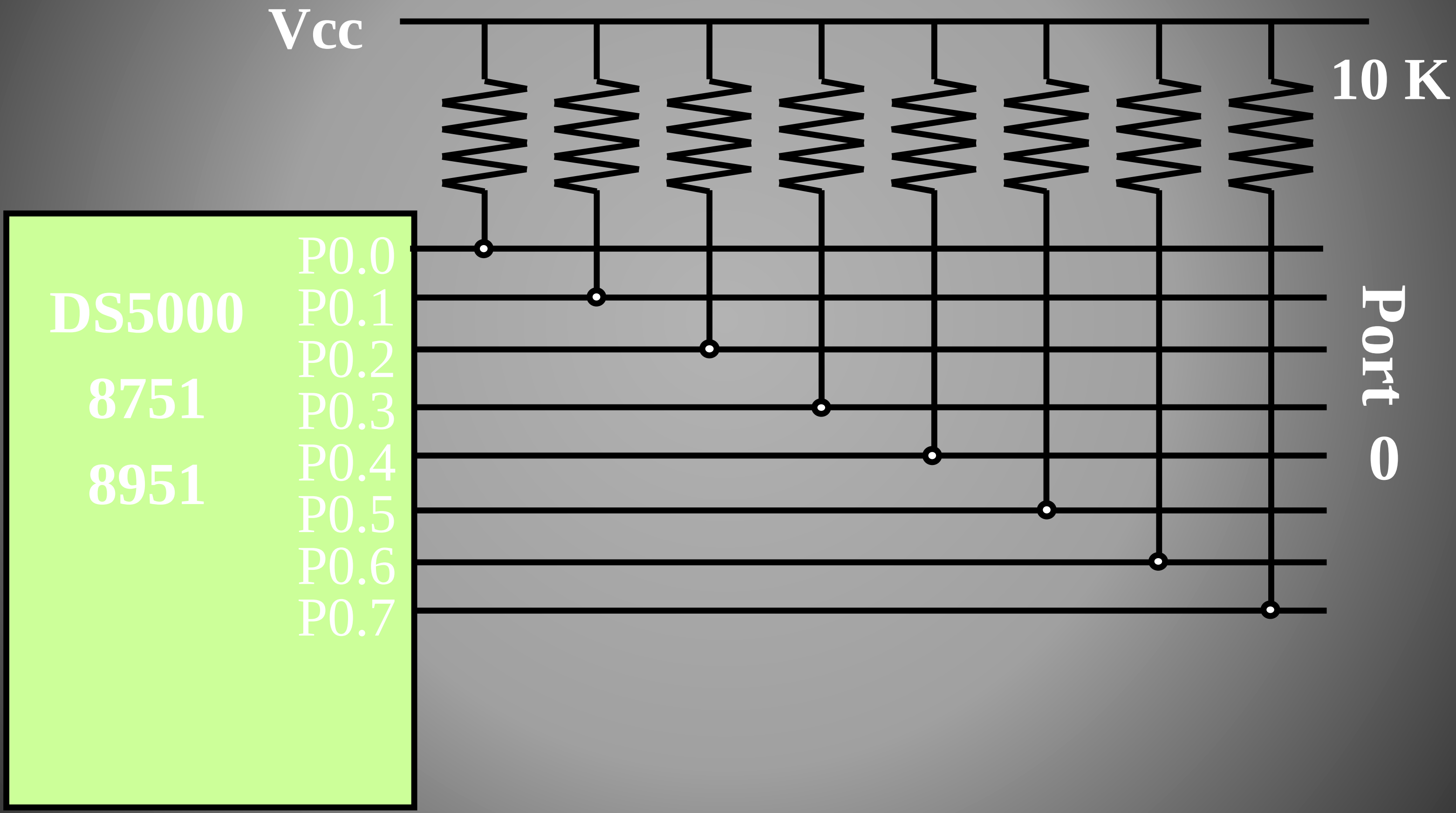
I/O Ports

- **Port 0（pins 32-39）**
- When connecting an 8051 to an external memory, the 8051 uses ports to send addresses and read instructions.
  - 16-bit address：P0 provides both address A0-A7, P2 provides address A8-A15.
  - Also, P0 provides data lines D0-D7.
- When P0 is used for address/data multiplexing, it is connected to the 74LS373 to latch the address.
- **Port 1（pins 1-8）**
- Port 1 is denoted by P1.
  - P1.0 ~ P1.7
  - P1 as an output port (i.e., write CPU data to the external pin)
  - P1 as an input port (i.e., read pin data into CPU bus)

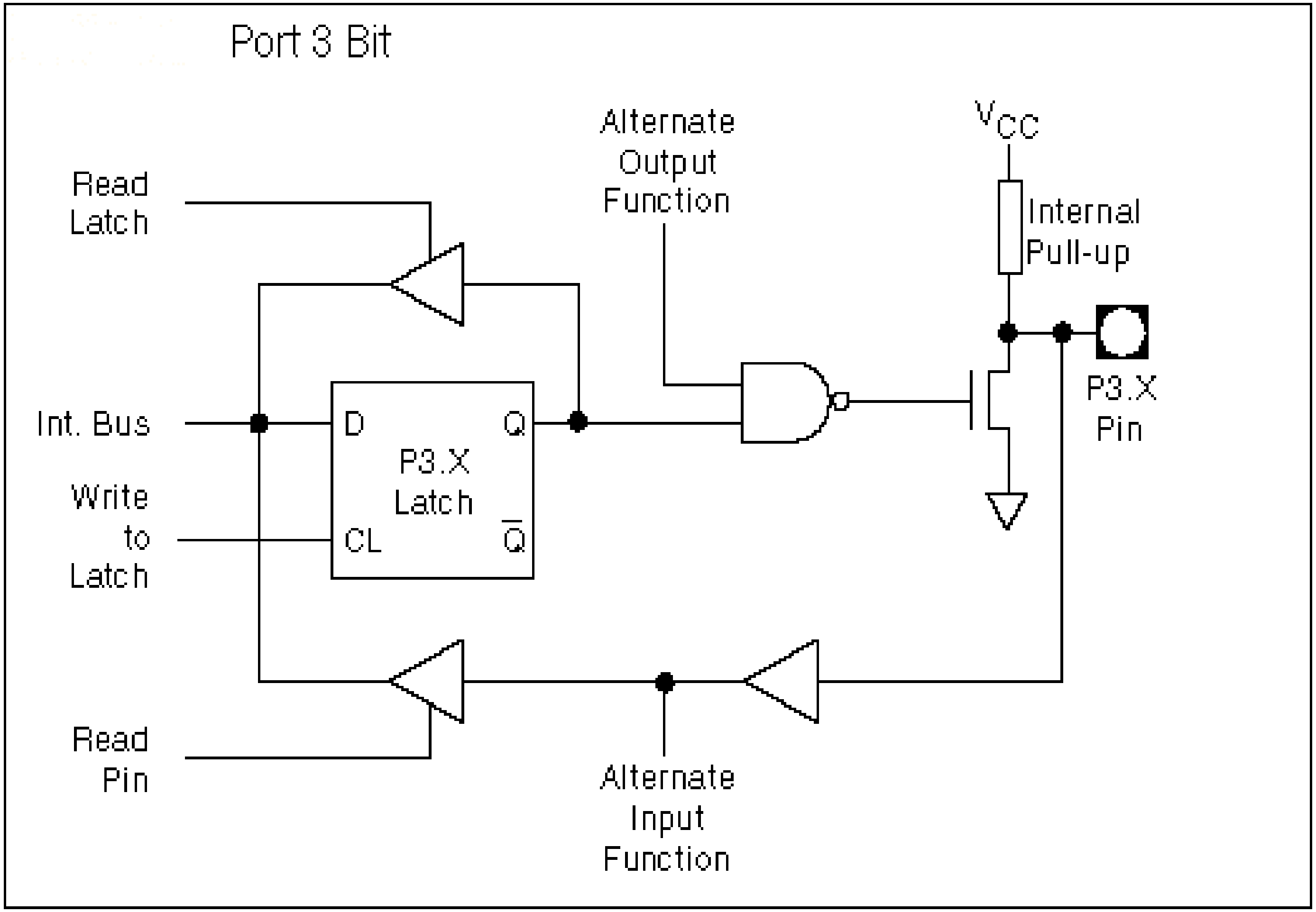
Port 3（pins 10-17）

- Although port 3 is configured as an output port upon reset, this is not the way it is most commonly used.
- Port 3 has the additional function of providing signals.
  - Serial communications signal：RxD, TxD
  - External interrupt：/INT0, /INT1
  - Timer/counter：T0, T1
  - External memory accesses ：/WR, /RD

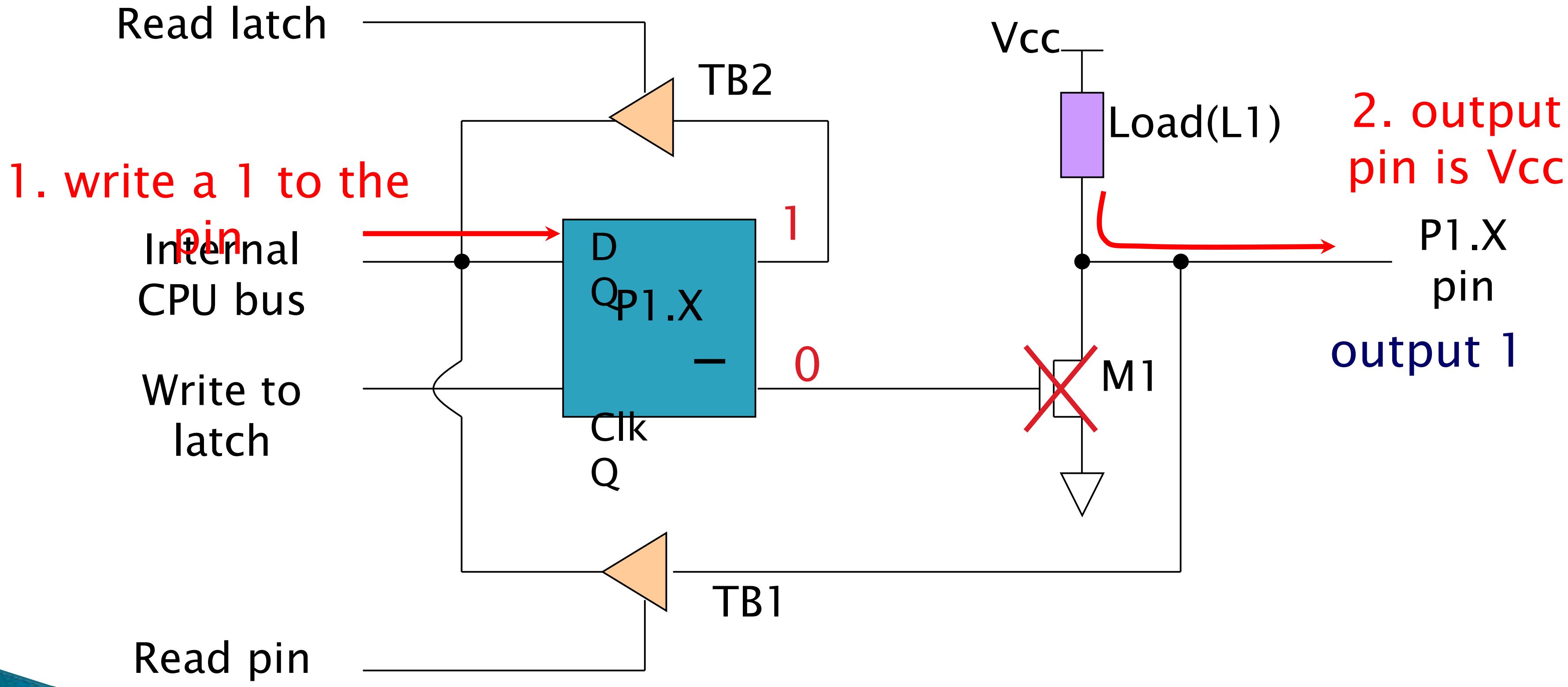
Port 0 with Pull-Up Resistors



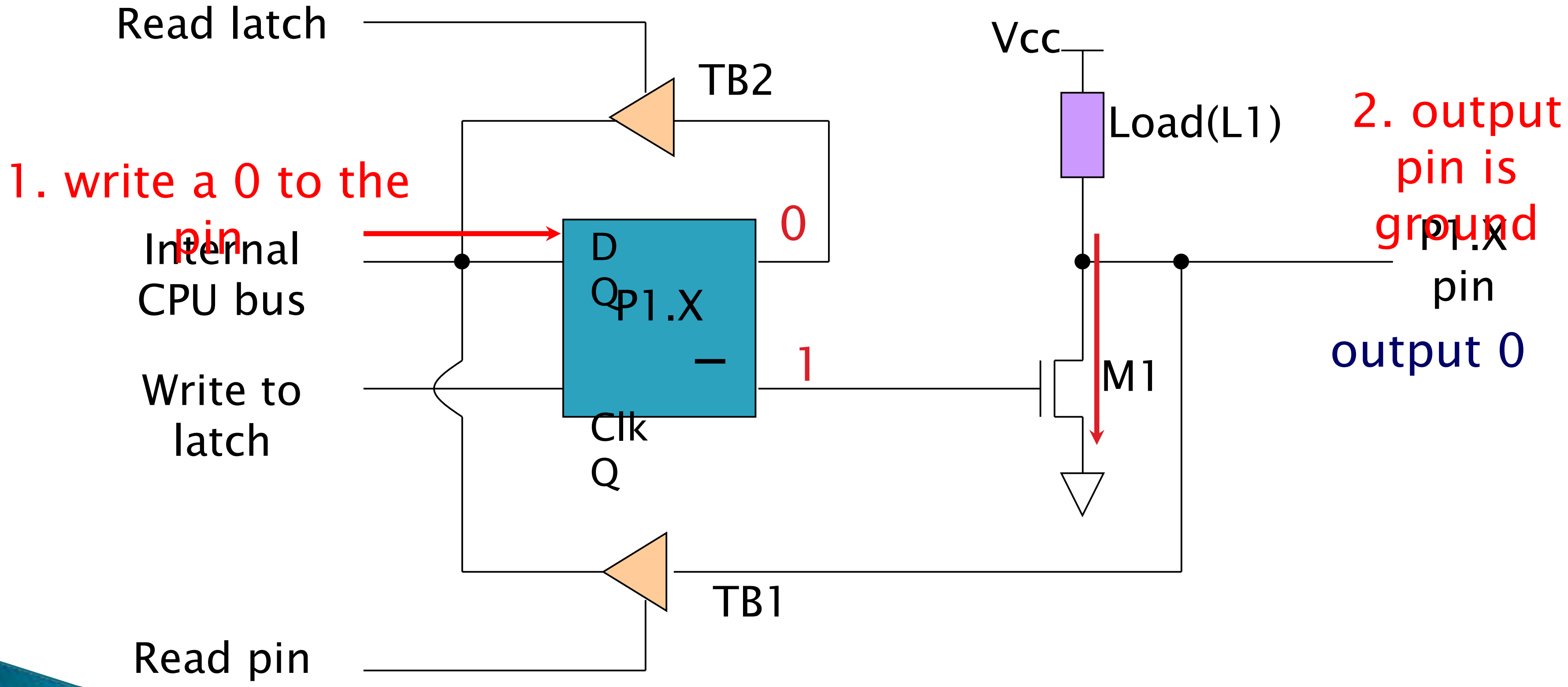
8051 Port 3 Bit Latches and I/O Buffers



Writing “1” to Output Pin P1.X



Writing "0" to Output Pin P1.X

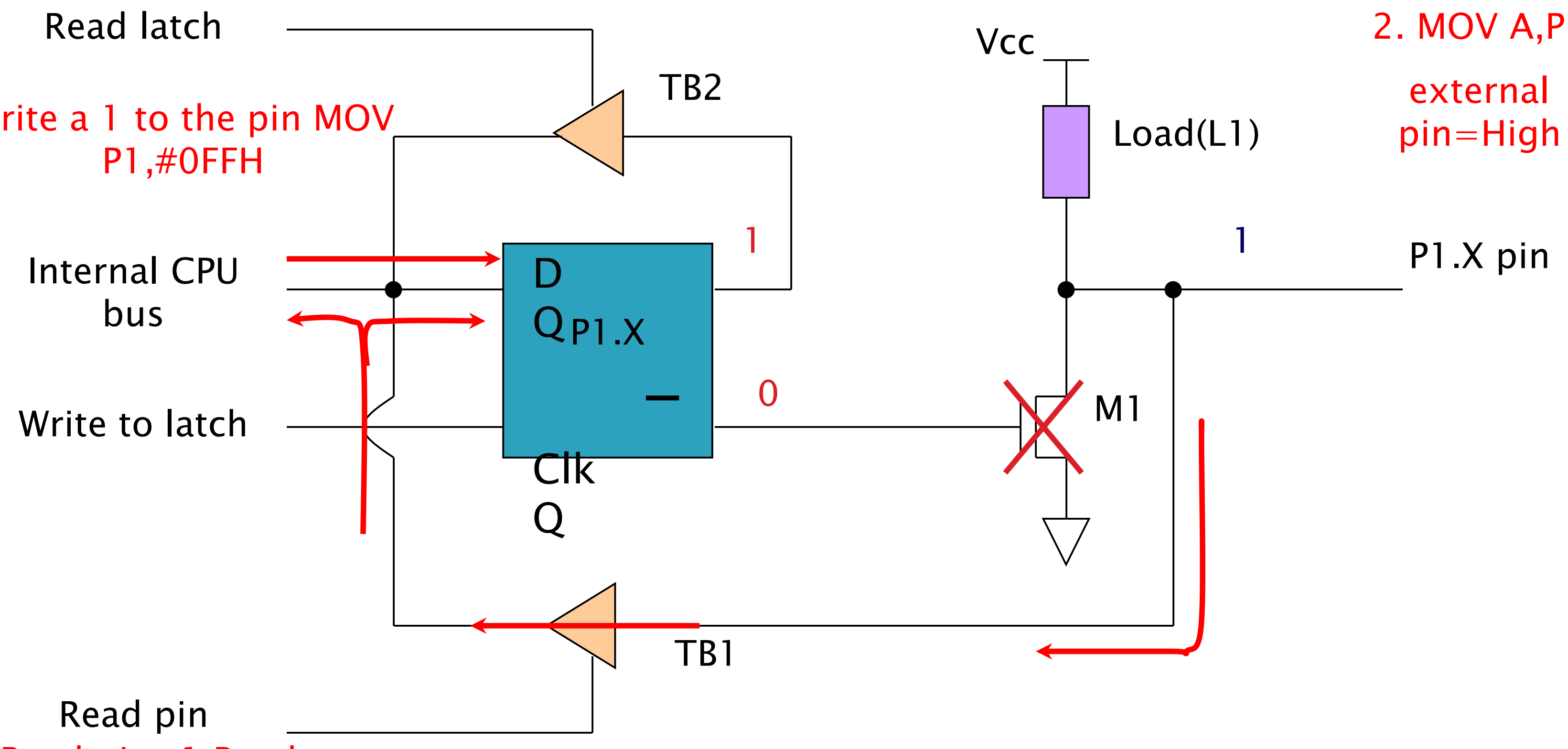


1. write a 0 to the pin

2. output pin is ground

output 0

Reading "High" at Input Pin



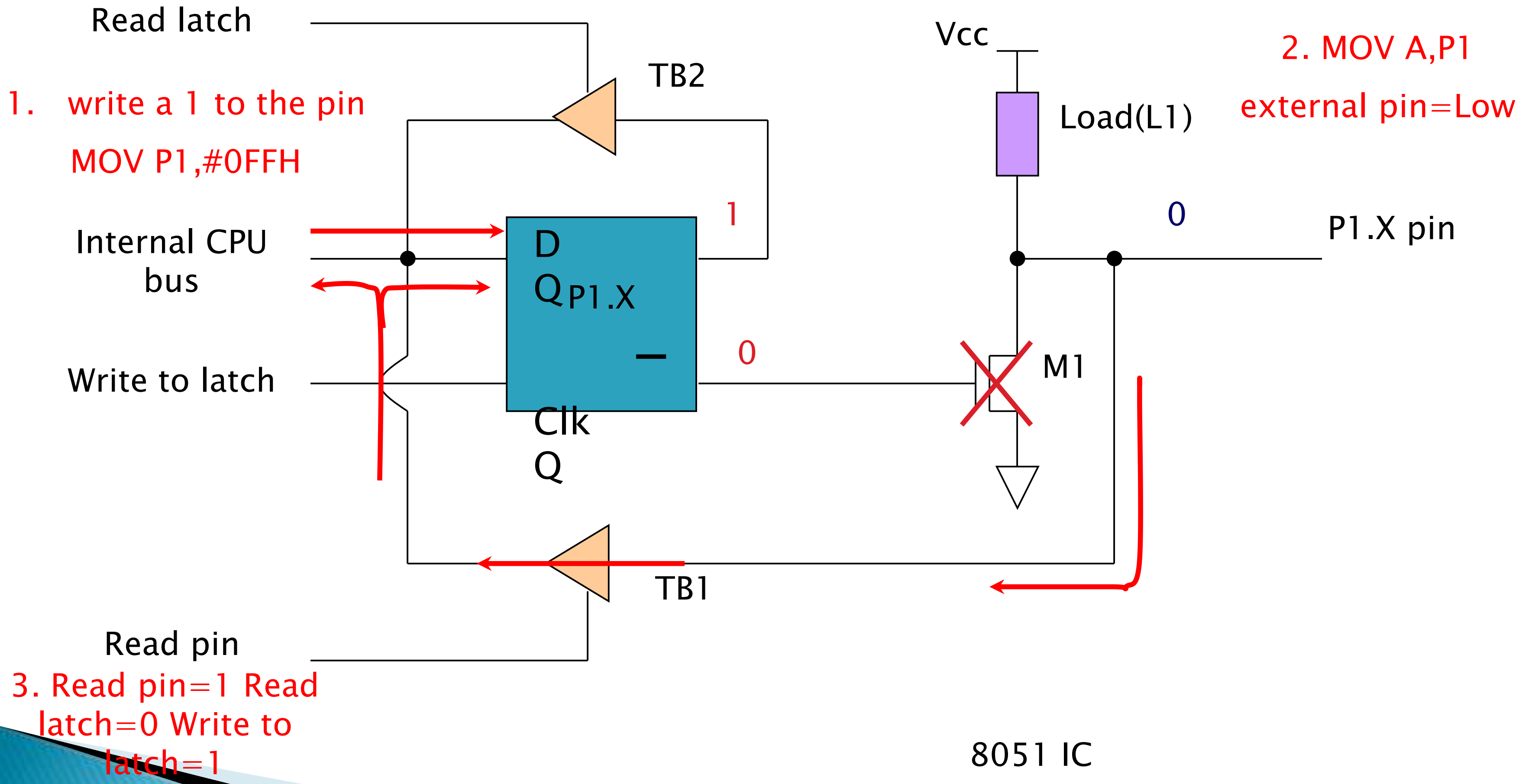
1. write a 1 to the pin MOV P1,#0FFH

2. MOV A,P1

external pin=High

3. Read pin=1 Read latch=0 Write to latch=1

Reading "Low" at Input Pin



1. write a 1 to the pin

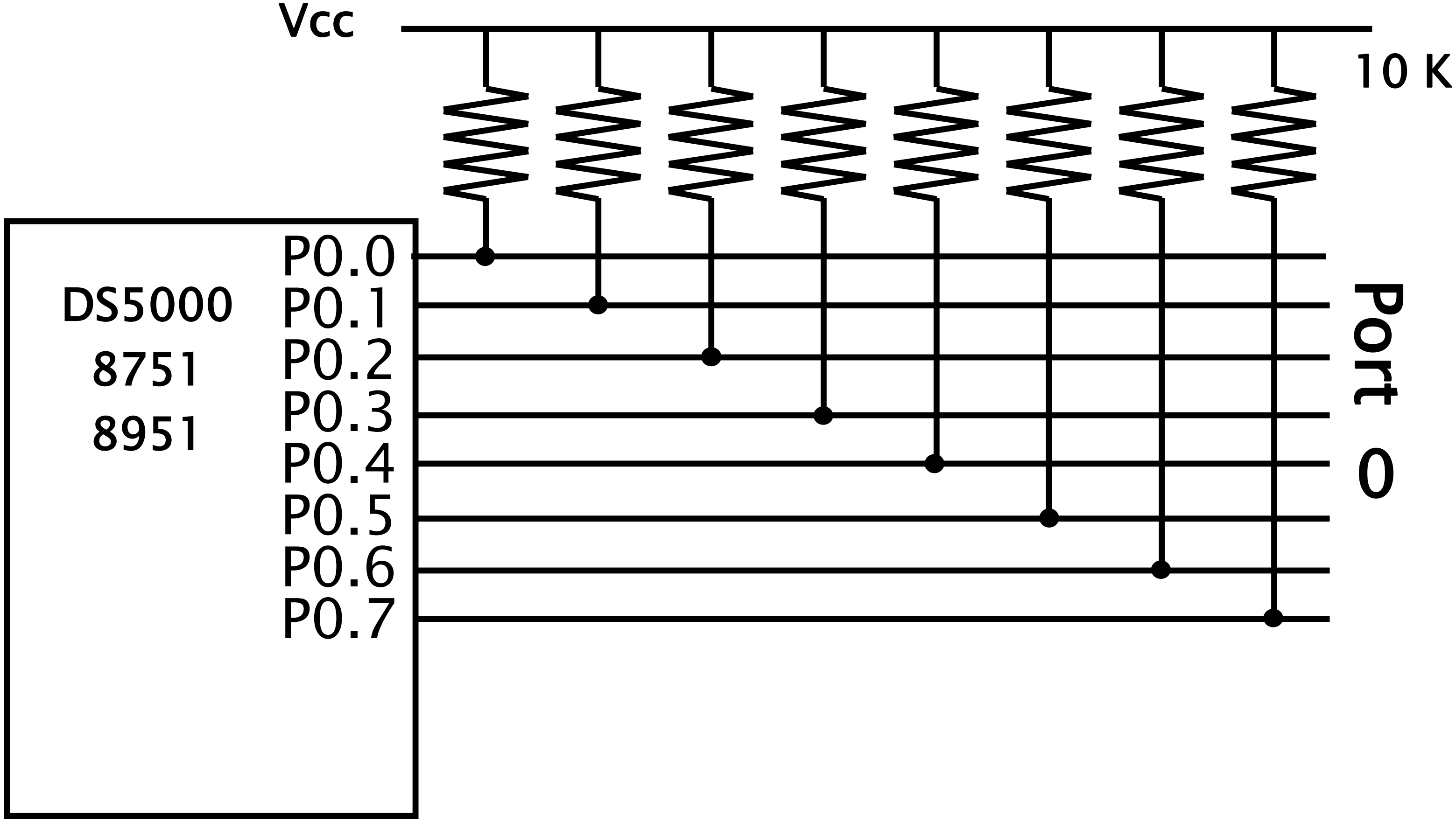
MOV P1,#0FFH

2. MOV A,P1

external pin=Low

3. Read pin=1 Read latch=0 Write to latch=1

Port 0 with Pull-Up Resistors



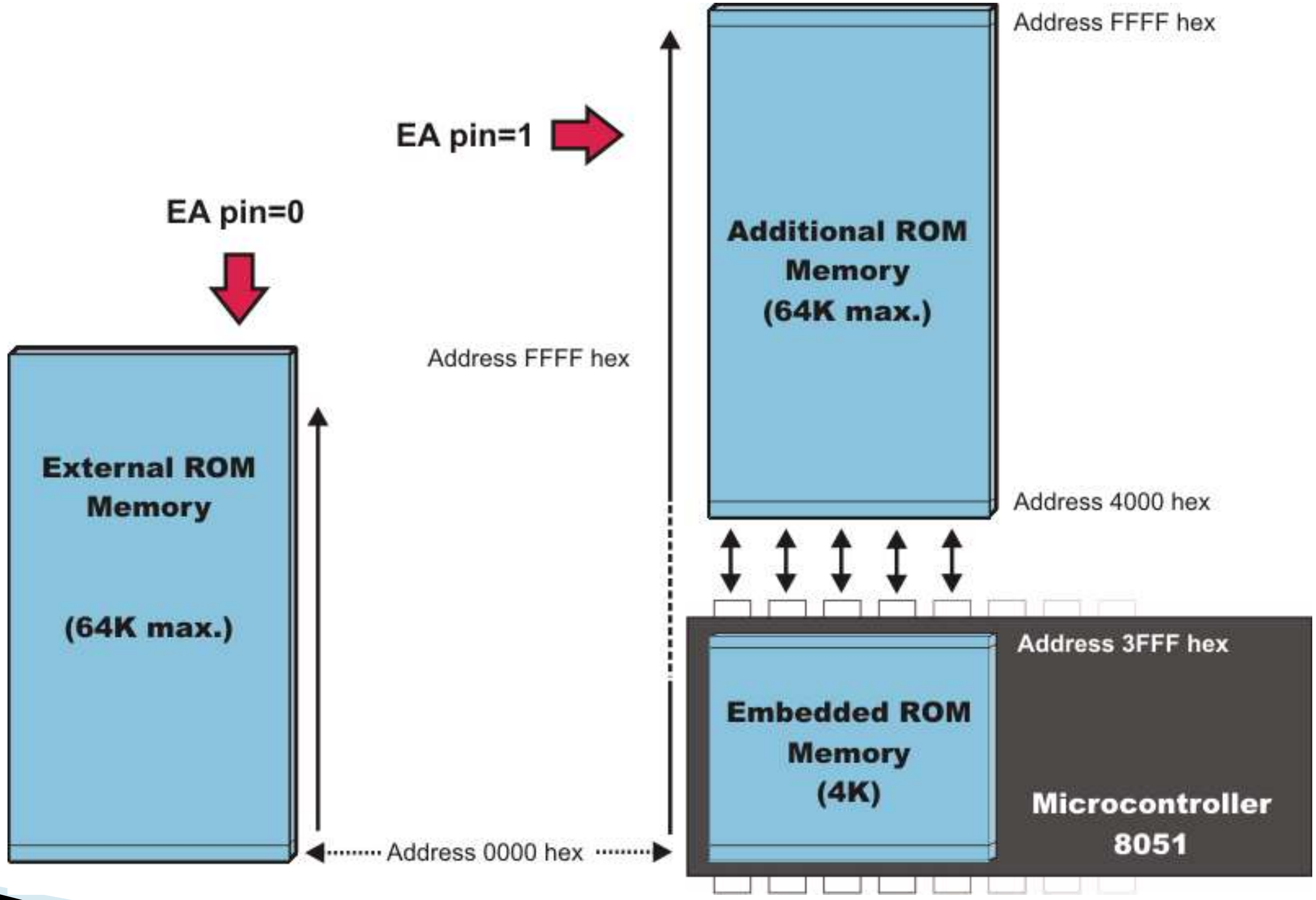
8051 Memory Organization

- The 8051 microcontroller's memory is divided into Program Memory and Data Memory.
- Program Memory (ROM) is used for permanent saving program being executed,
- Data Memory (RAM) is used for temporarily storing and keeping intermediate results and variables.

Program Memory (ROM)/Code Memory

- Program Memory (ROM) is used for permanent saving program (CODE) being executed. The memory is read only.
- Depending on the settings made in compiler, program memory may also used to store a constant variables. The 8051 executes programs stored in program memory only.
- Code memory type specifier is used to refer to program memory.
- 8051 memory organization allows external program memory to be added.
- How does the microcontroller handle external memory depends on the **pin EA logical state.**

Program Memory

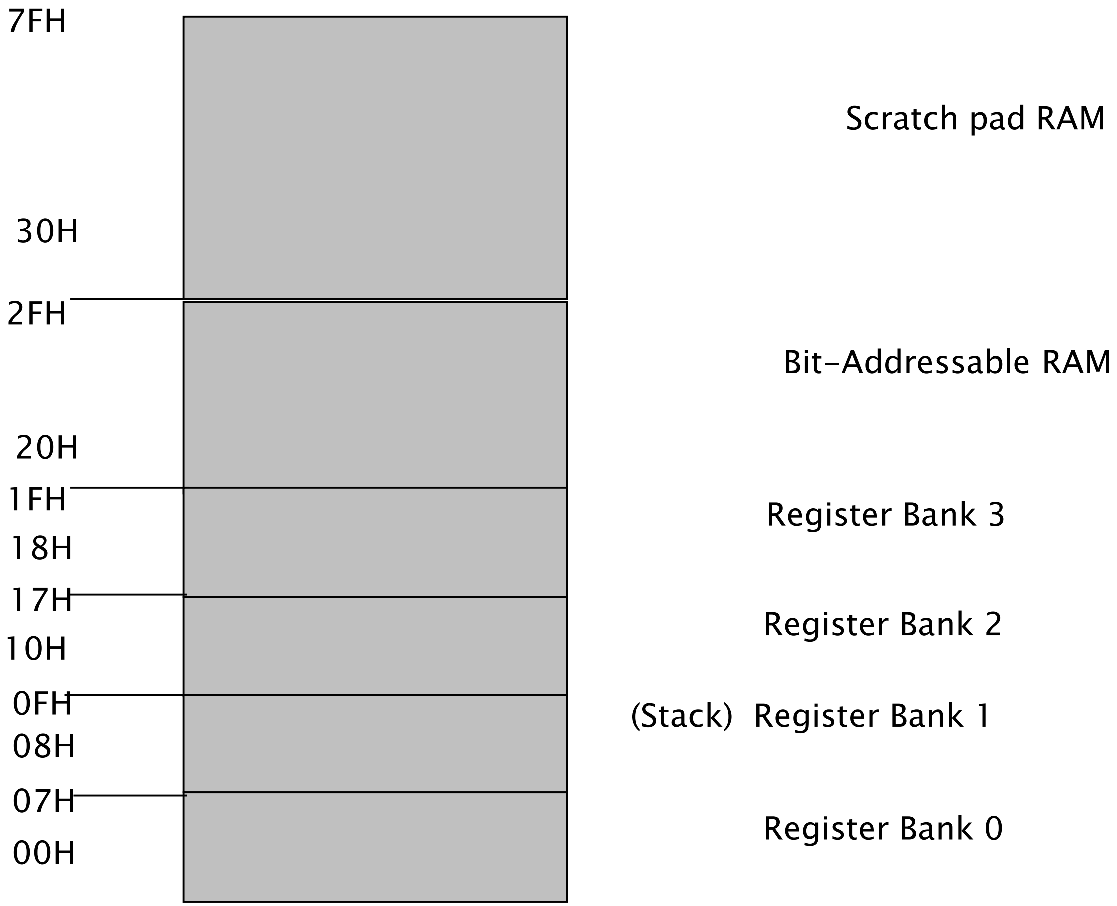


Internal Data Memory

- Up to 256 bytes of internal data memory is available
- Locations available to the user occupy addressing space from 0 to 7Fh, i.e. first 128 registers and this part of RAM is divided in several blocks.
- The first 128 bytes of internal data memory are both directly and indirectly addressable.
- The upper 128 bytes of data memory (from 0x80 to 0xFF) can be addressed only indirectly.

Memory Organization

- RAM memory space allocation in the 8051



7FH

Scratch pad RAM

30H

2FH

Bit-Addressable RAM

20H

1FH

Register Bank 3

18H

17H

Register Bank 2

10H

0FH

(Stack) Register Bank 1

08H

07H

Register Bank 0

00H

Bit Addressable Memory

- Memory block in the range of 20h to 2Fh is bit-addressable, which means that each bit being there has its own address from 0 to 7Fh.
- Since there are 16 such registers, this block contains in total of 128 bits with separate addresses (Bit 0 of byte 20h has the bit address 0, and bit 7 of byte 2Fh has the bit address 7Fh).

External Data Memory

- Access to external memory is slower than access to internal data memory.
- There may be up to 64K Bytes of external data memory.
- Several 8051 devices provide on-chip XRAM space that is accessed with the same instructions as the traditional external data space.
- This XRAM space is typically enabled via proper setting of SFR register and overlaps the external memory space.
- Setting of that register must be manually done in code, before any access to external memory or XRAM space is made.

Address Multiplexing for External Program/Code Memory

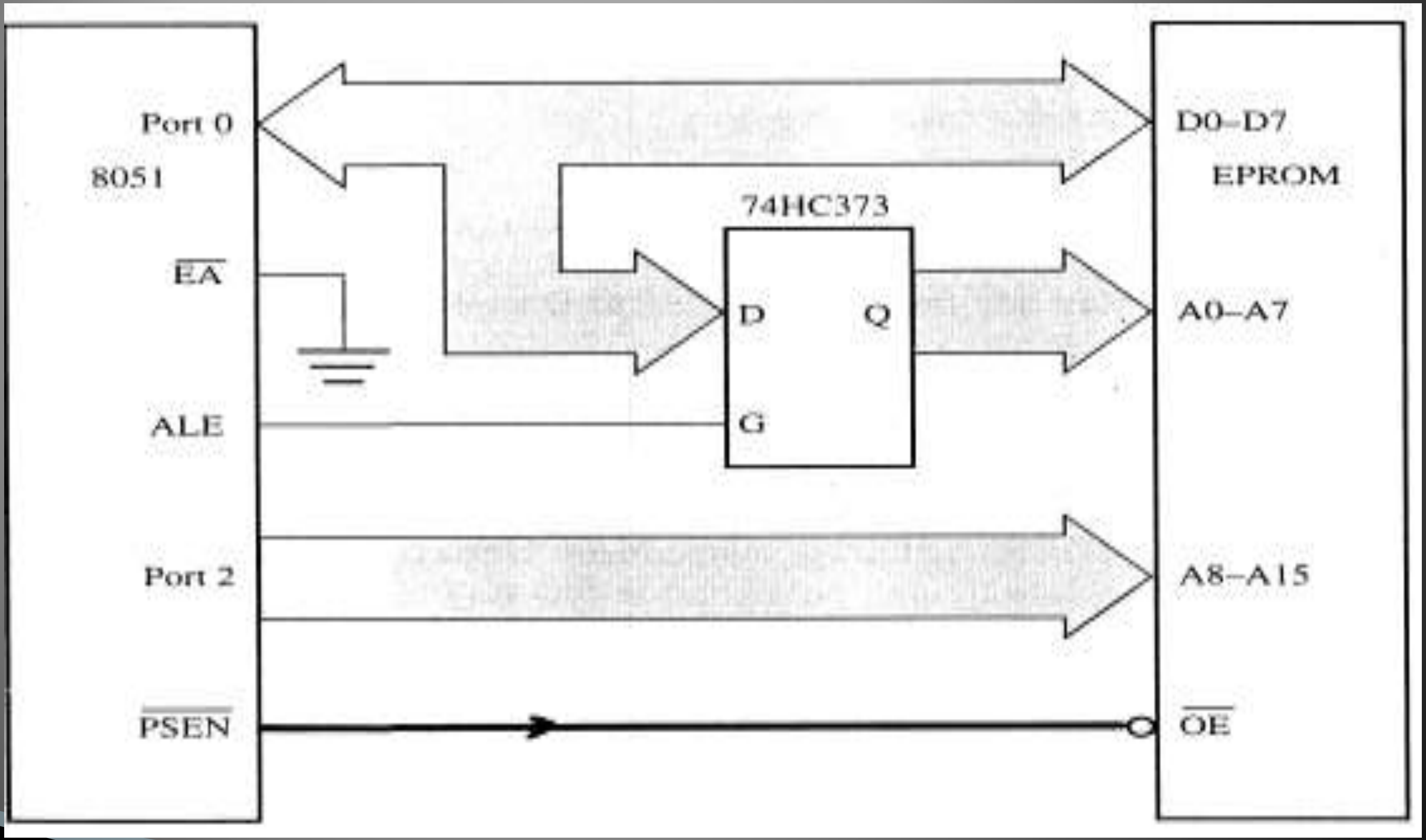


Figure 2-8

Accessing external code memory

Accessing External Data Memory

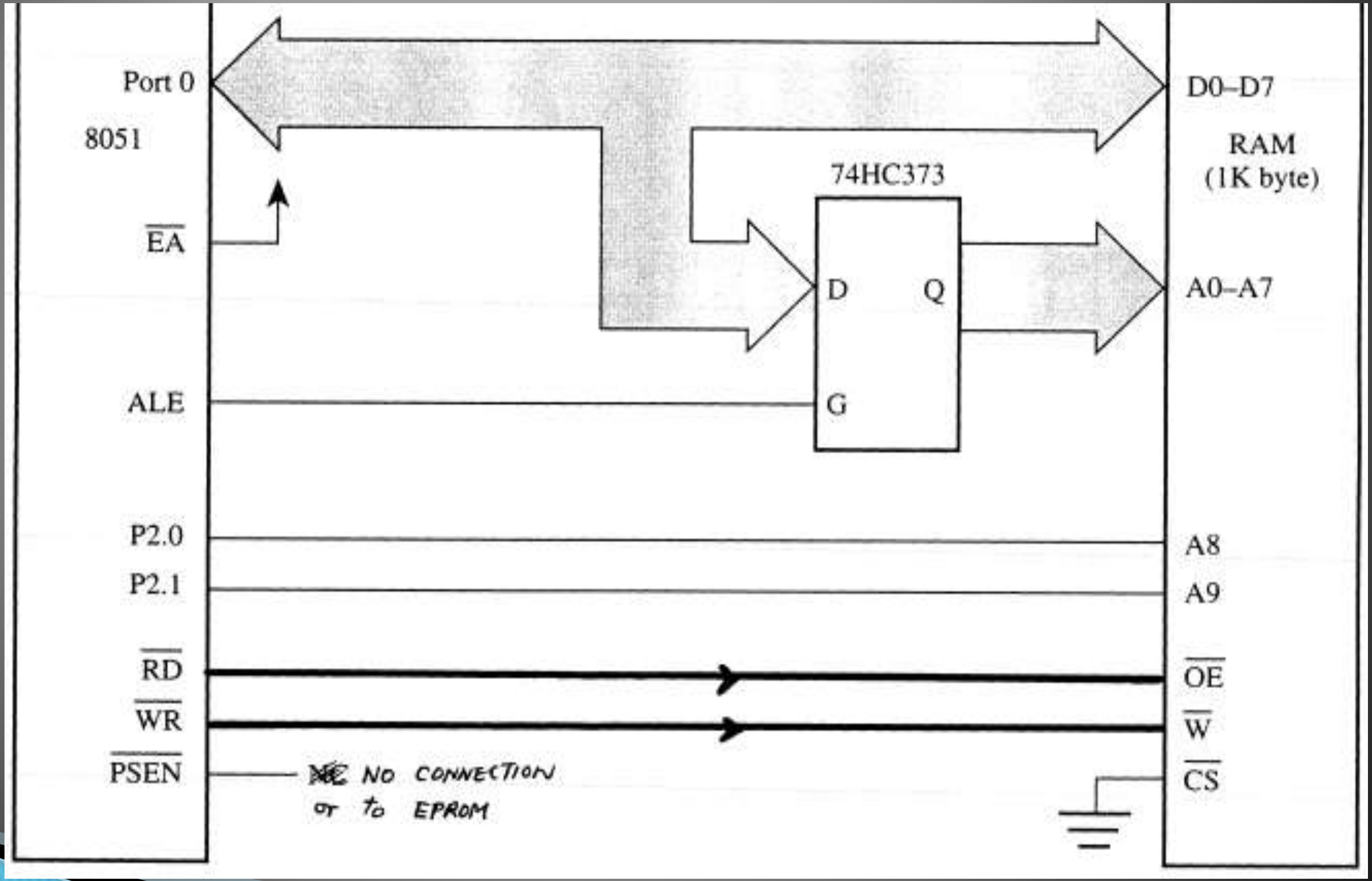


Figure 2-11

Interface to 1K RAM

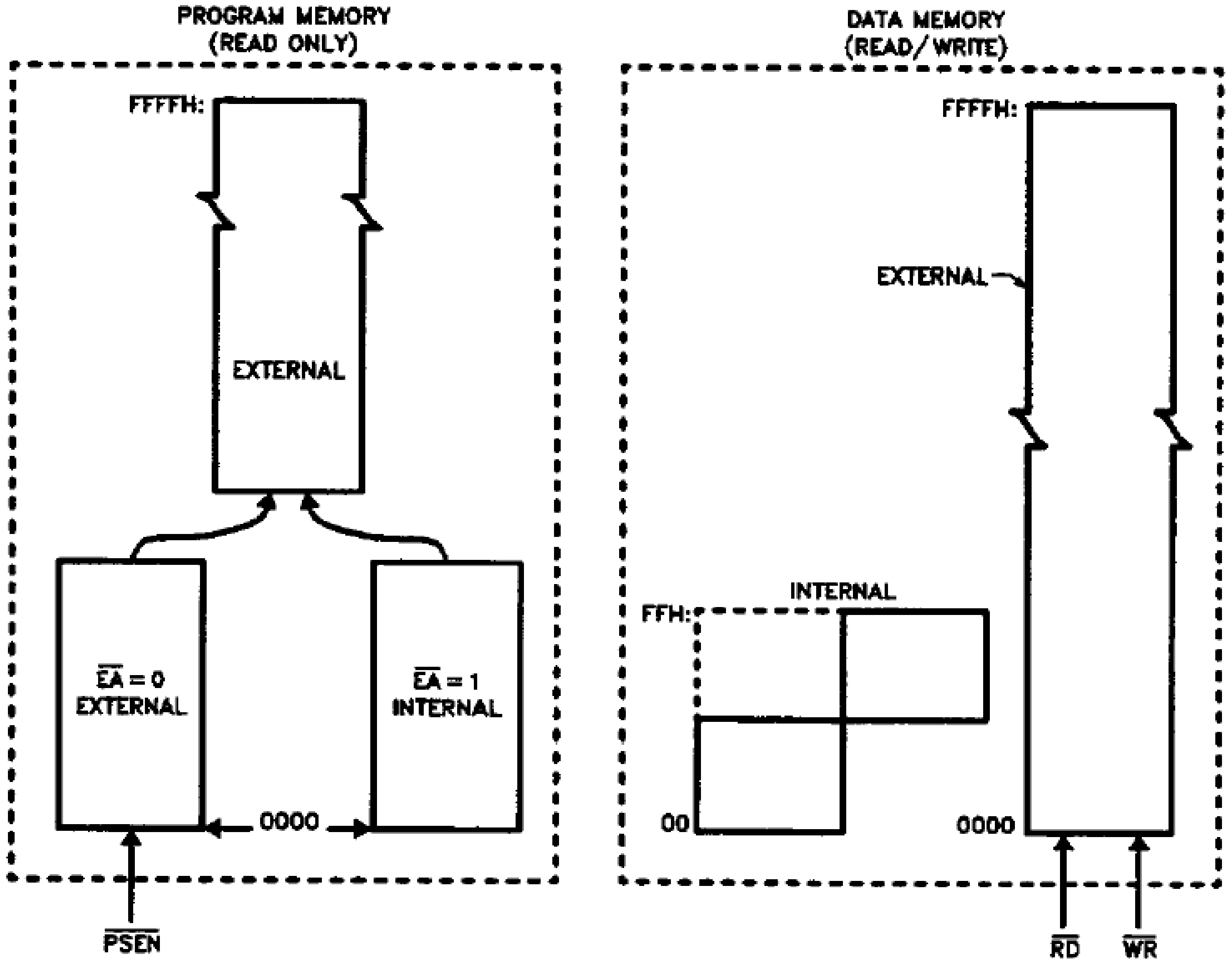


Figure 2. MCS®-51 Memory Structure

Special Function Registers

- SFRs which are also bit addressable

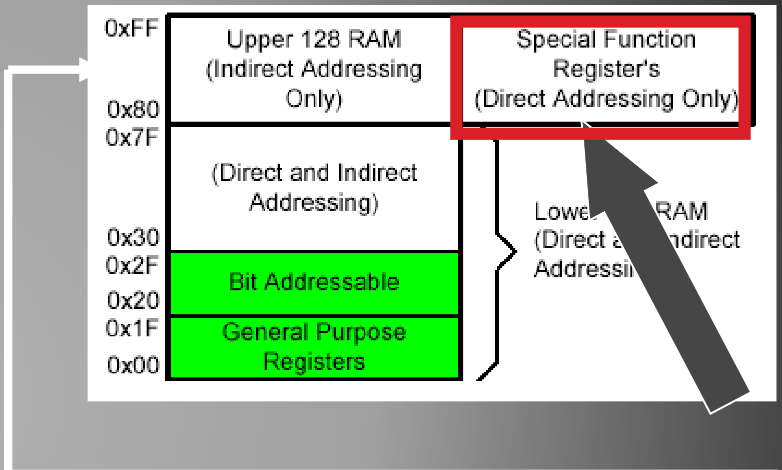
  A, B, IP, IE, TCON, SCON, PSW, P0, P1, P2, P3
- Other SFRs

  TMOD, THO, TLO, TH1, TL1, SBUF, PCON, SP, DPTR

Special Function Registers

- ❑DATA registers

- ❑CONTROL registers
  - ✣Timers
  - ✣Serial ports
  - ✣Interrupt system
  - ✣Analog to Digital converter
  - ✣Digital to Analog converter
  - ✣Etc.


0xFF
0x80
0x7F
0x30
0x2F
0x20
0x1F
0x00
Upper 128 RAM (Indirect Addressing Only)
Special Function Register's (Direct Addressing Only)
(Direct and Indirect Addressing)
Bit Addressable
General Purpose Registers
Lowe RAM (Direct a ndirect Addressi

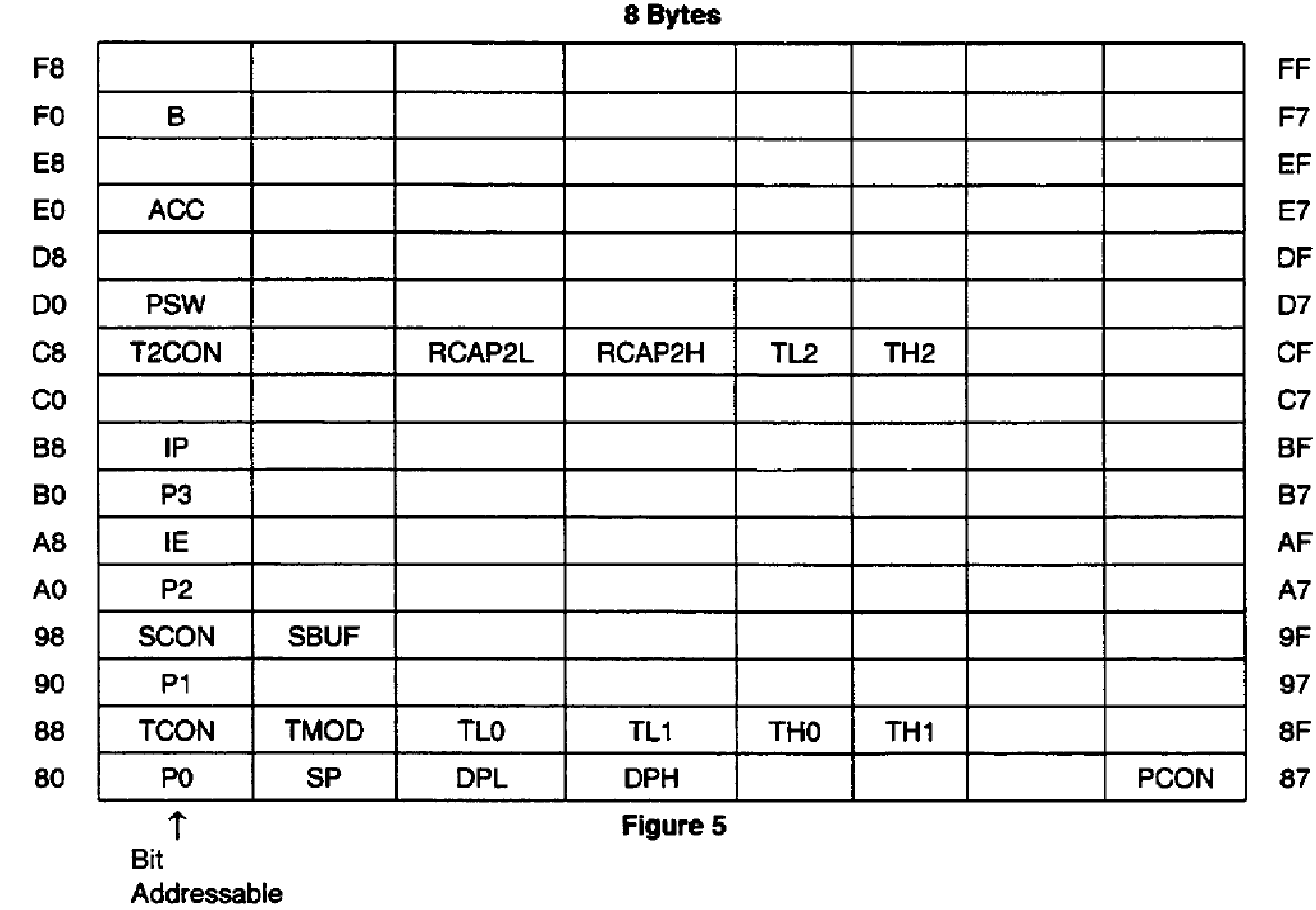
Addresses 80h – FFh

Direct Addressing used to access SPRs

Special Function Registers

Name	Function	Name	Function
A	Accumulator	SBUF	Serial Port data buffer
B	Arithmetic	SP	Stack Pointer
DPH	Addressing Ext Memory	TMOD	Timer/Counter mode cntrl
DPL	Addressing Ext Memory	TCON	Timer/Counter cntrl
IE	Interrupt enable	TL0	Timer0 lower byte
IP	Interrupt Priority	TH0	Timer0 higher byte
P0	I/O Port Latch	TL1	Timer1 lower byte
P1	I/O Port Latch	TH1	Timer1 higher byte
P2	I/O Port Latch		
P3	I/O Port Latch		
PCON	Power Control		
PSW	Pgm Status		

SFR MEMORY MAP

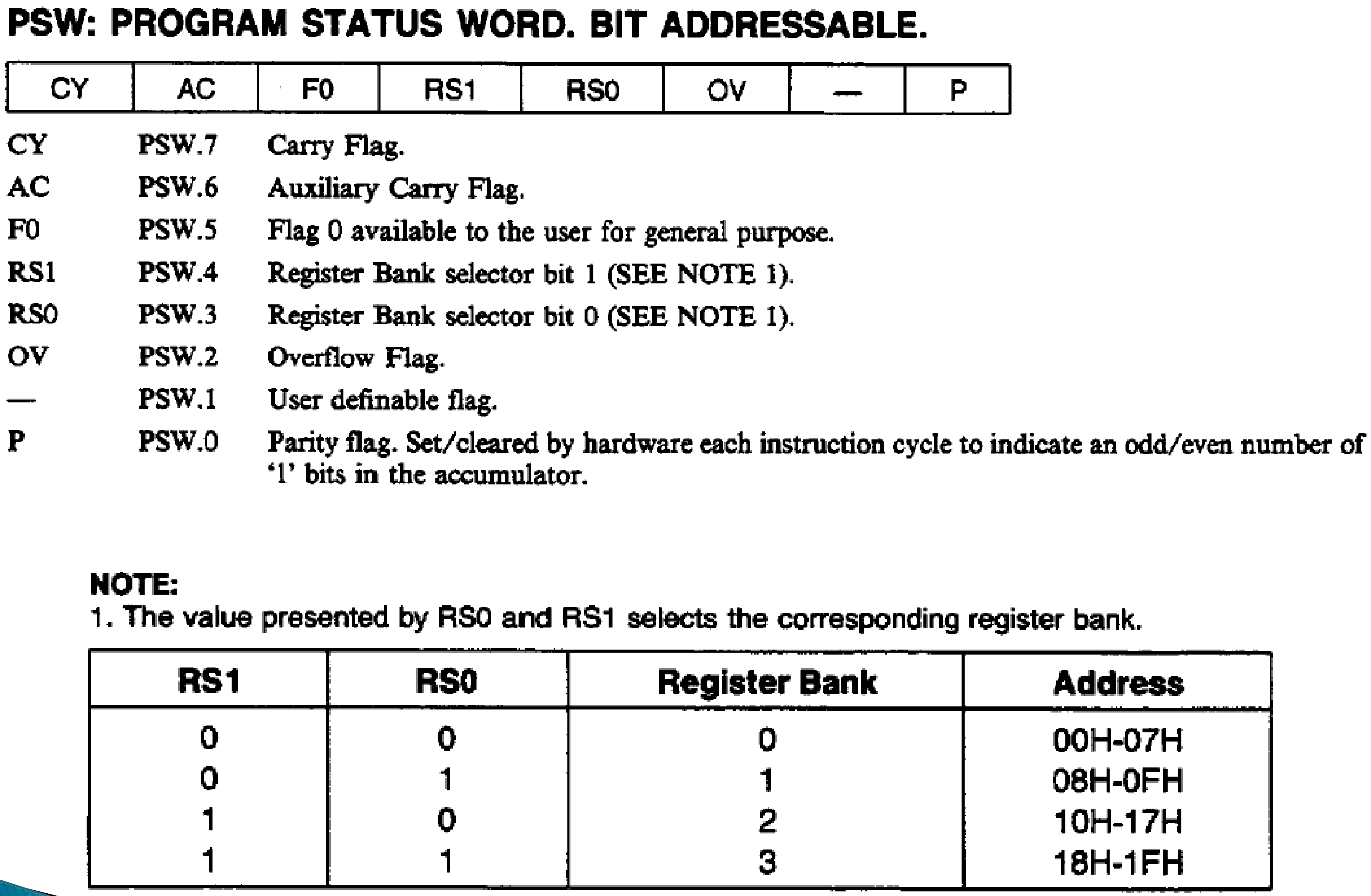


8 Bytes

F8									FF
F0	B								F7
E8									EF
E0	ACC								E7
D8									DF
D0	PSW								D7
C8	T2CON		RCAP2L	RCAP2H	TL2	TH2			CF
C0									C7
B8	IP								BF
B0	P3								B7
A8	IE								AF
A0	P2								A7
98	SCON	SBUF							9F
90	P1								97
88	TCON	TMOD	TL0	TL1	TH0	TH1			8F
80	P0	SP	DPL	DPH				PCON	87

↑
Bit
Addressable

Figure 5



PSW: PROGRAM STATUS WORD. BIT ADDRESSABLE.

CY	AC	F0	RS1	RS0	OV	—	P

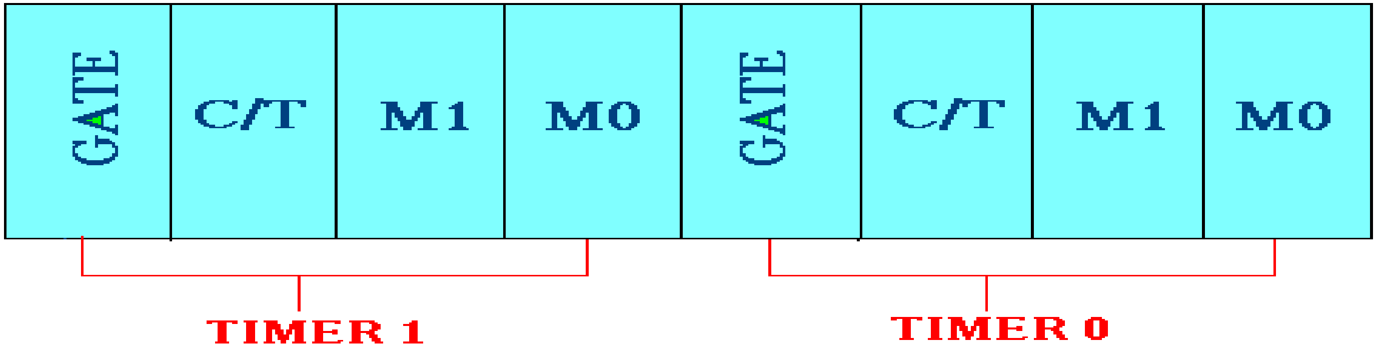
CY	PSW.7	Carry Flag.
AC	PSW.6	Auxiliary Carry Flag.
F0	PSW.5	Flag 0 available to the user for general purpose.
RS1	PSW.4	Register Bank selector bit 1 (SEE NOTE 1).
RS0	PSW.3	Register Bank selector bit 0 (SEE NOTE 1).
OV	PSW.2	Overflow Flag.
—	PSW.1	User definable flag.
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of '1' bits in the accumulator.

NOTE:

1. The value presented by RS0 and RS1 selects the corresponding register bank.

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

TMOD Register:



GATE	C/T	M1	M0	GATE	C/T	M1	M0

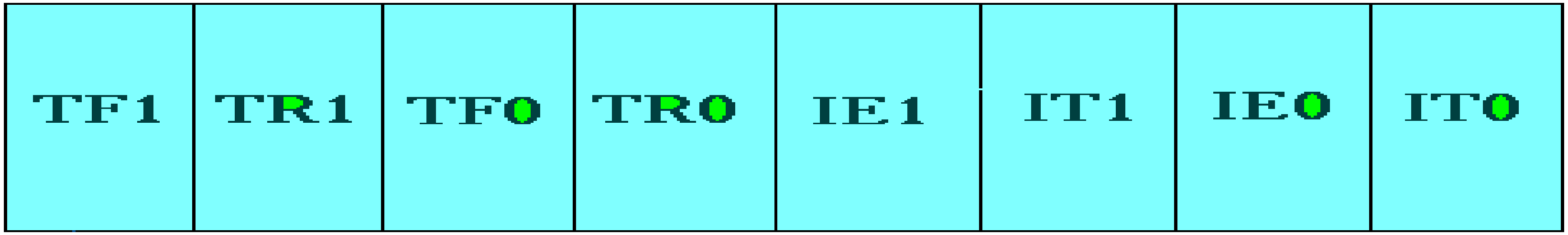
TIMER 1

TIMER 0

- **Gate :** When set, timer only runs while INT(0,1) is high.
- **C/T :** Counter/Timer select bit.
- **M1 :** Mode bit 1.
- **M0 :** Mode bit 0.

M1	M0	MODE
0	0	13-bit timer mode
0	1	16-bit timer mode
1	0	8-bit auto-reload mode
1	1	split mode

TCON Register:



TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

- TF1: Timer 1 overflow flag.
- TR1: Timer 1 run control bit.
- TF0: Timer 0 overflag.
- TR0: Timer 0 run control bit.
- IE1: External interrupt 1 edge flag.
- IT1: External interrupt 1 type flag.
- IE0: External interrupt 0 edge flag.
- IT0: External interrupt 0 type flag.

CHAPTER 7

Instruction Set and Addressing Modes of 8051

Addressing Modes

The way in which the instruction is specified.

- Immediate
- Register
- Direct
- Register Indirect
- Indexed

Immediate Addressing Mode

- Immediate Data is specified in the instruction itself

- Egs:

```
MOV  A,#65H
MOV  A,#’A’
MOV  R6,#65H
MOV  DPTR,#2343H
MOV  P1,#65H
```

Register Addressing Mode

MOV Rn, A ;n=0,..,7

ADD A, Rn

MOV DPL, R6

~~MOV DPTR, A~~

~~MOV Rm, Rn~~

Direct Addressing Mode

Although the entire of 128 bytes of RAM can be accessed using direct addressing mode, it is most often used to access RAM loc. 30 – 7FH.

```
MOV  R0, 40H
MOV  56H, A
MOV  A, 4              ; ≡ MOV A, R4
MOV  6, 2              ; copy R2 to R6
                       ; MOV  R6,R2 is invalid !
```

Register Indirect Addressing Mode

- In this mode, register is used as a pointer to the data.

MOV A,@Ri

; move content of RAM loc. Where address is held by Ri into A

(i=0 or 1)

MOV @R1,B

In other word, the content of register R0 or R1 is sources or target in MOV, ADD and SUBB insructions.

⌦ jump

Indexed Addressing Mode And On-Chip ROM Access

- This mode is widely used in accessing data elements of look-up table entries located in the program (code) space ROM at the 8051

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

A= content of address A +DPTR from ROM

Note:

Because the data elements are stored in the program (code) space ROM of the 8051, it uses the instruction MOVC instead of MOV. The “C” means code.

The 8051
Assembly Language

Overview

- Data transfer instructions
- Addressing modes
- Data processing (arithmetic and logic)
- Program flow instructions

Data Transfer Instructions

- MOV dest, source   dest □   source
- Stack instructions

```
PUSH byte        ;increment stack pointer,
                       ;move byte on stack
POP byte         ;move from stack to byte,
                       ;decrement stack pointer
```

- Exchange instructions

```
XCH a, byte      ;exchange accumulator and byte
XCHD a, byte     ;exchange low nibbles of
                       ;accumulator and byte
```

Addressing Modes

Immediate Mode – specify data by its value

```
mov A, #0          ;put 0 in the accumulator
                   ;A = 00000000
mov R4, #11h       ;put 11hex in the R4 register
                    ;R4 = 00010001
mov B, #11          ;put 11 decimal in b register
                    ;B = 00001011
mov DPTR,#7521h   ;put 7521 hex in DPTR
                   ;DPTR = 0111010100100001
```

Addressing Modes

Immediate Mode – continue

```
        MOV DPTR,#7521h
        MOV DPL,#21H
        MOV DPH, #75
```

```
        COUNT EGU 30
         ~
         ~
        mov R4, #COUNT
```

```
        MOV DPTR,#MYDATA
        ~
        ~
        0RG 200H
MYDATA:DB “IRAN”
```

Addressing Modes

- Register Addressing – either source or destination is one of CPU register

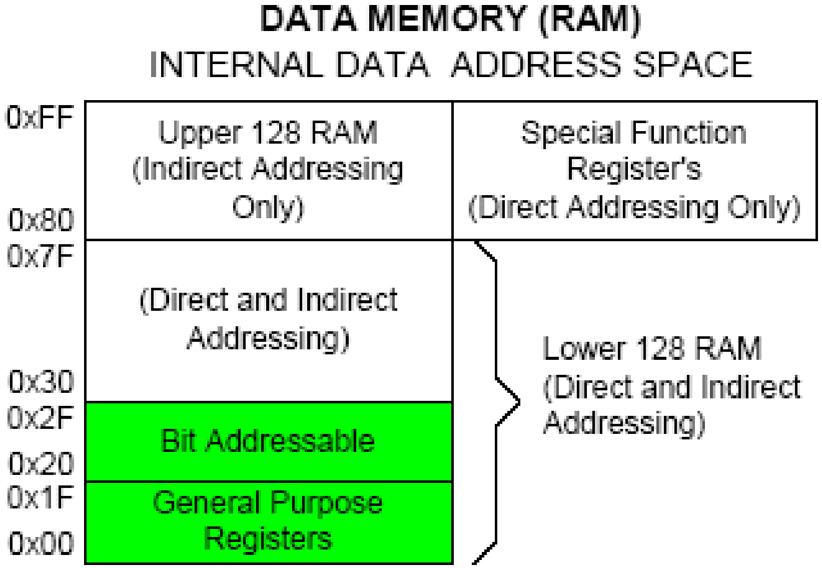
```
MOV R0,A
MOV A,R7
ADD A,R4
ADD A,R7
MOV DPTR,#25F5H
MOV R5,DPL
MOV R,DPH
```

Note that MOV R4,R7 is incorrect

Addressing Modes

Direct Mode – specify data by its 8-bit address
Usually for 30h-7Fh of RAM

```
Mov a, 70h          ; copy contents of RAM at 70h to a
Mov R0,40h          ; copy contents of RAM at 70h to a
Mov 56h,a           ; put contents of a at 56h to a
Mov 0D0h,a          ; put contents of a into PSW
```


DATA MEMORY (RAM)
INTERNAL DATA ADDRESS SPACE
0xFF
0x80
0x7F
0x30
0x2F
0x20
0x1F
0x00
Upper 128 RAM (Indirect Addressing Only)
Special Function Register's (Direct Addressing Only)
(Direct and Indirect Addressing)
Bit Addressable
General Purpose Registers
Lower 128 RAM (Direct and Indirect Addressing)

Addressing Modes

Direct Mode – play with R0-R7 by direct address

MOV A,4 ≡ MOV A,R4

MOV A,7 ≡ MOV A,R7

MOV 7,2 ≡ ~~MOV R7,R6~~

```
MOV R2,#5     ;Put 5 in R2
MOV R2,5      ;Put content of RAM at 5 in R2
```

Addressing Modes

Register Indirect – the address of the source or destination is specified in registers

Uses registers R0 or R1 for 8-bit address:

```
mov psw, #0          ; use register bank 0
mov r0, #0x3C
mov @r0, #3          ; memory at 3C gets #3
                     ; M[3C] □  3
```

Uses DPTR register for 16-bit addresses:

```
mov dptr, #0x9000    ; dptr □  9000h
movx a, @dptr        ; a □  M[9000]
```

Note that 9000 is an address in external memory

Addressing Modes

Register Indexed Mode – source or destination address is the sum of the base address and the accumulator(Index)

- Base address can be DPTR or PC

```
mov dptr, #4000h
mov a, #5
movc a, @a + dptr   ;a □  M[4005]
```

Acc Register

- A register can be accessed by direct and register mode

- This 3 instruction has same function with different code

```
0703 E500         mov a,00h
0705 8500E0       mov acc,00h
0708 8500E0       mov 0e0h,00h
```

- Also this 3 instruction

```
070B E9           mov a,r1
070C 89E0         mov acc,r1
070E 89E0         mov 0e0h,r1
```

SFRs Address

- B – always direct mode – except in MUL & DIV

```
0703 8500F0          mov b,00h
0706 8500F0          mov 0f0h,00h

0709 8CF0            mov b,r4
070B 8CF0            mov 0f0h,r4
```

- P0~P3 – are direct address

```
0704 F580          mov p0,a
0706 F580          mov 80h,a
0708 859080        mov p0,p1
```

- Also other SFRs (pcon, tmod, psw,….)

8051 Instruction Format

- immediate addressing

Op code	Immediate data

```
add a,#3dh          ;machine code=243d
```

- Direct addressing

Op code	Direct address

```
mov r3,0E8h         ;machine code=ABE8
```

Stack

- Stack-oriented data transfer
  - Only one operand (direct addressing)
  - SP is other operand – register indirect – implied
- Direct addressing mode must be used in Push and Pop

```
mov sp, #0x40    ; Initialize SP
push 0x55        ; SP □  SP+1, M[SP] □  M[55]
                 ; M[41] □  M[55]
pop b            ; b □  M[55]
```

Note: can only specify RAM or SFRs (direct mode) to push or pop. Therefore, to push/pop the accumulator, must use acc, not a

Stack (push,pop)

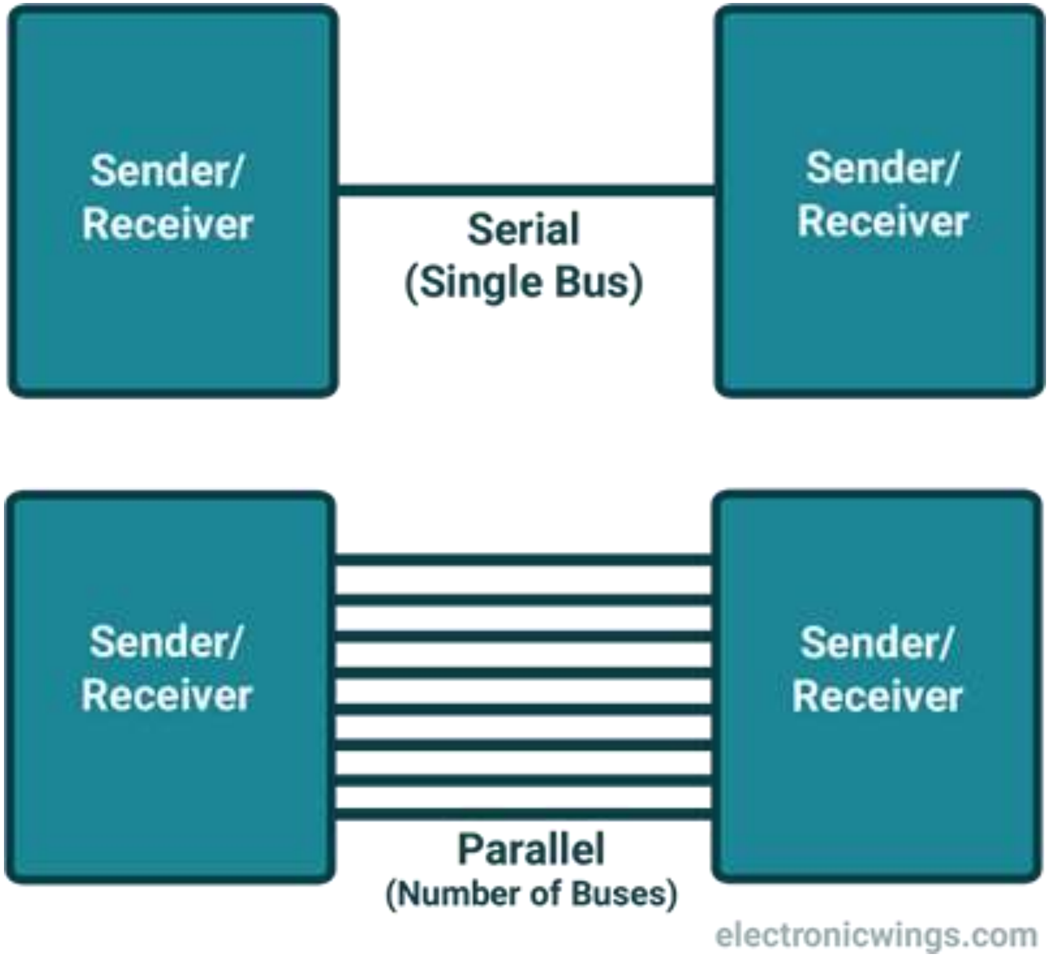
- Therefore

```
Push a    ;is invalid
Push r0   ;is invalid
Push r1   ;is invalid
push acc  ;is correct
Push psw  ;is correct
Push b    ;is correct
Push 13h
Push 0
Push 1
Pop  7
Pop  8
Push 0e0h  ;acc
Pop  0f0h  ;b
```

Serial Port operation

- Serial communication means transfer data bit by bit serially at a time, where as in parallel communication, the number of bits that can be transferred at a time depends upon the number of data lines available for communication.
- Two methods of serial communication are
- Synchronous Communication: Transfer of bulk data in framed structure at a time
- Asynchronous Communication: Transfer of a byte data in framed structure at a time
- 8051 has built in UART with RXD (serial data receive pin) and TXD (serial data transmit pin) on PORT3.0 and PORT3.1 respectively.

Communication Mode



Asynchronous communication

- Asynchronous serial communication is widely used for byte oriented transmission.
- Frame structure in Asynchronous communication:
- **START bit:** It is a bit with which serial communication start and it is always low.
- **Data bits packet**: Data bits can be 5 to 9 bits packet. Normally we use 8 data bit packet, which is always sent after START bit.
- **STOP bit**: This is one or two bits. It is sent after data bits packet to indicate end of frame. Stop bit is always logic high.
- In asynchronous serial communication frame, first START bit followed by data byte and at last STOP bit, forms a 10-bit frame. Sometimes last bit is also used as parity bit.

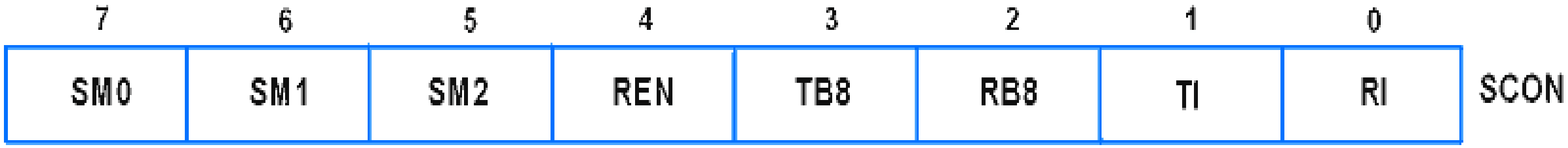
Data transmission rate

- Data transmission rate is measured in bits per second (bps). In binary system it is also called as baud rate (number of signal changes per second).
- Standard baud rates supported are 1200, 2400, 4800, 19200, 38400, 57600, and 115200. Normally most of the time 9600 bps is used when speed is not a big issue.

Serial communication Registers

- **SBUF: Serial Buffer Register**
- This is the serial communication data register used to transmit or receive data through it.
- **SCON: Serial Control Register**
- Serial control register SCON is used to set serial communication operation modes. Also it is used to control transmit and receive operations.

SCON



7	6	5	4	3	2	1	0	
SM0	SM1	SM2	REN	TB8	RB8	TI	RI	SCON

Mode	SM0	SM1	Mode
0	0	0	1/12 of Osc frequency shift register mode fixed baud rate
1	0	1	8-bit UART with timer 1 determined baud rate
2	1	0	9-bit UART with 1/32 of Osc fixed baud rate
3	1	1	9-bit UART with timer 1 determined baud rate

Bit 7:6 - SM0:SM1: Serial Mode Specifier

Normally mode-1 (SM0 =0, SM1=1) is used with 8 data bits, 1 start bit and 1 stop bit.

SCON

- **Bit 5 – SM2**: for Multiprocessor Communication
- This bit enables multiprocessor communication feature in mode 2 & 3.
- **Bit 4 – REN**: Receive Enable
- 1 = Receive enable
- 0 = Receive disable
- **Bit 3 – TB8**: 9th Transmit Bit
- This is 9th bit which is to be transmitted in mode 2 & 3 (9-bit mode)
- **Bit 2 – RB8**: 9th Receive Bit
- This is 9th received bit in mode 2 & 3 (9-bit mode), where as in mode 1, if SM2 = 0 then RB8 hold stop bit that received
- **Bit 1 – TI**: Transmit Interrupt Flag
- This bit indicates transmission is complete and gets set after transmitting the byte from buffer. Normally TI (Transmit Interrupt Flag) is set by hardware at the end of 8th bit in mode 0 and at the beginning of stop bit in other modes.
- **Bit 0 – RI**: Receive Interrupt Flag
- This bit indicates reception is complete and gets set after receiving the complete byte in buffer. Normally RI (Receive Interrupt Flag) is set by hardware in receiving mode at the end of 8th bit in mode 0 and at the stop bit receive time in other modes.
-

Timer Operations

- The 8051 has two counters/timers which can be used either as timer to generate a time delay or as counter to count events happening outside the microcontroller.
- he 8051 has two timers: timer0 and timer1. They can be used either as timers or as counters. Both timers are 16 bits wide. Since the 8051 has an 8-bit architecture, each 16-bit is accessed as two separate registers of low byte and high byte.

Timer0 registers

- **Timer0 registers** is a 16 bits register and accessed as low byte and high byte. The low byte is referred as a TL0 and the high byte is referred as TH0. These registers can be accessed like any other registers.

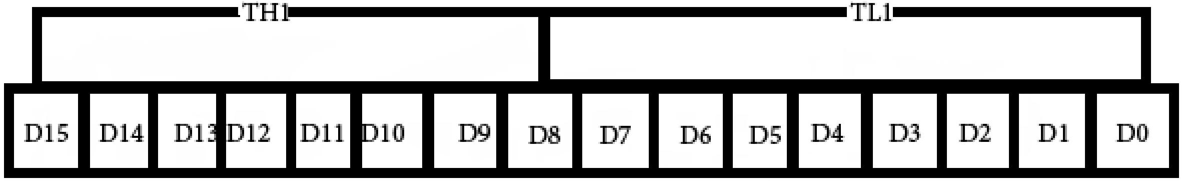
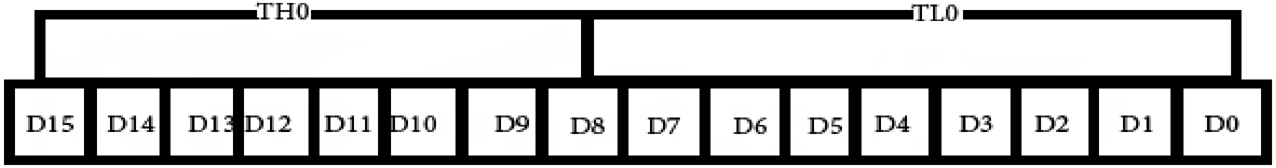


Fig. Timer1

Timer1 registers

- **Timer1 registers** is also a 16 bits register and is split into two bytes, referred to as TL1 and TH1.



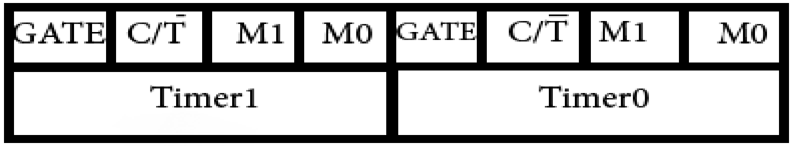
TH0 TL0

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

Fig. Timer0

TMOD (timer mode) Register:

- This is an 8-bit register which is used by both timers 0 and 1 to set the various timer modes. In this TMOD register, lower 4 bits are set aside for timer0 and the upper 4 bits are set aside for timer1
- In upper or lower 4 bits, first bit is a GATE bit. Every timer has a means of starting and stopping. Some timers do this by software, some by hardware, and some have both software and hardware controls.
- The hardware way of starting and stopping the timer by an external source is achieved by making GATE=1 in the TMOD register. And if we change to GATE=0 then we do no need external hardware to start and stop the timers



GATE	$C/\overline{T}$	M1	M0	GATE	$C/\overline{T}$	M1	M0
Timer1				Timer0			

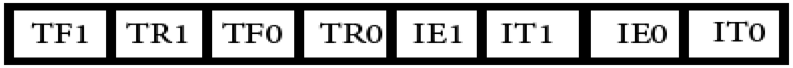
Fig. TMOD Register

TMOD

- The second bit is C/T bit and is used to decide whether a timer is used as a time delay generator or an event counter. If this bit is 0 then it is used as a timer and if it is 1 then it is used as a counter.
- In upper or lower 4 bits, the last bits third and fourth are known as M1 and M0 respectively. These are used to select the timer mode.

M0	**M1**	**Mode**	**Operating Mode**
0	0	0	13-bit timer mode,
0	1	1	16-bit timer mode,
1	0	2	8-bit auto reload
1	1	3	Spilt timer mode.

TCON register–



TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Fig. TCON Register

TCON is 8-bit control register and contains timer and interrupt flags.

Bit 7 - TF1: Timer1 Overflow Flag

1 = Timer1 overflow occurred (i.e. Timer1 goes to its max and roll over back to zero).

0 = Timer1 overflow not occurred.

It is cleared through software. In Timer1 overflow interrupt service routine, this bit will get cleared automatically while exiting from ISR.

Bit 6 - TR1: Timer1 Run Control Bit

1 = Timer1 start.

0 = Timer1 stop.

It is set and cleared by software.

Bit 5 – TF0: Timer0 Overflow Flag

1 = Timer0 overflow occurred (i.e. Timer0 goes to its max and roll over back to zero).

0 = Timer0 overflow not occurred.

It is cleared through software. In Timer0 overflow interrupt service routine, this bit will get cleared automatically while exiting from ISR.

TCON register

- **Bit 4 – TR0:** Timer0 Run Control Bit
- 1 = Timer0 start.
- 0 = Timer0 stop.
- It is set and cleared by software.
- **Bit 3 – IE1:** External Interrupt1 Edge Flag
- 1 = External interrupt1 occurred.
- 0 = External interrupt1 Processed.
- It is set and cleared by hardware.
- **Bit 2 – IT1:** External Interrupt1 Trigger Type Select Bit
- 1 = Interrupt occur on falling edge at INT1 pin.
- 0 = Interrupt occur on low level at INT1 pin.
- **Bit 1 – IE0:** External Interrupt0 Edge Flag
- 1 = External interrupt0 occurred.
- 0 = External interrupt0 Processed.
- It is set and cleared by hardware.
- **Bit 0 – IT0:** External Interrupt0 Trigger Type Select Bit
- 1 = Interrupt occur on falling edge at INT0 pin.
- 0 = Interrupt occur on low level at INT0 pin.

Interrupts in 8051

- Interrupts in 8051 microcontroller are more desirable to reduce the regular status checking of the interfaced devices or inbuilt devices.
- Interrupt is an event that temporarily suspends the main program, passes the control to a special code section, executes the event-related function and resumes the main program flow where it had left off.

Basic Types

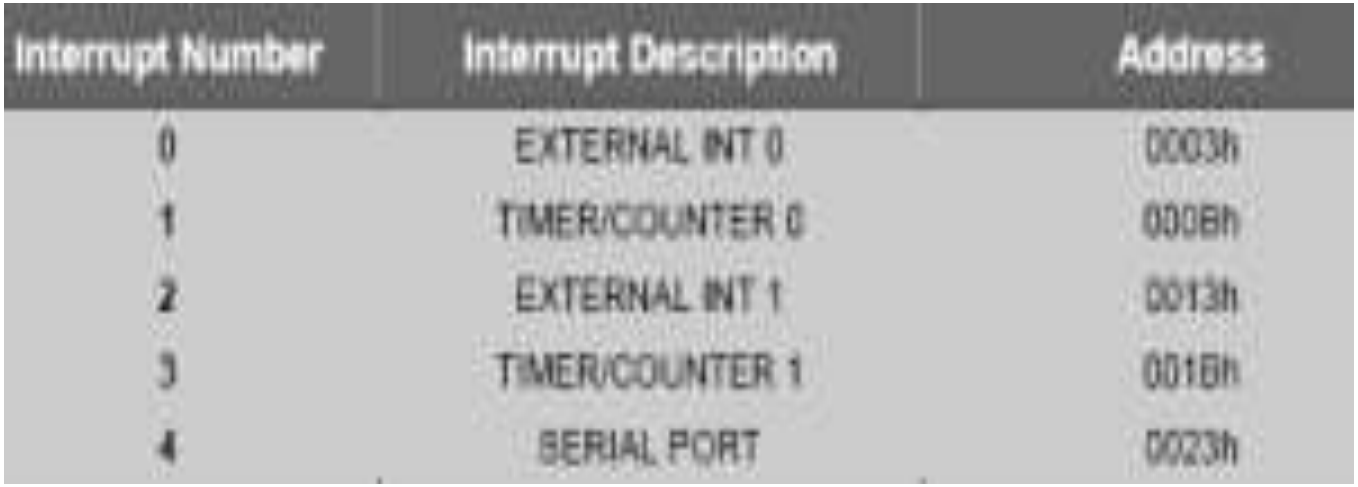
- Interrupts are of different types like software and hardware, maskable and non-maskable, fixed and vector interrupts, and so on.
- Interrupt Service Routine (ISR) comes into the picture when interrupt occurs, and then tells the processor to take appropriate action for the interrupt, and after ISR execution, the controller jumps into the main program.

Interrupt Sources

- 8051 has 5 sources of interrupts
  - Timer 0 overflow(T0)
  - Timer 1 overflow(T1)
  - External Interrupt 0(INT0)
  - External Interrupt 1(INT1)
  - Serial Port events(TI/RI)
  - The Timer and Serial interrupts are internally generated by the microcontroller, whereas the external interrupts are generated by additional interfacing devices or switches that are externally connected to the microcontroller. These external interrupts can be edge triggered or level triggered. When an interrupt occurs, the microcontroller executes the interrupt service routine so that memory location corresponds to the interrupt that enables it. The Interrupt corresponding to the memory location is given in the interrupt vector table below.

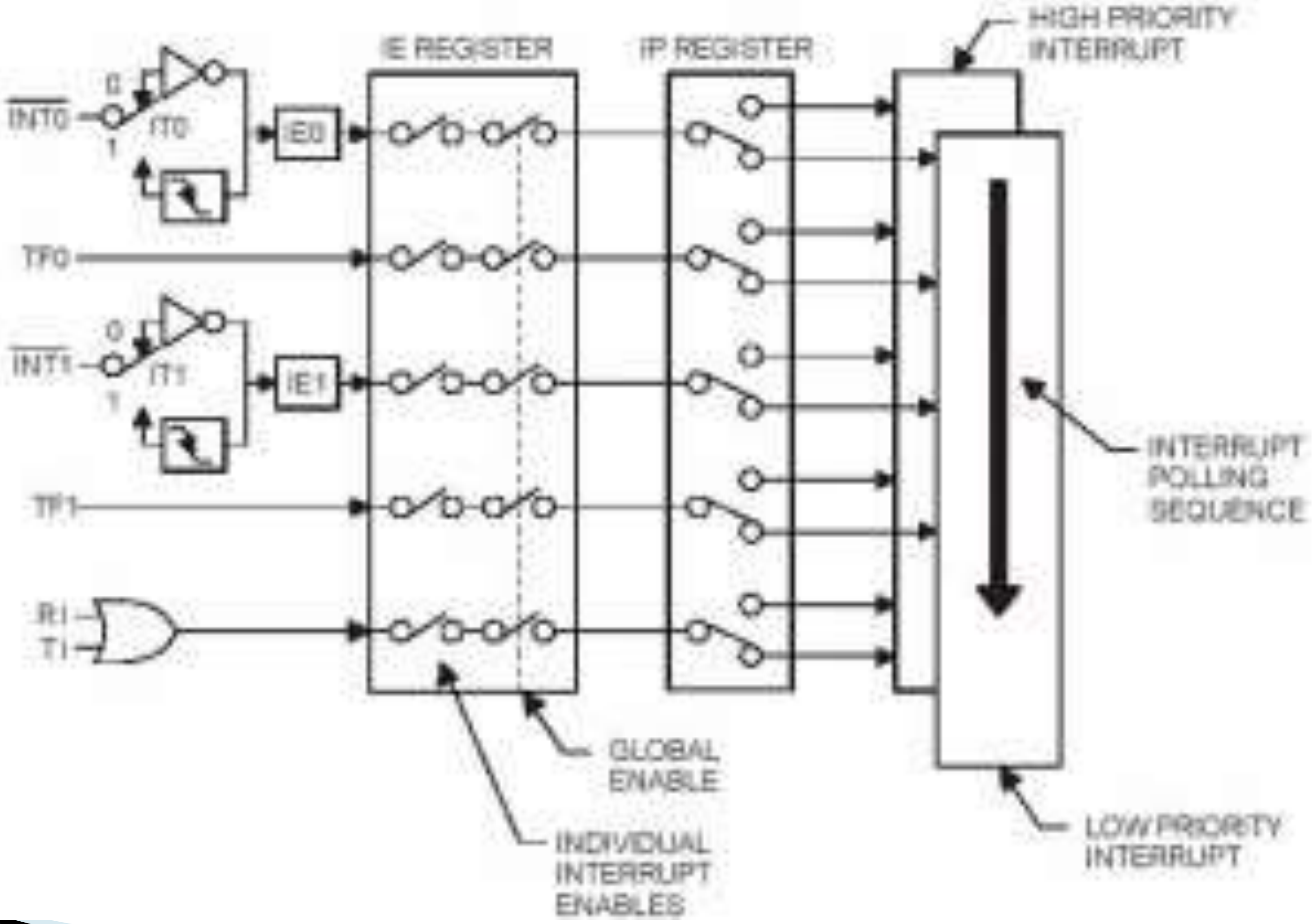
Interrupt Priorities

- What if two interrupt sources interrupt at the same time?
- The interrupt with the highest PRIORITY gets serviced first.
- All interrupts have a default priority order.
- Priority can also be set to “high” or “low”.



Interrupt Number	Interrupt Description	Address
0	EXTERNAL INT 0	0003h
1	TIMER/COUNTER 0	000Bh
2	EXTERNAL INT 1	0013h
3	TIMER/COUNTER 1	001Bh
4	SERIAL PORT	0023h

Interrupt Structure



Interrupt Enable (IE) Register:

- This register is responsible for enabling and disabling the interrupt. It is a bit addressable register in which EA must be set to one for enabling interrupts. The corresponding bit in this register enables particular interrupt like timer, external and serial inputs. In the below IE register, bit corresponding to 1 activates the interrupt and 0 disables the interrupt.

Interrupt Enable (IE) Register

- This register is responsible for enabling and disabling the interrupt. It is a bit addressable register in which EA must be set to one for enabling interrupts. The corresponding bit in this register enables particular interrupt like timer, external and serial inputs. In the below IE register, bit corresponding to 1 activates the interrupt and 0 disables the interrupt.

Interrupt Enable Register :

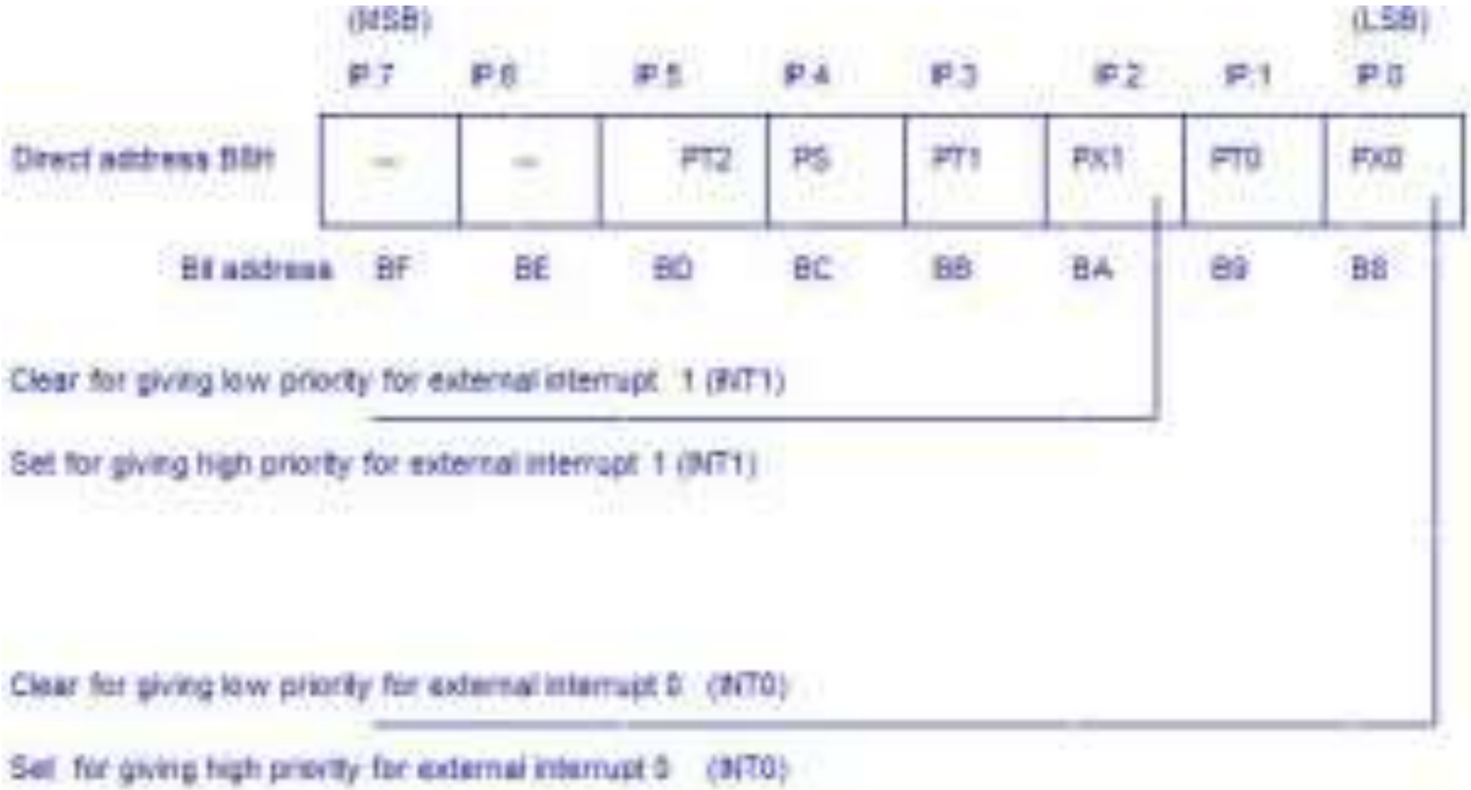
EA	—	ET2	ES	ET1	EX1	ET0	EX0

- EA **: Global enable/disable**.
- **--- : Undefined**.
- ET2 :Enable Timer 2 interrupt.
- ES :Enable Serial port interrupt.
- ET1 :Enable Timer 1 interrupt.
- EX1 :Enable External 1 interrupt.
- ET0 : Enable Timer 0 interrupt.
- EX0 : Enable External 0 interrupt**.**

Interrupt Priority Register (IP):

- It is also possible to change the priority levels of the interrupts by setting or clearing the corresponding bit in the Interrupt priority (IP) register as shown in the figure.
- This allows the low priority interrupt to interrupt the high-priority interrupt, but prohibits the interruption by another low-priority interrupt. Similarly, the high-priority interrupt cannot be interrupted.
- If these interrupt priorities are not programmed, the microcontroller executes in predefined manner and its order is INT0, TF0, INT1, TF1, and SI.

Interrupt Priority Register



	(MSB)							(LSB)
	IP.7	IP.6	IP.5	IP.4	IP.3	IP.2	IP.1	IP.0
Direct address B8H	—	—	PT2	PS	PT1	PX1	PT0	PX0
Bit address	BF	BE	BD	BC	BB	BA	B9	B8

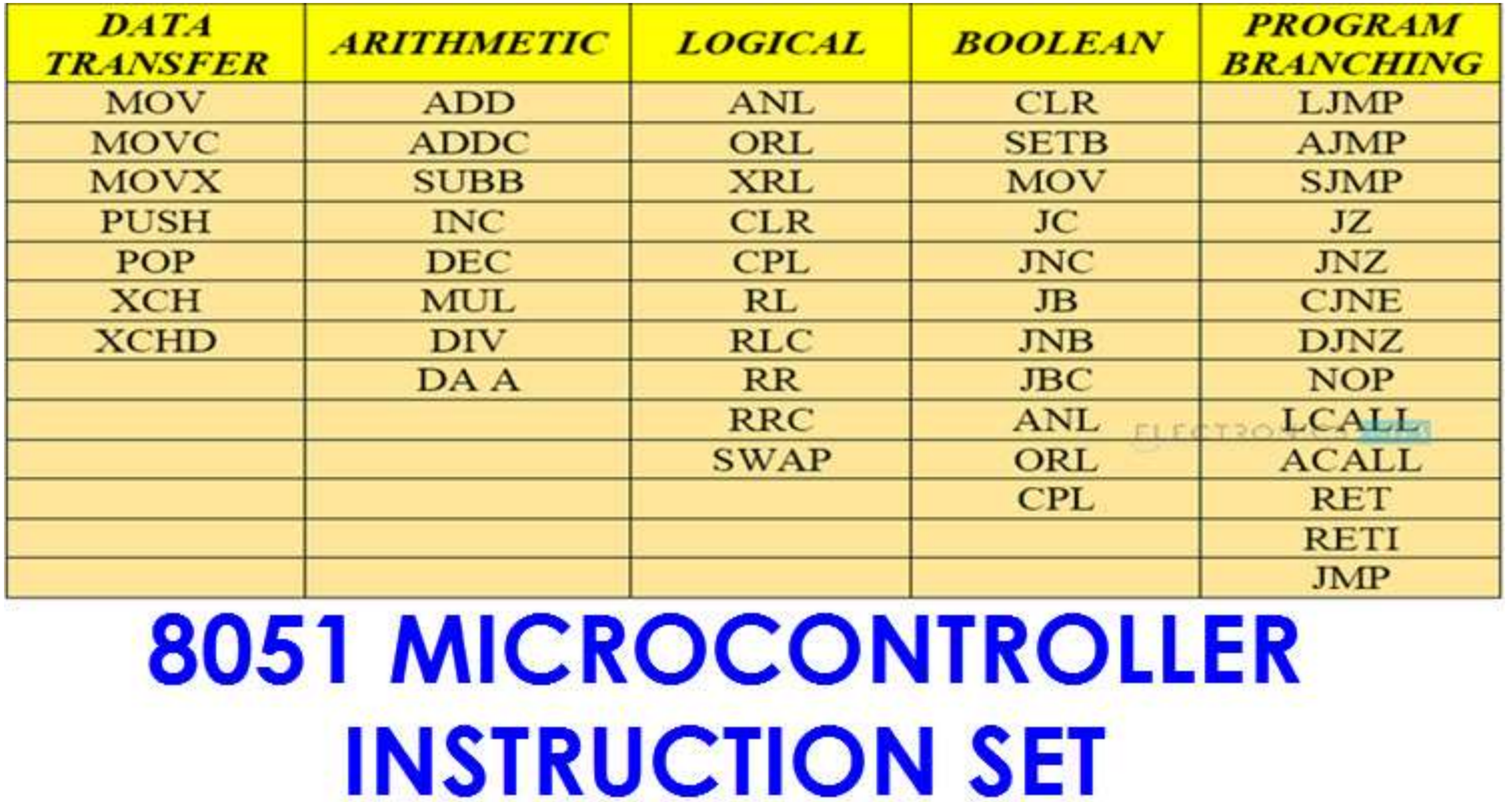
Clear for giving low priority for external interrupt 1 (INT1)

Set for giving high priority for external interrupt 1 (INT1)

Clear for giving low priority for external interrupt 0 (INT0)

Set for giving high priority for external interrupt 0 (INT0)

Instruction Set



DATA TRANSFER	ARITHMETIC	LOGICAL	BOOLEAN	PROGRAM BRANCHING
MOV	ADD	ANL	CLR	LJMP
MOVC	ADDC	ORL	SETB	AJMP
MOVX	SUBB	XRL	MOV	SJMP
PUSH	INC	CLR	JC	JZ
POP	DEC	CPL	JNC	JNZ
XCH	MUL	RL	JB	CJNE
XCHD	DIV	RLC	JNB	DJNZ
	DA A	RR	JBC	NOP
		RRC	ANL	LCALL
		SWAP	ORL	ACALL
			CPL	RET
				RETI
				JMP

8051 MICROCONTROLLER
INSTRUCTION SET

MOV Instruction

- **Operation:MOV**
- **Function:**Move memory
- **Syntax:**MOV *operand1*,*operand2*

- **Description:** MOV copies the value of *operand2* into *operand1*. The value of *operand2* is not affected.
- Both *operand1* and *operand2* must be in Internal RAM.
- No flags are affected unless the instruction is moving the value of a bit into the carry bit in which case the carry bit is affected or unless the instruction is moving a value into the PSW register (which contains all the program flags).

MOV

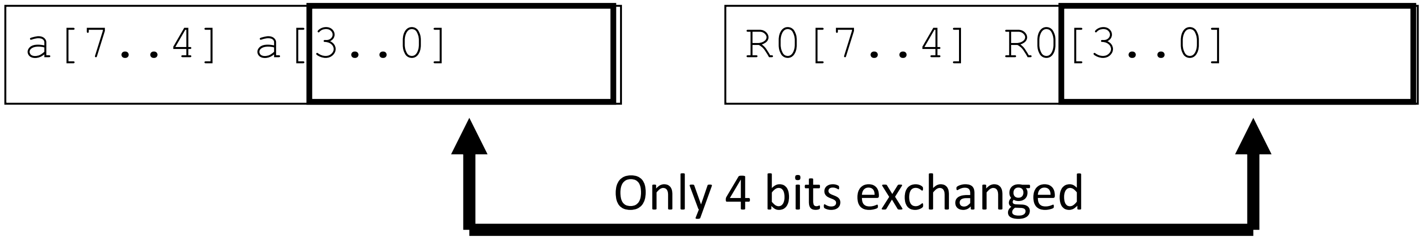
- No flags are affected unless the instruction is moving the value of a bit into the carry bit in which case the carry bit is affected or unless the instruction is moving a value into the PSW register (which contains all the program flags).

- MOV @R0,#*data*
- MOV @R0,A
- MOV A,#*data*
- MOV A,@R1
- MOV A,R0

Exchange Instructions

two way data transfer

```
XCH a, 30h          ; a □ □  M[30]
XCH a, R0           ; a □ □  R0
XCH a, @R0          ; a □ □  M[R0]
XCHD a, R0          ; exchange “digit”
```



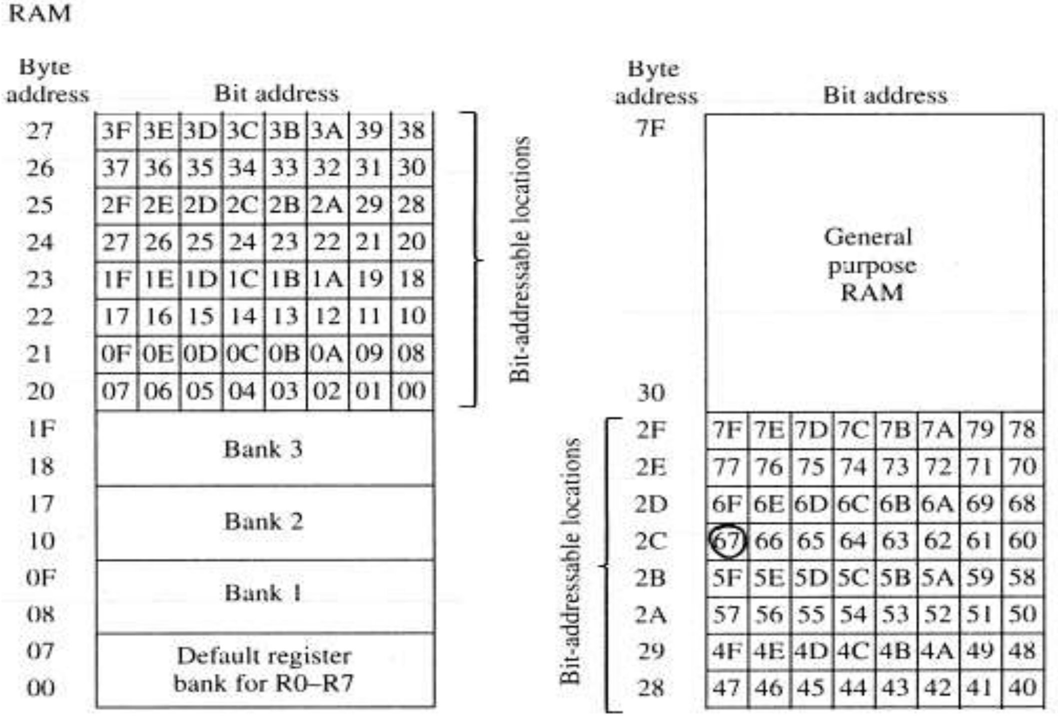
Bit-Oriented Data Transfer

- transfers between individual bits.
- Carry flag (C) (bit 7 in the PSW) is used as a single-bit accumulator
- RAM bits in addresses 20-2F are bit addressable

```
mov C, P0.0

mov C, 67h

mov C, 2ch.7
```



RAM

Byte address	Bit address							
27	3F	3E	3D	3C	3B	3A	39	38
26	37	36	35	34	33	32	31	30
25	2F	2E	2D	2C	2B	2A	29	28
24	27	26	25	24	23	22	21	20
23	1F	1E	1D	1C	1B	1A	19	18
22	17	16	15	14	13	12	11	10
21	0F	0E	0D	0C	0B	0A	09	08
20	07	06	05	04	03	02	01	00
1F 18	Bank 3							
17 10	Bank 2							
0F 08	Bank 1							
07 00	Default register bank for R0–R7							

Bit-addressable locations

Byte address	Bit address							
7F 30	General purpose RAM							
2F	7F	7E	7D	7C	7B	7A	79	78
2E	77	76	75	74	73	72	71	70
2D	6F	6E	6D	6C	6B	6A	69	68
2C	67	66	65	64	63	62	61	60
2B	5F	5E	5D	5C	5B	5A	59	58
2A	57	56	55	54	53	52	51	50
29	4F	4E	4D	4C	4B	4A	49	48
28	47	46	45	44	43	42	41	40

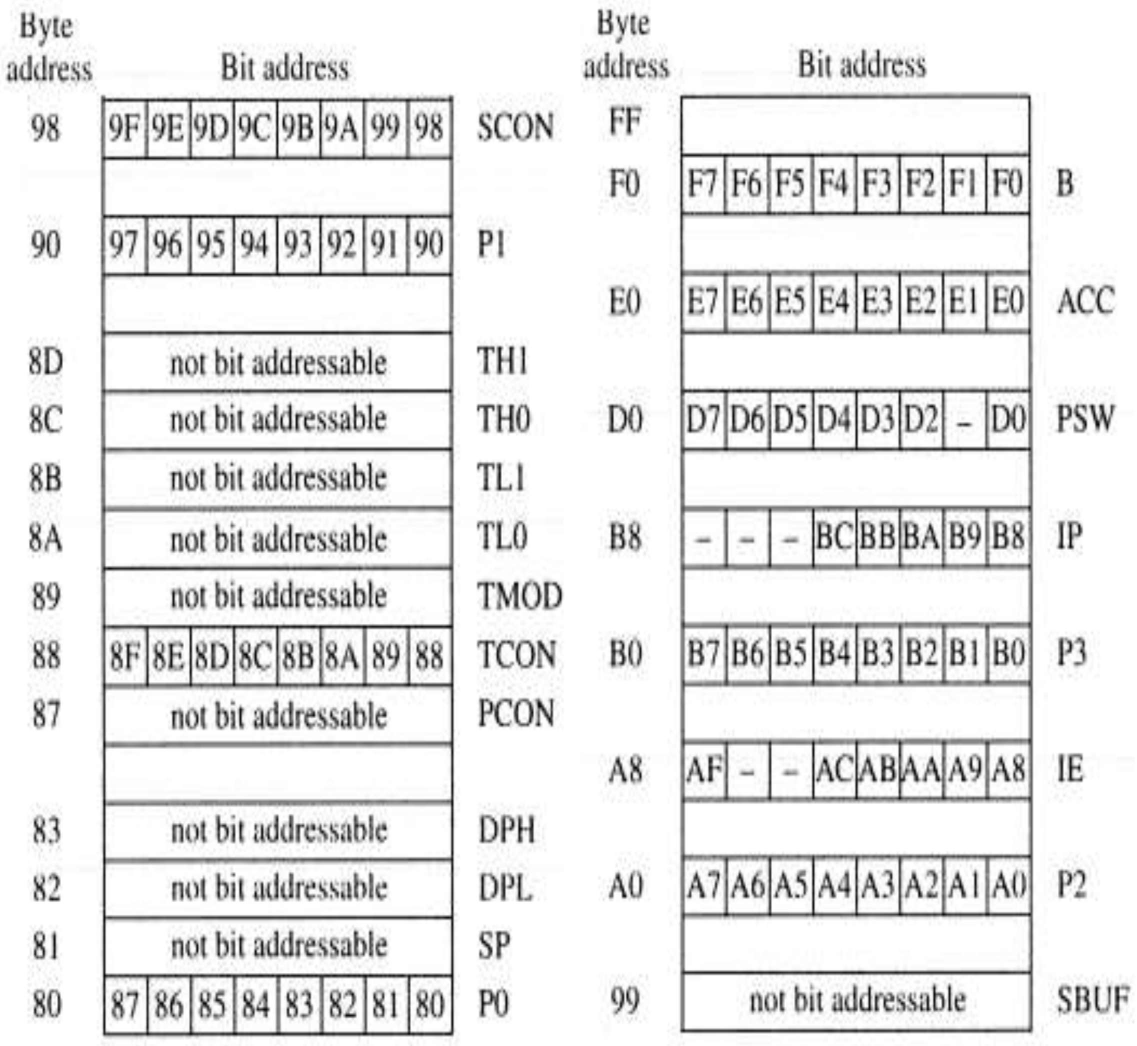
Bit-addressable locations

SFRs that are Bit Addressable

- SFRs with addresses ending in 0 or 8 are bit-addressable.

  (80, 88, 90, 98, etc)

- Notice that all 4 parallel I/O ports are bit addressable

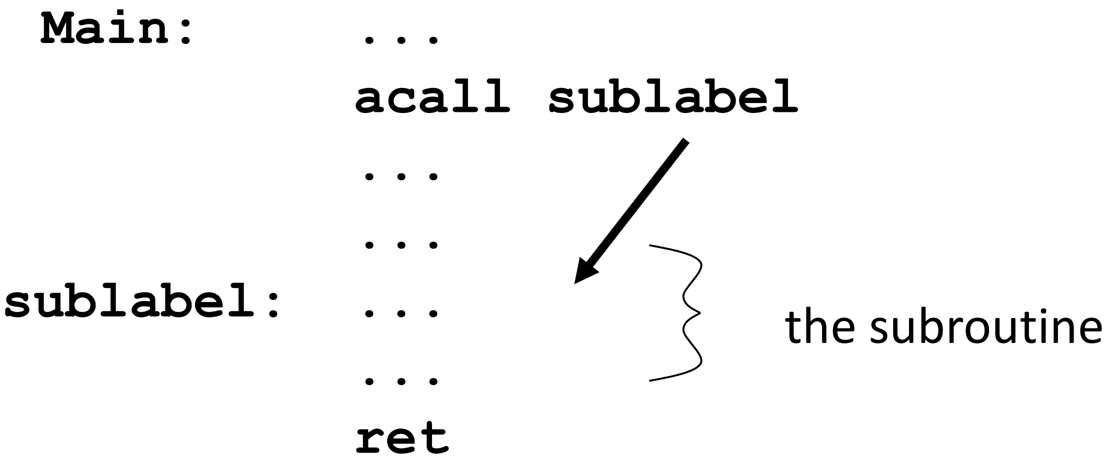


Byte address	Bit address								
98	9F	9E	9D	9C	9B	9A	99	98	SCON
90	97	96	95	94	93	92	91	90	P1
8D	not bit addressable								TH1
8C	not bit addressable								TH0
8B	not bit addressable								TL1
8A	not bit addressable								TL0
89	not bit addressable								TMOD
88	8F	8E	8D	8C	8B	8A	89	88	TCON
87	not bit addressable								PCON
83	not bit addressable								DPH
82	not bit addressable								DPL
81	not bit addressable								SP
80	87	86	85	84	83	82	81	80	P0

Byte address	Bit address								
FF									
F0	F7	F6	F5	F4	F3	F2	F1	F0	B
E0	E7	E6	E5	E4	E3	E2	E1	E0	ACC
D0	D7	D6	D5	D4	D3	D2	-	D0	PSW
B8	-	-	-	BC	BB	BA	B9	B8	IP
B0	B7	B6	B5	B4	B3	B2	B1	B0	P3
A8	AF	-	-	AC	AB	AA	A9	A8	IE
A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
99	not bit addressable								SBUF

Subroutines

- Subroutines allow us to have "structured" assembly language programs.
- This is useful for breaking a large design into manageable parts.
- It saves code space when subroutines can be called many times in the same program.



```
 Main:       ...
             acall sublabel
             ...
             ...
sublabel:    ...
             ...
             ret
```

the subroutine

Subroutine - Example

```
square:  push b
              mov  b,a
              mul  ab
              pop  b
              ret
```

- **8 byte and 11 machine cycle**
- SP is initialized to 07 after reset.(Same address as R7)
- With each push operation 1st , pc is increased
- When using subroutines, the stack will be used to store the PC, so it is very important to initialize the stack pointer. Location 2Fh is often used.

```
mov SP, #2Fh
```

Example of delay

```
      mov a,#0aah
Back1:mov p0,a
      lcall delay1
      cpl a
      sjmp back1
Delay1:mov r0,#0ffh;1cycle
Here: djnz r0,here ;2cycle
      ret          ;2cycle
      end
```

Delay=1+255*2+2=513 cycle

```
Delay2:
       mov r6,#0ffh
back1: mov r7,#0ffh ;1cycle
Here:  djnz r7,here ;2cycle
       djnz r6,back1;2cycle
       ret          ;2cycle
       end
```

Delay=1+(1+255*2+2)*255+2
     =130818 machine cycle

Arithmetic Instructions

Mnemonic	Description
`ADD A, byte`	add A to byte, put result in A
`ADDC A, byte`	add with carry
`SUBB A, byte`	subtract with borrow
`INC A`	increment A
`INC byte`	increment byte in memory
`INC DPTR`	increment data pointer
`DEC A`	decrement accumulator
`DEC byte`	decrement byte
`MUL AB`	multiply accumulator by b register
`DIV AB`	divide accumulator by b register
`DA A`	decimal adjust the accumulator

ADD Instructions

```
add a, byte         ; a □  a + byte
addc a, byte              ; a □  a + byte + C
```

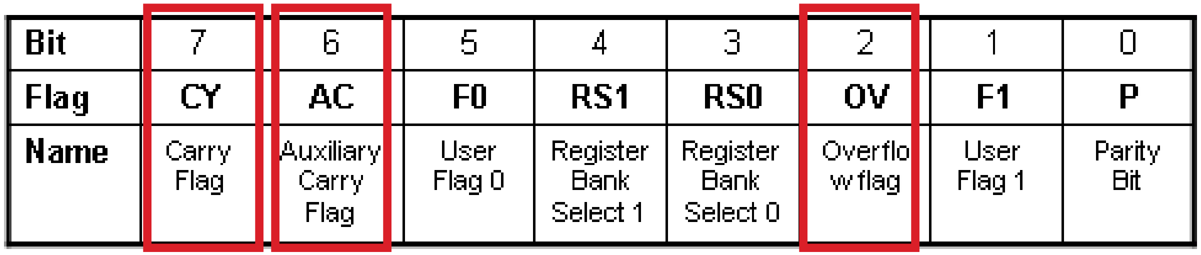
These instructions affect 3 bits in PSW:

C = 1 if result of add is greater than FF

AC = 1 if there is a carry out of bit 3

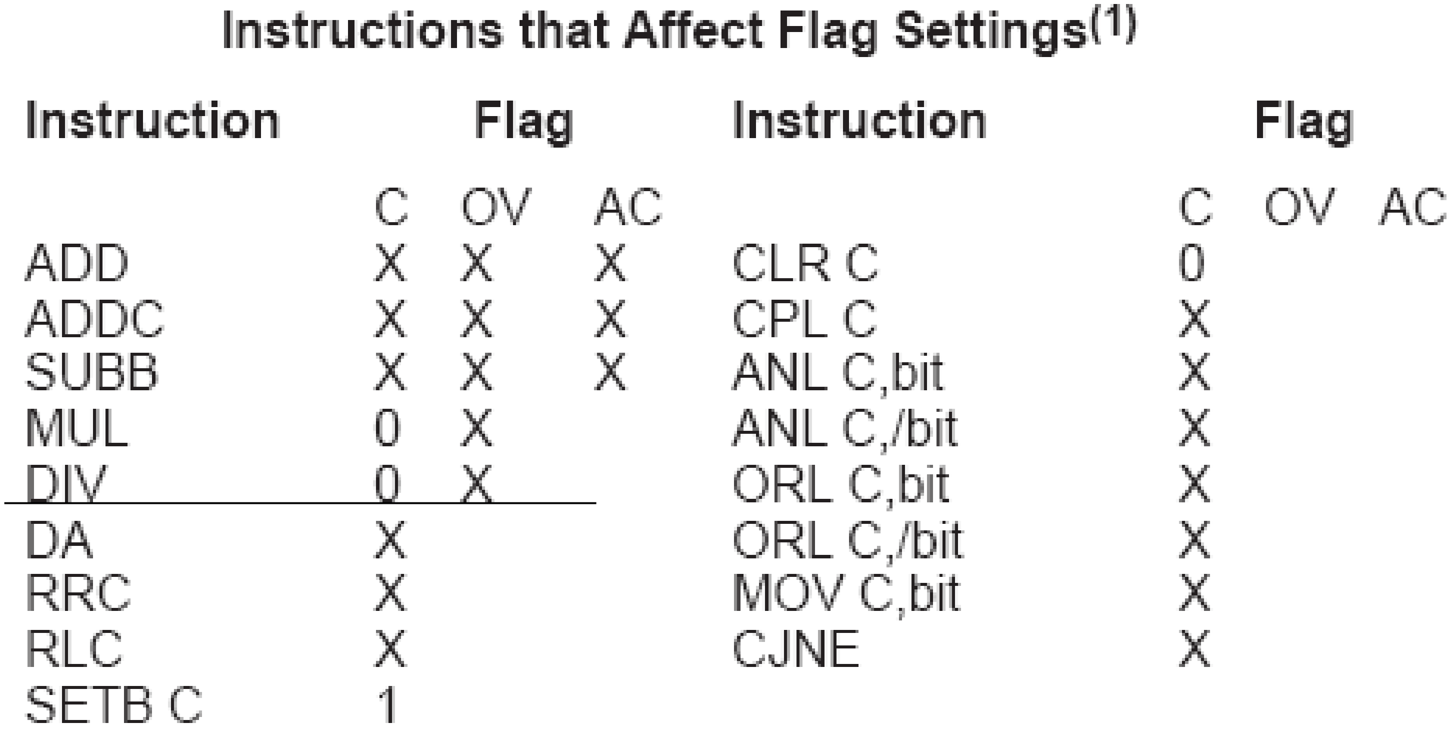
OV = 1 if there is a carry out of bit 7, but not from bit 6, or visa versa.

Program Status Word (PSW)



Bit	7	6	5	4	3	2	1	0
Flag	**CY**	**AC**	**F0**	**RS1**	**RS0**	**OV**	**F1**	**P**
Name	Carry Flag	Auxiliary Carry Flag	User Flag 0	Register Bank Select 1	Register Bank Select 0	Overflo w flag	User Flag 1	Parity Bit

Instructions that Affect PSW bits



Instructions that Affect Flag Settings[1]

Instruction	Flag			Instruction	Flag		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLR C	0		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	0	X		ANL C,/bit	X		
DIV	0	X		ORL C,bit	X		
DA	X			ORL C,/bit	X		
RRC	X			MOV C,bit	X		
RLC	X			CJNE	X		
SETB C	1						

ADD Examples

```
mov a, #3Fh
add a, #D3h
```

- What is the value of the C, AC, OV flags after the second instruction is executed?

```
0011 1111
1101 0011
0001 0010
```

```
C = 1
AC = 1
OV = 0
```

Signed Addition and Overflow

2’s complement:

0000 0000	00	0	
…			
0111 1111	7F	127	
1000 0000		80	-128
…			
1111 1111		FF	-1

```
 0111 1111  (positive 127)
 0111 0011  (positive 115)
 1111 0010  (overflow
cannot represent 242 in 8
bits 2’s complement)
```

```
 1000 1111  (negative 113)
 1101 0011  (negative  45)
 0110 0010  (overflow)
```

```
0011 1111  (positive)
1101 0011  (negative)
0001 0010 (never overflows)
```

Addition Example

```
; Computes Z = X + Y
; Adds values at locations 78h and 79h and puts them in 7Ah
;-----------------------------------------------------------------
    ----
X          equ       78h
Y          equ       79h
Z          equ       7Ah
;-----------------------------------------------------------------
    ---
           org 00h
           ljmp Main
;-----------------------------------------------------------------
    ---
           org 100h
Main:
           mov a, X
           add a, Y
           mov Z, a
           end
```

The 16-bit ADD example

```
; Computes Z = X + Y     (X,Y,Z are 16 bit)
;-------------------------------------------------------
    ------------
X          equ       78h
Y          equ       7Ah
Z          equ       7Ch
;-------------------------------------------------------
    -----------
           org 00h
           ljmp Main
;-------------------------------------------------------
    -----------
           org 100h
Main:
           mov a, X
           add a, Y
           mov Z, a
            mov a, X+1
           adc a, Y+1
           mov Z+1, a
           end
```

Subtract

SUBB A, byte	subtract with borrow

Example:

```
SUBB A, #0x4F    ;A □  A – 4F – C
```

Notice that

There is no subtraction WITHOUT borrow.
Therefore, if a subtraction without borrow is desired,
it is necessary to clear the C flag.

Example:

```
Clr  c
SUBB A, #0x4F     ;A □  A – 4F
```

Increment and Decrement

INC A	increment A
INC byte	increment byte in memory
INC DPTR	increment data pointer
DEC A	decrement accumulator
DEC byte	decrement byte

- The increment and decrement instructions do NOT affect the C flag.
- Notice we can only INCREMENT the data pointer, not decrement.

Example: Increment 16-bit Word

- Assume 16-bit word in R3:R2

```
mov a, r2
add a, #1          ; use add rather than increment to affect C
mov r2, a
mov a, r3
addc a, #0         ; add C to most significant byte
mov r3, a
```

Multiply

When multiplying two 8-bit numbers, the size of the maximum product is 16-bits

FF x FF = FE01

(255 x 255 = 65025)

```
MUL AB          ; BA □  A * B
```

Note : B gets the High byte

A gets the Low byte

Division

- Integer Division

```
DIV AB     ; divide A by B

A □  Quotient(A/B)
B □  Remainder(A/B)
```

OV – used to indicate a divide by zero condition.
C – set to zero

Decimal Adjust

```
DA a    ; decimal adjust a
```

Used to facilitate BCD addition.
Adds “6” to either high or low nibble after an addition
to create a valid BCD number.

Example:

```
        mov a, #23h
        mov b, #29h
        add a, b            ; a □  23h + 29h = 4Ch (wanted 52)
        DA a                ; a □  a + 6 = 52
```

Logic Instructions

- Bitwise logic operations
  - (AND, OR, XOR, NOT)
- Clear
- Rotate
- Swap

Logic instructions do NOT affect the flags in PSW

Bitwise Logic

ANL □ AND

ORL □ OR

XRL □ XOR

CPL □ Complement

Examples:

ANL 0000111
1
1010110
0
0000110
0

ORL 0000111
1
1010110
0
1010111
1

XRL 0000111
1
1010110
0
1010001
1

CPL 1010110
0
0101001
1

Address Modes with Logic

ANL – AND

ORL – OR

XRL – eXclusive oR

a, byte

direct, reg. indirect, reg, immediate

byte, a

direct

byte, #constant

CPL – Complement

a ex: cpl a

Uses of Logic Instructions

- Force individual bits low, without affecting other bits.

```
anl PSW, #0xE7        ;PSW AND 11100111
```

- Force individual bits high.

```
orl PSW, #0x18        ;PSW OR 00011000
```

- Complement individual bits

```
xrl P1, #0x40               ;P1 XRL 01000000
```

Other Logic Instructions

CLR - clear

RL – rotate left

RLC – rotate left through Carry

RR – rotate right

RRC – rotate right through Carry

SWAP – swap accumulator nibbles

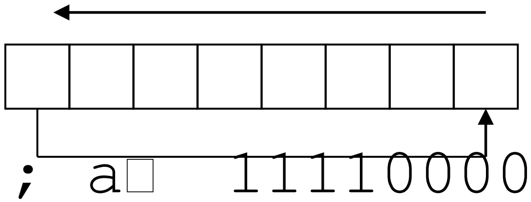
CLR (Set all bits to 0)

CLR A	
CLR byte	(direct mode)
CLR Ri	(register mode)
CLR @Ri	(register indirect mode)

Rotate

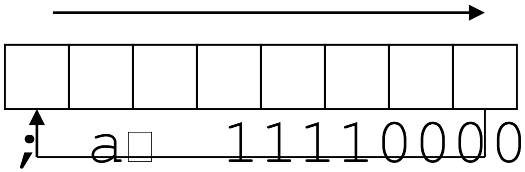
- Rotate instructions operate only on **a**

RL a



```
Mov  a,#0xF0        ;  a□   11110000
RR  a               ;  a□   11100001
```

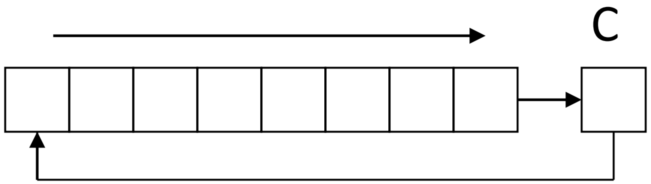
RR a



```
Mov  a,#0xF0        ;  a□   11110000
RR  a               ;  a□   01111000
```

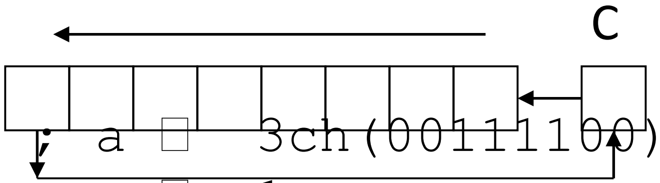
Rotate through Carry

RRC a



```
mov a, #0A9h        ; a □  A9
add a, #14h         ; a □  BD (10111101), C□ 0
rrc a               ; a □   01011110, C□ 1
```

RLC a



```
mov a, #3ch         ; a □  3ch(00111100)
setb c              ; c □  1
rlc a               ; a □   01111001, C□ 1
```

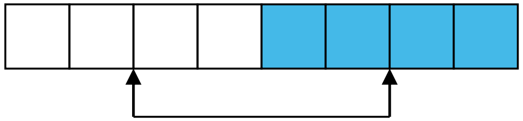
Rotate and Multiplication/Division

- Note that a shift left is the same as multiplying by 2, shift right is divide by 2

```
mov a, #3       ; A□  00000011 (3)
clr C           ; C□  0
rlc a           ; A□  00000110 (6)
rlc a           ; A□  00001100 (12)
rrc a           ; A□  00000110 (6)
```

Swap

SWAP a



```
mov a, #72h          ; a □  27h
swap a                ; a □  27h
```

Bit Logic Operations

- Some logic operations can be used with single bit operands

```
ANL C, bit
ORL C, bit
CLR C
CLR bit
CPL C
CPL bit
SETB C
SETB bit
```

- “bit” can be any of the bit-addressable RAM locations or SFRs.

Shift/Mutliply Example

- Program segment to multiply by 2 and add 1.

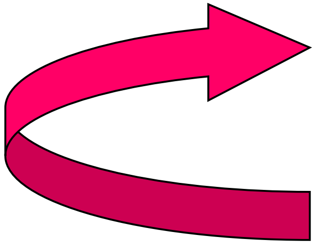
Program Flow Control

- Unconditional jumps (“go to”)
- Conditional jumps
- Call and return

Unconditional Jumps

- **SJMP <rel addr>** ; Short jump, relative address is 8-bit 2’s complement number, so jump can be up to 127 locations forward, or 128 locations back.
- **LJMP <address 16>** ; Long jump
- **AJMP <address 11>** ; Absolute jump to anywhere within 2K block of program memory
- **JMP @A + DPTR** ; Long indexed jump

Infinite Loops



```
Start: mov C, p3.7
       mov p1.6, C
       sjmp Start
```

Microcontroller application programs are almost always infinite loops!

Re-locatable Code

Memory specific NOT Re-locatable (machine code)

```
        org 8000h
Start:  mov C, p1.6
        mov p3.7, C
        ljmp Start
        end
```

Re-locatable (machine code)

```
        org 8000h
Start:  mov C, p1.6
        mov p3.7, C
        sjmp Start
        end
```

Jump table

```
            Mov dptr,#jump_table
            Mov a,#index_number
            Rl  a
            Jmp @a+dptr
               ...
Jump_table: ajmp case0
            ajmp case1
            ajmp case2
            ajmp case3
```

Conditional Jump

- These instructions cause a jump to occur only if a condition is true. Otherwise, program execution continues with the next instruction.

```
loop: mov a, P1
      jz  loop     ; if a=0, goto loop,
                   ; else goto next instruction
      mov b, a
```

- There is no zero flag (z)
- Content of A checked for zero on time

Conditional jumps

Mnemonic	Description
JZ <rel addr>	Jump if a = 0
JNZ <rel addr>	Jump if a != 0
JC <rel addr>	Jump if C = 1
JNC <rel addr>	Jump if C != 1
JB <bit>, <rel addr>	Jump if bit = 1
JNB <bit>,<rel addr>	Jump if bit != 1
JBC <bir>, <rel addr>	Jump if bit =1, &clear bit
CJNE A, direct, <rel addr>	Compare A and memory, jump if not equal

Example: Conditional Jumps

```
if (a = 0) is true
    send a 0 to LED
else
    send a 1 to LED
```

```
          jz led_off
          Setb P1.6
          sjmp skipover
 led_off: clr P1.6
          mov A, P0
skipover:
```

More Conditional Jumps

Mnemonic	Description
CJNE A, #data <rel addr>	Compare A and data, jump if not equal
CJNE Rn, #data <rel addr>	Compare Rn and data, jump if not equal
CJNE @Rn, #data <rel addr>	Compare Rn and memory, jump if not equal
DJNZ Rn, <rel addr>	Decrement Rn and then jump if not zero
DJNZ direct, <rel addr>	Decrement memory and then jump if not zero

Iterative Loops

For A = 0 to 4 do

{…}

```
        clr a
loop:   ...
        ...
        inc a
        cjne a, #4, loop
```

For A = 4 to 0 do

{…}

```
        mov R0, #4
loop:   ...
        ...
        djnz R0, loop
```

Iterative Loops(examples)

```
        mov a,#50h
        mov b,#00h
        cjne a,#50h,next
        mov  b,#01h
next:   nop
        end
```

```
        mov a,#0aah
        mov b,#10h
Back1:mov r6,#50
Back2:cpl a
        djnz r6,back2
        djnz b,back1
        end
```

```
        mov a,#25h
        mov r0,#10h
        mov r2,#5
Again:   mov @ro,a
        inc r0
        djnz r2,again
        end
```

```
        mov a,#0h
        mov r4,#12h
Back:   add a,#05
        djnz r4,back
        mov  r5,a
        end
```

Call and Return

- Call is similar to a jump, but
  - Call pushes PC on stack before branching

```
acall <address 11>          ; stack □  PC
                            ; PC □  address 11 bit

lcall <address 16>          ; stack □  PC
                             ; PC □  address 16 bit
```

Return

- Return is also similar to a jump, but
  - Return instruction pops PC from stack to get address to jump to

```
ret                 ; PC □  stack
```

CHAPTER 8

Assembly language programming

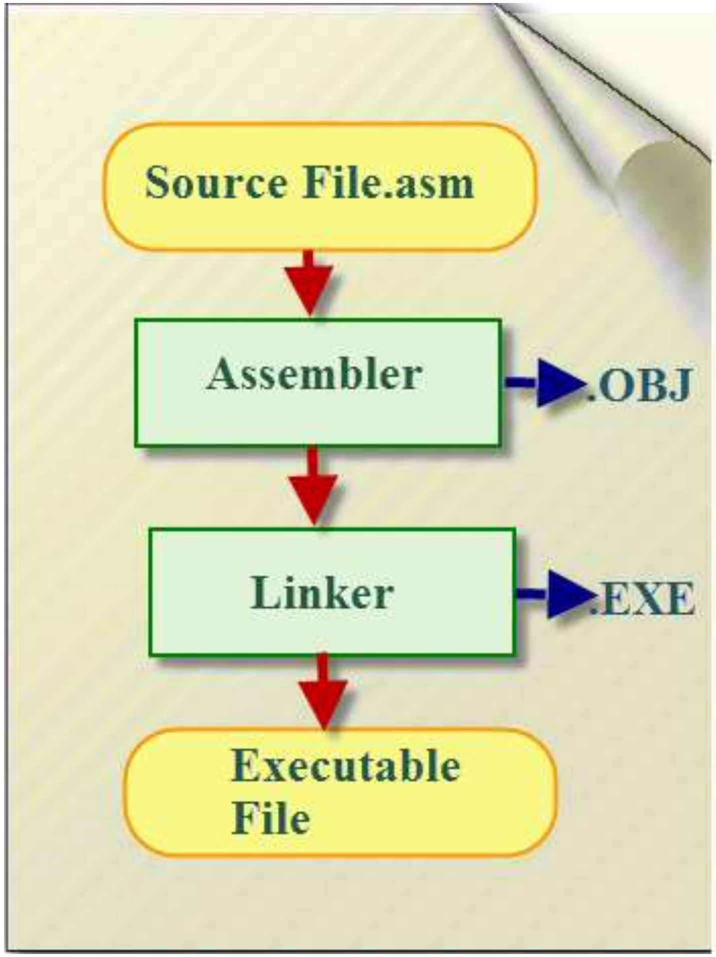
Contents

- Assembly language programming
- Data Transfer operations
- Input/Output operations

8051 Programming in Assembly Language

- The assembly language is a fully hardware related programming language.
- The embedded designers must have sufficient knowledge on hardware of particular processor or controllers before writing the program.
- The assembly language is developed by mnemonics; therefore, users cannot understand it easily to modify the program.

Process



Rules of Assembly Language

- The assembly code must be written in upper case letters
- The labels must be followed by a colon (label:)
- All symbols and labels must begin with a letter
- All comments are typed in lower case
- The last line of the program must be the END directive

Assembler Directives:

- The assembling directives give the directions to the CPU. The 8051 microcontroller consists of various kinds of assembly directives to give the direction to the control unit. The most useful directives are 8051 programming, such as:
- ORG
- DB
- EQU
- END

Assembler Directives

- **ORG(origin):** This directive indicates the start of the program. This is used to set the register address during assembly. For example; ORG 0000h tells the compiler all subsequent code starting at address 0000h.
- **Syntax**: ORG 0000h
- **DB(define byte)**: The define byte is used to allow a string of bytes. For example, print the "EDGEFX" wherein each character is taken by the address and finally prints the "string" by the DB directly with double quotes.
- **Syntax:**
- ORG 0000h
- MOV a, #00h
  ————–
  DB"EDGEFX"

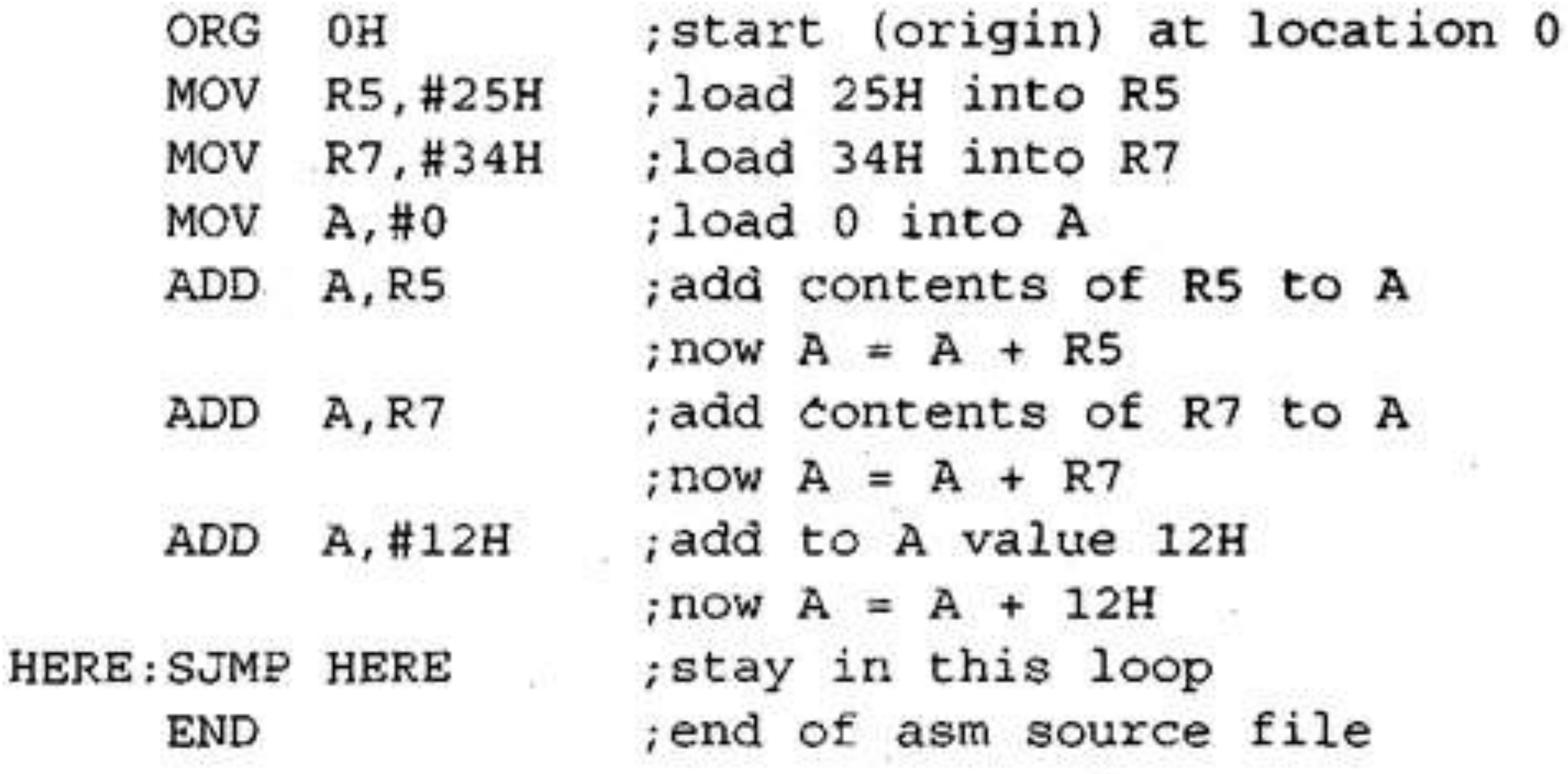
Assembler Directives

- **EQU (equivalent):** The equivalent directive is used to equate address of the variable.
- **Syntax:**
- reg equ,09h
  ————————-
  ————————-
  MOV reg,#2h
- **END:**The END directive is used to indicate the end of the program.
- **Syntax:**
- reg equ,09h
- ————————-
  ————————-
  MOV reg,#2h
  END

Structure of Assembly language

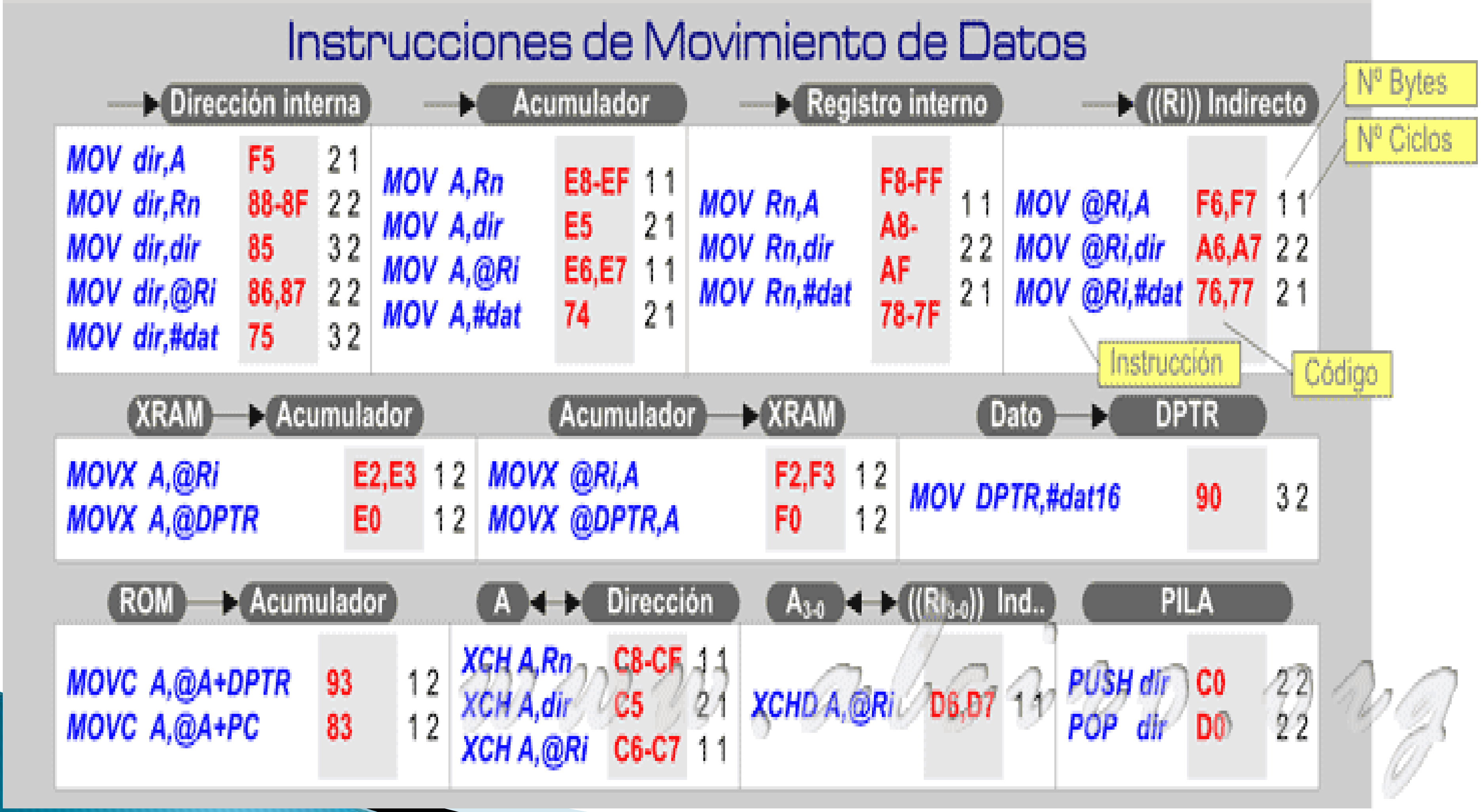
- An Assembly language program consists of, among other things, a series of lines of Assembly language instructions.
- An Assembly language instruction consists of a mnemonic, optionally followed by one or two operands.
- The operands are the data items being manipulated, and the mnemonics are the commands to the CPU, telling it what to do with those items.

Example



```
     ORG  0H           ;start (origin) at location 0
     MOV  R5,#25H      ;load 25H into R5
     MOV  R7,#34H      ;load 34H into R7
     MOV  A,#0         ;load 0 into A
     ADD  A,R5         ;add contents of R5 to A
                       ;now A = A + R5
     ADD  A,R7         ;add contents of R7 to A
                       ;now A = A + R7
     ADD  A,#12H       ;add to A value 12H
                       ;now A = A + 12H
HERE:SJMP HERE         ;stay in this loop
     END               ;end of asm source file
```

Data Transfer Instructions



Instrucciones de Movimiento de Datos

Nº Bytes

Nº Ciclos

Instrucción

Código

→ Dirección interna

MOV dir,A	F5	2	1
MOV dir,Rn	88-8F	2	2
MOV dir,dir	85	3	2
MOV dir,@Ri	86,87	2	2
MOV dir,#dat	75	3	2

→ Acumulador

MOV A,Rn	E8-EF	1	1
MOV A,dir	E5	2	1
MOV A,@Ri	E6,E7	1	1
MOV A,#dat	74	2	1

→ Registro interno

MOV Rn,A	F8-FF	1	1
MOV Rn,dir	A8-AF	2	2
MOV Rn,#dat	78-7F	2	1

→ ((Ri)) Indirecto

MOV @Ri,A	F6,F7	1	1
MOV @Ri,dir	A6,A7	2	2
MOV @Ri,#dat	76,77	2	1

XRAM → Acumulador

MOVX A,@Ri	E2,E3	1	2
MOVX A,@DPTR	E0	1	2

Acumulador → XRAM

MOVX @Ri,A	F2,F3	1	2
MOVX @DPTR,A	F0	1	2

Dato → DPTR

MOV DPTR,#dat16	90	3	2

ROM → Acumulador

MOVC A,@A+DPTR	93	1	2
MOVC A,@A+PC	83	1	2

A ↔ Dirección

XCH A,Rn	C8-CF	1	1
XCH A,dir	C5	2	1
XCH A,@Ri	C6-C7	1	1

A_{3-0} ↔ ((Ri_{3-0})) Ind..

XCHD A,@Ri	D6,D7	1	1

PILA

PUSH dir	C0	2	2
POP dir	D0	2	2

Data Transfer Instructions

- **MOV destination, source.** Data movement in the internal RAM. This type of instructions supported by virtually all addresses, direct, indirect, recording and immediate.
- **MOV** A,P0 ; Mueve el contenido del puerto 0 al acumulador
  MOV R1,A ; Mueve el contenido del Acumulador al registro 1
- **MOVX.** Data movement in the external RAM (XRAM). This type of motion only supports indirect addressing, register 8bit by R0 or R1 and 16-bit register via the DPTR.
- **MOV** DPTR,#2000H ; Mover al registro apuntador DPTR el dato inmediato 2000H (dirección)
  MOVX A,@DPTR ; Mover el contenido de la memoria que apunta el DPTR (2000H) al Acumulador
- **MOVC.** Allows movement of the accumulator ROM. By this statement can make the manipulation or movement of tables from the program memory.
- **XCH.** Swaps the contents of the accumulator and the internal RAM.
- **XCHD.** Swaps the contents of the first 4 bits of the Accumulator with internal RAM.
- **PUSH and POP.** To transfer data to the *stack.*

Data Transfer

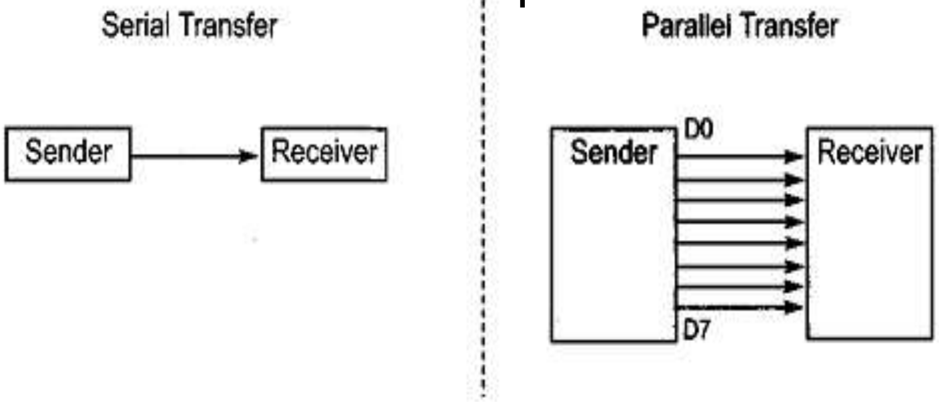
- Computers transfer data in two ways: parallel and serial.
- In parallel data transfers, often 8 or more lines (wire conductors) are used to transfer data to a device that is only a few feet away. Examples of parallel transfers are printers and hard disks; each uses cables with many wire strips. Although in such cases a lot of data can be transferred in a short amount of time by using many wires in parallel, the distance cannot be great.
- To transfer to a device located many meters away, the serial method is used. In serial communication, the data is sent one bit at a time, in contrast to parallel communication, in which the data is sent a byte or more at a time.
- Serial communication of the 8051 is the topic of this chapter. The 8051 has serial communication capability built into it, thereby making possible fast data transfer using only a few wires.

BASICS OF SERIAL COMMUNICATION

- When a microprocessor communicates with the outside world, it provides the data in byte-sized chunks. In some cases, such as printers, the information is simply grabbed from the 8-bit data bus and presented to the 8-bit data bus of the printer.
- This can work only if the cable is not too long, since long cables diminish and even distort signals. Furthermore, an 8-bit data path is expensive.
- For these reasons, serial communication is used for transferring data between two systems located at distances of hundreds of feet to millions of miles apart. Figure 10-1 diagrams serial versus parallel data transfers.

SERIAL COMMUNICATION

The fact that serial communication uses a single data line instead of the 8-bit data line of parallel communication not only makes it much cheaper but also enables two computers located in two different cities to communicate over the telephone
.For serial data communication to work, the byte of data must be converted to serial bits using a parallel-in-serial-out shift register; then it can be transmitted over a single data line. This also means that at the receiving end there must be a serial-in-parallel-out shift register to receive the serial data and pack them into a byte.

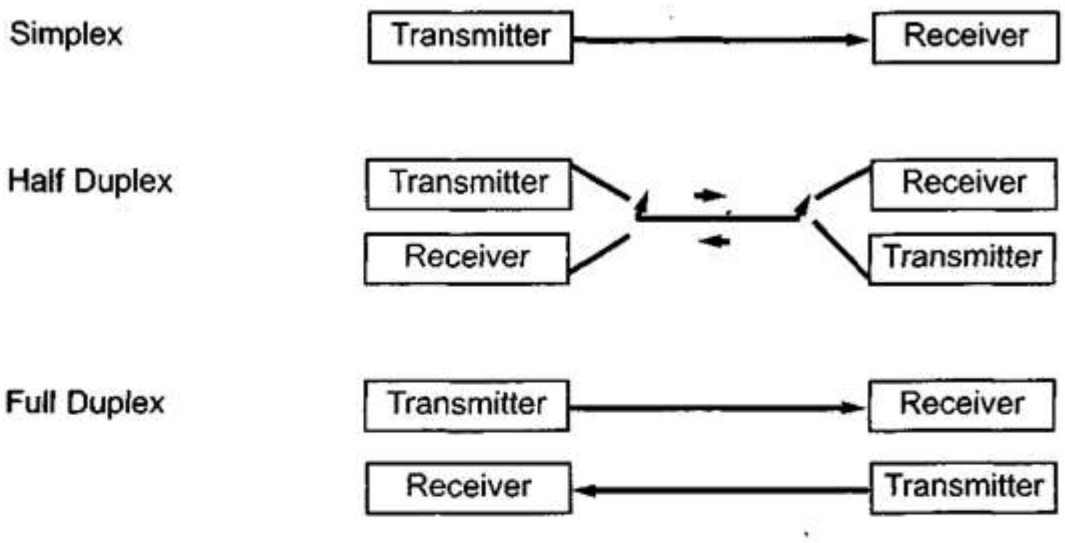


Serial data communication Methods

- Serial data communication uses two methods, asynchronous and synchronous.
  The *synchronous* method transfers a block of data (characters) at a time, while the *asynchronous* method transfers a single byte at a time. It is possible to write software to use either of these methods, but the programs can be tedious and long. For this reason, there are special 1C chips made by many manufacturers for serial data communications

Modes

. These chips are commonly referred to as UART (universal asynchronous receiver-transmitter) and USART (universal synchronous-asynchronous receiver-transmitter).



Data Transfer Operations in 8051

- Data transfer instructions are responsible for transferring data between various memory storing elements like registers, RAM, and ROM. The execution time of these instructions varies based on how complex an operation they have to perform.
- In the table given in next slides, we have listed all the data transfer instruction. In the table [A]= Accumulator; [Rn]=Register in RAM; DPTR=Data Pointer; PC=Program Counter
- Lets take all these Data Transfer instructions one by one.

Data Transfer Operations in 8051

- 1) MOV Instruction–The MOV instruction has two operands, the source, and the destination. The second operand is the source, whereas the first one is the destination. This instruction uses various addressing modes to move data in the RAM space of the microcontroller.
- Examples-MOV A, R0 //Moves data from the register R0 to the accumulator
- MOV R0,50H //Moves data stored in memory location 50H to Ro
- MOV A,@R0 //Uses data stored in R0 register as an address and moves the data at that location to the accumulator

Data Transfer Operations in 8051

OPCODE	OPERAND	DESCRIPTION
MOV	A,Rn	Moves data from registers in register banks of RAM to accumulator
MOV	A, Address	Moves data from an address in the RAM space to the accumulator
MOV	A,@Rn	Uses data stored in a register as an address and moves the data at that address to the accumulator

Data Transfer Operations in 8051

OPCODE	OPERAND	DESCRIPTION
MOV	A,#Data	Moves data given by programmer directly to the accumulator
MOV	Rn,A	Moves data from the accumulator to registers in register bank
MOV	Rn,Address	Moves data from an address in the RAM space to a register in the register banks

Data Transfer Operations in 8051

OPCODE	OPERAND	DESCRIPTION
MOV	Rn,#Data	Moves data given by a programmer directly to a register in the register banks
MOV	Address,A	Moves data to an address in the RAM space from the Accumulator
MOV	Address,Rn	Moves data to an address in the RAM space from a register in the register banks

Data Transfer Operations in 8051

OPCODE	OPERAND	DESCRIPTION
MOV	Address,Address	Moves data from one address to the other
MOV	Address,@Ri	Uses data stored in a register as an address and moves the data at that address to a register in the register bank
MOV	Address,#Data	Moves data given by the programmer directly to an address

Data Transfer Operations in 8051

OPCODE	OPERAND	DESCRIPTION
MOV	@Rn,A	Moves data from the accumulator to an address which is stored in a register
MOV	@Rn,Address	Moves data from an address to an address which is stored in a register
MOV	@Rn,#Data	Moves data given by the programmer to an address which is stored in the register

Data Transfer Operations in 8051

- **MOVC Instruction**
- MOVC instruction is responsible for moving data from the Program memory (Flash memory) to the RAM for processing it.
- **Example**
- MOVC A,@A+DPTR
- MOVC A,@A+PC
- The table for this instruction is on the next slide

Data Transfer Operations in 8051

OPCODE	OPERAND	DESCRIPTION
MOVC	A, @A+DPTR	Moves data to accumulator from a address stored in the memory location (internal ROM) at A+DPTR
MOVC	A, @A+PC	Moves data to accumulator from a address stored in the memory location(internal ROM) at A+PC

Data Transfer Operations in 8051

- **MOVX Instruction**
- The 8051 microcontroller in most cases has an on-chip 4K flash memory, but due to its 16-bit address bus, it can access 64k memory locations. Due to this reason, the 8051 can be interfaced with external memory using ports 0 and 2. To access data in this external memory, the MOVX instruction is used
- The Table for this instruction starts from the next slide.

Data Transfer Operations in 8051

OPCODE	OPERAND	DESCRIPTION
MOVX	A, @Rn	Moves data to accumulator from a memory location (External ROM)
MOVX	@Rn,A	Moves data to Memory location (External ROM) from a register in the register bank
MOVX	A, @DPTR	Moves data to accumulator from a memory location (External ROM) pointed by the Data Pointer
MOVX	@DPTR,A	Moves data to Memory location (External ROM)

Data Transfer Operations in 8051

- **Stack operations**
- The RAM of the 8051 microcontroller is home to a set of 32 general-purpose registers (00H-1FH). These registers are 8 bit wide and are bundled in groups of 8 forming four register banks. Stack operations can be used to place data into these registers in an efficient manner. These stack operations use special commands (PUSH, POP) to place and extract data from these general purposes registers.
- **The PUSH operation**
- The PUSH operation is used to place data into the stack. When this command is given the value of address stored in the stack pointer is increased by one. After incrementing the address in the stack pointer, data is placed at that memory location. For example, when the 8051 is powered up, it holds the address 07H. When it receives the first PUSH instruction, the address is updated to 08H, and data is stored in that location.

Data Transfer Operations in 8051

- **Example** (R6 contains 80H and stack pointer points at 07H)
- PUSH 6; This instruction moves data stored in register R6 to 08H
- **The POP operation**
- The POP operation is used to extract data that is stored in the stack. This operation is the complete opposite of the PUSH operation. It extracts the data from the location which the stack pointer points to and then decreases the value of the SP by 1.
- **Example** (Stack pointer points at memory location 08H which contains 50H)
- POP 6; register R6 now contains the data 50H and the stack pointer points to 07H

Data Transfer Operations in 8051

OPCODE	OPERAND	DESCRIPTION
PUSH	Rn	Places value at the top of the stack
POP	Rn	Extracts the data from the top of the stack

Data Transfer Operations in 8051

- **Exchange Instructions**
- This operation is used to exchange data between the source and the destination operands.
- **Example**
- XCH A, R0; exchanges the data stored in the accumulator and R0
- XCHD A,@R0; Exchanges the lower four bits of a memory location stored in a register with the accumulator

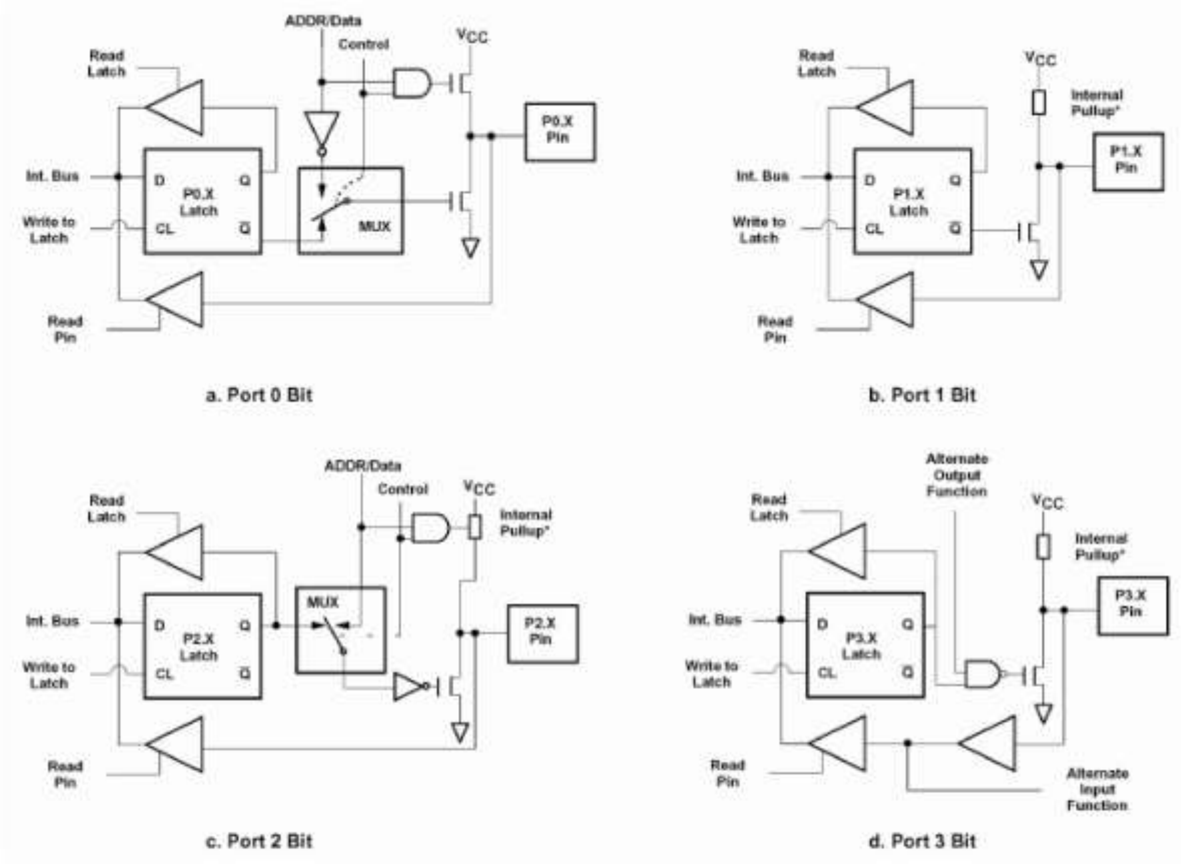
Data Transfer Operations in 8051

OPCODE	OPERAND	DESCRIPTION
XCH	A,Rn	Exchanges the value between a register and the accumulator
XCH	A,Address	Exchanges the value between the accumulator and a memory location in the RAM
XCHD	A,@Rn	Exchanges the value between the accumulator and a memory location stored in the register
XCHD	A,@Rn	Exchanges the lower four bits of a memory location stored in a

Input Output Operations in 8051

- The 8051 has four important ports. Port 0, Port 1, Port 2 and Port 3. These ports allow the microcontroller to connect with the outside world. The four ports of 8051 microcontrollers have certain specific functions and corresponding features. In this post, we will have a look at the purpose of each of these ports.
- **What are the features of the four ports of 8051?**
- Each port has 8 pins. Thus the four ports jointly comprise 32 pins.
- All ports are bidirectional.
- They are constructed with a D type output latch. They have output drivers and input buffers.
- We can modify their functions using software and hardware that they connect to.
- All the ports are configured as input ports on Reset.
- To configure ports as an input port 1 must be written to that port
- To configure it as an output port 0 must be written to it.

Input Output Operations in 8051



a. Port 0 Bit

b. Port 1 Bit

c. Port 2 Bit

d. Port 3 Bit

Input Output Operations in 8051

- Port 0 Features
- Address is 80H
- Construction: Port 0 has a D-type latch, unidirectional buffer, and 2 FETs at each pin. It does not have an internal pull-up resistor. An external pull-up resistor is needed when Port 0 is defined as an output port.
- Port 0 of the 8051 has two main functions: To be used as a simple input-output port and to access external memory in conjunction with Port 2.
- **Functions of Port 0**
- **Simple I/O port:**
- When we use Port 0 as an input port, the internal latch should know that it’s being used for input, and thus, a digital 1 (FFH) is written at the port address of 80H. This turns off the transistors causing the pin to float in high impedance state connecting it to the input buffer. We can read data from ‘Read Pin Data’/’Read Latch Bit.’

Input Output Operations in 8051

- When we use Port 0 as an output port, the latch programmed to 0 will turn on. Consequently, the FET will connect to GND. We will require an external pull up resistor(10k Ohm) here to give a logic ‘1’ for using Port 0 as an output port.
- When the 8051 wants to access external memory, the address for the memory generates due to Port 0 and Port 2. We get the lower half of the address from Port 0 and the upper half from Port 2. This is done using ALE pulses, which help to latch the address to the external bus. Once done, the Port 0 goes back to being an input port to read data from that memory.

Input Output operations in 8051

- **Working of port 0**
- As mentioned above port zero has a lot up its sleeve, from reading data to addresses it does a lot of things for the microcontroller. Therefore it is imperative for us to get a deeper understanding of the workings of this port.
- To configure port 0 as an input port the internal bus writes 1 to the D flip flop and the control pin is set to 0(Upper FET is OFF). The mux is connected to Q'(0) of the D flip flop as the control pin is 0. Due to this, the pin is connected to the input buffer which can be read to get the input data.
- To use the port as an output port 0 is written to the D flip flop with the control signal being set to 0. This enables the lower FET and disables the upper FET due to this the pin gets connected to the ground and a zero is written to the output device. To write a 1 to the external device the microcontroller writes 1 to the D flip flop which drives the pin to a high impedance state as it is not connected to either VCC or ground. To solve this problem a pull-up resistor is connected to the output pin which pulls the value to 5v or logic 1.
- For reading Addresses or data from external memory the Control bit is set to set to 1 which connects the Mux to Data/address pin. The ALE pin is used to latch the address and once that is done the port is used for data transfer.

Input Output Operations in 8051

- **What are the features and functions of Port 1 in 8051?**
- **Features of Port 1:**
- Address is 90H
- Construction: Port 1 has one D latch, two unidirectional buffers, 1 FET, and one internal pull-up resistor at each pin.
- It has only one function – to act as an Input-Output port.
- **The function of Port 1 – I/O port:**
- When Port 1 is functioning in the capacity of an input port, a digital ‘1’ (FFH) is written to the latch. At 90H. This turns off the transistor, and the pin floats in a high impedance state. Consequently, it connects to the input buffer.
- When Port 1 is functioning in the capacity of an output port, the latch is given a ‘LOW’ signal (00H). This turns the FER (Field Effect Transistor) o. The pull-up resistor is OFF, and the port is used as an output port.

Input Output Operations in 8051

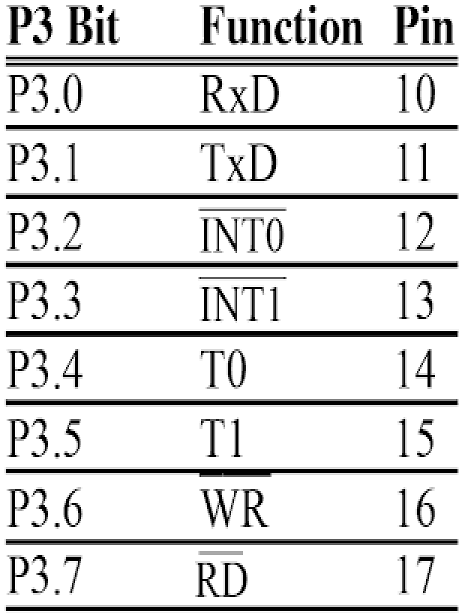
- **What are the features and functions of Port 2 in 8051?**
- **Features of Port 2**
- Address is 10H
- Construction: Port 2 has a D type latch, 1 FET, an internal pull-up resistor, two unidirectional buffers, and a Control Logic block.
- Its main functions are kind of similar to those of Port 0. It can be used as an input-output port. And can access external memory in conjunction with Port 0.
- **Functions of Port 2**
- **I/O port:**
- Quite similar to Port 0. The only difference here is that in Port 2, we use one FET with an internal pull-up resistor instead of the two FETs we saw in Port 0.
- **Memory Access:**
- Port 2 is used in conjunction with Port 0 to generate the upper address of the external memory location that needs to be accessed. However, one key difference is that it doesn’t need to turnaround and get a 1 in the latch immediately for input as in Port 0. It can remain stable.

Input Output Operations in 8051

- **What are the features and functions of Port 3 in 8051?**
- **Features of Port 3**
- Address is B0H
- Construction: The third Port of 8051 has a D-type latch. In addition to that, it has three unidirectional buffers. A FET with an internal pull-up resistor. Additionally, it also has a NAND gate connected to the FET.
- Port 3 performs two main functions, as we will see below.
- **Functions of Port 3**
- **I/O port**
- Just like Port 2, Port 3 can function as an input-output port.
- **Alternate SFR function**
- The input to SFR 1, we get the output of latch as 1, which turns on the NAND gate, and depending on the value of 'Alternate Output Pin,' FET will be wither ON/OFF.

Input Output Operations in 8051

RXD: this is used for a serial input port
TXD: this is used for serial output port
INT0: this used for an external interrupt 0
INT1: this used for external interrupt 1
T0: Timer 0 external input
T1: Timer 1 external input
WR: external data memory write strobe
RD: external data memory Read strobe



P3 Bit	Function	Pin
P3.0	RxD	10
P3.1	TxD	11
P3.2	$\overline{\text{INT0}}$	12
P3.3	$\overline{\text{INT1}}$	13
P3.4	T0	14
P3.5	T1	15
P3.6	$\overline{\text{WR}}$	16
P3.7	$\overline{\text{RD}}$	17

Instruction sets for Ports

INSTRUCTION	EXAMPLE	EXPLANATION
MOV	MOV A,Port	Moves data from a given port to the accumulator
JNB	JNB Port,Address	Checks the value in the input buffer. If the value is zero then it transfers the control to the given address
JB	JB Port,Address	Checks the value in the input buffer. If the value is not zero then it transfers the control to the given address
CJNE	CJNE A,Port,Address	Checks the value in the input buffer of a port. Compares it to the value in the accumulator. If the values are not the same then the control is transferred to a given address

Instruction sets for Ports

INSTRUCTION	EXAMPLE	EXPLANATION
ANL	ANL P1, A	Performs logical AND between the value stored in the latch of the port and the accumulator
ORL	ORL P1, A	Performs logical OR between the value stored in the latch of the port and the accumulator. After this, it writes the new value to the latch
XRL	XRL P1, A	Performs logical XOR between the value stored in the latch of the port and the accumulator. After this, it writes the new value to the latch
JBC	JBC,Port,Address	If the value at a given port is 1 then it jumps to a given address and then clears the latch

Instruction sets for Ports

INSTRUCTION	EXAMPLE	EXPLANATION
CPL	CPL,Port	Complements the data in the latch
MOV	MOV Port,C	Reads the value at the latch of a given port and then transfers it to the carry flag
CLR	CLR Port	Clears the value stored at the latch of a port
SETB	SETB Port	Sets the value of a latch

Chapter 9

Design and Interfacing with 8051

Contents

- keypad interface
- 7- segment interface
- LCD Interfacing
- Stepper motor Interfacing

Interfacing the keyboard to the 8051

- At the lowest level, keyboards are organized in a matrix of rows and columns.
- The CPU accesses both rows and columns through ports; therefore, with two 8-bit ports, an 8 x 8 matrix of keys can be connected to a microprocessor.
- When a key is pressed, a row and a column make a contact; otherwise, there is no
- In such systems, it is the function of programs stored in the EPROM of the microcontroller to scan the keys continuously, identify which one has been activated, and present it to the motherboard. In this section we look at the mechanism by which the 8051 scans and identifies the key.

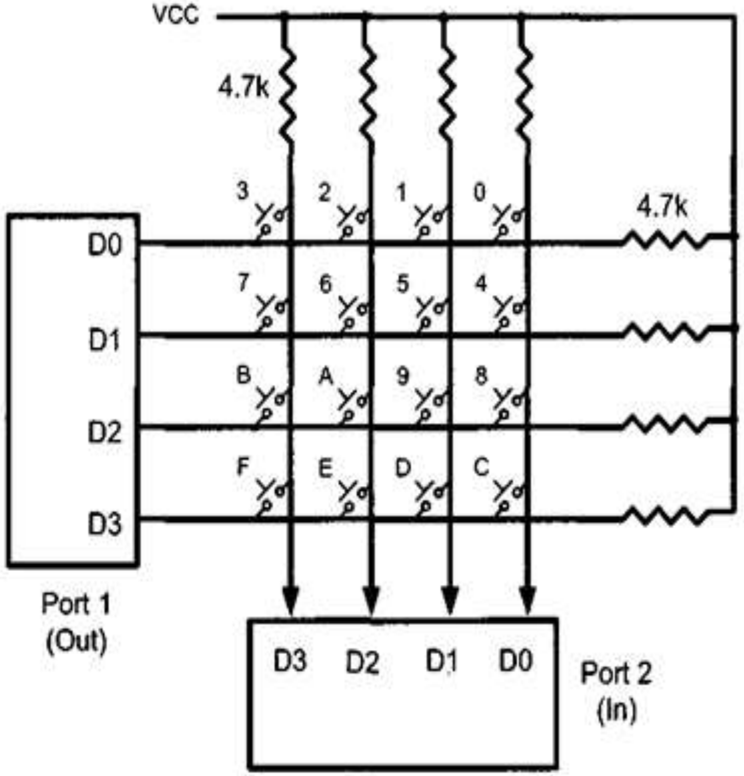
4x4 Keypad

- 4 x 4 matrix connected to two ports. The rows are connected to an output port and the columns are connected to an input port.
- If no key has been pressed, reading the input port will yield 1 s for all columns since they are all connected to high (V_{cc}).
- If all the rows are grounded and a key is pressed, one of the columns will have 0 since the key pressed provides the path to ground.
- It is the function of the microcontroller to scan the keyboard continuously to detect and identify the key pressed.

Keypad

- Keypad is used as an input device to read the key pressed by user and to process it.
- 4x4 keypad consists of 4 rows and 4 columns. Switches are placed between the rows and columns. A key press establishes a connection between corresponding row and column between which the switch is placed.
- To read the key press, we need to configure the rows as outputs and columns as inputs.
- Columns are read after applying signals to the rows in order to determine whether or not a key is pressed and if pressed, which key is pressed.

Interfacing



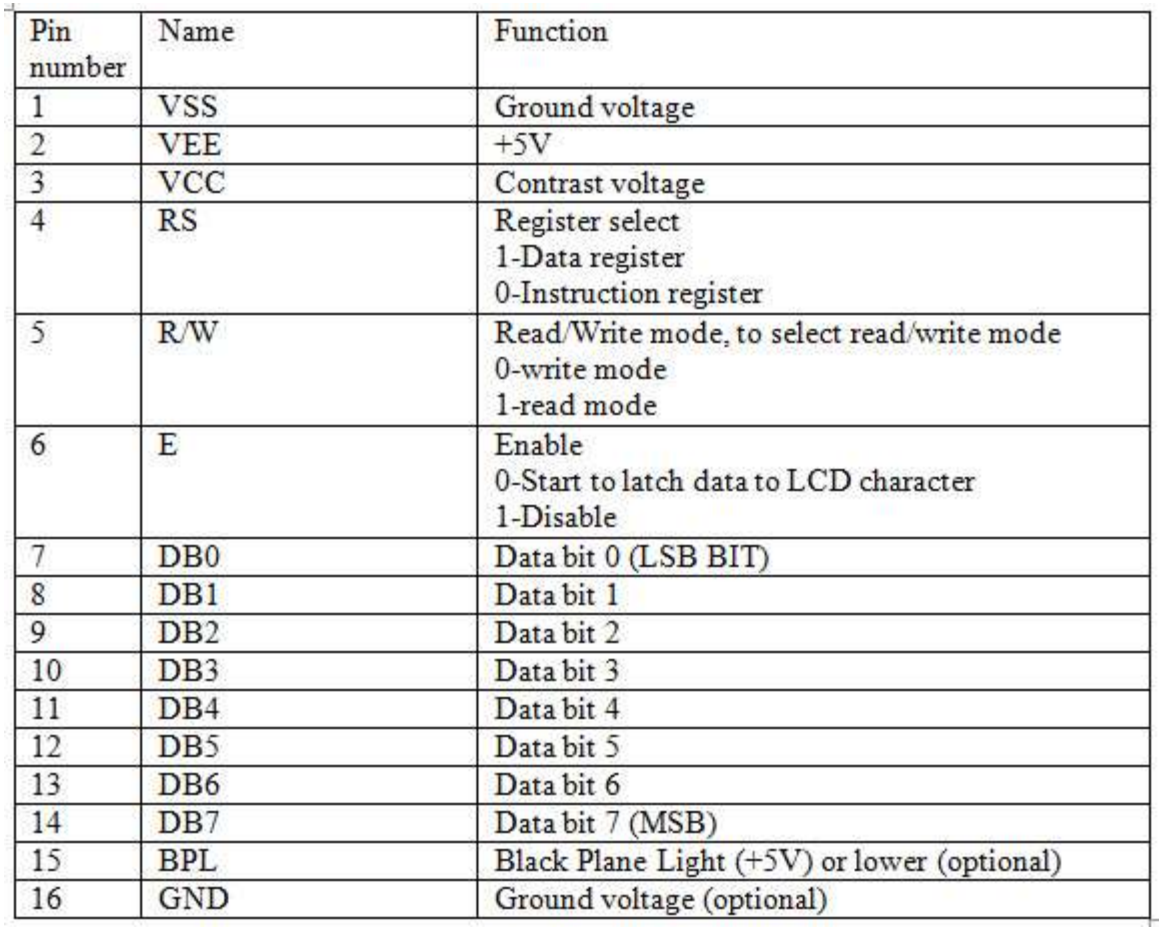
LCD Interfacing

- Display units are the most important output devices in embedded projects and electronics products. 16x2 LCD is one of the most used display unit. 16x2 LCD means that there are two rows in which 16 characters can be displayed per line, and each character takes 5X7 matrix space on LCD.

LCD Pins

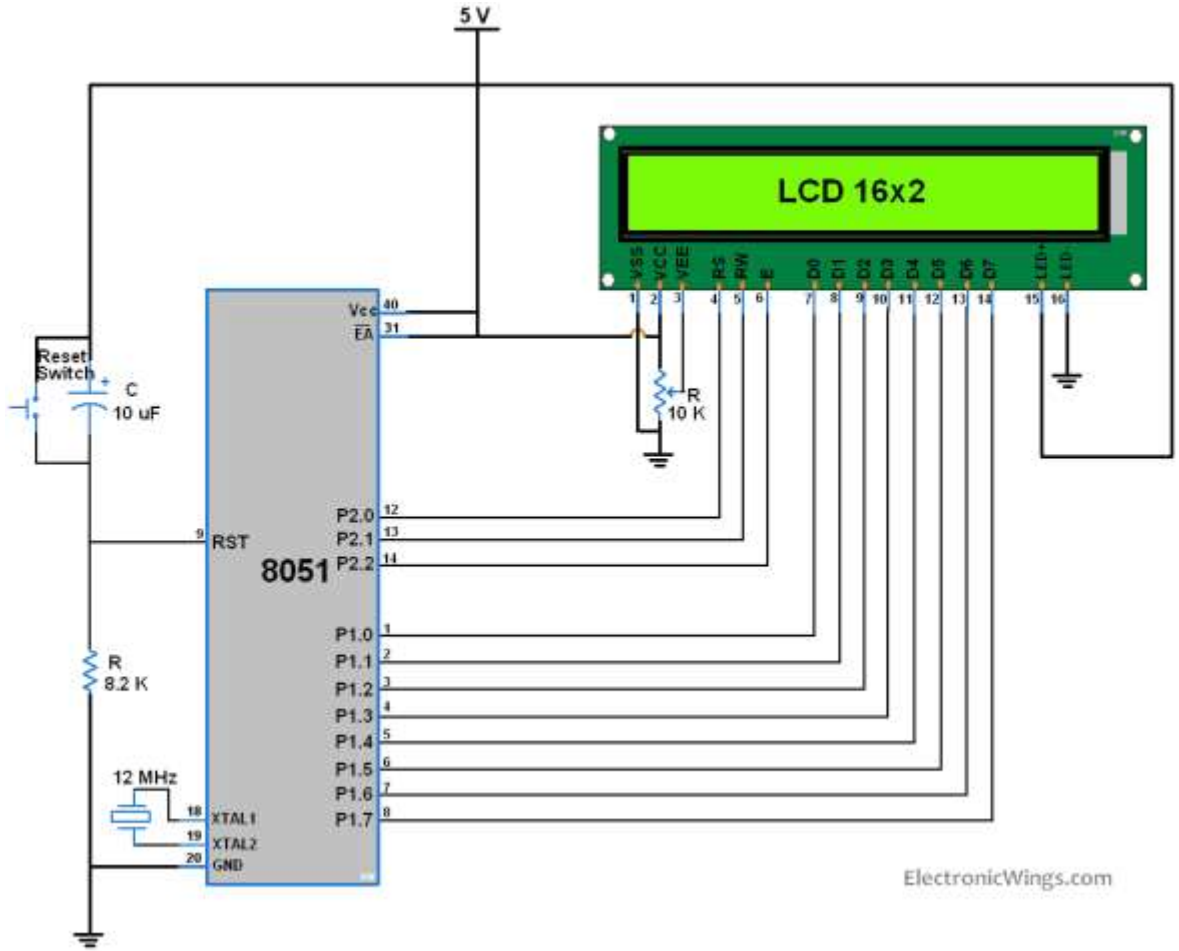
- LCD 16x2 is 16 pin device which has 8 data pins (D0-D7) and 3 control pins (RS, RW, EN). The remaining 5 pins are for supply and backlight for the LCD.
- The control pins help us configure the LCD in command mode or data mode. They also help configure read mode or write mode and also when to read or write.
- LCD 16x2 can be used in 4-bit mode or 8-bit mode depending on the requirement of the application. In order to use it we need to send certain commands to the LCD in command mode and once the LCD is configured according to our need, we can send the required data in data mode.

LCD PINS



Pin number	Name	Function
1	VSS	Ground voltage
2	VEE	+5V
3	VCC	Contrast voltage
4	RS	Register select 1-Data register 0-Instruction register
5	R/W	Read/Write mode, to select read/write mode 0-write mode 1-read mode
6	E	Enable 0-Start to latch data to LCD character 1-Disable
7	DB0	Data bit 0 (LSB BIT)
8	DB1	Data bit 1
9	DB2	Data bit 2
10	DB3	Data bit 3
11	DB4	Data bit 4
12	DB5	Data bit 5
13	DB6	Data bit 6
14	DB7	Data bit 7 (MSB)
15	BPL	Black Plane Light (+5V) or lower (optional)
16	GND	Ground voltage (optional)

Interfacing



LCD Initialization

- For initializing the LCD, the following are the steps that are given below and these steps are the same for almost all the applications.
- Send 38H to the 8-bit data line for initialization
- Send 0FH for making LCD ON, cursor ON, cursor blinking ON
- Send 06H for incrementing cursor position
- Send 01H for clearing the display and return the cursor

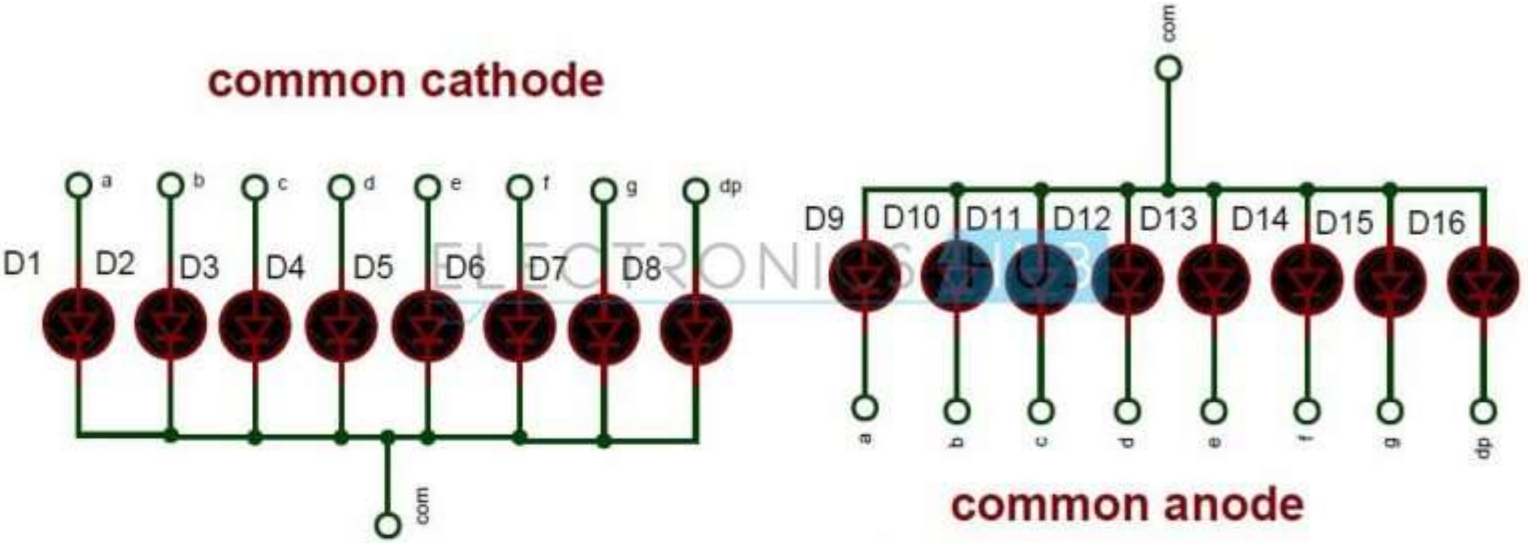
Steps for Sending Data to the LCD

- The logic state of these pins that make the module to determine whether a given data input is a data or command to be displayed.
- Make R/W low
- Make RS=1, if the data byte is a data to be displayed and make
- RS=0, if the data byte is a command.
- Place data byte on the data register
- Then pulse E from high to low
- Repeat the above steps for sending other data

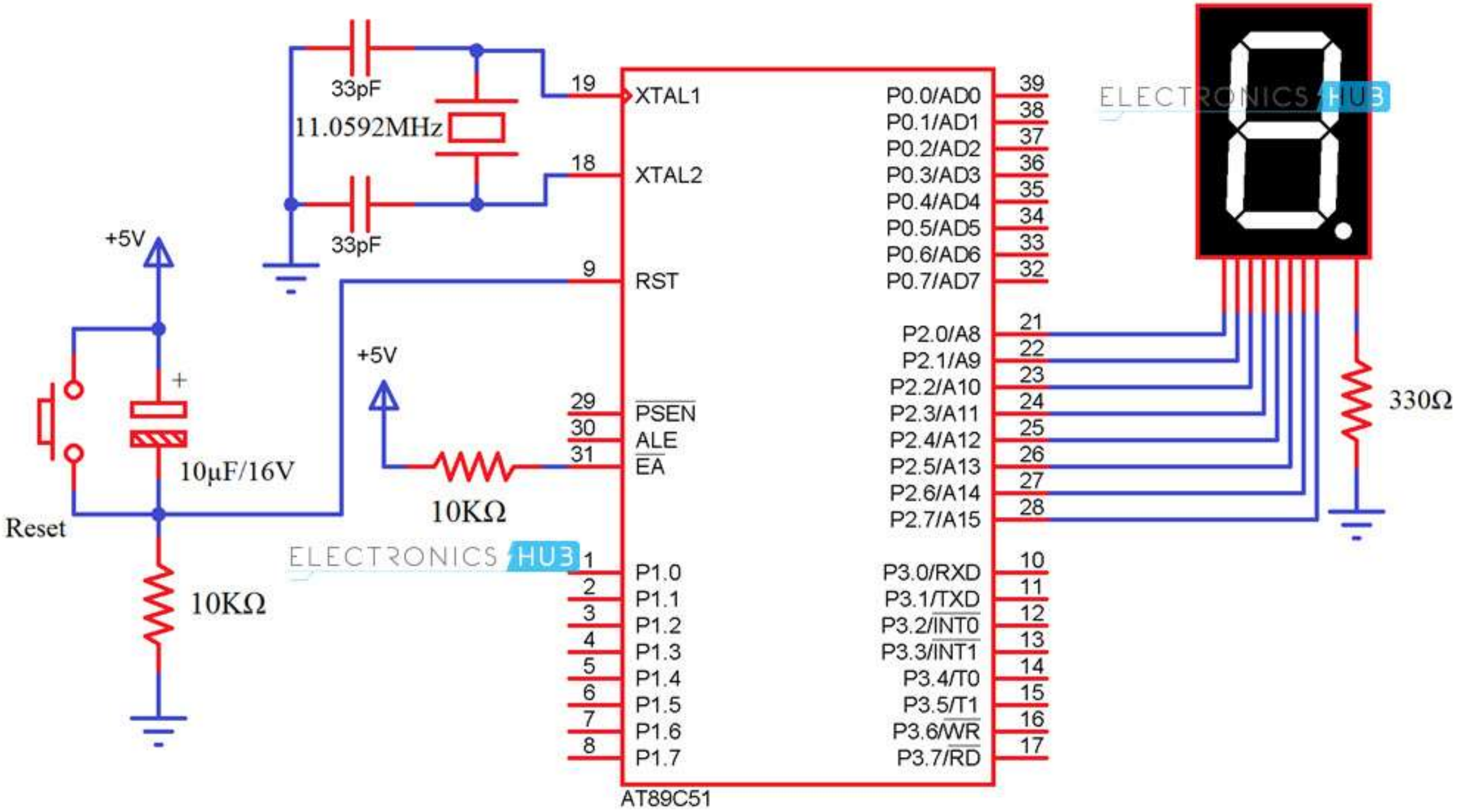
Seven segment displays

- Seven segment displays internally consist of 8 LEDs. In these LEDs, 7 LEDs are used to indicate the digits 0 to 9 and single LED is used for indicating decimal point. Generally seven segments are two types, one is common cathode and the other is common anode.
- In common cathode, all the cathodes of LEDs are tied together and labeled as com. and the anode are left alone. In common anode, seven segment display all the anodes are tied together and cathodes are left freely. Below figure shows the internal connections of seven segment Display.

Seven segment displays



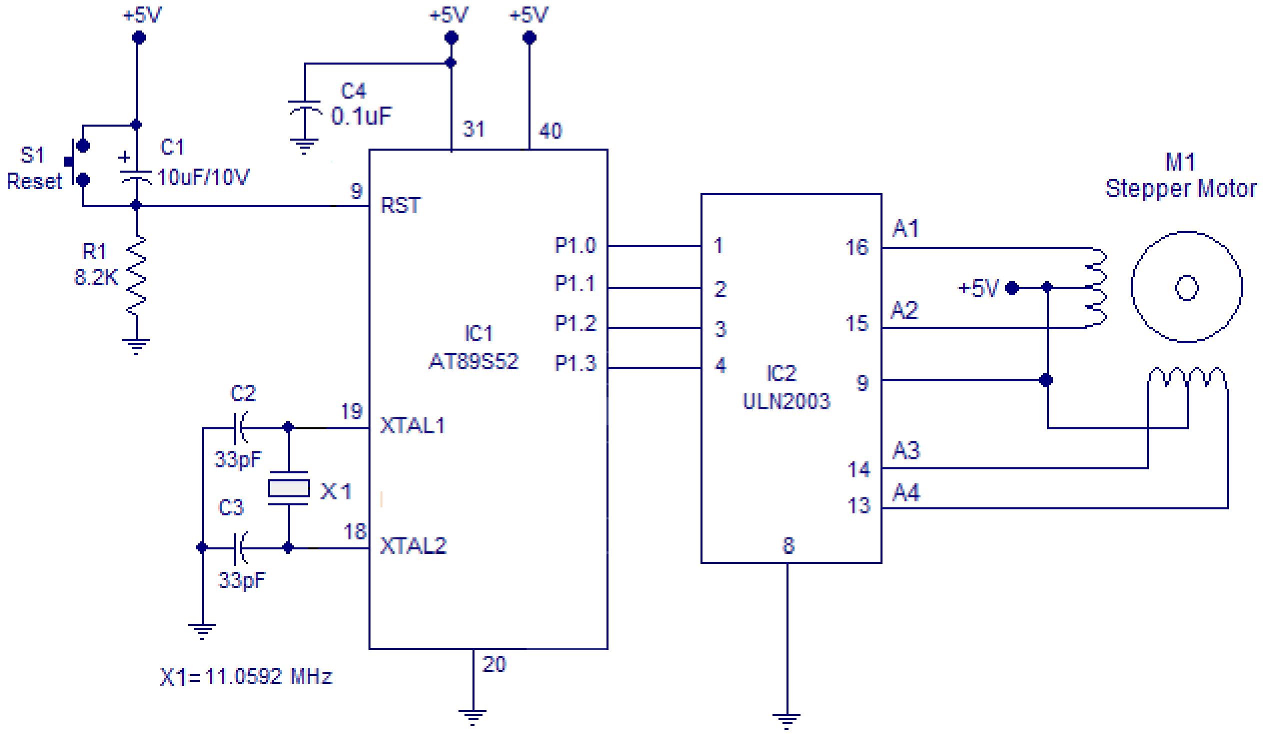
Interfacing 7 Segment Display to 8051



Stepper Motor

- A stepper motor is a brushless and synchronous motor which divides the complete rotation into number of steps. Each stepper motor will have some fixed step angle and motor rotates at this angle.
- To interface a Stepper Motor with 8051 two different drivers: L293D and ULN 2003 are required.
- The main principle of these circuits is to rotate the stepper motor step wise at a particular step angle.
- The step size of the motor is determined by the number of phases and the number of teeth on the rotor. Step size is the angular displacement of the rotor in one step.

Stepper Motor



Interfacing stepper motor to 8051

- P1.0, P1.1, P1.2 and P1.3 pins are used for controlling the phases A1, A2, A3 and A4 of the stepper motor respectively.
- ULN2003 is used for driving the individual phases of the stepper motor. ULN2003 is used for driving high current loads such as relays and motors.
- ULN2003 has 8 individual channels each with 1A capacity. The channels can be paralleled to increase the current capacity. Each channels are fitted with individual freewheeling diodes.
- The ULN2003 is operated in current sinking mode. Each channel is activated by giving a logic LOW at the corresponding input.
- For example if we make pin 1 of ULN2003 LOW, phase A1 of the stepper motor gets switched ON.

Chapter 10

Application of Micro controllers

Microcontroller

- Microcontroller is termed as “**Computer-on-a-Chip**“. It is named so, because not only the CPU, but RAM, ROM, I/O ports, Timer/Counter, Serial I/Os all are put together on a single microcontroller chip. Microcontrollers are task specific and are essentially used for making *Embedded Systems*.

Classification According to Number of Bits

- The bits in microcontroller are 8-bits, 16-bits and 32-bits microcontroller.
- In **8-bit** microcontroller, the point when the internal bus is 8-bit then the ALU is performs the arithmetic and logic operations.
- The examples of 8-bit microcontrollers are Intel 8031/8051, PIC1x and Motorola MC68HC11 families.

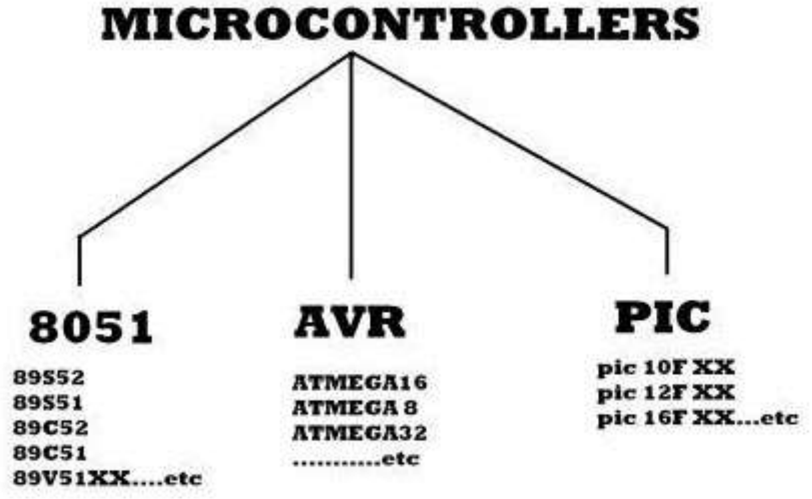
16-bit microcontroller

- The **16-bit** microcontroller performs greater precision and performance as compared to 8-bit. For example 8 bit microcontrollers can only use 8 bits, resulting in a final range of 0×00 – 0xFF (0-255) for every cycle.
- In contrast, 16 bit microcontrollers with its 16 bit data width has a range of 0×0000 – 0xFFFF (0-65535) for every cycle. A longer timer most extreme worth can likely prove to be useful in certain applications and circuits. It can automatically operate on two 16 bit numbers.
- Some examples of 16-bit microcontroller are 16-bit MCUs are extended 8051XA, PIC2x, Intel 8096 and Motorola MC68HC12 families.

32-bit Microcontroller

- The **32-bit** microcontroller uses the 32-bit instructions to perform the arithmetic and logic operations.
- These are used in automatically controlled devices including implantable medical devices, engine control systems, office machines, appliances and other types of embedded systems.
- Some examples are Intel/Atmel 251 family, PIC3x.

Types of Microcontroller



MICROCONTROLLERS

8051

89S52
89S51
89C52
89C51
89V51XX....etc

AVR

ATMEGA16
ATMEGA 8
ATMEGA32
............etc

PIC

pic 10F XX
pic 12F XX
pic 16F XX...etc

Applications of AVR Microcontroller

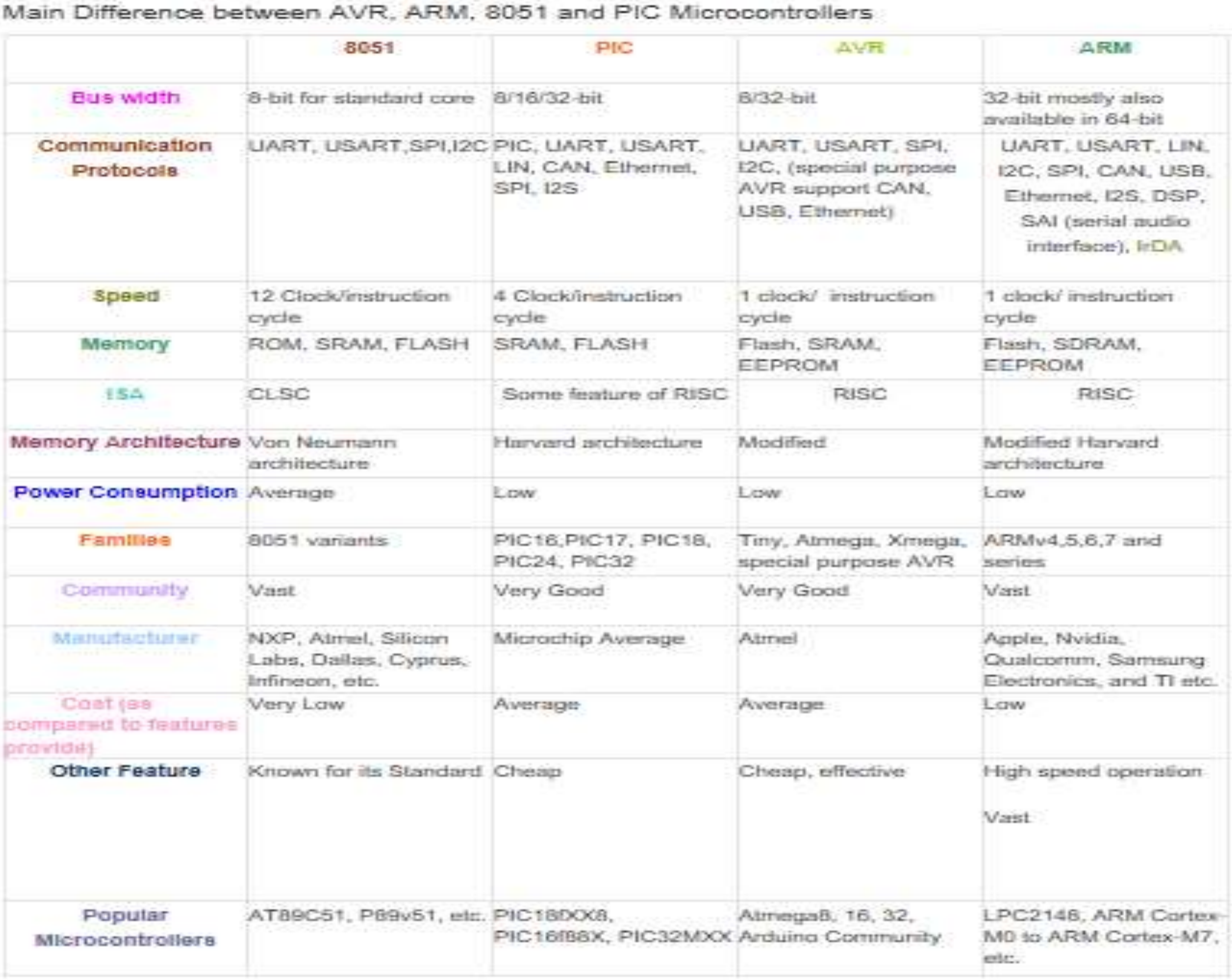
- There are many applications of AVR microcontroller; they are used in home automation, touch screen, automobiles, medical devices and defense.

PIC Microcontroller

- **PIC microcontrollers** (Programmable Interface Controllers), are electronic circuits that can be programmed to carry out a vast range of tasks. They can be programmed to be timers or to control a production line and much more.
- **Arduino** is based on the Atmel Atmega series **microcontrollers** while **PIC**(Pheripheral Interface **Controller**) is a **microcontroller** family specially designed for peripheral interfaces.

Features of PIC

- Flash **memory** (program **memory**, programmed using MPLAB **devices**)
- SRAM (data **memory**)
- EEPROM **memory** (programmable at run-time)
- Sleep mode (power savings)
- Watchdog timer.
- Various crystal or RC oscillator configurations, or an external **clock**.



Main Difference between AVR, ARM, 8051 and PIC Microcontrollers

	8051	PIC	AVR	ARM
Bus width	8-bit for standard core	8/16/32-bit	8/32-bit	32-bit mostly also available in 64-bit
Communication Protocols	UART, USART,SPI,I2C	PIC, UART, USART, LIN, CAN, Ethernet, SPI, I2S	UART, USART, SPI, I2C, (special purpose AVR support CAN, USB, Ethernet)	UART, USART, LIN, I2C, SPI, CAN, USB, Ethernet, I2S, DSP, SAI (serial audio interface), IrDA
Speed	12 Clock/instruction cycle	4 Clock/instruction cycle	1 clock/ instruction cycle	1 clock/ instruction cycle
Memory	ROM, SRAM, FLASH	SRAM, FLASH	Flash, SRAM, EEPROM	Flash, SDRAM, EEPROM
ISA	CLSC	Some feature of RISC	RISC	RISC
Memory Architecture	Von Neumann architecture	Harvard architecture	Modified	Modified Harvard architecture
Power Consumption	Average	Low	Low	Low
Families	8051 variants	PIC16,PIC17, PIC18, PIC24, PIC32	Tiny, Atmega, Xmega, special purpose AVR	ARMv4,5,6,7 and series
Community	Vast	Very Good	Very Good	Vast
Manufacturer	NXP, Atmel, Silicon Labs, Dallas, Cyprus, Infineon, etc.	Microchip Average	Atmel	Apple, Nvidia, Qualcomm, Samsung Electronics, and TI etc.
Cost (as compared to features provide)	Very Low	Average	Average	Low
Other Feature	Known for its Standard	Cheap	Cheap, effective	High speed operation Vast
Popular Microcontrollers	AT89C51, P89v51, etc.	PIC18DXX8, PIC16f88X, PIC32MXX	Atmega8, 16, 32, Arduino Community	LPC2148, ARM Cortex-M0 to ARM Cortex-M7, etc.

Applications

8051 Products

- **Consumer Electronics Products**:
- Toys, Cameras, Robots, Washing Machine, Microwave Ovens etc. [any automatic home appliance]
- **2. Instrumentation and Process Control**:
- Oscilloscopes, Multi-meter, Leakage Current Tester, Data Acquisition and Control etc.
- **3. Medical Instruments**:
- ECG Machine, Accu-Check etc.
- **4. Communication**:
- Cell Phones, Telephone Sets, Answering Machines etc.
- **5. Office Equipment**:
- Fax, Printers etc.
- **6. Multimedia Application**:
- Mp3 Player, PDAs etc.
- **7. Automobile**:
- Speedometer, Auto-breaking system etc.