Microcontrollers

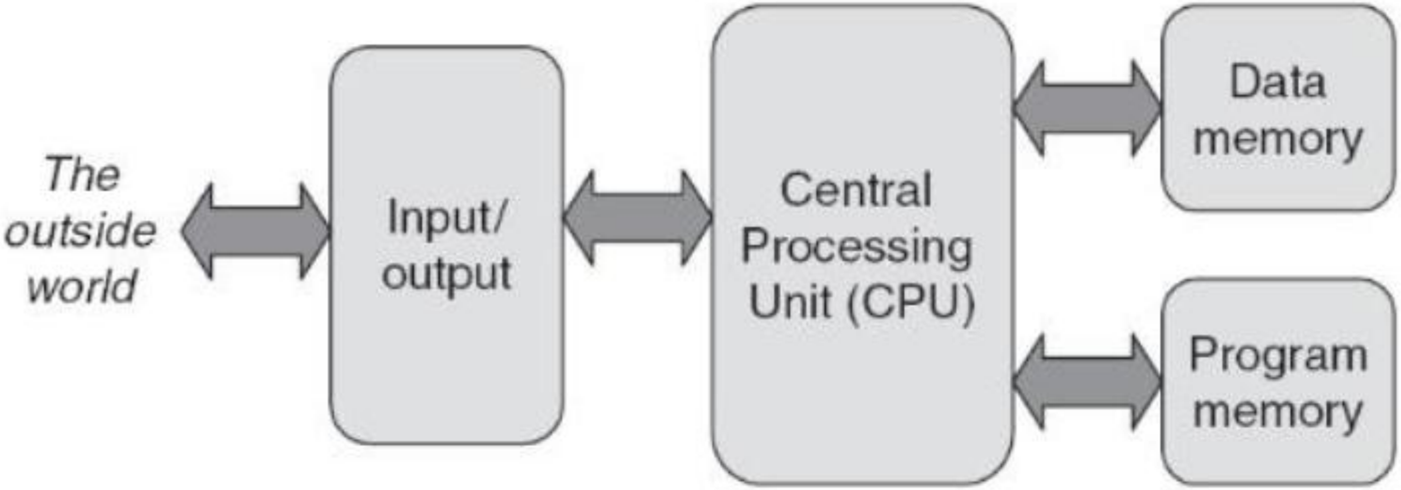
A microcontroller is a compact integrated circuit designed to govern a specific operation in an embedded system. A typical microcontroller includes a processor, memory and input/output (I/O) peripherals on a single chip.

8051 is one of the first and most popular microcontrollers also known as MCS-51. Intel introduced it in the year 1981.

These microcontrollers were named 80C51, where C in the name tells that it is based on CMOS technology.

It is an 8-bit microcontroller which means the data bus is 8-bit. Therefore, it can process 8 bits at a time.

Computing Essentials



Components of a Microprocessor/Controller

CPU: Central Processing Unit

I/O: Input /Output

Bus: Address bus & Data bus

Memory: RAM & ROM

Timer

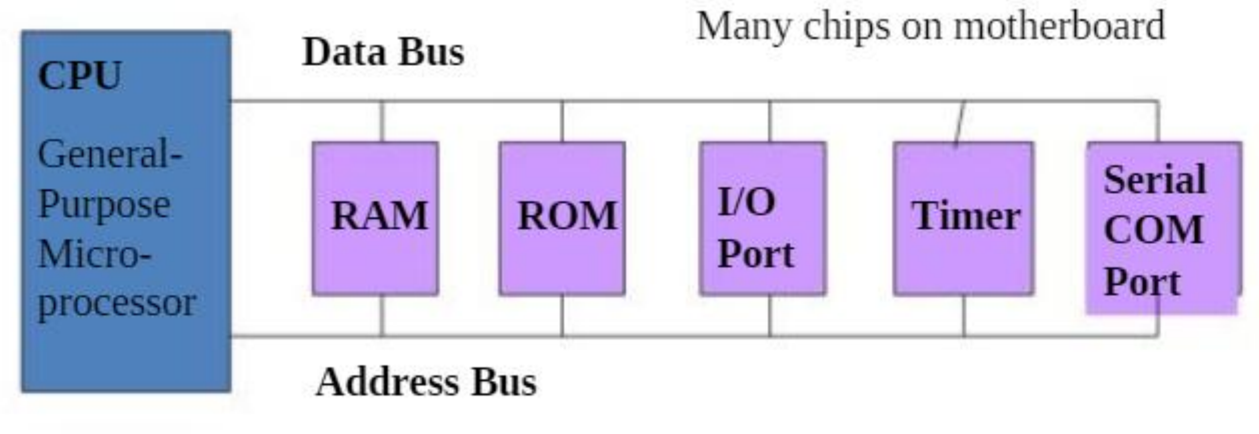
Interrupt

Serial Port

Parallel Port

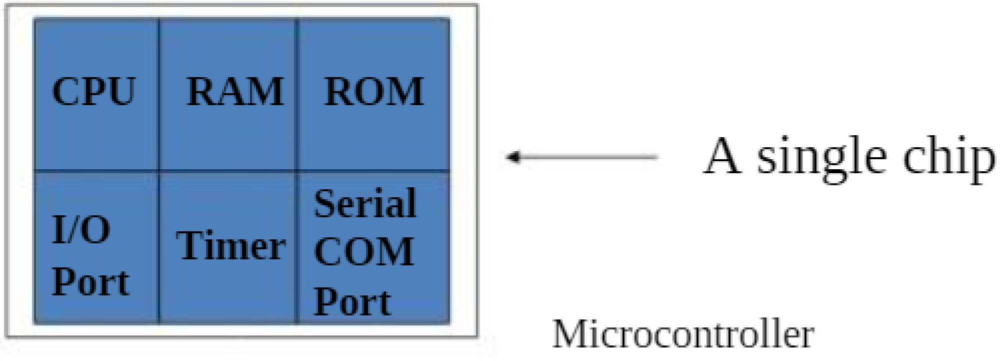
General-purpose microprocessor

- CPU for Computers
- Commonly no RAM, ROM, I/O on CPU chip itself



Microcontroller

- A single-chip computer
- On-chip RAM, ROM, I/O ports

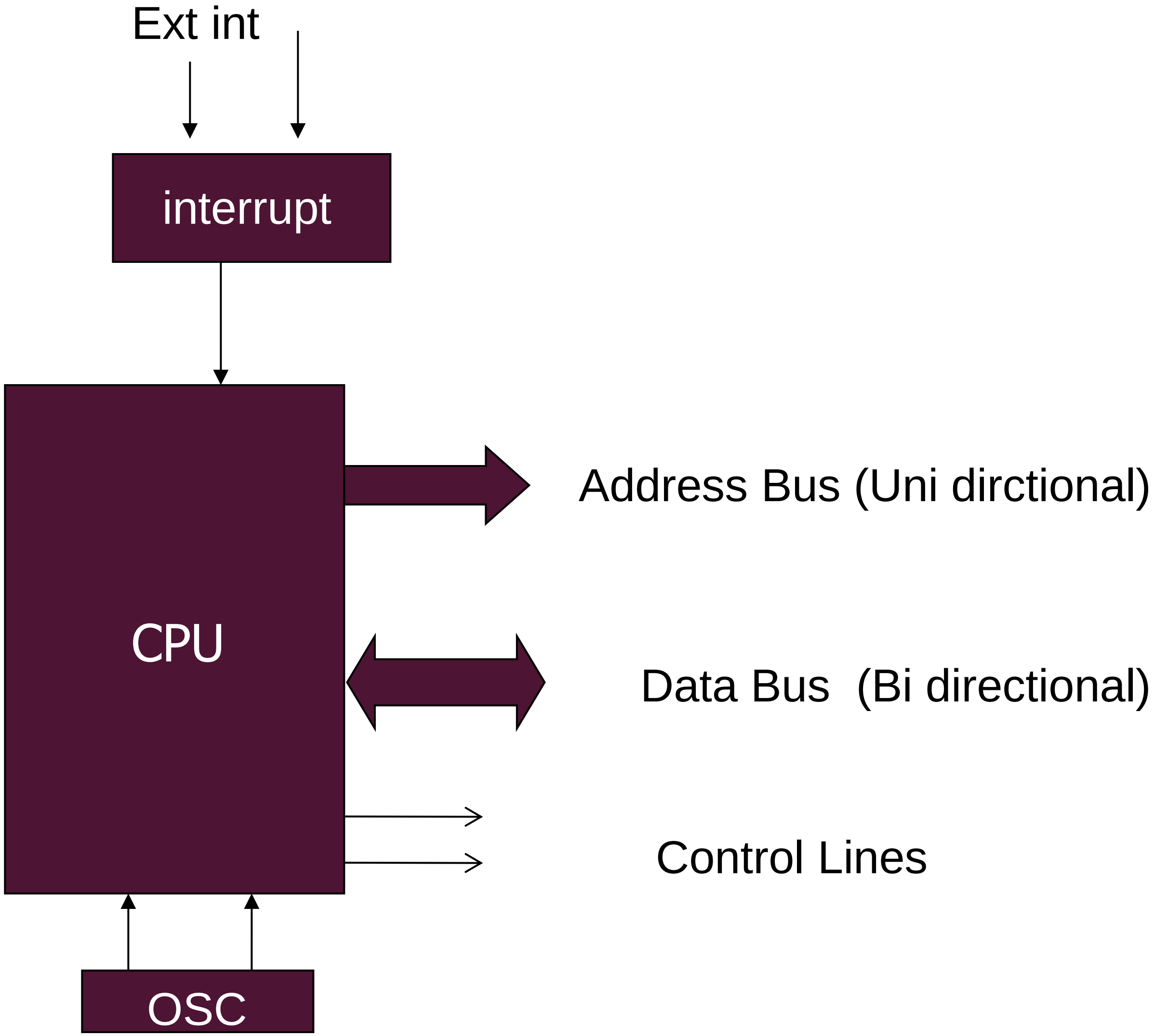


CPU	RAM	ROM
I/O Port	Timer	Serial COM Port

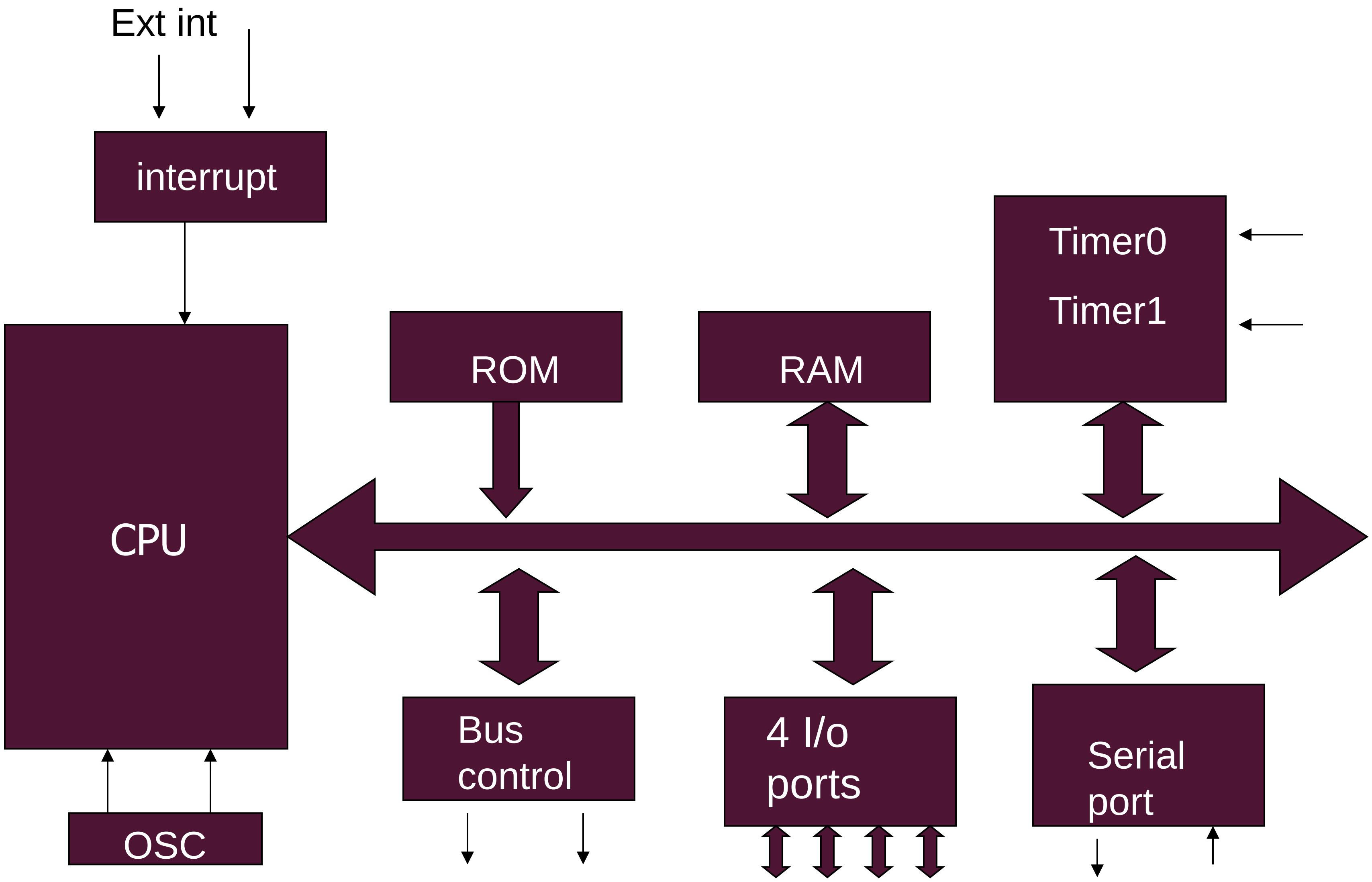
A single chip

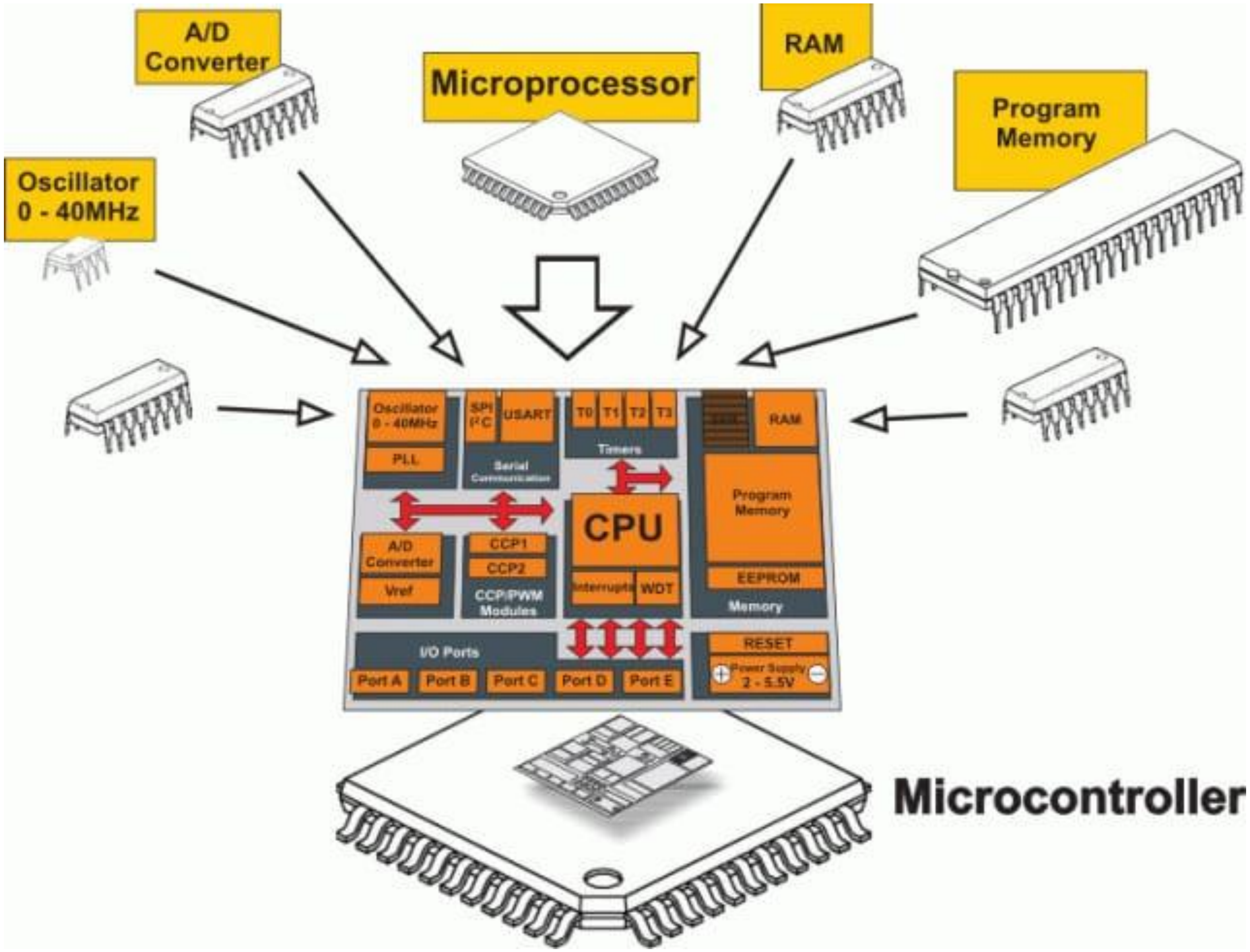
Microcontroller

MicroProcessor



Microcontroller




A/D Converter
Microprocessor
RAM
Program Memory
Oscillator 0 - 40MHz
Oscillator 0 - 40MHz
PLL
SPI I²C
USART
Serial Communication
T0
T1
T2
T3
Timers
RAM
Program Memory
CPU
A/D Converter
Vref
CCP1
CCP2
CCP/PWM Modules
Interrupts
WDT
EEPROM
Memory
I/O Ports
Port A
Port B
Port C
Port D
Port E
RESET
Power Supply 2 - 5.5V
Microcontroller

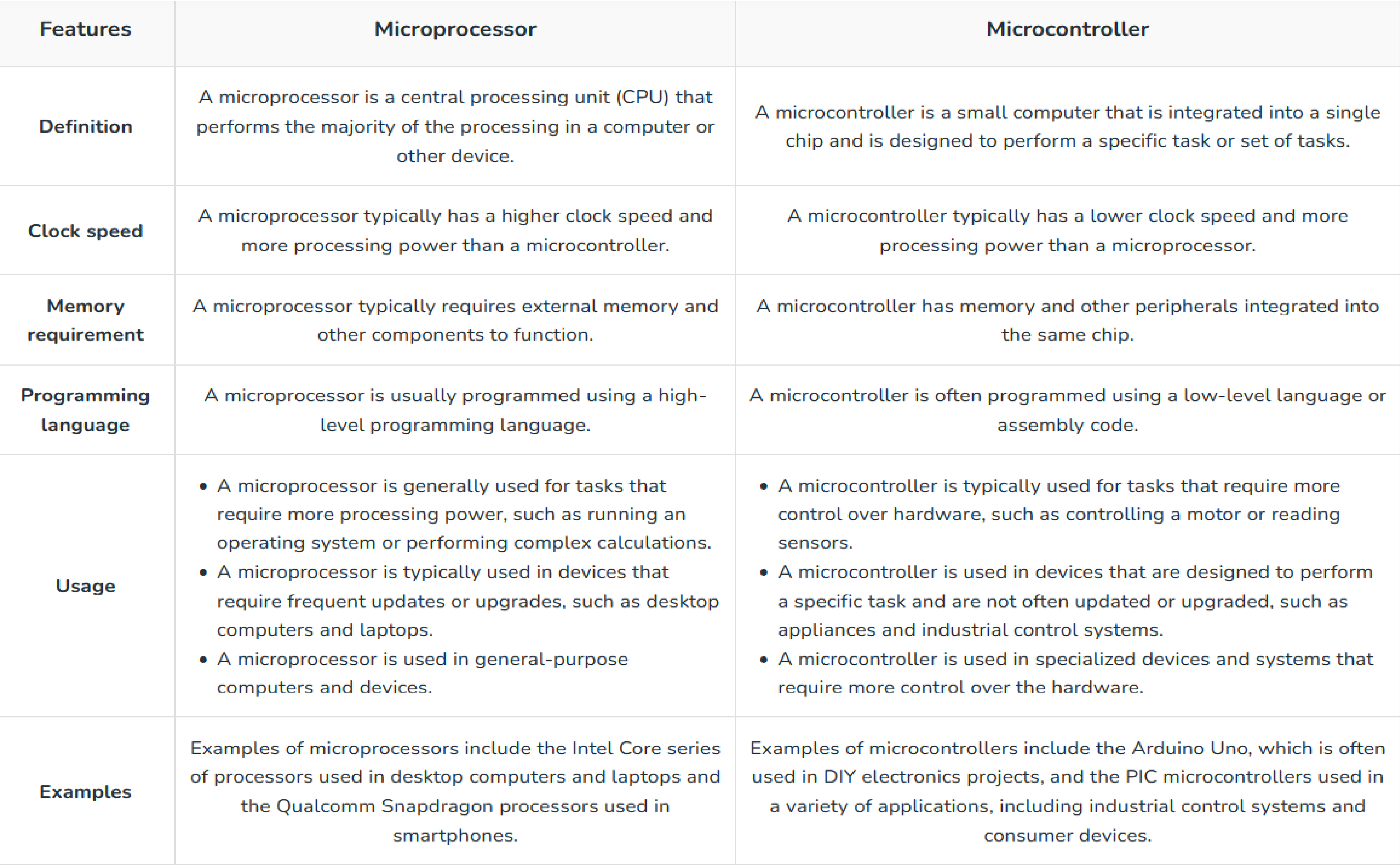
Microprocessors and Microcontrollers

- The microprocessor is a processor on one silicon chip.
- A uP system is uP plus all the components it requires to do a certain task
- The microcontrollers are used in embedded computing.
- The microcontroller is a microprocessor with added circuitry.
- They are a complete computer on a chip containing direct input and output capability and memory along with the uP on a single chip.
- Many times they contain other specialized application-specific components as well

MICROPROCESSOR VS MICROCONTROLLER

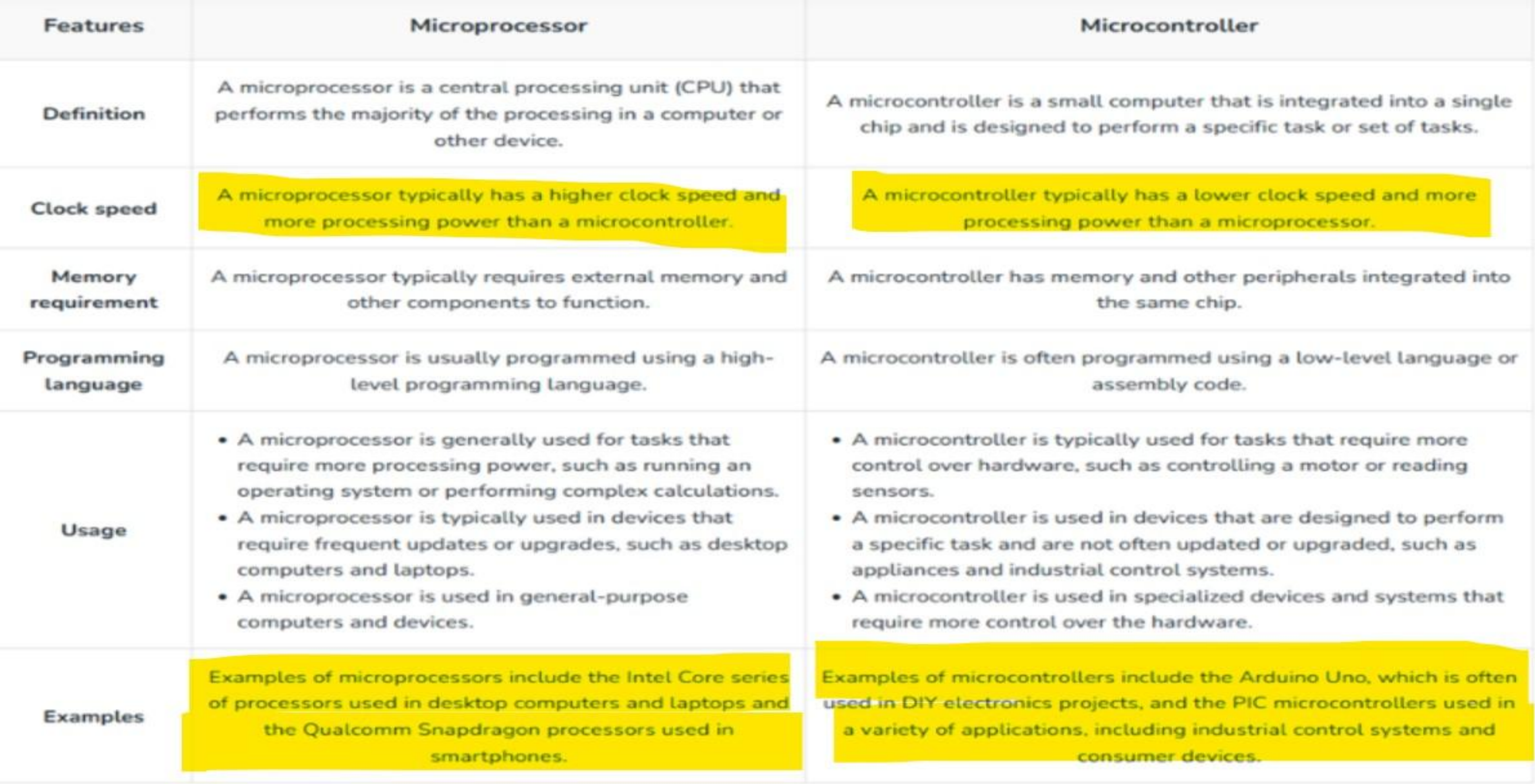
Microprocessor	**Microcontroller**
Since memory and I/O are connected externally, the circuit becomes large in size.	Since memory and I/O are present together, the internal circuit is small in size.
Cost is high	Cost is low
RAM, ROM, I/O units, and other peripherals are not embedded on a single chip.	RAM, ROM, CPU and other peripherals are embedded on a single chip.
Used in personal computers.	Used in embedded systems.
Uses an external bus.	Uses an internal controlling bus.
Based on the Von Neumann model	Based on the Harvard architecture
It is a CPU on a single silicon-based integrated chip.	It is a byproduct of the development of microprocessors with a CPU along with other peripherals.
Complex and expensive due to a large number of instructions to process.	Simple and inexpensive due to less number of instructions to process.
Can run at a very high speed.	Can run up to 200MHz or more.

Microprocessors and Microcontrollers



Features	Microprocessor	Microcontroller
Definition	A microprocessor is a central processing unit (CPU) that performs the majority of the processing in a computer or other device.	A microcontroller is a small computer that is integrated into a single chip and is designed to perform a specific task or set of tasks.
Clock speed	A microprocessor typically has a higher clock speed and more processing power than a microcontroller.	A microcontroller typically has a lower clock speed and more processing power than a microprocessor.
Memory requirement	A microprocessor typically requires external memory and other components to function.	A microcontroller has memory and other peripherals integrated into the same chip.
Programming language	A microprocessor is usually programmed using a high-level programming language.	A microcontroller is often programmed using a low-level language or assembly code.
Usage	• A microprocessor is generally used for tasks that require more processing power, such as running an operating system or performing complex calculations. • A microprocessor is typically used in devices that require frequent updates or upgrades, such as desktop computers and laptops. • A microprocessor is used in general-purpose computers and devices.	• A microcontroller is typically used for tasks that require more control over hardware, such as controlling a motor or reading sensors. • A microcontroller is used in devices that are designed to perform a specific task and are not often updated or upgraded, such as appliances and industrial control systems. • A microcontroller is used in specialized devices and systems that require more control over the hardware.
Examples	Examples of microprocessors include the Intel Core series of processors used in desktop computers and laptops and the Qualcomm Snapdragon processors used in smartphones.	Examples of microcontrollers include the Arduino Uno, which is often used in DIY electronics projects, and the PIC microcontrollers used in a variety of applications, including industrial control systems and consumer devices.

Microprocessors and Microcontrollers



Features	Microprocessor	Microcontroller
Definition	A microprocessor is a central processing unit (CPU) that performs the majority of the processing in a computer or other device.	A microcontroller is a small computer that is integrated into a single chip and is designed to perform a specific task or set of tasks.
Clock speed	A microprocessor typically has a higher clock speed and more processing power than a microcontroller.	A microcontroller typically has a lower clock speed and more processing power than a microprocessor.
Memory requirement	A microprocessor typically requires external memory and other components to function.	A microcontroller has memory and other peripherals integrated into the same chip.
Programming language	A microprocessor is usually programmed using a high-level programming language.	A microcontroller is often programmed using a low-level language or assembly code.
Usage	• A microprocessor is generally used for tasks that require more processing power, such as running an operating system or performing complex calculations. • A microprocessor is typically used in devices that require frequent updates or upgrades, such as desktop computers and laptops. • A microprocessor is used in general-purpose computers and devices.	• A microcontroller is typically used for tasks that require more control over hardware, such as controlling a motor or reading sensors. • A microcontroller is used in devices that are designed to perform a specific task and are not often updated or upgraded, such as appliances and industrial control systems. • A microcontroller is used in specialized devices and systems that require more control over the hardware.
Examples	Examples of microprocessors include the Intel Core series of processors used in desktop computers and laptops and the Qualcomm Snapdragon processors used in smartphones.	Examples of microcontrollers include the Arduino Uno, which is often used in DIY electronics projects, and the PIC microcontrollers used in a variety of applications, including industrial control systems and consumer devices.

Microcontroller

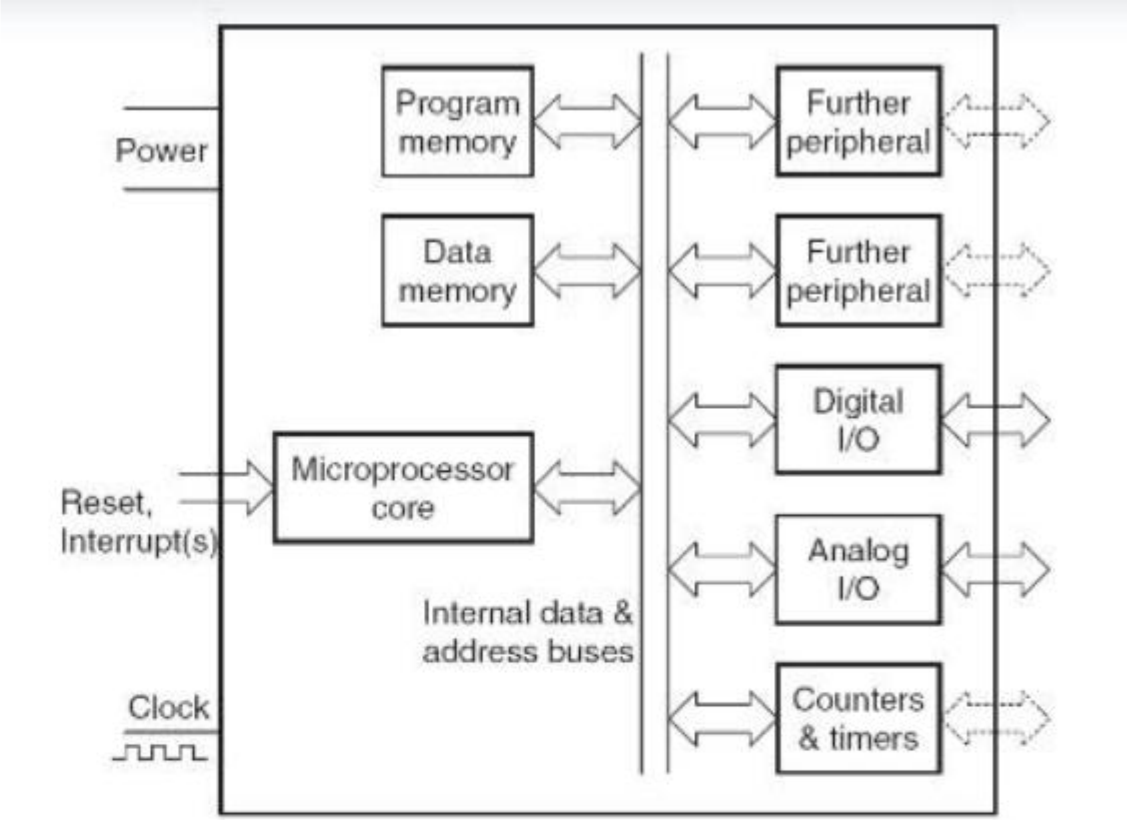
- A microcontroller is a small computer integrated onto a single chip.
- It combines one or more CPUs (processor cores) with memory and programmable input/output peripherals.
- Microcontrollers have contributed exponentially in the development of many computer applications, industrial instrumentation and controls.
- The whole CPU of a computer gets fabricated on a single chip using LSI and VLSI technology with the advancement of semiconductor technology.

Features of Microcontroller

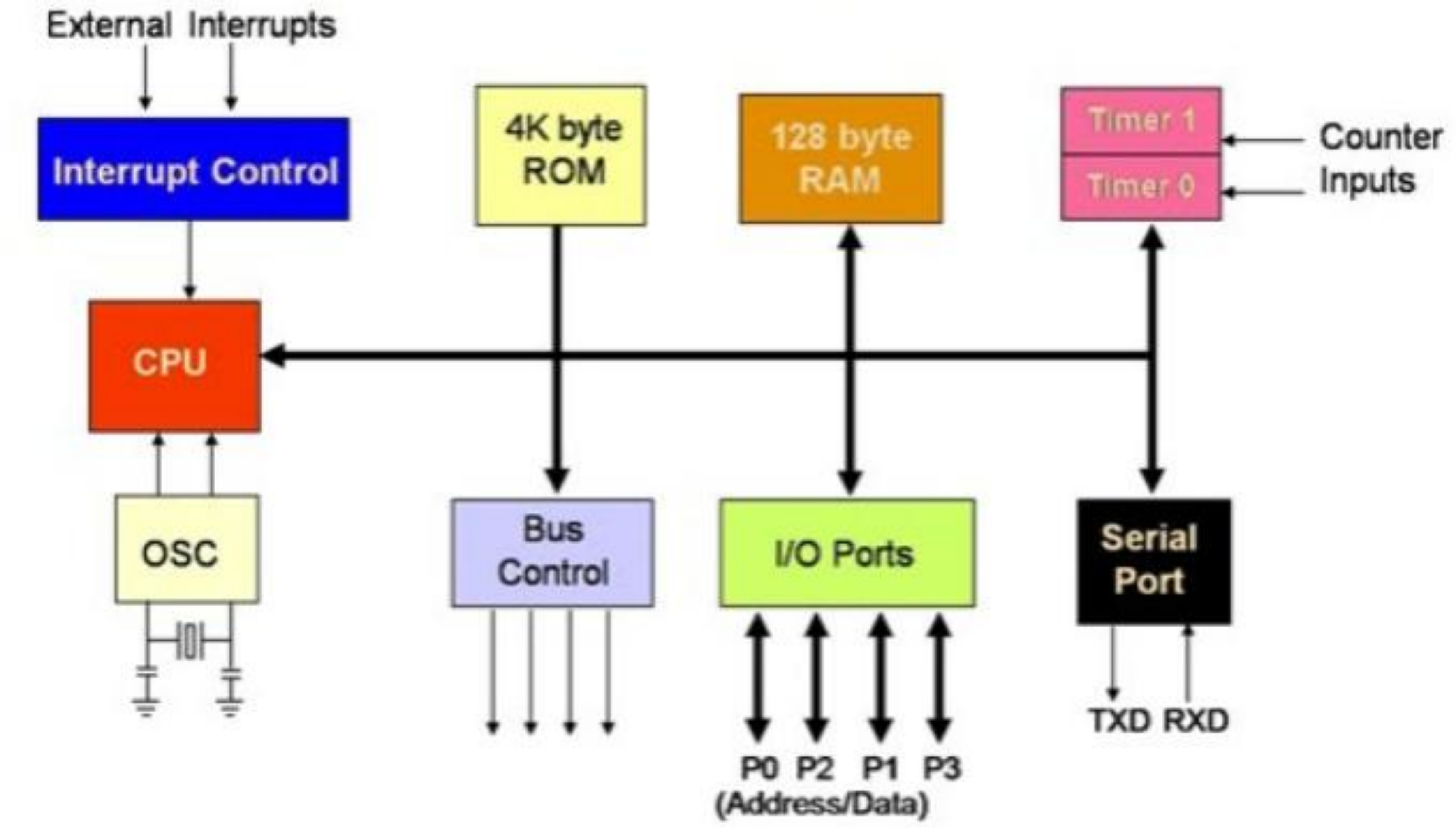
Some features of microcontrollers include:

- **On-chip peripherals:** Many microcontrollers have a variety of on-chip peripherals, such as timers, serial ports, and analog-to-digital converters, which allow them to interface with external devices.
- **Memory:** Microcontrollers have both program memory, which stores the instructions that are executed by the processor, and data memory, which is used to store variables and other data.
- **Input/output (I/O) pins:** Microcontrollers have a set of I/O pins that can be used to interface with external devices, such as sensors or actuators.
- **Low power consumption:** Microcontrollers are designed to be low-power, which makes them suitable for use in battery-powered devices.
- **Cost:** Microcontrollers are typically less expensive than general-purpose processors, as they are designed for specific tasks and do not have as many capabilities.
- **Size:** Microcontrollers are small, which makes them suitable for use in compact devices.
- **Flexibility:** Microcontrollers are highly flexible and can be programmed to perform a wide range of tasks

Microcontrollers



Architecture of Microcontrollers



Architecture of Microcontrollers

- **CPU (Central Processing Unit)**: CPU act as a mind of any processing machine. It synchronizes and manages all processes that are carried out in microcontroller. User has no power to control the functioning of CPU. It interprets the program stored in ROM and carries out from storage and then performs it projected duty. CPU manage the different types of registers available in 8051 microcontroller.

- **Interrupts**: Interrupts is a sub-routine call that given by the microcontroller when some other program with high priority is request for acquiring the system buses the n interrupts occur in current running program.

- Timer 0 overflow interrupt - TF0
- Timer 1 overflow interrupt - TF1
- External hardware interrupt - INT0
- External hardware interrupt - INT1
- Serial communication interrupt - RI/TI

- **Memory**: Microcontroller also required memory for storage of data and operands for the short duration. In microcontroller 8051 there is code or program memory of 4 KB that is it has 4 KB ROM and it also comprise of data memory (RAM) of 128 bytes.

- **Bus** : Bus is a group of wires which uses as a communication canal or acts as means of data transfer. The different bus configuration includes 8, 16 or more cables. Therefore, a bus can bear 8 bits, 16 bits all together.

- 8051 microcontrollers is consisting of 16 bit address bus. It is generally be used for transferring the data from Central Processing Unit to Memory. The data bus is of 8 bits. It is generally be used for transferring the data from one peripherals position to other peripherals.

- **Oscillator**: To perform timer operation inside microcontroller it required externally connected or on-chip oscillator. Microcontroller is used inside an embedded system for managing the function of devices. Therefore, 8051 uses the two 16 bit counters and timers. GENERATES PULSE TIMELY

Memory Organization

- The 8051 has two types of memory and these are Program Memory and Data Memory.
- Program Memory (ROM) is used to permanently save the program being executed.
- Data Memory (RAM) is used for temporarily storing data and intermediate results created and used during the operation of the microcontroller
- Depending on the model(of 8051) in use at most a few Kb of ROM and 128 or 256 bytes of RAM is used.
- All 8051 microcontrollers have a 16-bit addressing bus and are capable of addressing 64 kb memory.

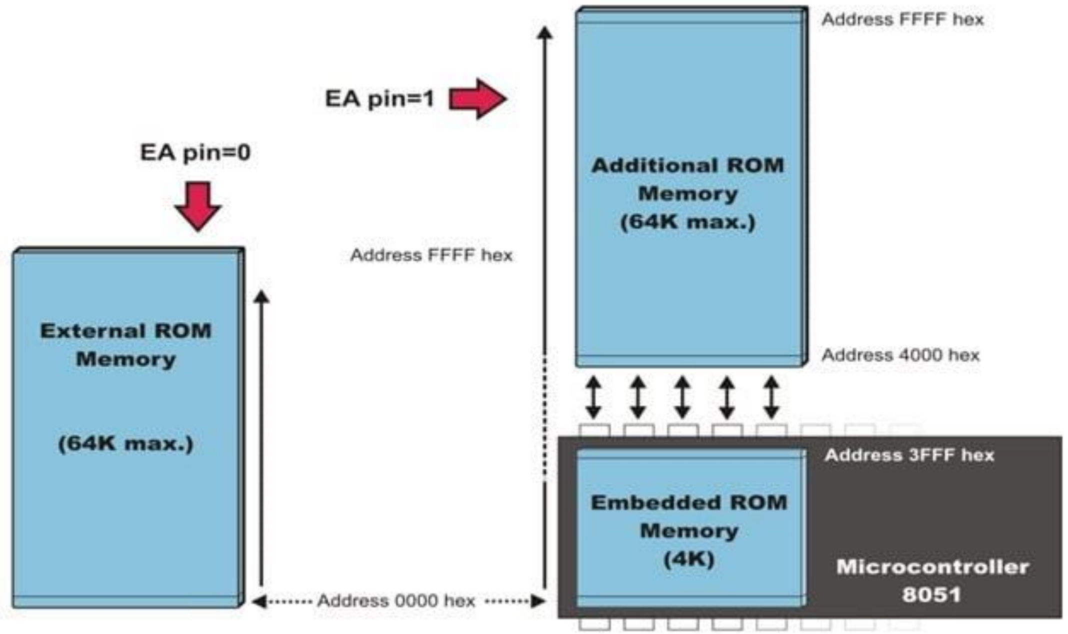
Memory Organization

Program Memory

- The first models of the 8051 microcontroller family did not have internal program memory. It was added as an external separate chip.
- Even though such an amount of memory is sufficient for writing most of the programs, there are situations when it is necessary to use additional memory as well. A typical example is so called lookup tables.
- They are used in cases when equations describing some processes are too complicated or when there is no time for solving them. In such cases all necessary estimates and approximates are executed in advance and the final results are put in the tables

Memory Organization

Program Memory

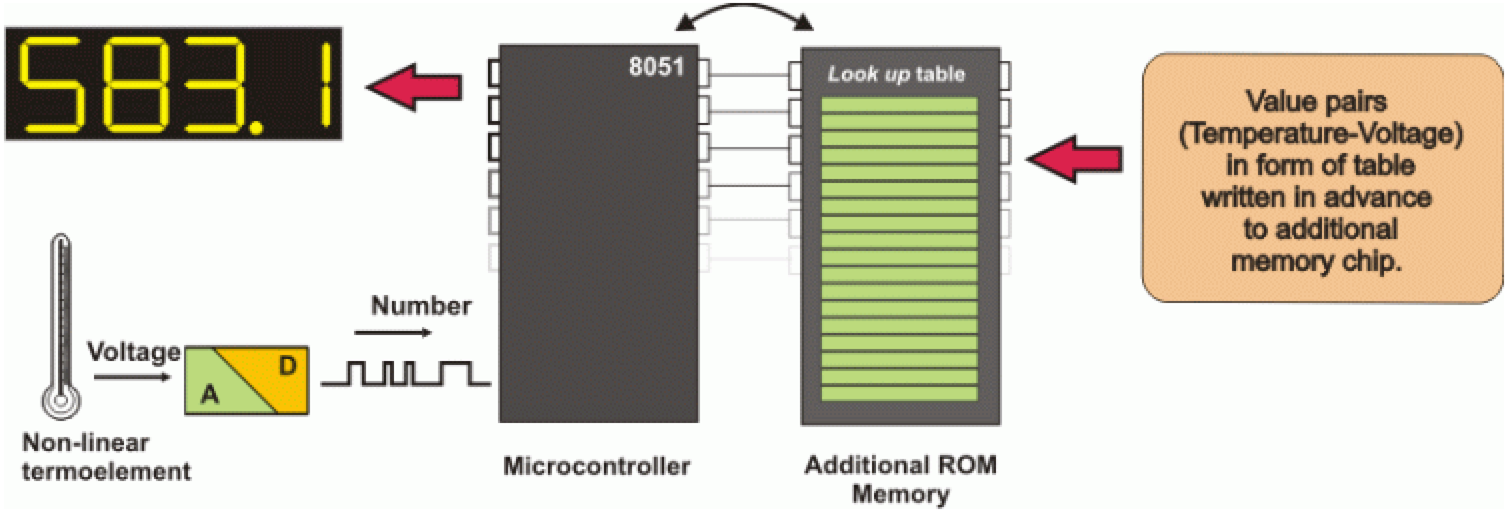


EA=0 In this case, the microcontroller completely ignores internal program memory and executes only the program stored in external memory.

EA=1 In this case, the microcontroller executes first the program from built-in ROM, then the program stored in external memory. In both cases, P0 and P2 are not available for use since being used for data and address transmission. Besides, the ALE and PSEN pins are also used.

Memory Organization

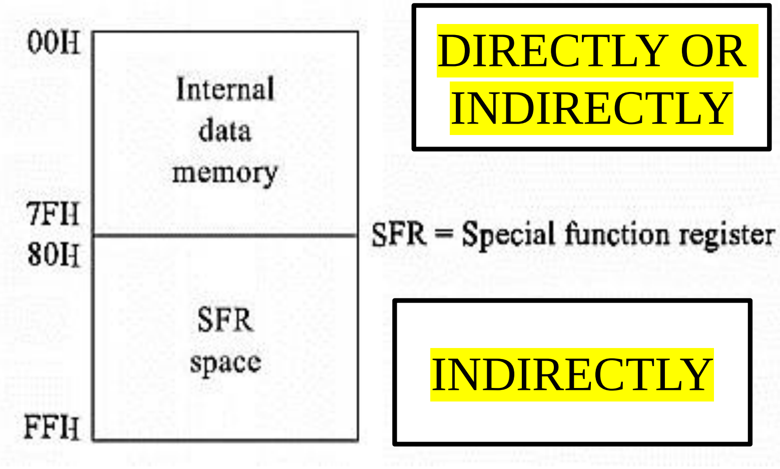
Program Memory


583.1
8051
Look up table
Value pairs (Temperature-Voltage) in form of table written in advance to additional memory chip.
Number
Voltage
D
A
Non-linear termoelement
Microcontroller
Additional ROM Memory

Memory Organization

Data Memory

- Data Memory is used for temporarily storing data and intermediate results created and used during the operation of the microcontroller. Besides, RAM memory built in the 8051 family includes many registers such as hardware counters and timers, input/output ports, serial data buffers etc.
- Up to 256 bytes of internal data memory are available depending on the 8051 derivative.
- These are a set of eight registers and a scratch pad memory. These eight registers are R0 toR7. The address range 00H to 07H is used to access the registers, and the rest are scratch pad memory.



DIRECTLY OR INDIRECTLY

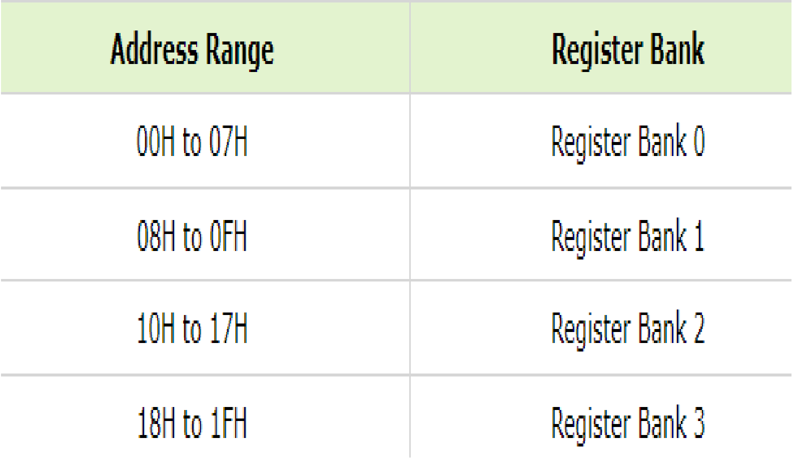
SFR = Special function register

INDIRECTLY

Memory Organization

Data Memory

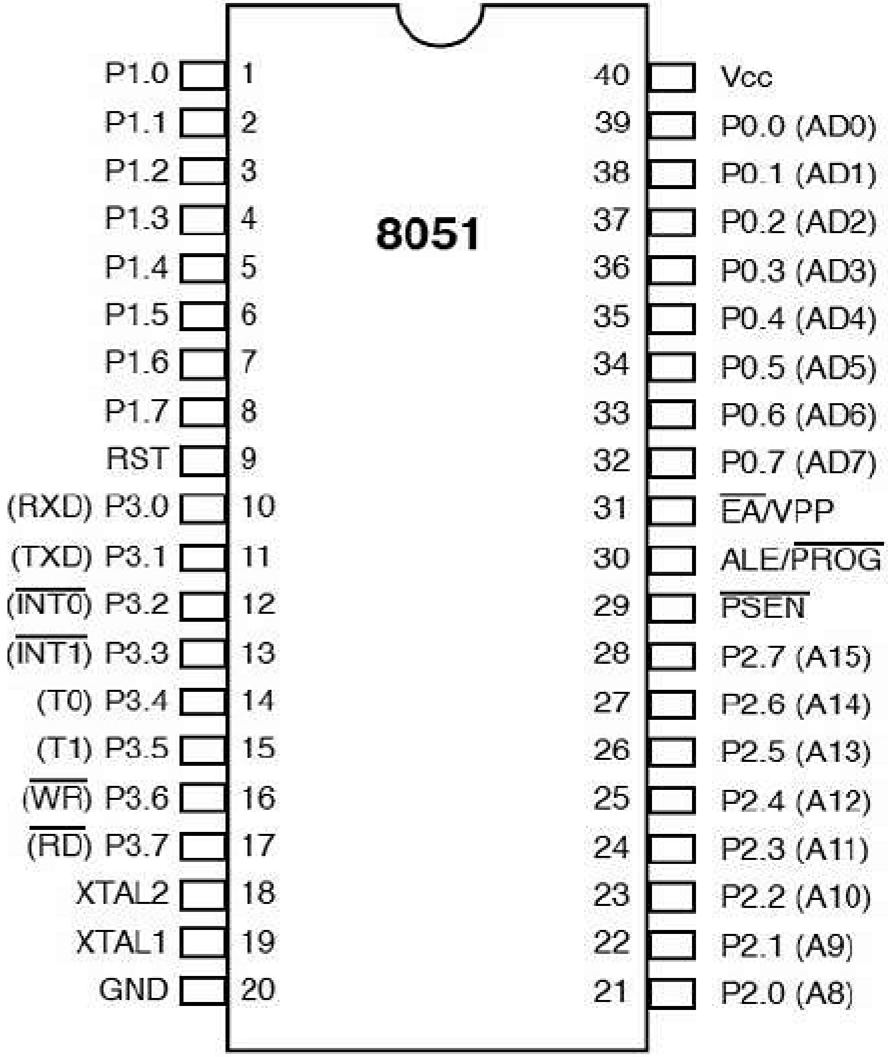
- 8051 Provides four register bank, but only one register bank can be used at any point in time. To select the register bank, two bits of PSW (Program Status Word) are used.
- The interrupt program can use one bank, and the interrupt Service Subroutine (ISS) can access another bank for better performance. As there are four banks, so for nested interrupts these can be used.
- 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.
- The 8051 provides 128 bytes of memory for Special Function Registers (SFRs). SFRs are bit, byte, or word-sized registers that are used to control timers, counters, serial I/O, port I/O, and peripherals



Address Range	Register Bank
00H to 07H	Register Bank 0
08H to 0FH	Register Bank 1
10H to 17H	Register Bank 2
18H to 1FH	Register Bank 3

Pin diagram of 8051 Microcontroller

- The 8051 microcontroller is a popular 8-bit microcontroller widely used in embedded systems. It is a single-chip microcontroller with a Harvard architecture that includes a CPU, RAM, ROM, and several peripherals.
- The 8051 microcontroller has a 40-pin dual in-line package (DIP) that provides various inputs and outputs for communication with external devices.
- These 40 pins serve different functions like read, write, I/O operations, interrupts etc.
- 8051 has four I/O ports wherein each port has 8 pins which can be configured as input or output depending upon the logic state of the pins

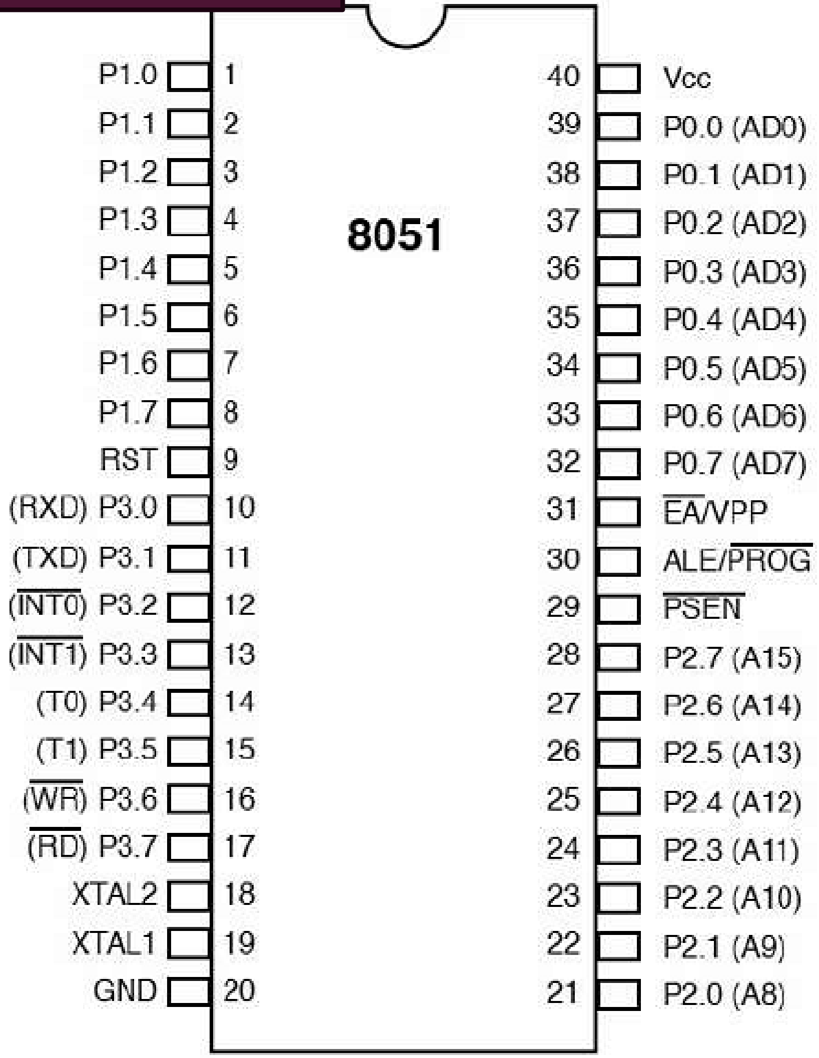


8051 Pin configuration

TIP TO LEARN:
0-> OUTPUT
1-> INPUT

Description of Pins

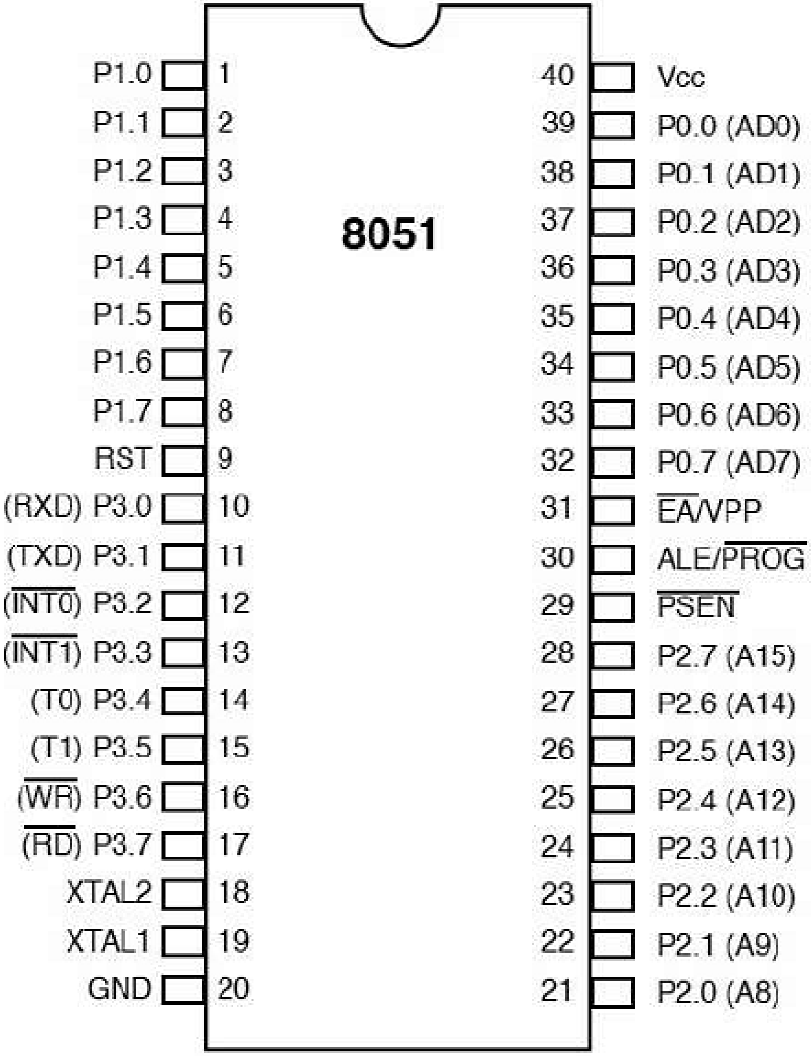
- **Pin 1 to Pin 8 (Port 1)** – Pin 1 to Pin 8 are assigned to Port 1 for simple I/O operations. They can be configured as input or output pins depending on the logic control i.e. if logic zero (0) is applied to the I/O port it will act as an output pin and if logic one (1) is applied the pin will act as an input pin. These pins are also referred to as P1.0 to P1.7 (where P1 indicates that it is a pin in port 1 and the number after '.' tells the pin number i.e. 0 indicates first pin of the port. So, P1.0 means first pin of port 1, P1.1 means second pin of the port 1 and so on). These pins are bidirectional pins.



Description of Pins

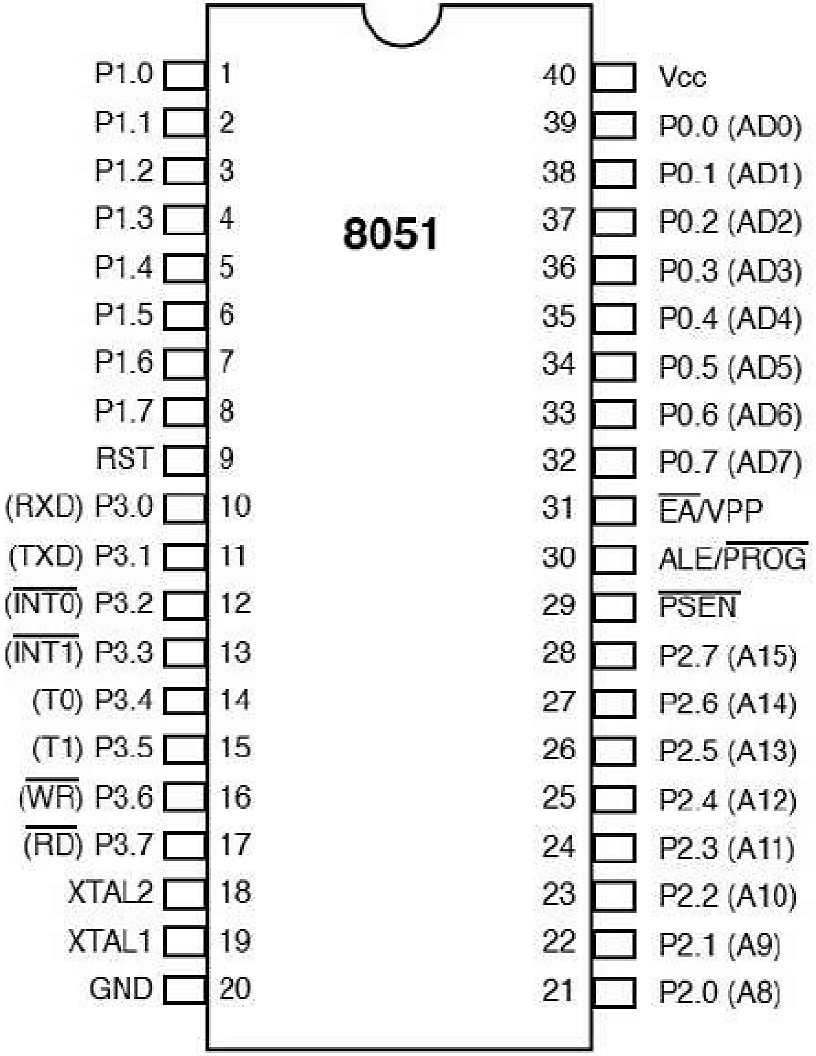
- **Pin 9 (RST)** – Reset pin.

It is an active-high, input pin. Therefore if the RST pin is high for a minimum of 2 machine cycles, the microcontroller will reset i.e. it will close and terminate all activities. It is often referred as "power-on-reset" pin because it is used to reset the microcontroller to it's initial values when power is on (high).



Description of Pins

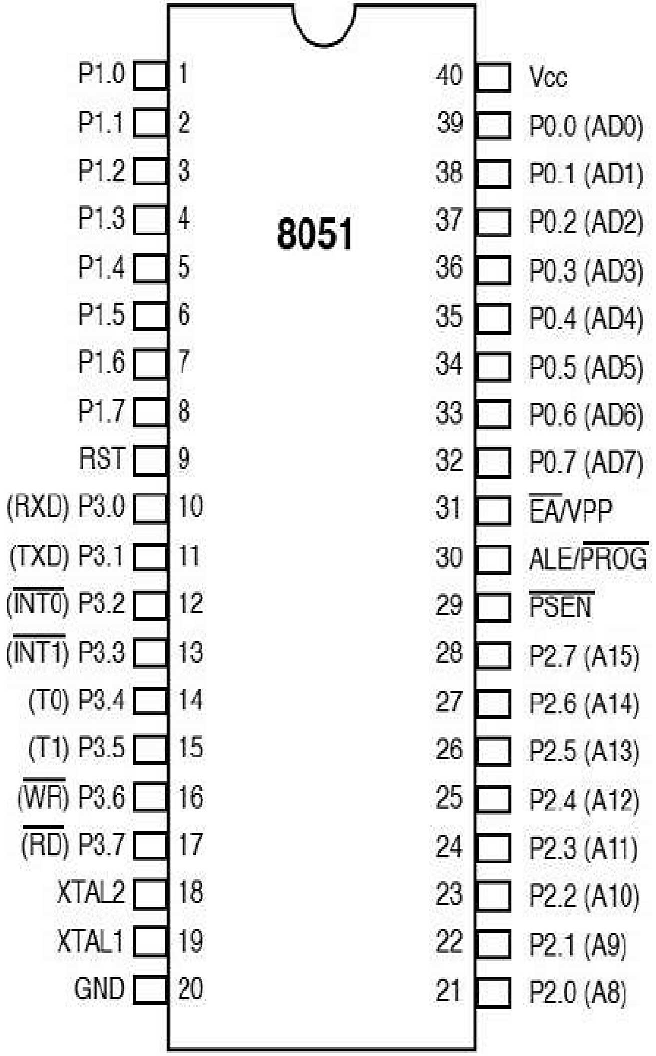
- **Pin 10 to Pin 17 (Port 3)** – Pin 10 to pin 17 are port 3 pins which are also referred to as P3.0 to P3.7. These pins are similar to port 1 and can be used as universal input or output pins. These pins are bidirectional pins.



Description of Pins

These pins also have some additional functions which are as follows:

- **P3.0 (RXD) :** 10th pin is RXD (serial data receive pin) which is for serial input. Through this input signal microcontroller receives data for serial communication.
- **P3.1 (TXD) :** 11th pin is TXD (serial data transmit pin) which is serial output pin. Through this output signal microcontroller transmits data for serial communication.
- **P3.2 and P3.3 (INT0′, INT1′) :** 12th and 13th pins are for External Hardware Interrupt 0 and Interrupt 1 respectively. When this interrupt is activated(i.e. when it is low), 8051 gets interrupted in whatever it is doing and jumps to the vector value of the interrupt (0003H for INT0 and 0013H for INT1) and starts performing Interrupt Service Routine (ISR) from that vector location.
- **P3.4 and P3.5 (T0 and T1) :** 14th and 15th pin are for Timer 0 and Timer 1 external input. They can be connected with 16 bit timer/counter.
- **P3.6 (WR') :** 16th pin is for external memory write i.e. writing data to the external memory.
- **P3.7 (RD') :** 17th pin is for external memory read i.e. reading data from external memory.



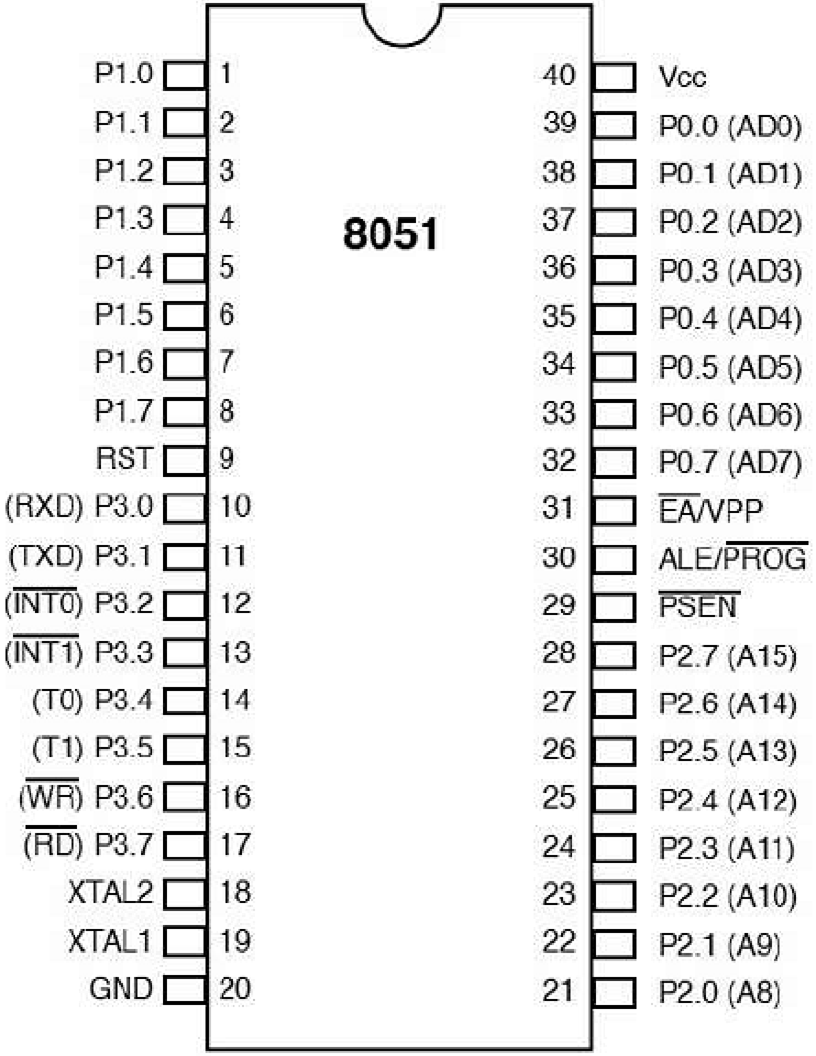
Description of Pins

Pin 18 and Pin 19 (XTAL2 And XTAL1) – These pins are connected to an external oscillator which is generally a quartz crystal oscillator. They are used to provide an external clock frequency of 4MHz to 30MHz. TO CONNECT PULSE OSCILATOR FROM OUTSIDE

Pin 20 (GND) – This pin is connected to the ground. It has to be provided with 0V power supply. Hence it is connected to the negative terminal of the power supply.

Pin 21 to Pin 28 (Port 2) – Pin 21 to pin 28 are port 2 pins also referred to as P2.0 to P2.7. When additional external memory is interfaced with the 8051 microcontroller, pins of port 2 act as higher-order address bytes. These pins are bidirectional.

INPUT /OUTPUT OR COMMUNICATE WITH EXTERNAL MEMORY

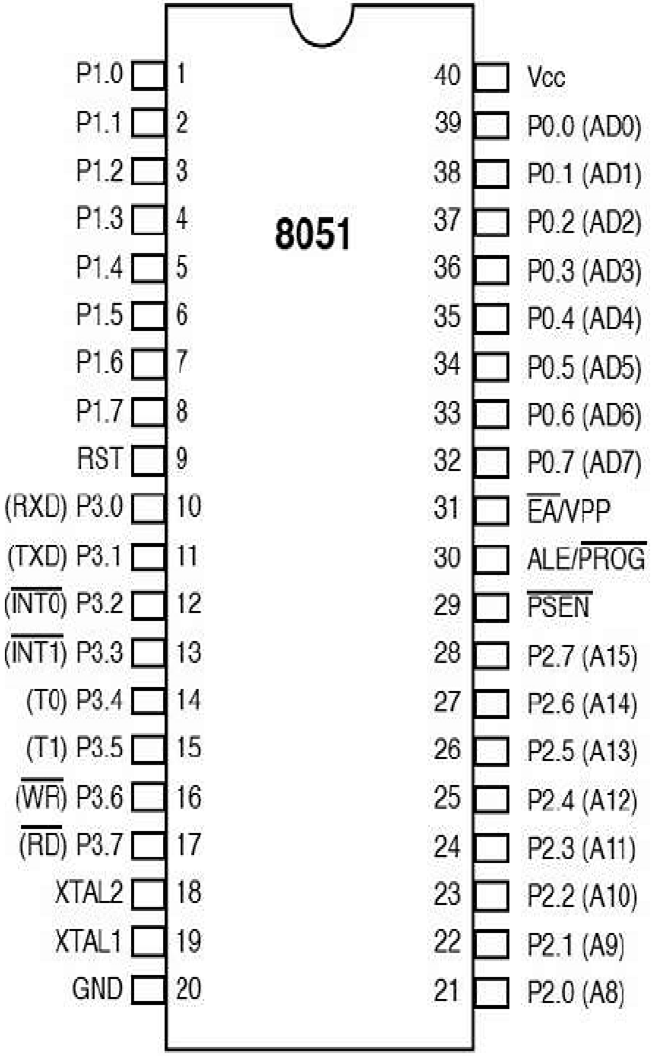


Description of Pins

Pin 29 (PSEN) – PSEN stands for Program Store Enable. It is output, active-low(0) pin. This is used to read external memory. In 8031 based system where external ROM holds the program code, this pin is connected to the OE pin of the ROM.

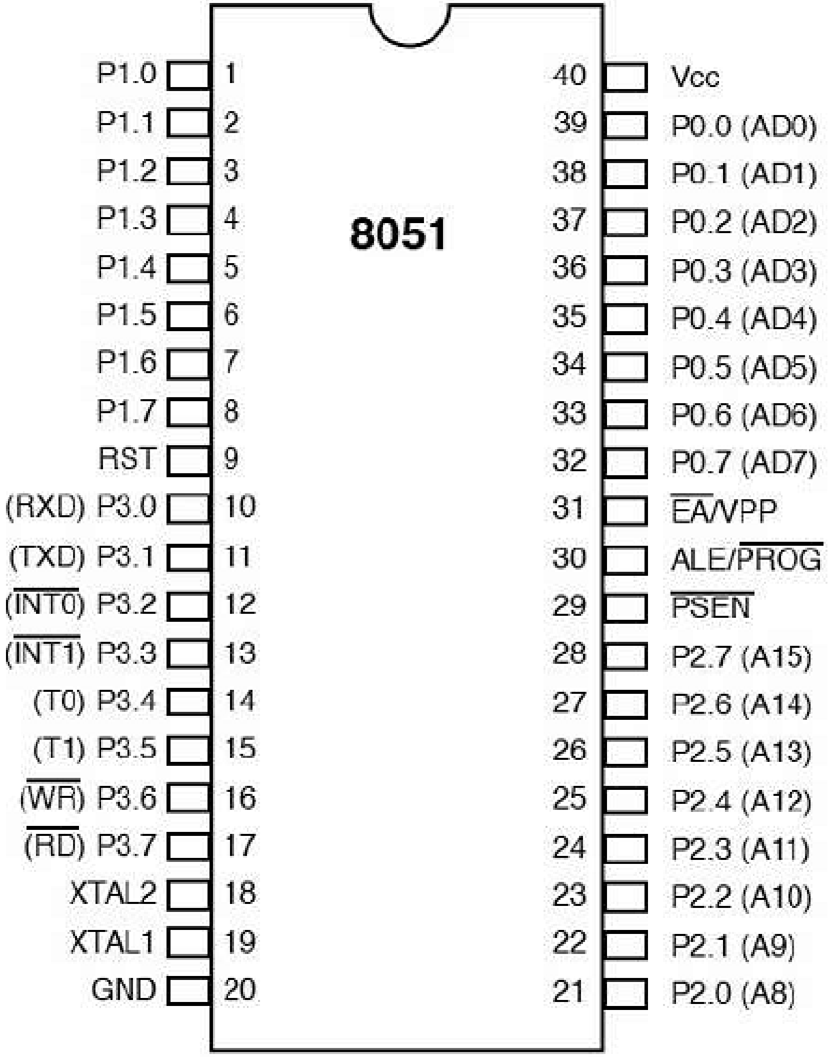
Pin 30 (ALE/ PROG) – ALE stands for Address Latch Enable. It is input, active-high(1) pin.USED TO DESCIDE TO TALK WITH WHICH CHIP WHEN MANY MEMORY CHIPS ARE ATTACHED. This pin is used to distinguish between memory chips when multiple memory chips are used. It is also used to de-multiplex the multiplexed address and data signals available at port 0. During flash programming i.e. Programming of EPROM, this pin acts as program pulse input (PROG).

Pin 31 (EA/ VPP) – EA stands for External Access input. It is used to enable/disable external memory interfacing. In 8051, EA is connected to Vcc as it comes with on-chip ROM to store programs. For other family members such as 8031 and 8032 in which there is no on-chip ROM, the EA pin is connected to the GND.



Description of Pins

- **Pin 32 to Pin 39 (Port 0)** – Pin 32 to pin 39 are port 0 pins also referred to as P0.0 to P0.7. They are bidirectional input/output pins. They don’t have any internal pull-ups. Hence, 10 K? pull-up registers are used as external pull-ups. Port 0 is also designated as AD0-AD7 because 8051 multiplexes address and data through port 0 to save pins.
- **Pin 40 (VCC)** – This pin provides power supply voltage i.e. +5 Volts to the circuit.



Ports in 8051

- **Pin configuration**, i.e. the pin can be configured as 1 for input and 0 for output as per the logic state.
  - **Input/Output (I/O) pin** − All the circuits within the microcontroller must be connected to one of its pins except P0 port because it does not have pull-up resistors built-in.
  - **Input pin** − Logic 1 is applied to a bit of the P register. The output FE transistor is turned off and the other pin remains connected to the power supply voltage over a pull-up resistor of high resistance.
- **Port 0** − The P0 (zero) port is characterized by two functions
  - When the external memory is used then the lower address byte (addresses A0A7) is applied on it, else all bits of this port are configured as input/output.
  - When P0 port is configured as an output then other ports consisting of pins with built-in pull-up resistor connected by its end to 5V power supply, the pins of this port have this resistor left out.

Ports in 8051

- **Port 1**

P1 is a true I/O port as it doesn’t have any alternative functions as in P0, but this port can be configured as general I/O only. It has a built-in pull-up resistor and is completely compatible with TTL circuits.

- **Port 2**

P2 is similar to P0 when the external memory is used. Pins of this port occupy addresses intended for the external memory chip. This port can be used for higher address byte with addresses A8-A15. When no memory is added then this port can be used as a general input/output port similar to Port 1.

- **Port 3**

In this port, functions are similar to other ports except that the logic 1 must be applied to appropriate bit of the P3 register.

Storage Registers in 8051

- Accumulator
- R register
- B register
- Data Pointer (DPTR)
- Program Counter (PC)
- Stack Pointer (SP)

Storage Registers in 8051

- Accumulator

The accumulator, register A, is used for all arithmetic and logic operations. If the accumulator is not present, then every result of each calculation (addition, multiplication, shift, etc.) is to be stored into the main memory. Access to main memory is slower than access to a register like the accumulator because the technology used for the large main memory is slower (but cheaper) than that used for a register

- "R" Registers

The "R" registers are a set of eight registers, namely, R0, R1 to R7. These registers function as auxiliary or temporary storage registers in many operations

Storage Registers in 8051

- "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 used only by two 8051 instructions: MUL A,B and DIV A,B. 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. Apart from using MUL and DIV instructions, the "B" register is often used as yet another temporary storage register, much like a ninth R register.

- Data Pointer

The Data Pointer (DPTR) is the 8051's only user-accessible 16-bit (2-byte) register. The Accumulator, R0–R7 registers and B register are 1-byte value registers. DPTR is meant for pointing to data. It is used by the 8051 to access external memory using the address indicated by DPTR. DPTR is the only 16-bit register available and is often used to store 2-byte values.

Storage Registers in 8051

- Program Counter (PC)

The Program Counter (PC) is a 2-byte address which tells the 8051 where the next instruction to execute can be found in the memory.

- Stack Pointer (SP)

The Stack Pointer tells the location from where the next value is to be removed from the stack. When a value is pushed onto the stack, the value of SP is incremented and then the value is stored at the resulting memory location. When a value is popped off the stack, the value is returned from the memory location indicated by SP, and then the value of SP is decremented.

Addressing Modes in 8051

Method of specifying the data to be operated by the instruction

CPU can access the data in various ways.

-- data could be in reg./Memory/immediate value

Is a way the MC to access data / operand from internal memory /external memory (or)Register specific Ports

The use of efficient modes of a program (addressing mode) will increase the processing speed of CPU as well as processing time can be shortened

Addressing Modes in 8051

- Register Addressing Mode
- Direct Addressing Mode
- Register indirect Addressing Mode
- Immediate Addressing Mode
- Indexed Addressing mode
- Implied Addressing mode

Register Addressing mode

- **Instruction will specify the name of the Reg. in which data is available**
- This addressing instruction involves information transfer between registers (at least one of the R0-R7 register involved.
- Source & Destination reg.'s match in Size:

Eg: **MOV A, R0**

MOV R2, A

ADD A, R5

Eg: ~~MOV R4, R5~~ (Invalid Instruction)

Direct Addressing mode

- The address of the data is directly specified in the instruction
  - MOV A, P3 ; Transfer the contents of Port 3 to the accumulator
  - MOV A, 020H ; Transfer the contents of RAM location 20H to the accumulator
  - MOV P1, AA H ; Transfer the contents of A to Port 1
  - MOV 20H, 40H ; Transfer the contents of the address 40H to the address 20H
  - ADD A, 55H ; Add the contents of A with the contents of the address 55H

MOV R0, 40H
MOV R4, 7FH
MOV 40H, A

Register Indirect Addressing mode

- In this mode, the source or destination address is given in the register. By using register indirect addressing mode, the internal or external addresses can be accessed. The R0 and R1 are used for 8-bit addresses, and DPTR is used for 16-bit addresses, no other registers can be used for addressing purposes.
- MOV0E5H, @R0;
- MOV@R1, 80H
- In the instructions, the @ symbol is used for register indirect addressing. In the first instruction, it is showing that the R0 register is used. If the content of R0 is 40H, then that instruction will take the data which is located at location 40H of the internal RAM.

Immediate Addressing mode

- In this Immediate Addressing Mode, the data is provided in the instruction itself.

  The data is provided immediately after the opcode.
- **MOVA, #0AFH;**
- **MOVR3, #45H;**
- **MOVDPTR, #FE00H;**

Indexed Addressing mode

- In the indexed addressing mode, the source memory can only be accessed from program memory only. The destination operand is always the register A..
- **MOVCA, @A+PC;**
- **MOVCA, @A+DPTR**

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

Implied Addressing mode

- In the implied addressing mode, there will be a single operand. These types of instruction can work on specific registers only. These types of instructions are also known as register specific instruction.
- **RLA;- rotate the accumulator left**
- **SWAPA;-interchanges the low-and high-order nibbles (four-bit fields) of the Accumulator (bits 3-0 and bits 7-4)**

Instruction set of 8051

The process of writing program for the microcontroller mainly consists of giving instructions (commands) in the specific order in which they should be executed in order to carry out a specific task.

- The first part of each instruction, called MNEMONIC refers to the operation an instruction performs (copy, addition, logic operation etc.)
- Depending on operation they perform, all instructions are divided in several groups:
  - Arithmetic Instructions
  - Branch Instructions
  - Data Transfer Instructions
  - Logic Instructions
  - Bit-oriented Instructions

- **Arithmetic Instructions**: Arithmetic instructions perform several basic operations such as addition, subtraction, division, multiplication etc. After execution, the result is stored in the first operand. For example:

  ADD A,R1 - The result of addition (A+R1) will be stored in the accumulator.

- **Branch Instructions:** There are two kinds of branch instructions:
  - Unconditional jump instructions: upon their execution a jump to a new location from where the program continues execution is executed.
  - Conditional jump instructions: a jump to a new program location is executed only if a specified condition is met. Otherwise, the program normally proceeds with the next instruction.

- **Data Transfer Instructions**

Data transfer instructions move the content of one register to another. The register the content of which is moved remains unchanged. If they have the suffix “X” (MOVX), the data is exchanged with external memory.

- **Logic Instructions**

Logic instructions perform logic operations upon corresponding bits of two registers. After execution, the result is stored in the first operand.

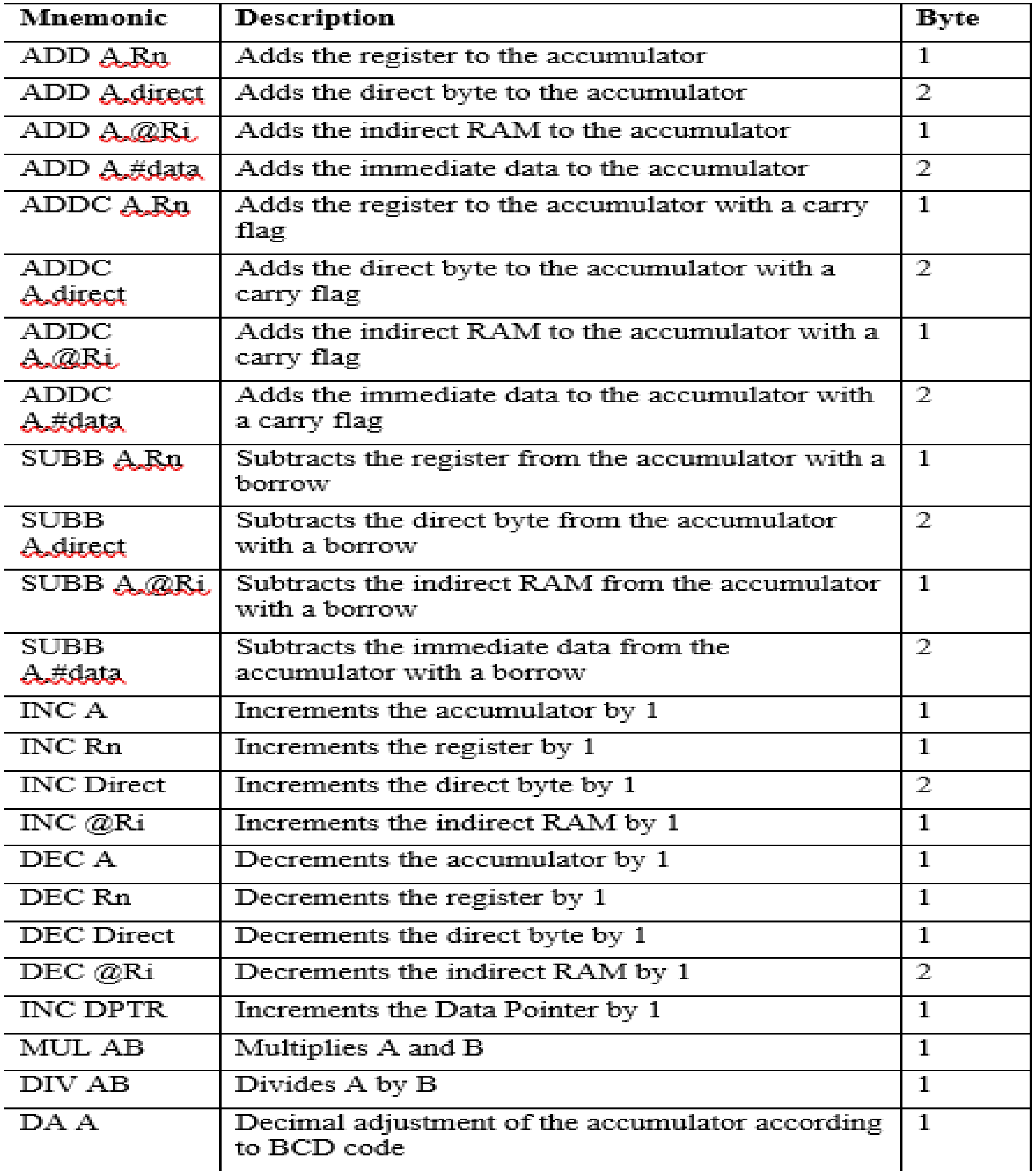
- **Bit-oriented Instructions**

Similar to logic instructions, bit-oriented instructions perform logic operations. The difference is that these are performed upon single bits.

Arithmetic Instructions: Arithmetic instructions perform several basic operations such as addition, subtraction, division, multiplication etc. After execution, the result is stored in the first operand.

For eg: ADD A,R1

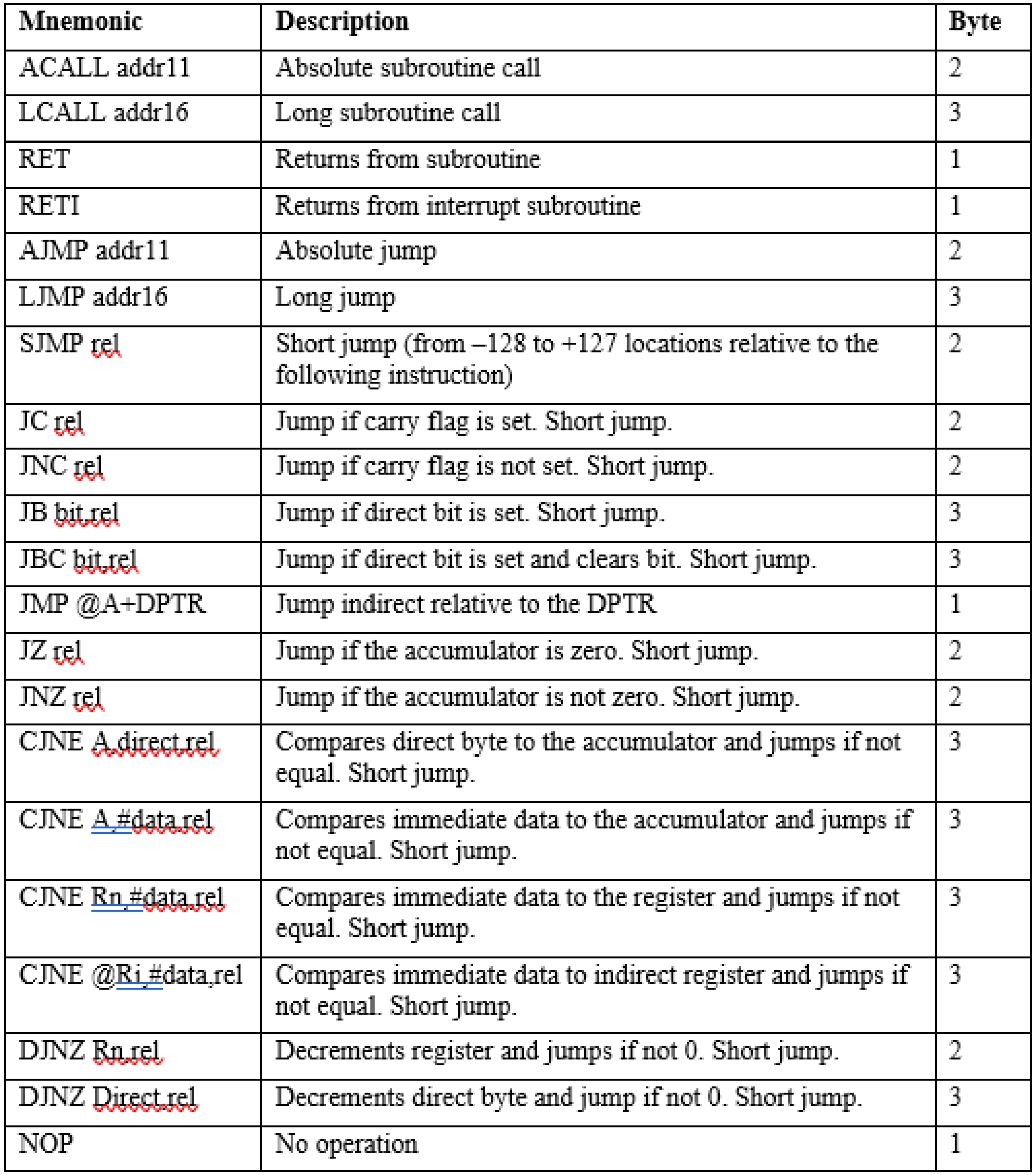
The result of addition (A+R1) will be stored in the accumulator.



Mnemonic	Description	Byte
ADD A,Rn	Adds the register to the accumulator	1
ADD A,direct	Adds the direct byte to the accumulator	2
ADD A,@Ri	Adds the indirect RAM to the accumulator	1
ADD A,#data	Adds the immediate data to the accumulator	2
ADDC A,Rn	Adds the register to the accumulator with a carry flag	1
ADDC A,direct	Adds the direct byte to the accumulator with a carry flag	2
ADDC A,@Ri	Adds the indirect RAM to the accumulator with a carry flag	1
ADDC A,#data	Adds the immediate data to the accumulator with a carry flag	2
SUBB A,Rn	Subtracts the register from the accumulator with a borrow	1
SUBB A,direct	Subtracts the direct byte from the accumulator with a borrow	2
SUBB A,@Ri	Subtracts the indirect RAM from the accumulator with a borrow	1
SUBB A,#data	Subtracts the immediate data from the accumulator with a borrow	2
INC A	Increments the accumulator by 1	1
INC Rn	Increments the register by 1	1
INC Direct	Increments the direct byte by 1	2
INC @Ri	Increments the indirect RAM by 1	1
DEC A	Decrements the accumulator by 1	1
DEC Rn	Decrements the register by 1	1
DEC Direct	Decrements the direct byte by 1	1
DEC @Ri	Decrements the indirect RAM by 1	2
INC DPTR	Increments the Data Pointer by 1	1
MUL AB	Multiplies A and B	1
DIV AB	Divides A by B	1
DA A	Decimal adjustment of the accumulator according to BCD code	1

- **Branch Instructions:** There are two kinds of branch instructions:

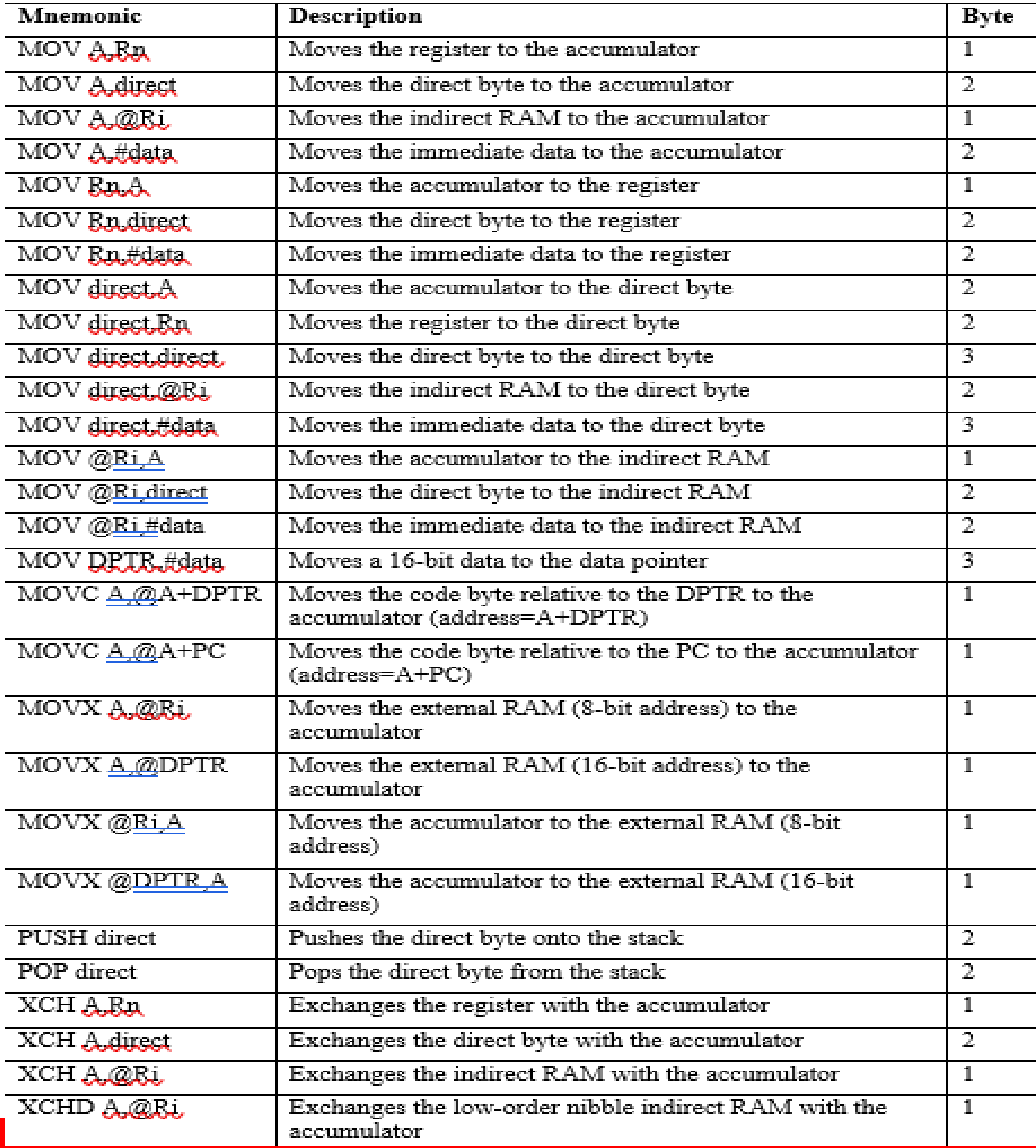
- Unconditional jump instructions: upon their execution a jump to a new location from where the program continues execution is executed.
- Conditional jump instructions: a jump to a new program location is executed only if a specified condition is met. Otherwise, the program normally proceeds with the next instruction.



Mnemonic	Description	Byte
ACALL addr11	Absolute subroutine call	2
LCALL addr16	Long subroutine call	3
RET	Returns from subroutine	1
RETI	Returns from interrupt subroutine	1
AJMP addr11	Absolute jump	2
LJMP addr16	Long jump	3
SJMP rel	Short jump (from –128 to +127 locations relative to the following instruction)	2
JC rel	Jump if carry flag is set. Short jump.	2
JNC rel	Jump if carry flag is not set. Short jump.	2
JB bit,rel	Jump if direct bit is set. Short jump.	3
JBC bit,rel	Jump if direct bit is set and clears bit. Short jump.	3
JMP @A+DPTR	Jump indirect relative to the DPTR	1
JZ rel	Jump if the accumulator is zero. Short jump.	2
JNZ rel	Jump if the accumulator is not zero. Short jump.	2
CJNE A,direct,rel	Compares direct byte to the accumulator and jumps if not equal. Short jump.	3
CJNE A,#data,rel	Compares immediate data to the accumulator and jumps if not equal. Short jump.	3
CJNE Rn,#data,rel	Compares immediate data to the register and jumps if not equal. Short jump.	3
CJNE @Ri,#data,rel	Compares immediate data to indirect register and jumps if not equal. Short jump.	3
DJNZ Rn,rel	Decrements register and jumps if not 0. Short jump.	2
DJNZ Direct,rel	Decrements direct byte and jump if not 0. Short jump.	3
NOP	No operation	1

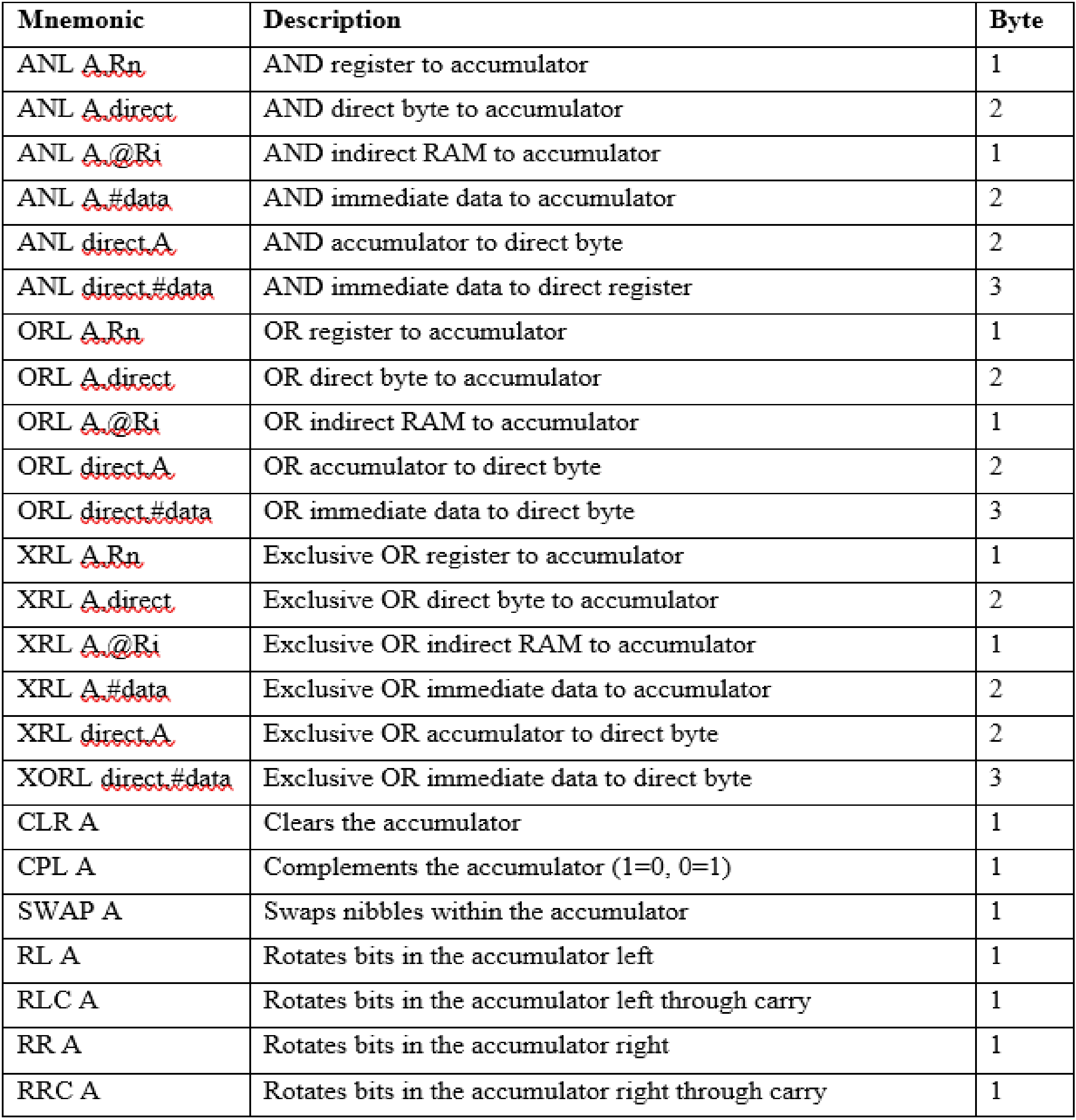
Data Transfer Instructions:

They move the content of one register to another. The register the content of which is moved remains unchanged. If they have the suffix “X” (MOVX), the data is exchanged with external memory.



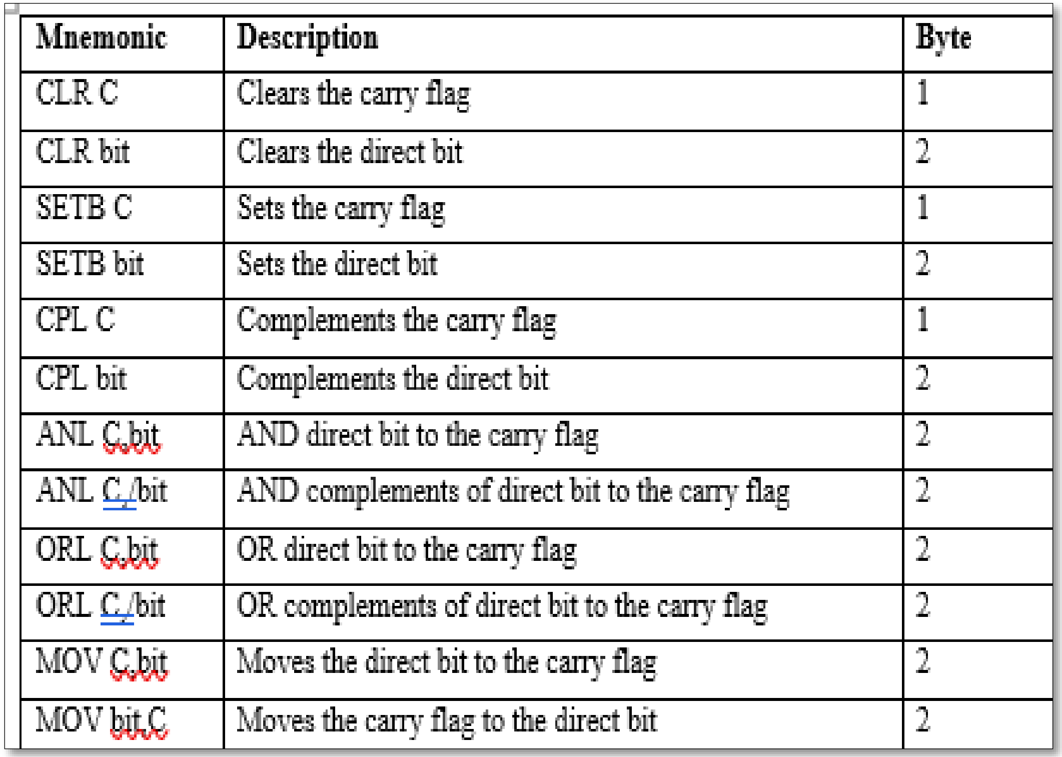
Mnemonic	Description	Byte
MOV A,Rn	Moves the register to the accumulator	1
MOV A,direct	Moves the direct byte to the accumulator	2
MOV A,@Ri	Moves the indirect RAM to the accumulator	1
MOV A,#data	Moves the immediate data to the accumulator	2
MOV Rn,A	Moves the accumulator to the register	1
MOV Rn,direct	Moves the direct byte to the register	2
MOV Rn,#data	Moves the immediate data to the register	2
MOV direct,A	Moves the accumulator to the direct byte	2
MOV direct,Rn	Moves the register to the direct byte	2
MOV direct,direct	Moves the direct byte to the direct byte	3
MOV direct,@Ri	Moves the indirect RAM to the direct byte	2
MOV direct,#data	Moves the immediate data to the direct byte	3
MOV @Ri,A	Moves the accumulator to the indirect RAM	1
MOV @Ri,direct	Moves the direct byte to the indirect RAM	2
MOV @Ri,#data	Moves the immediate data to the indirect RAM	2
MOV DPTR,#data	Moves a 16-bit data to the data pointer	3
MOVC A,@A+DPTR	Moves the code byte relative to the DPTR to the accumulator (address=A+DPTR)	1
MOVC A,@A+PC	Moves the code byte relative to the PC to the accumulator (address=A+PC)	1
MOVX A,@Ri	Moves the external RAM (8-bit address) to the accumulator	1
MOVX A,@DPTR	Moves the external RAM (16-bit address) to the accumulator	1
MOVX @Ri,A	Moves the accumulator to the external RAM (8-bit address)	1
MOVX @DPTR,A	Moves the accumulator to the external RAM (16-bit address)	1
PUSH direct	Pushes the direct byte onto the stack	2
POP direct	Pops the direct byte from the stack	2
XCH A,Rn	Exchanges the register with the accumulator	1
XCH A,direct	Exchanges the direct byte with the accumulator	2
XCH A,@Ri	Exchanges the indirect RAM with the accumulator	1
XCHD A,@Ri	Exchanges the low-order nibble indirect RAM with the accumulator	1

Logic Instructions : They perform logic operations upon corresponding bits of two registers. After execution, the result is stored in the first operand.

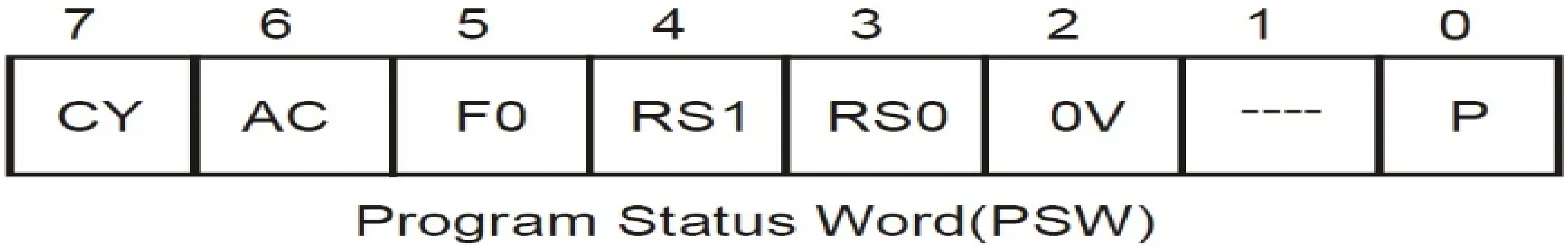


Mnemonic	Description	Byte
ANL A,Rn	AND register to accumulator	1
ANL A,direct	AND direct byte to accumulator	2
ANL A,@Ri	AND indirect RAM to accumulator	1
ANL A,#data	AND immediate data to accumulator	2
ANL direct,A	AND accumulator to direct byte	2
ANL direct,#data	AND immediate data to direct register	3
ORL A,Rn	OR register to accumulator	1
ORL A,direct	OR direct byte to accumulator	2
ORL A,@Ri	OR indirect RAM to accumulator	1
ORL direct,A	OR accumulator to direct byte	2
ORL direct,#data	OR immediate data to direct byte	3
XRL A,Rn	Exclusive OR register to accumulator	1
XRL A,direct	Exclusive OR direct byte to accumulator	2
XRL A,@Ri	Exclusive OR indirect RAM to accumulator	1
XRL A,#data	Exclusive OR immediate data to accumulator	2
XRL direct,A	Exclusive OR accumulator to direct byte	2
XORL direct,#data	Exclusive OR immediate data to direct byte	3
CLR A	Clears the accumulator	1
CPL A	Complements the accumulator (1=0, 0=1)	1
SWAP A	Swaps nibbles within the accumulator	1
RL A	Rotates bits in the accumulator left	1
RLC A	Rotates bits in the accumulator left through carry	1
RR A	Rotates bits in the accumulator right	1
RRC A	Rotates bits in the accumulator right through carry	1

- **Bit-oriented Instructions:** Similar to logic instructions, bit-oriented instructions perform logic operations. The difference is that these are performed upon single bits.



Mnemonic	Description	Byte
CLR C	Clears the carry flag	1
CLR bit	Clears the direct bit	2
SETB C	Sets the carry flag	1
SETB bit	Sets the direct bit	2
CPL C	Complements the carry flag	1
CPL bit	Complements the direct bit	2
ANL C,bit	AND direct bit to the carry flag	2
ANL C,/bit	AND complements of direct bit to the carry flag	2
ORL C,bit	OR direct bit to the carry flag	2
ORL C,/bit	OR complements of direct bit to the carry flag	2
MOV C,bit	Moves the direct bit to the carry flag	2
MOV bit,C	Moves the carry flag to the direct bit	2



7	6	5	4	3	2	1	0
CY	AC	F0	RS1	RS0	0V	----	P

Program Status Word(PSW)

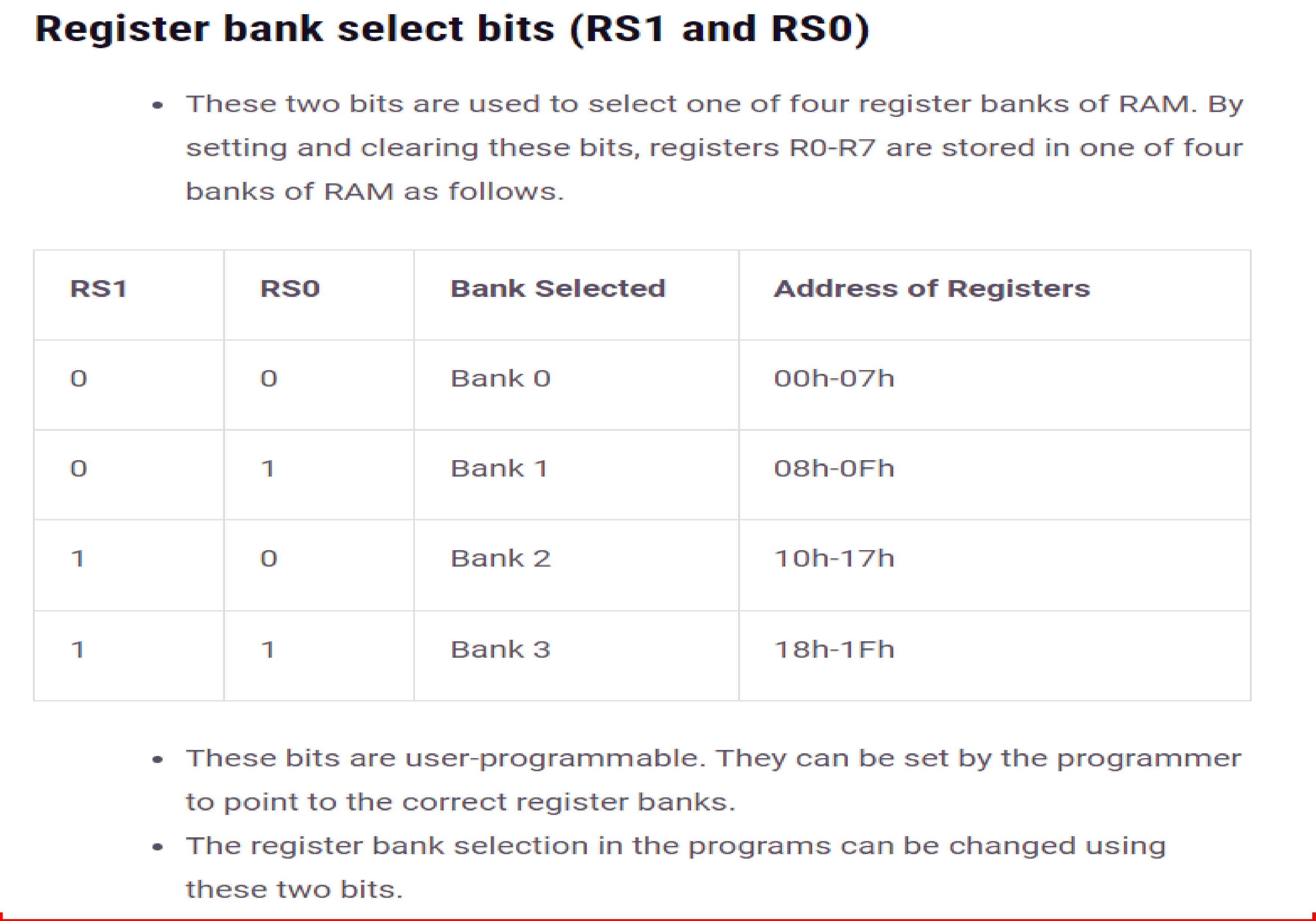
Figure 1: Format of PSW register in 8051 Microcontroller

Parity bit (P)

- This parity flag bit is used to show the number of 1s in the accumulator only. If the accumulator register contains an odd number of 1s, then this flag set to 1.
- If accumulator contains even number of 1s, then this flag cleared to 0.

Overflow flag (OV)

- This flag is set during ALU operations, to indicate overflow in the result. It is set to 1 if there is a carry out of either the D7 bit or the D6 bit of the accumulator.
- Overflow flag is set when arithmetic operations such as add and subtract result in sign conflict.

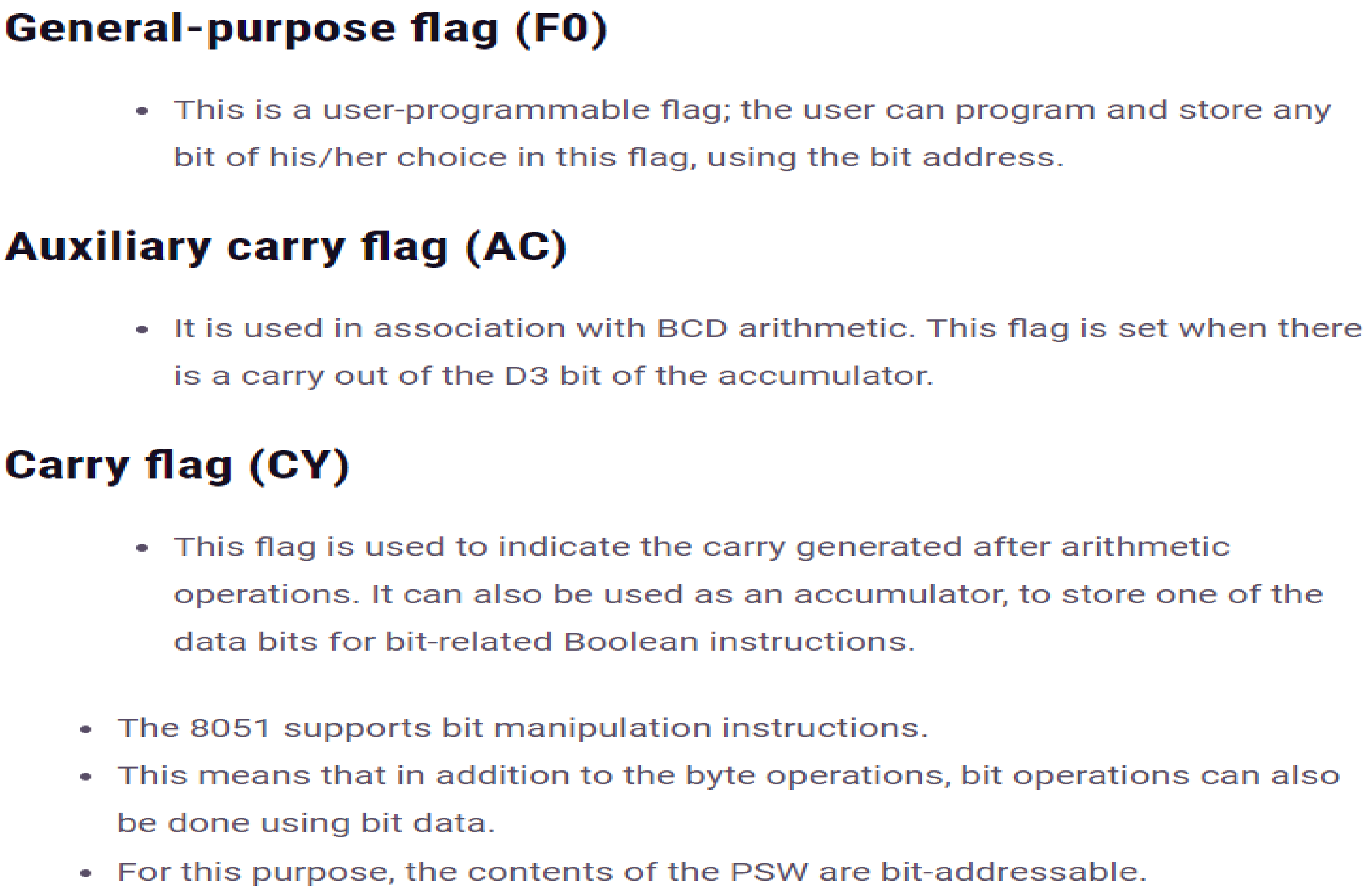


Register bank select bits (RS1 and RS0)

- These two bits are used to select one of four register banks of RAM. By setting and clearing these bits, registers R0-R7 are stored in one of four banks of RAM as follows.

RS1	RS0	Bank Selected	Address of Registers
0	0	Bank 0	00h-07h
0	1	Bank 1	08h-0Fh
1	0	Bank 2	10h-17h
1	1	Bank 3	18h-1Fh

- These bits are user-programmable. They can be set by the programmer to point to the correct register banks.
- The register bank selection in the programs can be changed using these two bits.



General-purpose flag (F0)

- This is a user-programmable flag; the user can program and store any bit of his/her choice in this flag, using the bit address.

Auxiliary carry flag (AC)

- It is used in association with BCD arithmetic. This flag is set when there is a carry out of the D3 bit of the accumulator.

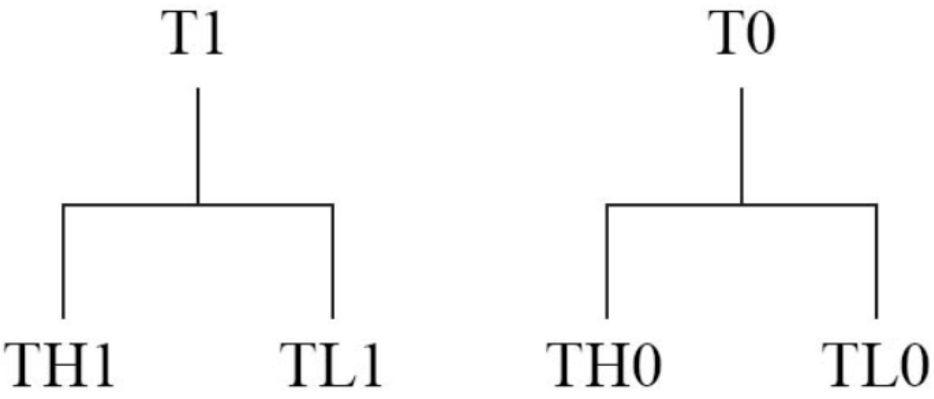
Carry flag (CY)

- This flag is used to indicate the carry generated after arithmetic operations. It can also be used as an accumulator, to store one of the data bits for bit-related Boolean instructions.

- The 8051 supports bit manipulation instructions.
- This means that in addition to the byte operations, bit operations can also be done using bit data.
- For this purpose, the contents of the PSW are bit-addressable.

Timers of 8051

- There are two timers in 8051, both are 16 bit timers . These are up counters. They can count from 0000 to FFFF (0 to 65,535). However, in each timer, one register is not enough to store 16bits, so both the timers have a total 4 SFRs, each SFR to store 8 bits. To trigger these timers they require a clock (either external or internal depends on many parameters).
- As 8051 has 8-bit architecture, each of these 16 bit timers can be accessed by two separate 8-bit registers.
- TMOD and TCON registers.



Timer Classification in 8051 Microcontroller

TL0 (Timer 0 Low Byte)

- TL0 is the lower 8-bit register of Timer 0.
- It stores the lower byte of the count value.
- When timer 0 functions as a 16-bit timer, TL0 increases first, and TH0 increases by one following an overflow (when 255 is achieved).

TH0 (Timer 0 High Byte)

- TH0 is the upper 8-bit register of Timer 0.
- It stores the higher byte of the count value.
- When TL0 overflows, Timer 0 can count up to 65,535 (FFFFH) before spilling since TH0 is increased.

TL1 (Timer 1 Low Byte)

- Timer 1's bottom 8-bit register is designated as TL1.
- It contains the count value's bottom byte.
- When using a 16-bit timer, TL1 increases first and then TH1 by 1 once TL1 reaches its maximum value of 255.

TH1 (Timer 1 High Byte)

- Timer 1's upper 8-bit register is designated as TH1.
- It contains the count value's higher byte.
- When TL1 overflows, TH1 increments, allowing Timer 1 to count up to 65,535 (FFFFH) before overflowing.

Timer Control Registers-TCON and TMOD

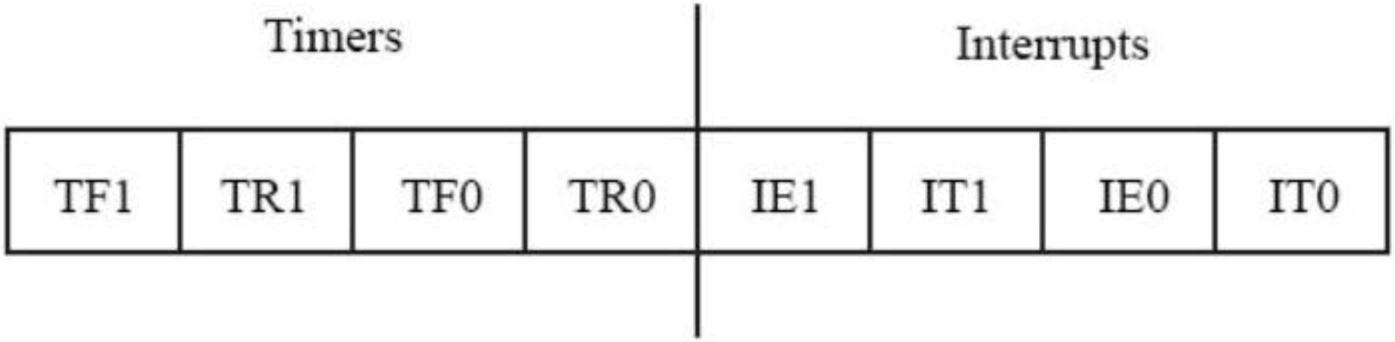
TCON and **TMOD** are the special function registers in the 8051 microcontroller. These are used to control the timers and counters.

1.TCON (Timer Control Register): The timers' start and stop are aided by this register. additionally indicates whether the timer is done counting.

2.TMOD (Timer Mode Register): The timers' mode is adjusted using this register. It whether the timers will record events occurring outside of the microcontroller or record

TCON (Timer Control Register) :

The 8051 microcontroller has a unique function register called the **TCON (Timer Control Register)**. In order to provide precise output, timers and counters are controlled by it. The data in the registers may overflow if these timers and counters are not under control. Thus, the TCON is utilized to control the timers and counters.



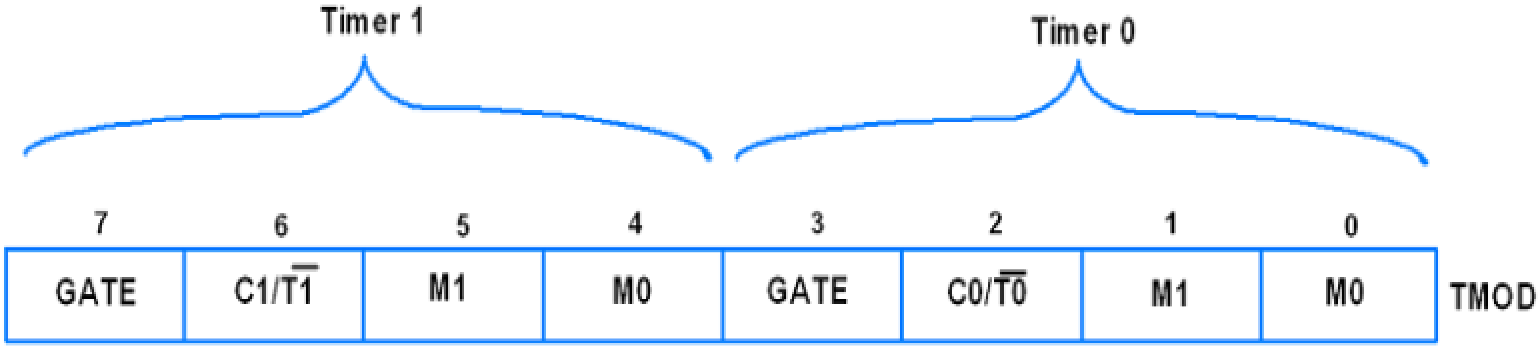
Timers				Interrupts			
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Tcon register in 8051 Mircrocontroller

TCON flag bits	Description
TF1	Timer overflow flag for timer 1. Whenever a timer overflow event occurs in timer 1, this flag becomes 1 and then interrupt is sent to the processor and as the processor goes to ISR this flag becomes zero. Note: Set to 1 when Timer 1 overflows else 0.
TF0	Timer overflow flag for timer 0. Whenever a timer overflow event occurs in timer 0, this flag becomes 1 and sends an interrupt to the processor and as the processor goes to ISR this flag becomes zero. Note:Set to 1 when Timer 0 overflows else 0.
TR1	Timer run for timer 1. Whenever the TR1 flag is 1, it means enabling the timer 1 (enabling timer / counter depends on another flag of another SFR). Note: Set to 1 to start Timer 1, and set to 0 to stop Timer 1.
TR0	Timer run for timer 0. Whenever the TR0 flag is 1, it means enabling the timer 0 (enabling timer / counter depends on another flag of another SFR). Note: Set to 1 to start Timer 0, and set to 0 to stop Timer 0.
IE0	This stands for interrupt external for INT0. whenever this flag is HIGH, it means an interrupt occurs at INT0. Note: Set to 1 when an external interrupt 0 occurs else 0.
IE1	This stands for interrupt external for INT1. whenever this flag is HIGH, it means an interrupt occurs at INT1. Note: Set to 1 when an external interrupt 1 occurs else 0.
IT1	**IT1** refers to the **interrupt type for INT1** (external interrupt 1). If **IT1 = 0**, the interrupt is **level triggered**. If **IT1 = 1**, the interrupt is **edge triggered**.
IT0	**IT0** refers to the **interrupt type for INT0** (external interrupt 0). If **IT0 = 0**, the interrupt is **level triggered**. If **IT0 = 1**, the interrupt is **edge triggered**.

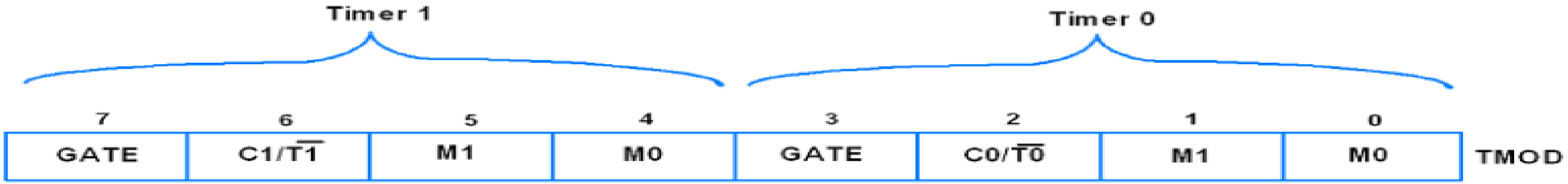
TMOD (Timer Mode Register):

The TMOD (Timer Mode Register) is a special function register in the 8051 microcontroller. Timer 0 and Timer 1 are the modes of operation that it is utilized to set. Whether a timer or counter needs to be set, it is done so using this register. The eight bits of the TMOD register are split into two sections: Timer 0 is controlled by the lower four bits, and Timer 1 is controlled by the upper four bits.



7	6	5	4	3	2	1	0
GATE	C1/$\overline{T1}$	M1	M0	GATE	C0/$\overline{T0}$	M1	M0

Its lower 4 bits are used for Timer0 and the upper 4 bits are used for Timer1



Timer 1				Timer 0				
7	6	5	4	3	2	1	0	
GATE	C1/$\overline{T1}$	M1	M0	GATE	C0/$\overline{T0}$	M1	M0	TMOD

Its lower 4 bits are used for Timer0 and the upper 4 bits are used for Timer1

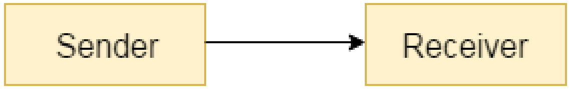
TMOD flag bits	**Description**
GATE	**1** = Enable Timer/Counter only when the INT0/INT1 **(external interrupt)**. pin is high and TR0/TR1 **(Timer run)** is set. ***(HIGH)*** **0** = Enable Timer/Counter when TR0/TR1 is set.
C/T(Counter/Timer)	Timer or Counter select bit **1** = Use as Counter **0** = Use as Timer
M0/M1	Timer/Counter mode select bit

To select modes:

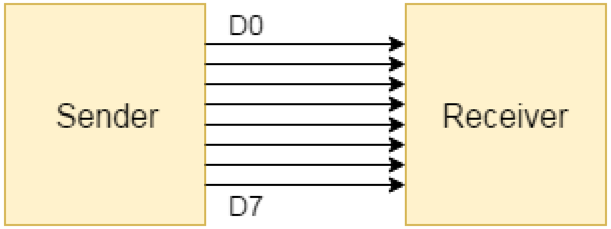
M1	**M0**	**Mode**
0	0	0 (13-bit timer mode)
0	1	1 (16-bit timer mode)
1	0	2 (8-bit auto-reload mode)
1	1	3 (split timer mode)

Serial Ports of 8051

- **Serial transfer**: In serial transfer, data is transfer to device located many meters away this method is used for long distance data transfer. Serial communication is mostly used for transmitting and receiving the signal. The 8051 microcontroller is consisting of **Universal Asynchronous Receiver Transmitter (UART)** used for serial communication. The signals are transmitted and received by the Rx and Tx pins of microcontroller.

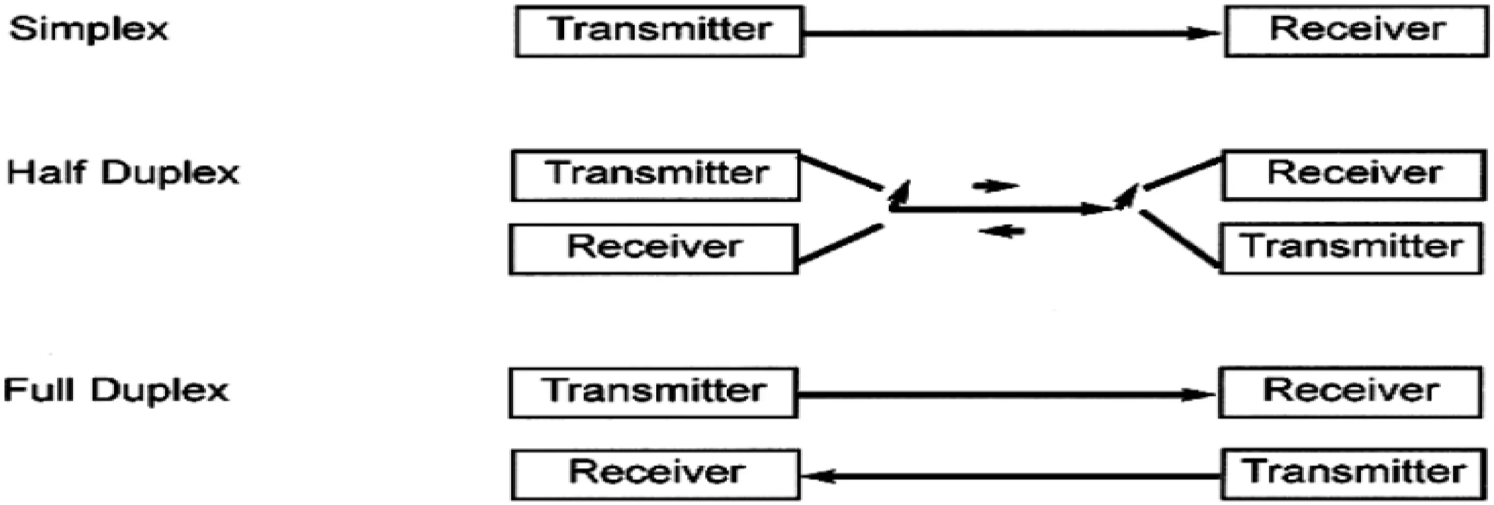


- **Parallel transfer**: In parallel transfer, data is transferred in 8 or more lines. In this wire conductor is used for transferring data to a device that is only a few feet away.



Modes of Serial transmission

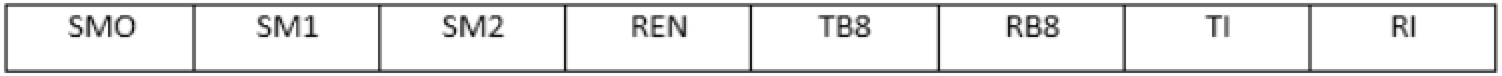
- There are two ways to transmit serial data: Simplex and Duplex.
- In simplex transmissions, the computer can only send data. There is only one wire.
- If the data can be transmitted and received, then it is a duplex transmission. Duplex transmissions can be half or full duplex depending on whether or not the data transfer can be simultaneous
- If the communication is only one way at a time, it is half duplex.
- If both sides can communicate at the same time, it is full duplex
- Full duplex requires two wire conductors for the data lines (in addition to the signal ground)



- Serial Communication can be : Asynchronous and Synchronous.
- **Synchronous Communication:** Transfer a block of data (characters) at a time. The events are referenced to a clock. Example: SPI bus, I2C bus.
- **Asynchronous Communication:** Transfer a single byte at a time. There is no clock. The bytes are separated by start and stop bits. Example: UART.
- The rate of data transfer in serial data communication is state as bps (bits per second)/ baud rate.
- **IC's for serial communication :** UART (Universal asynchronous receiver- transmitter), USART (Universal synchronous-asynchronous receiver-transmitter).
- 8051 uses SBUF (serial buffer) register to hold data, SCON (serial port control) register to control data communication and PCON (power mode control) register to control data rates.

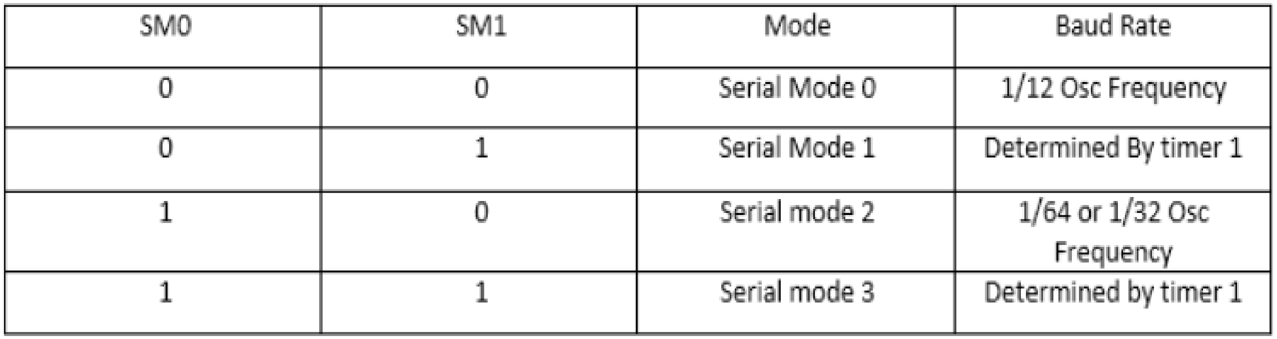
SCON register:

Serial port control and status register is the special function register SCON. It is a bit addressable register used to set the mode in which serial communication takes place in the controller..
This register contain not only the mode selection bits but also the 9th data bit for transmit and receive (TB8 and RB8) and the serial part interrupt bits (TI and RI)



SMO	SM1	SM2	REN	TB8	RB8	TI	RI

SMO – Serial port mode 0 shift register
SM1 – serial port mode 1 8 bit UAR + variable
SM2 – enable multiprocessor communication in mode 2/3
REN – set/ clear by software to enable/disable reception
TB8 – the 9th bit that will be transmitted in mode 2/3 set/clear by software.
RB8 – in mode 2/3 it is the 9th bit that was received in mode 1 if SM2 =0, RB8 is the stop bit that was received in mode it is not used.
TI – transmit interrupt flag set by hardware at the end of 8 bit time in mode 0 at the beginning of the stop bit in the other mode it must be cleared by software
RI – receive interrupt flag set by hardware and must be cleared by software.



SM0	SM1	Mode	Baud Rate
0	0	Serial Mode 0	1/12 Osc Frequency
0	1	Serial Mode 1	Determined By timer 1
1	0	Serial mode 2	1/64 or 1/32 Osc Frequency
1	1	Serial mode 3	Determined by timer 1

Serial Mode of 8051

1. **SM0, SM1:** Serial Mode Control Bits

2. **SM2:** Multiprocessor mode control bit, logic 1 enables Multiprocessor mode and 0 for normal mode.

3. **REN:** Enables Serial reception. If set, it enables the reception otherwise the reception is disabled.

4. **TB8:** It is the 9th bit of the data that is to be transmitted.

5. **RB8:** It is used in modes 2 and 3, it is the 9th bit received by the microcontroller.

6. **TI:** It is known as Transmit Interrupt flag which is set by hardware to indicate the end of a transmission. It has to be cleared by the software.

7. **RI:** It is known as Receive Interrupt flag which is set by hardware to indicate the end of a reception. It has to be cleared by the software.

Interrupt

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

Hardware Interrupt

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

Software Interrupt

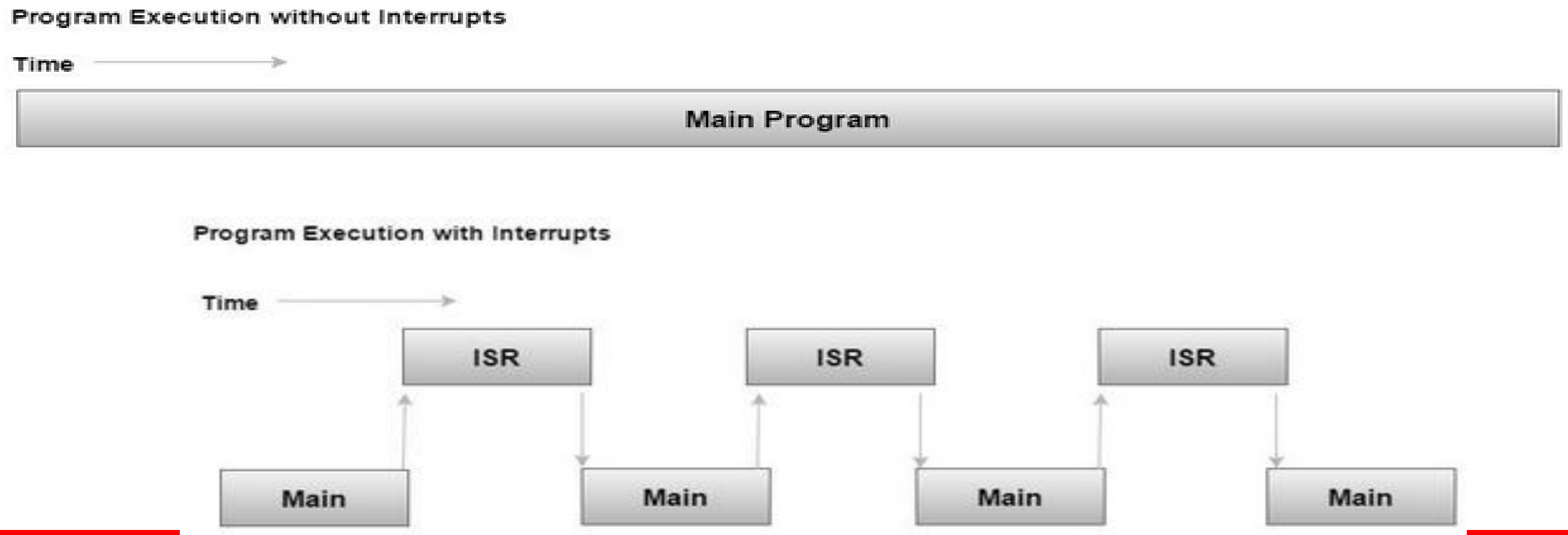
- A software interrupt is caused either by an exceptional condition or a special instruction in the instruction set which causes an interrupt when it is executed by the processor.
- For example, if the processor's arithmetic logic unit runs a command to divide a number by zero, to cause a divide-by-zero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

What is Polling?

- The state of continuous monitoring is known as **polling**. The microcontroller keeps checking the status of other devices; and while doing so, it does no other operation and consumes all its processing time for monitoring. This problem can be addressed by using interrupts.
- In the interrupt method, the controller responds only when an interruption occurs. Thus, the controller is not required to regularly monitor the status (flags, signals etc.) of interfaced and inbuilt devices.

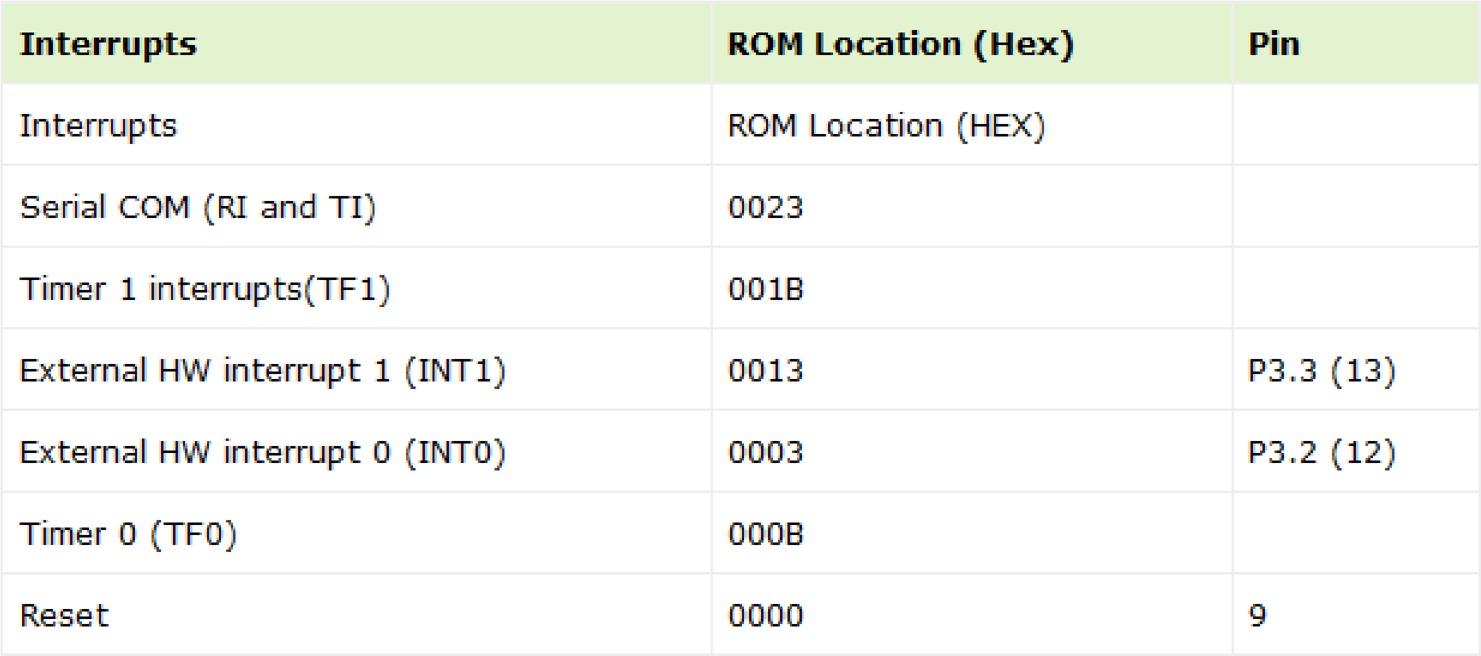
Interrupt Service Routine (ISR)

- For every interrupt, there must be an interrupt service routine (ISR), or **interrupt handler**.
- When an interrupt occurs, the microcontroller runs the ISR.
- For every interrupt, there is a fixed location in memory that holds the address of its ISR.
- The table of memory locations set aside to hold the addresses of ISRs is called as the Interrupt Vector Table.


Program Execution without Interrupts
Time
Main Program
Program Execution with Interrupts
Time
ISR
ISR
ISR
Main
Main
Main
Main

Interrupt Vector Table

There are six interrupts including RESET in 8051.



Interrupts	ROM Location (Hex)	Pin
Interrupts	ROM Location (HEX)	
Serial COM (RI and TI)	0023	
Timer 1 interrupts(TF1)	001B	
External HW interrupt 1 (INT1)	0013	P3.3 (13)
External HW interrupt 0 (INT0)	0003	P3.2 (12)
Timer 0 (TF0)	000B	
Reset	0000	9

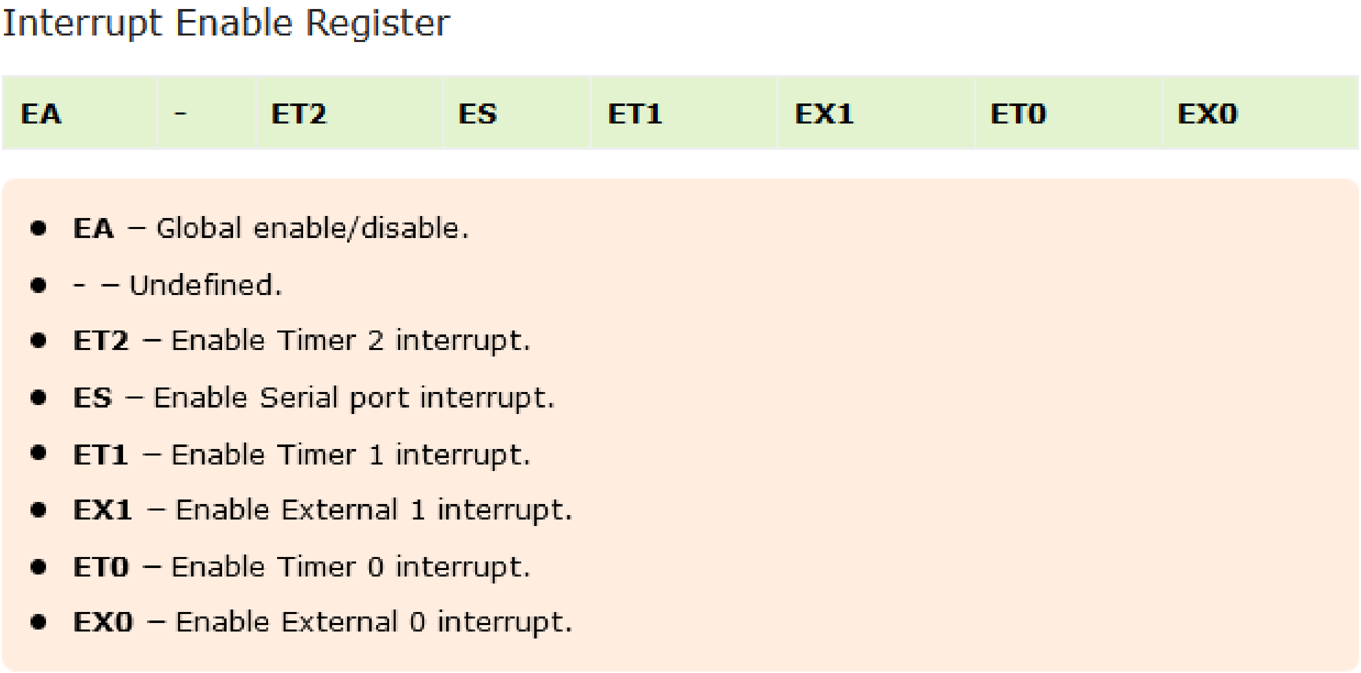
Steps to handle an Interrupts

There are six interrupts including RESET in 8051.

- The microcontroller closes the currently executing instruction and saves the address of the next instruction (PC) on the stack.
- It also saves the current status of all the interrupts internally (i.e., not on the stack).
- It jumps to the memory location of the interrupt vector table that holds the address of the **ISR**.
- The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it. It starts to execute the interrupt service subroutine, which is RETI (return from interrupt).
- Upon executing the RETI instruction, the microcontroller returns to the location where it was interrupted..
- First, it gets the program counter (PC) address from the stack by popping the top bytes of the stack into the PC. Then, it start to execute from that address

Enabling and Disabling the Interrupts

- Upon Reset, all the interrupts are disabled even if they are activated. The interrupts must be enabled using software in order for the microcontroller to respond to those interrupts.
- **IE (interrupt enable) register** is responsible for enabling and disabling the interrupt. IE is a bit addressable register.



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.

Enabling and Disabling the Interrupts

•To enable an interrupt, we take the following steps –

•Bit D7 of the IE register (EA) must be high to allow the rest of register to take effect.

• If EA = 1, interrupts will be enabled and will be responded to
If, their corresponding bits in IE are high.

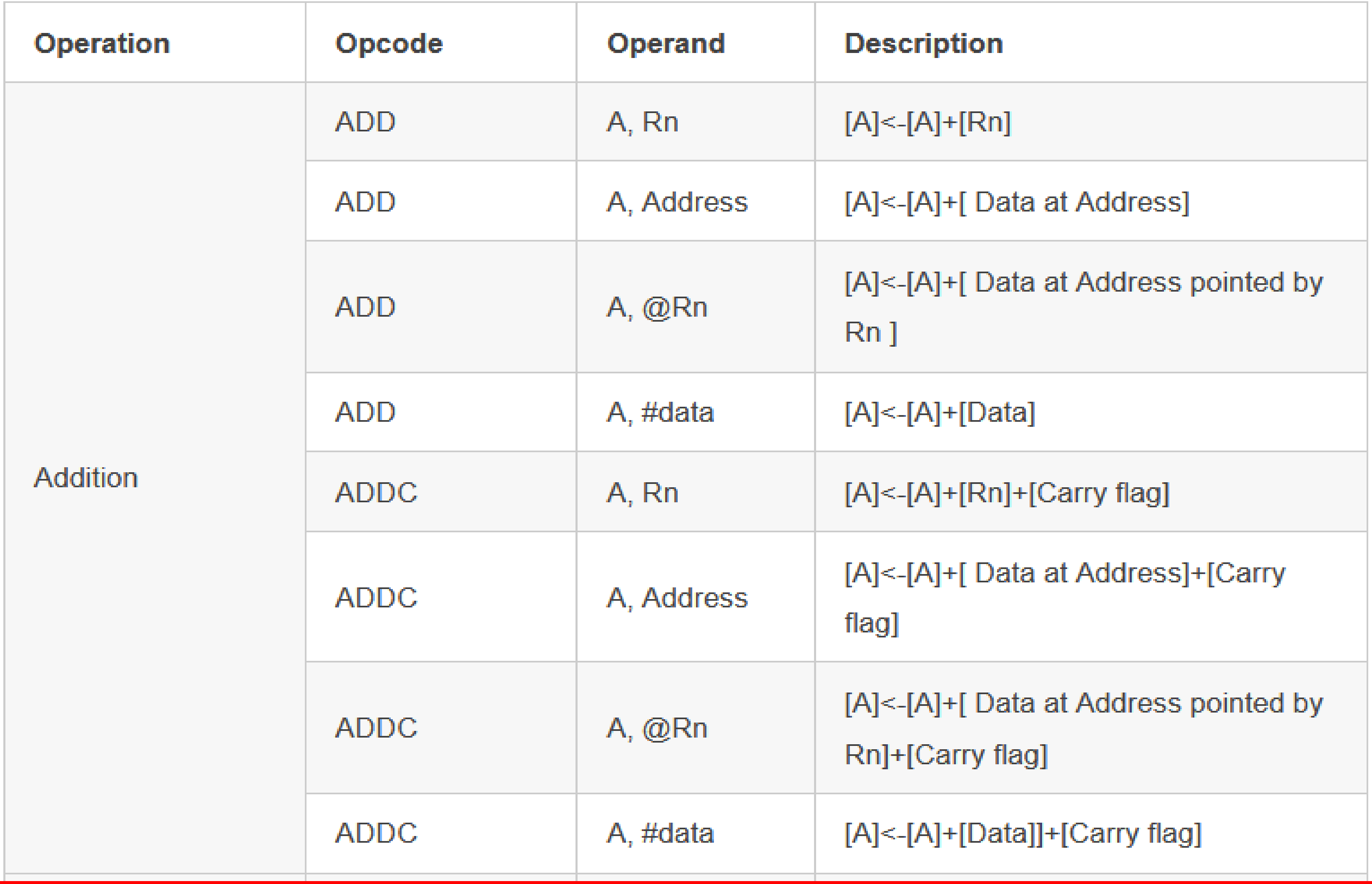
• If EA = 0, no interrupts will be responded
even if, their associated pins in the IE register are high.

List of Arithmetic instructions of 8051:

- In 8051 Microcontroller there are 24 different instructions under the Arithmetic Group.
- In total there are 64 opcodes.
- The Carry Flag (CY), Auxiliary Carry (AC)and Overflow flag (OV) are affected based on the result of ADD, ADDC, SUBB etc. instructions.

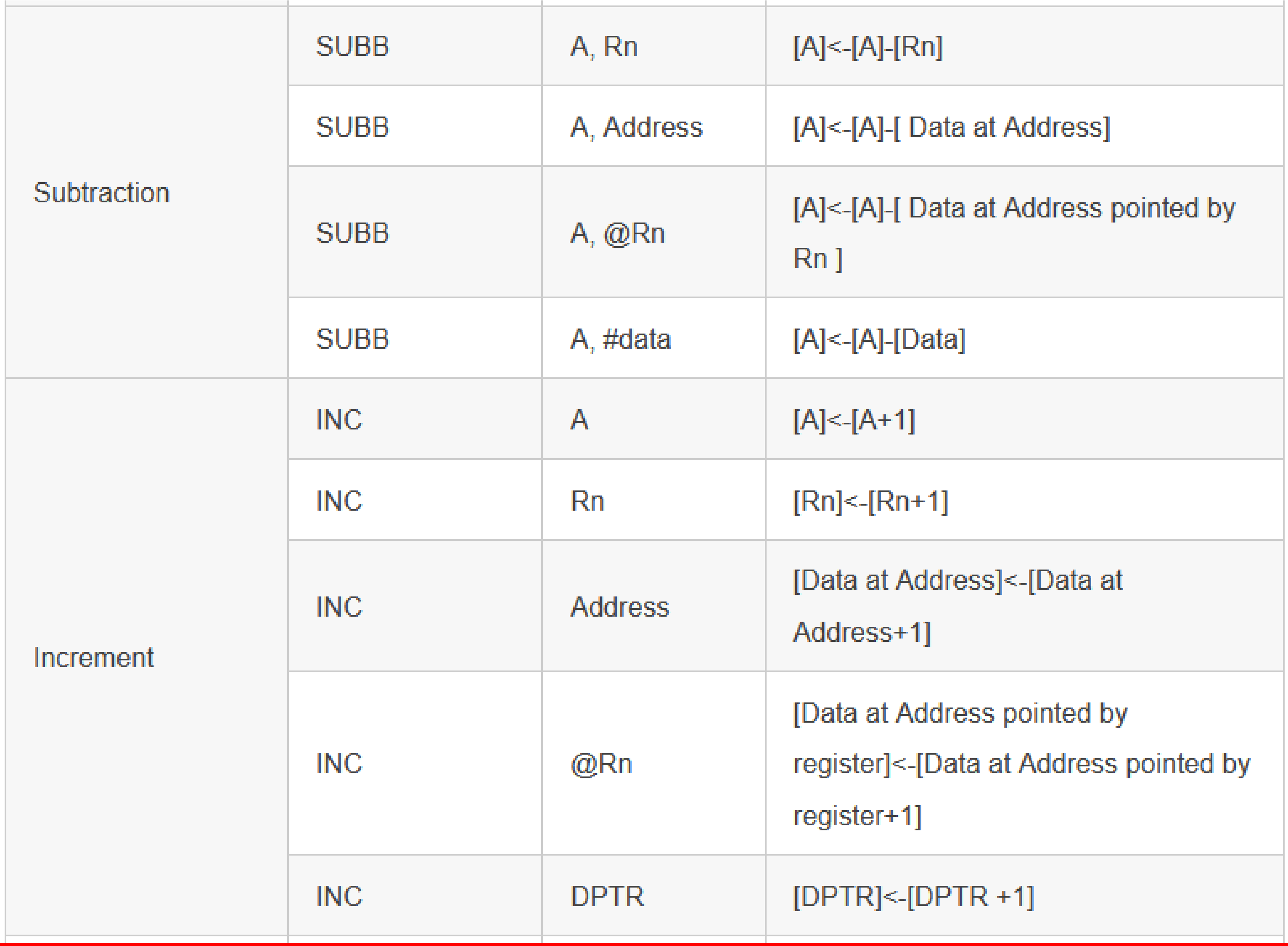
- ADD- Addition without carry
- ADDC- ADDC stands for addition with carry
- SUB- Subtraction
- SUBB- Subtraction with Borrow
- MUL-Multiplication
- DIV-Division

Arithmetic instruction: Addition



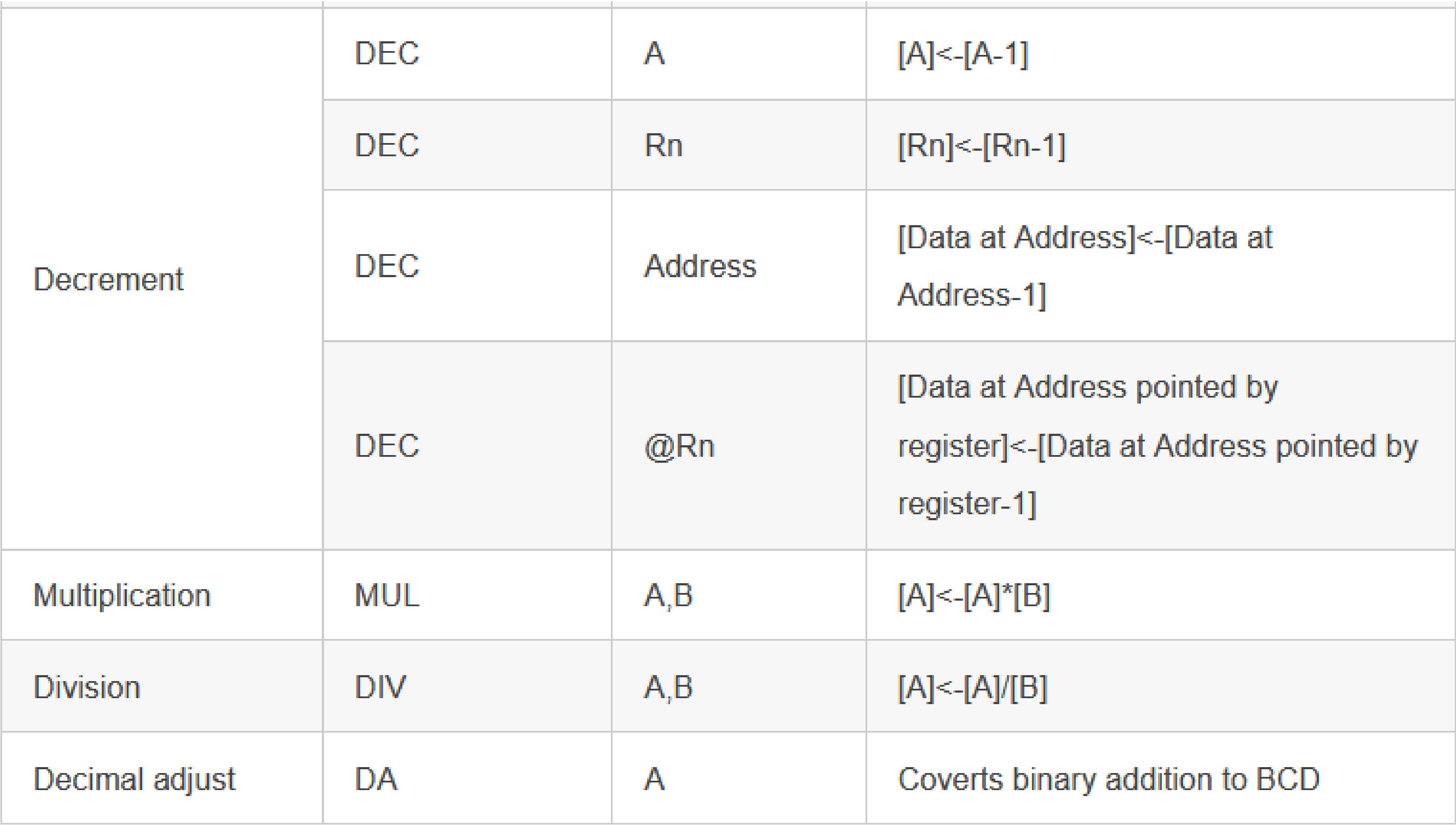
Operation	Opcode	Operand	Description
Addition	ADD	A, Rn	[A]<-[A]+[Rn]
	ADD	A, Address	[A]<-[A]+[Data at Address]
	ADD	A, @Rn	[A]<-[A]+[Data at Address pointed by Rn]
	ADD	A, #data	[A]<-[A]+[Data]
	ADDC	A, Rn	[A]<-[A]+[Rn]+[Carry flag]
	ADDC	A, Address	[A]<-[A]+[Data at Address]+[Carry flag]
	ADDC	A, @Rn	[A]<-[A]+[Data at Address pointed by Rn]+[Carry flag]
	ADDC	A, #data	[A]<-[A]+[Data]]+[Carry flag]

Arithmetic instruction: Subtraction and Increment



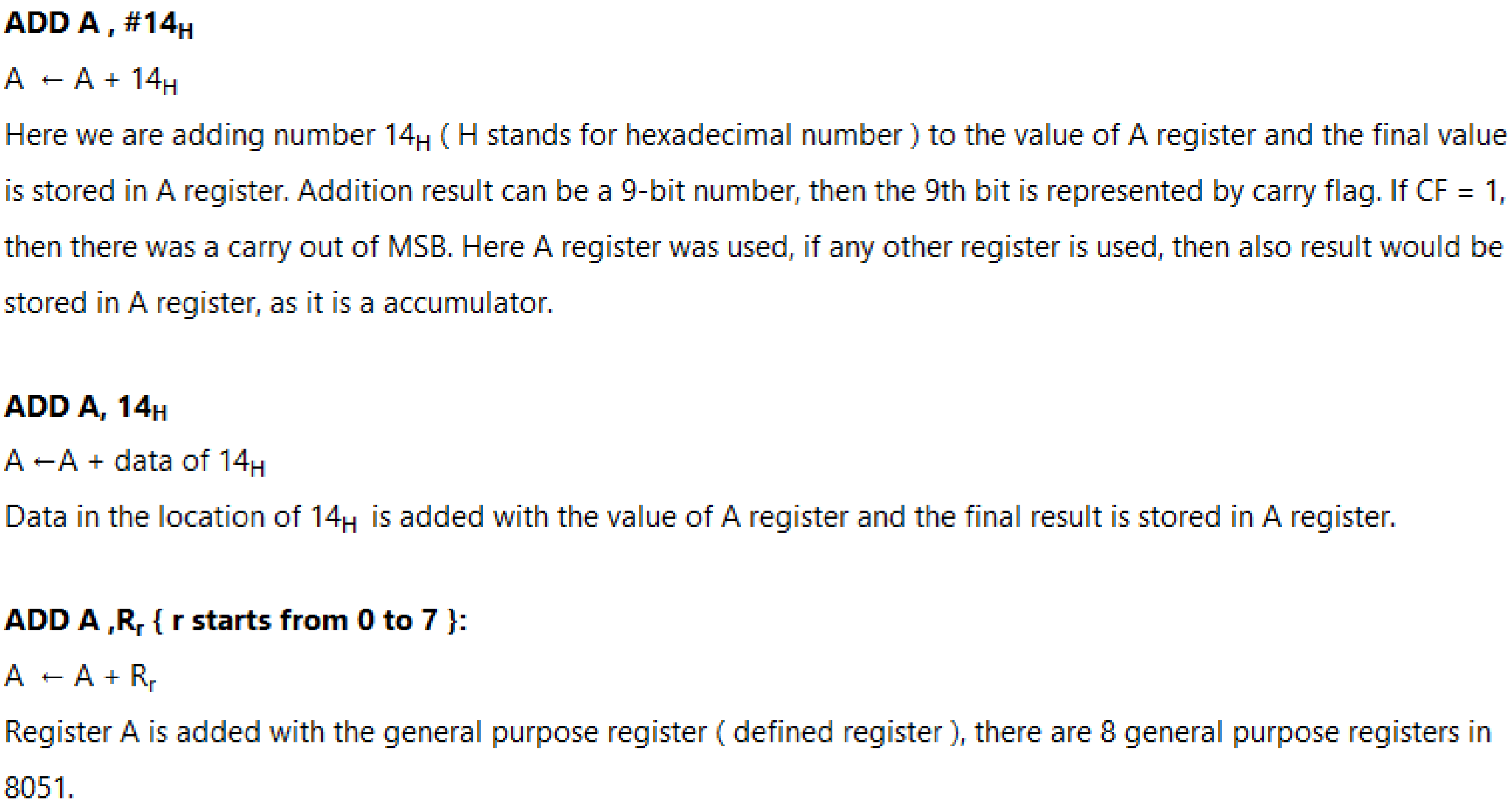
Subtraction	SUBB	A, Rn	[A]<-[A]-[Rn]
	SUBB	A, Address	[A]<-[A]-[Data at Address]
	SUBB	A, @Rn	[A]<-[A]-[Data at Address pointed by Rn]
	SUBB	A, #data	[A]<-[A]-[Data]
Increment	INC	A	[A]<-[A+1]
	INC	Rn	[Rn]<-[Rn+1]
	INC	Address	[Data at Address]<-[Data at Address+1]
	INC	@Rn	[Data at Address pointed by register]<-[Data at Address pointed by register+1]
	INC	DPTR	[DPTR]<-[DPTR +1]

Arithmetic instruction: Decrement, Multiplication, Division, and Decimal Adjust



Decrement	DEC	A	[A]<-[A-1]
	DEC	Rn	[Rn]<-[Rn-1]
	DEC	Address	[Data at Address]<-[Data at Address-1]
	DEC	@Rn	[Data at Address pointed by register]<-[Data at Address pointed by register-1]
Multiplication	MUL	A,B	[A]<-[A]*[B]
Division	DIV	A,B	[A]<-[A]/[B]
Decimal adjust	DA	A	Coverts binary addition to BCD

Arithmetic instruction: Few Examples



ADD A , #14_H

A ← A + 14_H

Here we are adding number 14_H (H stands for hexadecimal number) to the value of A register and the final value is stored in A register. Addition result can be a 9-bit number, then the 9th bit is represented by carry flag. If CF = 1, then there was a carry out of MSB. Here A register was used, if any other register is used, then also result would be stored in A register, as it is a accumulator.

ADD A, 14_H

A ←A + data of 14_H

Data in the location of 14_H is added with the value of A register and the final result is stored in A register.

ADD A ,R_r { r starts from 0 to 7 }:

A ← A + R_r

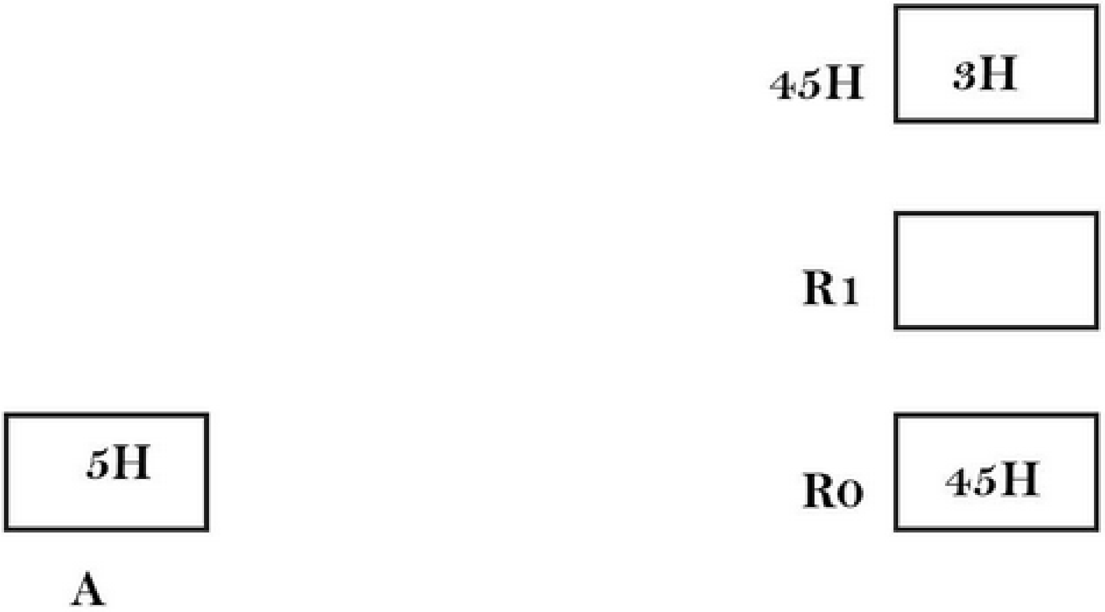
Register A is added with the general purpose register (defined register), there are 8 general purpose registers in 8051.

Arithmetic instruction: Few Examples

ADD A, @R_P { only R_0 and R_1 }:

A ← A + value of the location, stored in the RP register.

Example : ADD A, @R_0



The answer of this addition would be 5H + 3H , which is 8H.

Arithmetic instruction: Few Examples

ADDC A, #14_H :

ADDC stands for addition with carry.

A ← A + 14_H + CF CF = previous data stored in carry flag.

This instruction comes into play when we perform additions with carry.

Example : Add $2F_H$ and 13_H (hexadecimal addition)

2 F

+ 1 3

In this addition first F + 3 is added, which is 12 , 1 is carry which is stored in the carry flag. Next 8051 adds 2 + 1 + 1(carry stored in the carry flag).

ADDC A, 14_H :

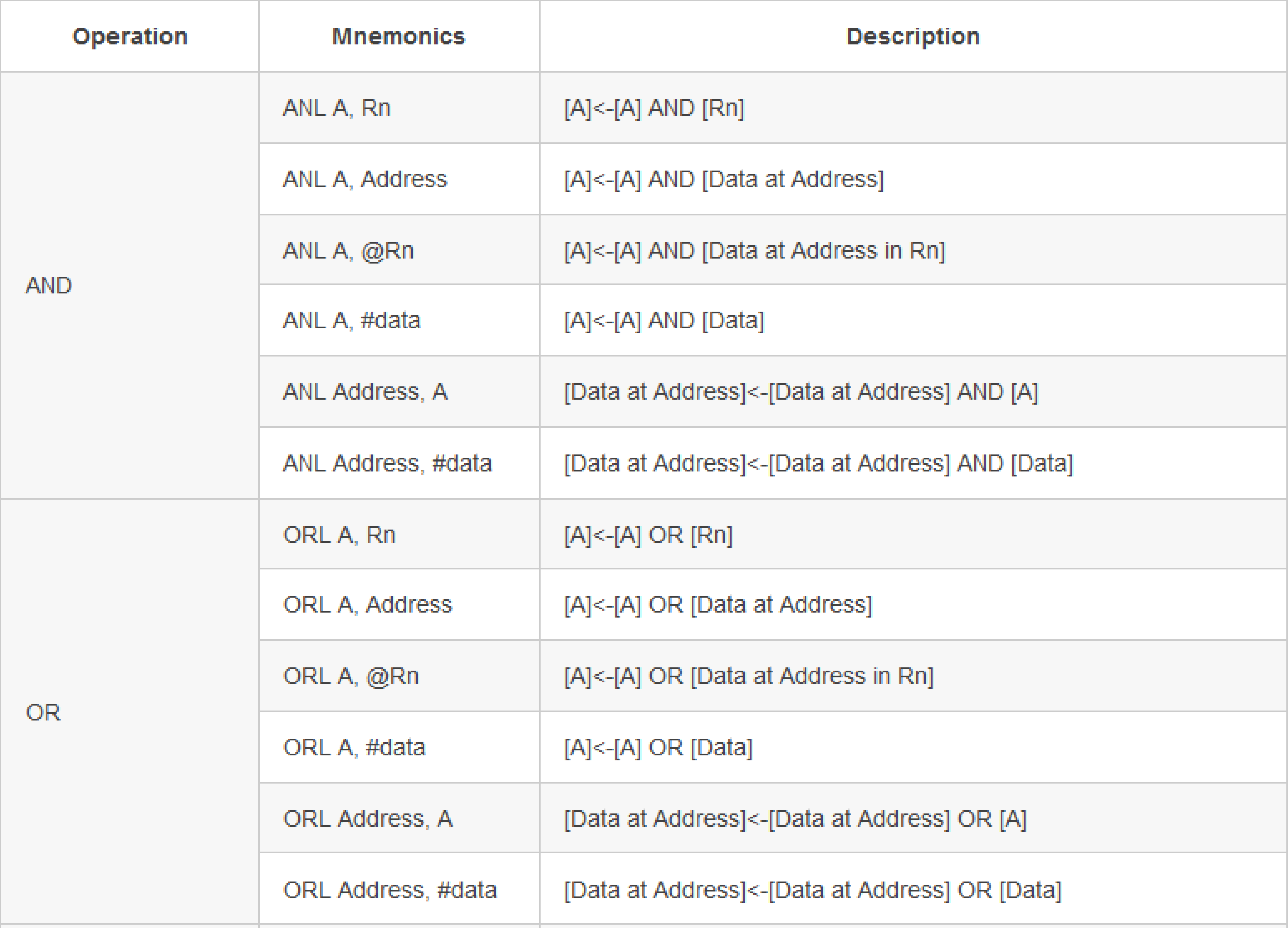
This is similar to the above instruction, but instead of adding directly 14_H ,here the data from that location is accessed. Final result is stored in A register.

List of Logical instructions of 8051:

- In 8051 Microcontroller there are 25 logical instructions in 8051. AND , OR, XOR , NOT …. These instructions come under logical instructions.
- In total there are 49 opcodes.
- The Carry Flag (CY) affects only by instruction RRC and RLC.

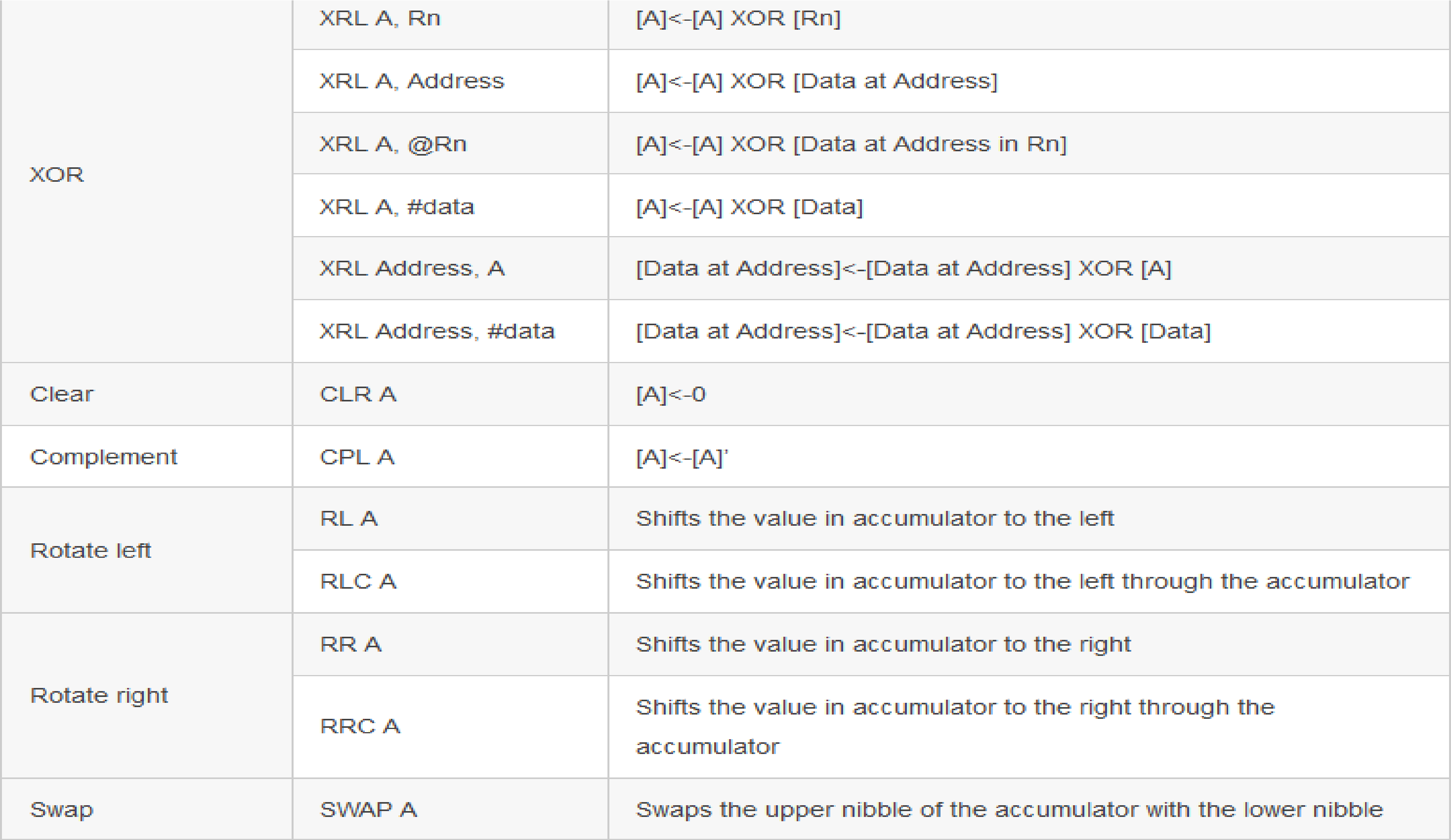
- ANL- AND Operation
- ORL-OR operation
- XRL- XOR Operation
- CLR-Clear
- CPL-Complement
- RL/RLC- Rotate Left
- RR/RRC- Rotate Right
- SWAP-Swap

Logical instruction: AND & OR



Operation	Mnemonics	Description
AND	ANL A, Rn	[A]<-[A] AND [Rn]
	ANL A, Address	[A]<-[A] AND [Data at Address]
	ANL A, @Rn	[A]<-[A] AND [Data at Address in Rn]
	ANL A, #data	[A]<-[A] AND [Data]
	ANL Address, A	[Data at Address]<-[Data at Address] AND [A]
	ANL Address, #data	[Data at Address]<-[Data at Address] AND [Data]
OR	ORL A, Rn	[A]<-[A] OR [Rn]
	ORL A, Address	[A]<-[A] OR [Data at Address]
	ORL A, @Rn	[A]<-[A] OR [Data at Address in Rn]
	ORL A, #data	[A]<-[A] OR [Data]
	ORL Address, A	[Data at Address]<-[Data at Address] OR [A]
	ORL Address, #data	[Data at Address]<-[Data at Address] OR [Data]

Logical instruction: XOR, Clear, Complement, Rotate Left and right,



XOR	XRL A, Rn	[A]<-[A] XOR [Rn]
	XRL A, Address	[A]<-[A] XOR [Data at Address]
	XRL A, @Rn	[A]<-[A] XOR [Data at Address in Rn]
	XRL A, #data	[A]<-[A] XOR [Data]
	XRL Address, A	[Data at Address]<-[Data at Address] XOR [A]
	XRL Address, #data	[Data at Address]<-[Data at Address] XOR [Data]
Clear	CLR A	[A]<-0
Complement	CPL A	[A]<-[A]'
Rotate left	RL A	Shifts the value in accumulator to the left
	RLC A	Shifts the value in accumulator to the left through the accumulator
Rotate right	RR A	Shifts the value in accumulator to the right
	RRC A	Shifts the value in accumulator to the right through the accumulator
Swap	SWAP A	Swaps the upper nibble of the accumulator with the lower nibble

Logical Instruction: Few Examples

ANL A, #14_H :

This is AND instruction performed between the data of register A, and the 14_H.

A ← A *14_H , the result is stored in A register, * is the symbol of and operation.

ANL A, R_0 :

This is AND instruction similar to above, but this is AND operation between A register and R_0 register. The result is stored in A register.

A ← A *R_0

ANL A, 14_H :

A ← A + [14_H]

Here the AND is performed between A register content and content of 14_H location. The result is stored in A register.

A ← A *[14_H]

Logical Instruction: Few Examples

ANL A, @R_0 :

Here @R_0 stands for the data of the location , stored in R_0.

ANL 14_H, A :

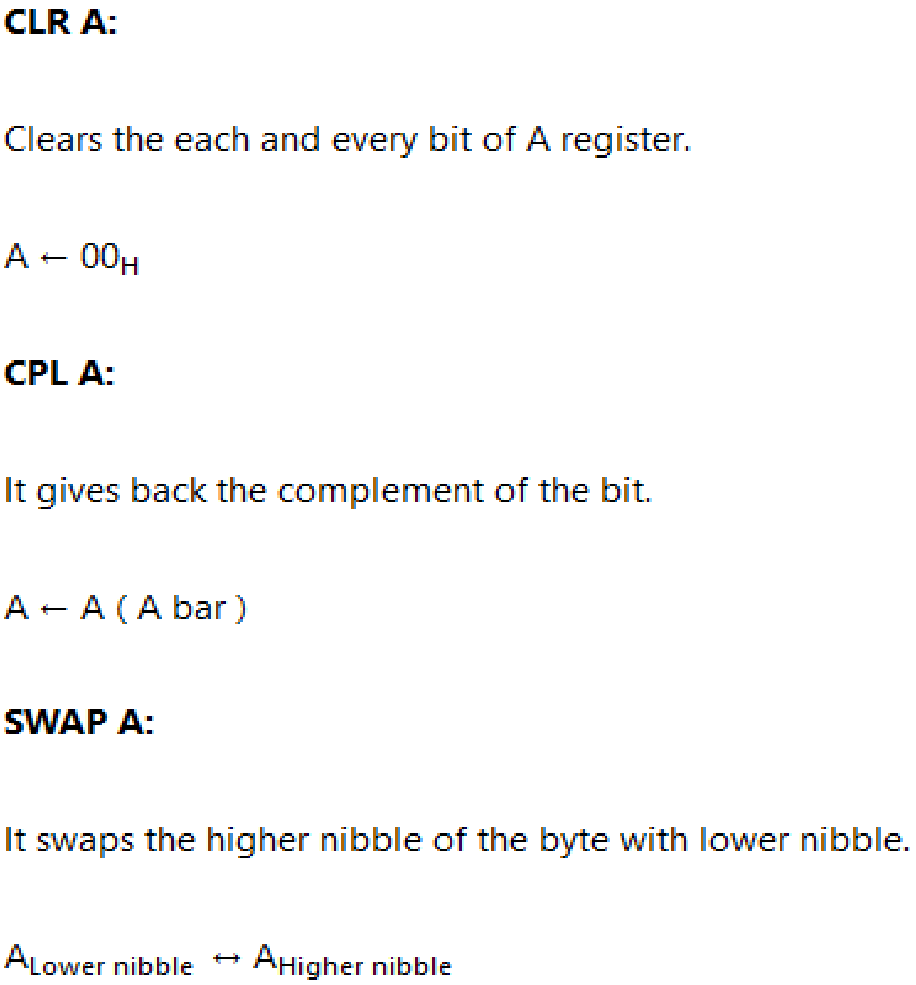
This is the same to **ANL A, 14_H ,** but in this instruction the final result is stored in 14_H location.

[14_H] ← A * [14_H]

ANL 14_H , #14_H :

[14_H] ← 14_H + [14_H] ; both are different here.

Logical Instruction: Few Examples



CLR A:

Clears the each and every bit of A register.

$A \leftarrow 00_H$

CPL A:

It gives back the complement of the bit.

$A \leftarrow A$ (A bar)

SWAP A:

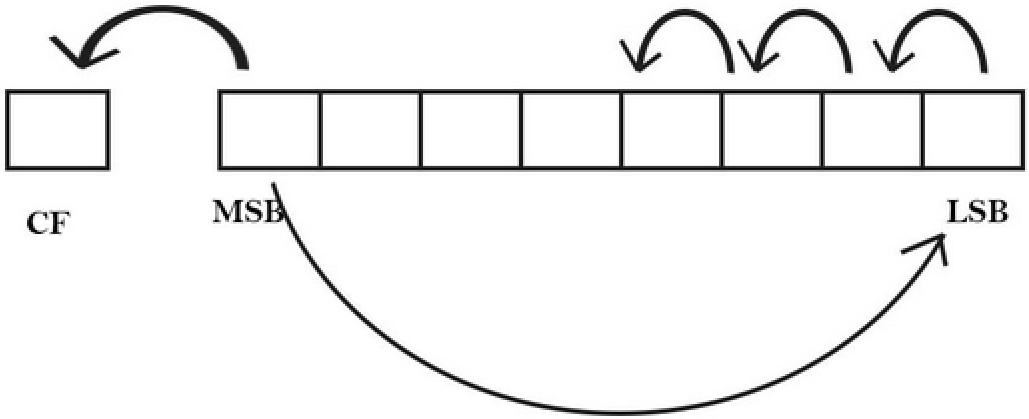
It swaps the higher nibble of the byte with lower nibble.

$A_{\text{Lower nibble}} \leftrightarrow A_{\text{Higher nibble}}$

Logical Instruction: Few Examples

RL A

This instruction is to shift all the bits from LSB to all the way left of it. Thus named rotate left.

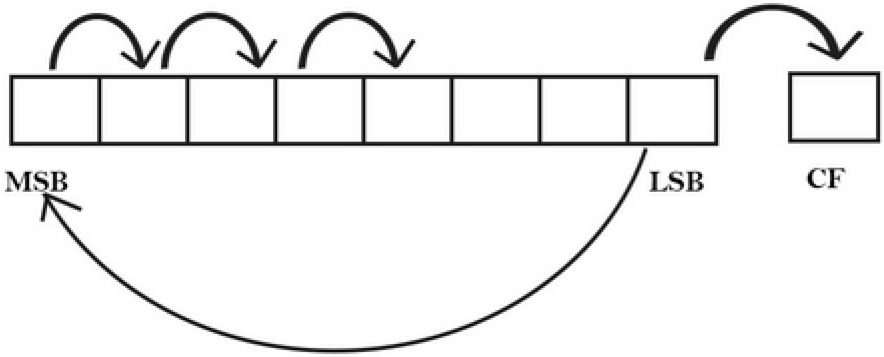


From the above figure, rotating left goes all up to MSB, and MSB value is assigned to LSB as well as Carry Flag, where the CF has lost its value.

Logical Instruction: Few Examples

RR A

This instruction is to shift all the bits from MSB to all the way right of it. Thus named rotate right.

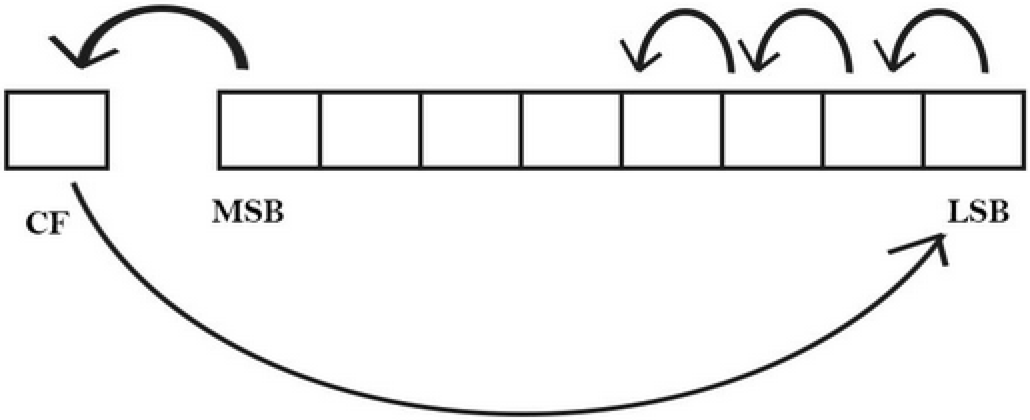


From the above figure, rotating right goes all up to LSB, and LSB value is assigned to MSB as well as Carry Flag, where the CF has lost its value.

Logical Instruction: Few Examples

RLC A

This instruction is to shift all the bits from LSB to all the way left of it including the carry flag in it. Thus named rotate left with carry.

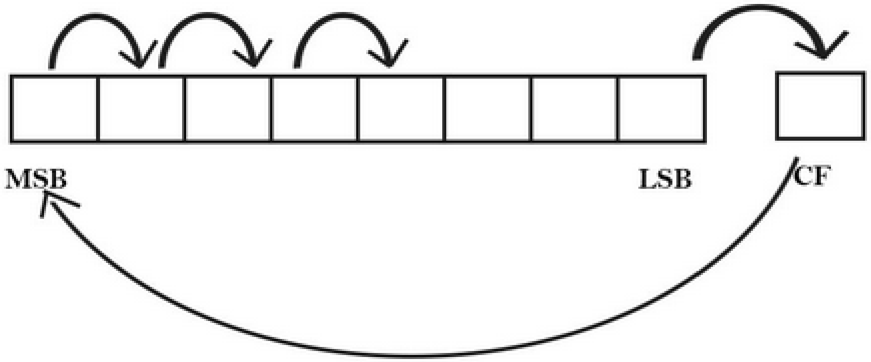


From the above figure, rotating left goes all up to CF, and then the value of CF is assigned to LSB. Here we don't lose the data of the CF flag register.

Logical Instruction: Few Examples

RRC A

This instruction is to shift all the bits from MSB to all the way left of it including the carry flag in it. Thus named rotate right with carry.



From the above figure, rotating right goes all up to CF, and then the value of CF is assigned to MSB. Here we don't lose the data of the CF flag register.

Thank you