MODULE: 4

8051 microcontroller

What is a Microcontroller?

A Microcontroller is a programmable digital processor with necessary peripherals. Both microcontrollers and microprocessors are complex sequential digital circuits meant to carry out job according to the program / instructions. Sometimes analog input/output interface makes a part of microcontroller circuit of mixed mode(both analog and digital nature).

Microcontrollers Vs Microprocessors

1. A microprocessor requires an external memory for program/data storage. Instruction execution requires movement of data from the external memory to the microprocessor or vice versa. Usually, microprocessors have good computing power and they have higher clock speed to facilitate faster computation.
2. A microcontroller has required on-chip memory with associated peripherals. A microcontroller can be thought of a microprocessor with inbuilt peripherals.
3. A microcontroller does not require much additional interfacing ICs for operation and it functions as a stand alone system. The operation of a microcontroller is multipurpose, just like a Swiss knife.
4. Microcontrollers are also called embedded controllers. A microcontroller clock speed is limited only to a few tens of MHz. Microcontrollers are numerous and many of them are application specific.

Development/Classification of microcontrollers (Invisible)

Microcontrollers have gone through a silent evolution (invisible). The evolution can be rightly termed as silent as the impact or application of a microcontroller is not well known to a common user, although microcontroller technology has undergone significant change since early 1970's. Development of some popular microcontrollers is given as follows.

Intel 4004	4 bit (2300 PMOS trans, 108 kHz)	1971
Intel 8048	8 bit	1976
Intel 8031	8 bit (ROM-less)	.
Intel 8051	8 bit (Mask ROM)	1980
Microchip PIC16C64	8 bit	1985
Motorola 68HC11	8 bit (on chip ADC)	.
Intel 80C196	16 bit	1982
Atmel AT89C51	8 bit (Flash memory)	.
Microchip PIC 16F877	8 bit (Flash memory + ADC)	.

Development of microprocessors (Visible)

Microprocessors have undergone significant evolution over the past four decades. This development is clearly perceptible to a common user, especially, in terms of phenomenal growth in capabilities of personal computers. Development of some of the microprocessors can be given as follows.

Intel 4004	4 bit (2300 PMOS transistors)	1971
Intel 8080 8085	8 bit (NMOS) 8 bit	1974
Intel 8088 8086	16 bit 16 bit	1978
Intel 80186 80286	16 bit 16 bit	1982
Intel 80386	32 bit (275000 transistors)	1985
Intel 80486 SX DX	32 bit 32 bit (built in floating point unit)	1989
Intel 80586 I MMX Celeron II III IV	64 bit	 1993 1997 1999 2000
Z-80 (Zilog)	8 bit	1976
Motorola Power PC 601 602 603	32-bit	 1993 1995

We use more number of microcontrollers compared to microprocessors. Microprocessors are primarily used for computational purpose, whereas microcontrollers find wide application in devices needing real time processing / control.

Application of microcontrollers are numerous. Starting from domestic applications such as in washing machines, TVs, airconditioners, microcontrollers are used in automobiles, process control industries , cell phones, electrical drives, robotics and in space applications.

Microcontroller Chips

Broad Classification of different microcontroller chips could be as follows:

- Embedded (Self -Contained) 8 - bit Microcontroller
- 16 to 32 Microcontrollers
- Digital Signal Processors

Features of Modern Microcontrollers

- Built-in Monitor Program
- Built-in Program Memory
- Interrupts
- Analog I/O
- Serial I/O
- Facility to Interface External Memory
- Timers

Internal Structure of a Microcontroller

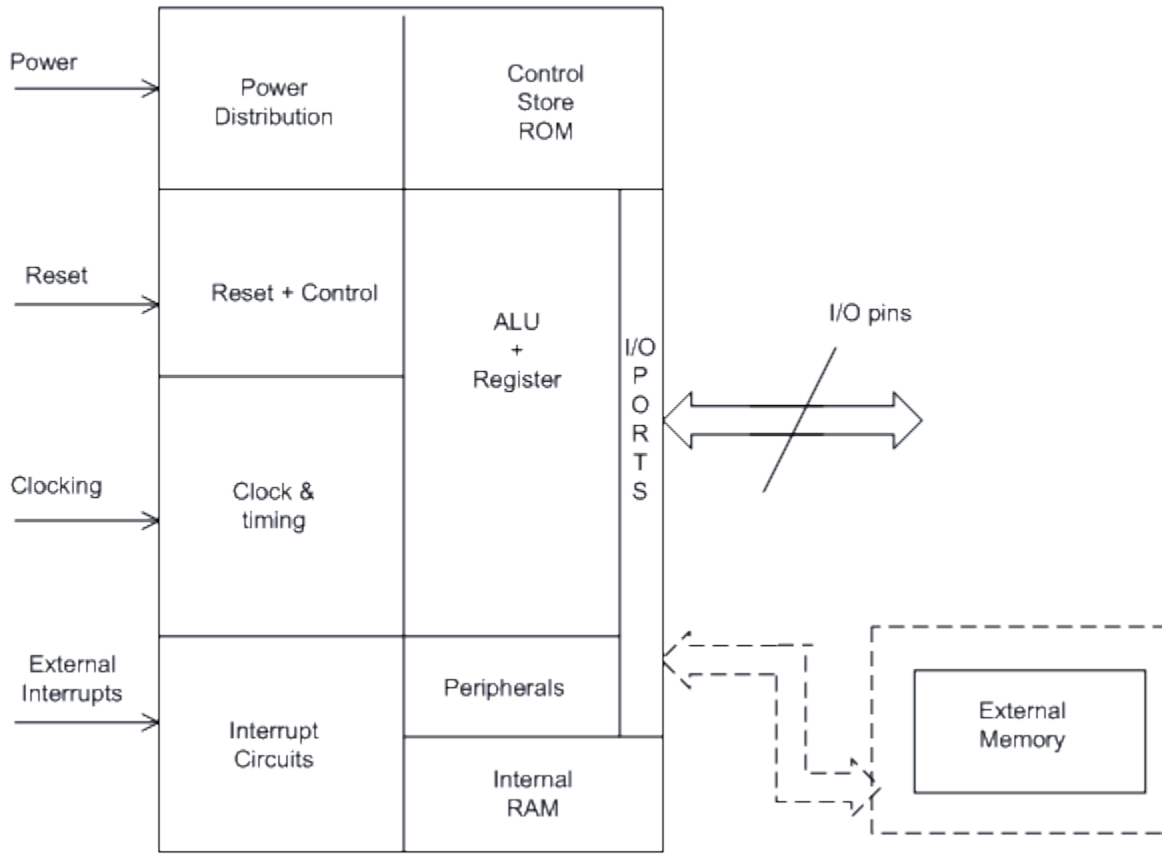

Fig. 4.1 Internal Structure of a Microcontroller

At times, a microcontroller can have external memory also (if there is no internal memory or extra memory interface is required). Early microcontrollers were manufactured using bipolar or NMOS technologies. Most modern microcontrollers are manufactured with CMOS technology, which leads to reduction in size and power loss. Current drawn by the IC is also reduced considerably from 10mA to a few micro Amperes in sleep mode(for a microcontroller running typically at a clock speed of 20MHz).

Harvard Architecture (Separate Program and Data Memory interfaces)

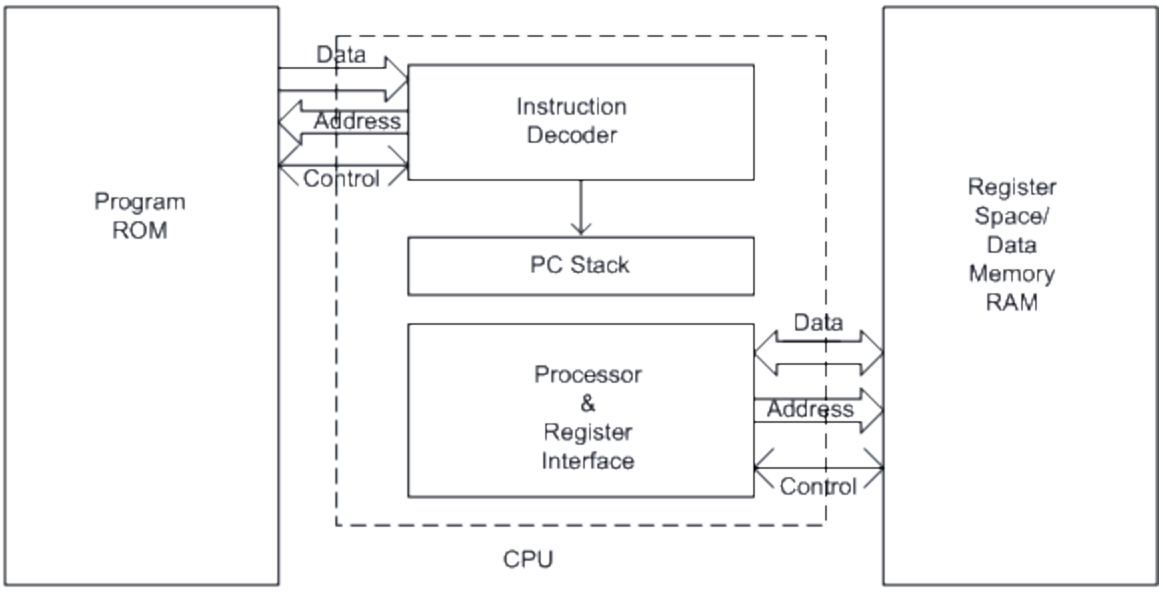

Fig. 4.2 Harvard Arcitecture

The same instruction (as shown under Princeton Architecture) would be executed as follows:

Cycle 1
- Complete previous instruction
- Read the "Move Data to Accumulator" instruction

Cycle 2
- Execute "Move Data to Accumulator" instruction
- Read next instruction

Hence each instruction is effectively executed in one instruction cycle, except for the ones that modify the content of the program counter. For example, the "jump" (or call) instructions takes 2 cycles. Thus, due to parallelism, Harvard architecture executes more instructions in a given time compared to Princeton Architecture.

Memory organization:

In the 8051, the memory is organized logically into program memory and data memory separately. The program memory is read-only type; the data memory is organized as read-write memory. Again, both program and data memories can be within the chip or outside.

Basic 8051 Architecture

The 8051 is an 8-bit microcontroller i.e. the data bus within and outside the chip is eight bits wide. The address bus of the 8051 is 16-bit wide. So it can address 64 KB of memory. The 8051 is a 40-pin chip as shown is figure below:

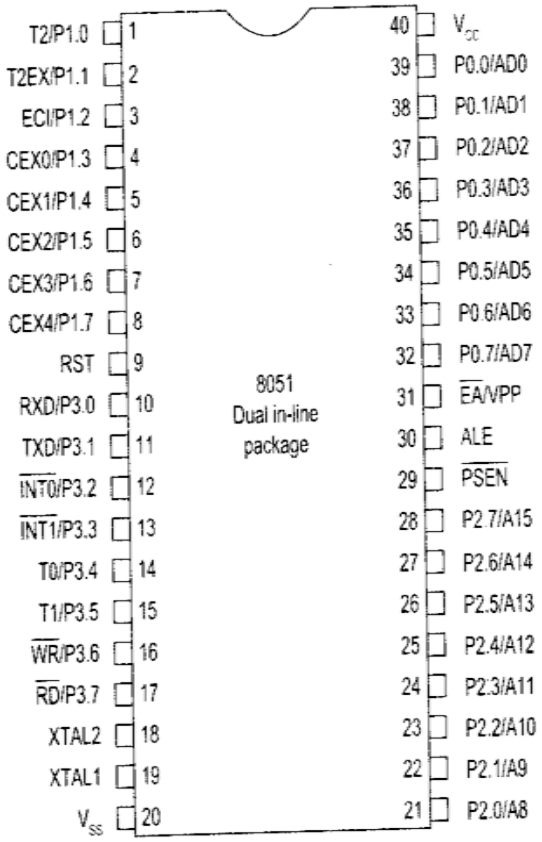

Fig 4.3 Pin details of 8051

8051 employs Harvard architecture. It has some peripherals such as 32 bit digital I/O, Timers and Serial I/O. The basic architecture of 8051 is given in fig 4.4.

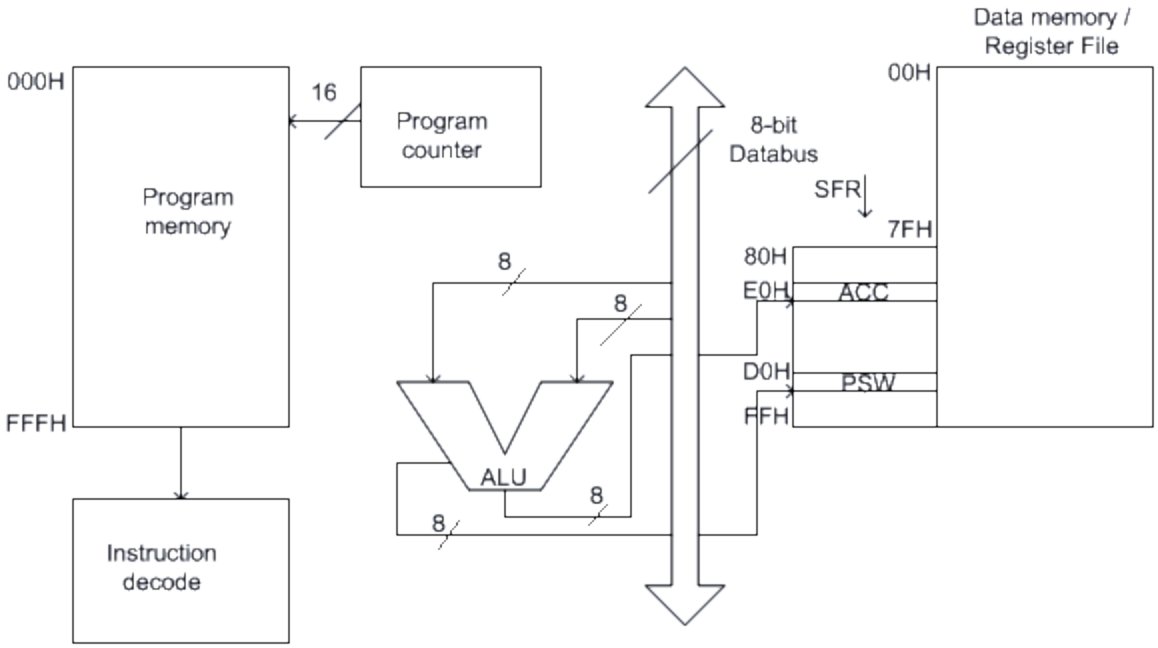

Fig 4.4 : Basic 8051 Architecture

Various features of 8051 microcontroller are given as follows.

- 8-bit CPU
- 16-bit Program Counter
- 8-bit Processor Status Word (PSW)
- 8-bit Stack Pointer
- Internal RAM of 128bytes
- Special Function Registers (SFRs) of 128 bytes
- 32 I/O pins arranged as four 8-bit ports (P0 - P3)
- Two 16-bit timer/counters : T0 and T1
- Two external and three internal vectored interrupts
- One full duplex serial I/O

8051 Clock and Instruction Cycle

In 8051, one instruction cycle consists of twelve (12) clock cycles. Instruction cycle is sometimes called as Machine cycle by some authors.

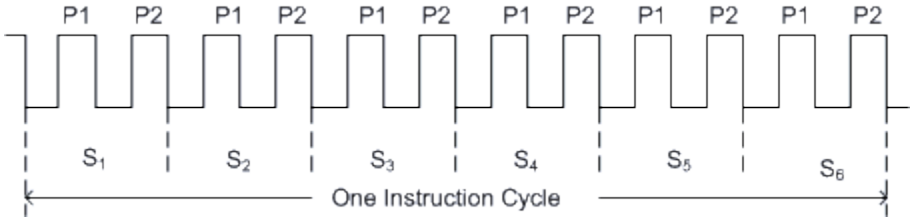

Fig 4.5 : Instruction cycle of 8051

In 8051, each instruction cycle has six states (S_1 - S_6). Each state has two pulses (P1 and P2)

128 bytes of Internal RAM Structure (lower address space)

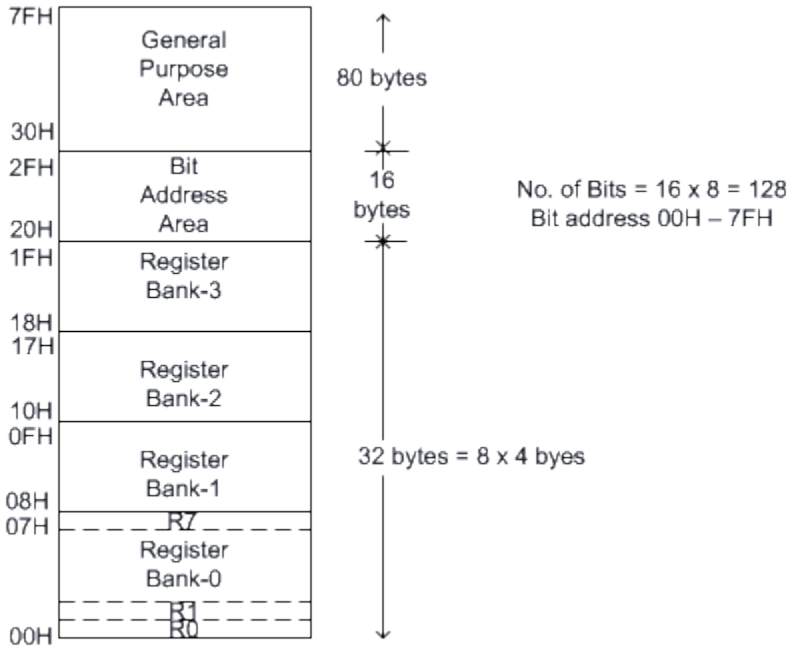

Fig 4.6: Internal RAM Structure

The lower 32 bytes are divided into 4 separate banks. Each register bank has 8 registers of one byte each. A register bank is selected depending upon two bank select bits in the PSW register. Next 16bytes are bit addressable. In total, 128bits (16X8) are available in bitaddressable area. Each bit can be accessed and modified by suitable instructions. The bit addresses are from 00H (LSB of the first byte in 20H) to 7FH (MSB of the last byte in 2FH). Remaining 80bytes of RAM are available for general purpose.

Internal Data Memory and Special Function Register (SFR) Map

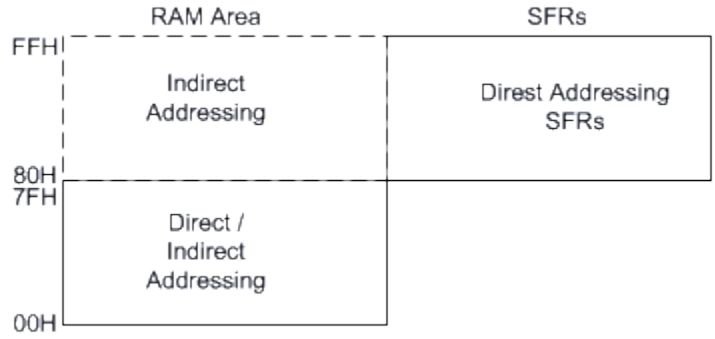

Fig 4.6 : Internal Data Memory Map

The special function registers (SFRs) are mapped in the upper 128 bytes of internal data memory address. Hence there is an address overlap between the upper 128 bytes of data RAM and SFRs. Please note that the upper 128 bytes of data RAM are present only in the 8052 family. The lower128 bytes of RAM (00H - 7FH) can be accessed both by direct or indirect addressing while the upper 128 bytes of RAM (80H - FFH) are accessed by indirect addressing.The SFRs (80H - FFH) are accessed by direct addressing only. This feature distinguishes the upper 128 bytes of memory from the SFRs, as shown in fig 4.6.

SFR Map

The set of Special Function Registers (SFRs) contains important registers such as Accumulator, Register B, I/O Port latch registers, Stack pointer, Data Pointer, Processor Status Word (PSW) and various control registers. Some of these registers are bit addressable (they are marked with a * in the diagram below). The detailed map of various registers is shown in the following figure.

Address

F8H								
F0H	B*							
E8H								
E0H	ACC*							
D8H								
D0H	PSW*							
C8H	(T2CON)*		(RCAP2L)	(RCAP2H)	(TL2)	(TH2)		
C0H								
B8H	IP*							
B0H	P3*							
A8H	IE*							
A0H	P2*							
98H	SCON*	SBUF						
90H	P1*							
88H	TCON*	TMOD	TL0	TL1	TH0	TH1		
80H	P0*	SP	DPL	DPH				PCON

Fig 4.7: SFR Map

It should be noted hat all registers appearing in the first column are bit addressable. The bit address of a bit in the register is calculated as follows.

Bit address of 'b' bit of register 'R' is

Address of register 'R' + b

where $0 \leq b \leq 7$

Processor Status Word (PSW) Address=D0H

CY	AC	F0	RS1	RS0	OV	-	P

Fi g 4.8: Processor Status Word

PSW register stores the important status conditions of the microcontroller. It also stores the bank select bits (RS1 & RS0) for register bank selection.

Interfacing External Memory

If external program/data memory are to be interfaced, they are interfaced in the following way.

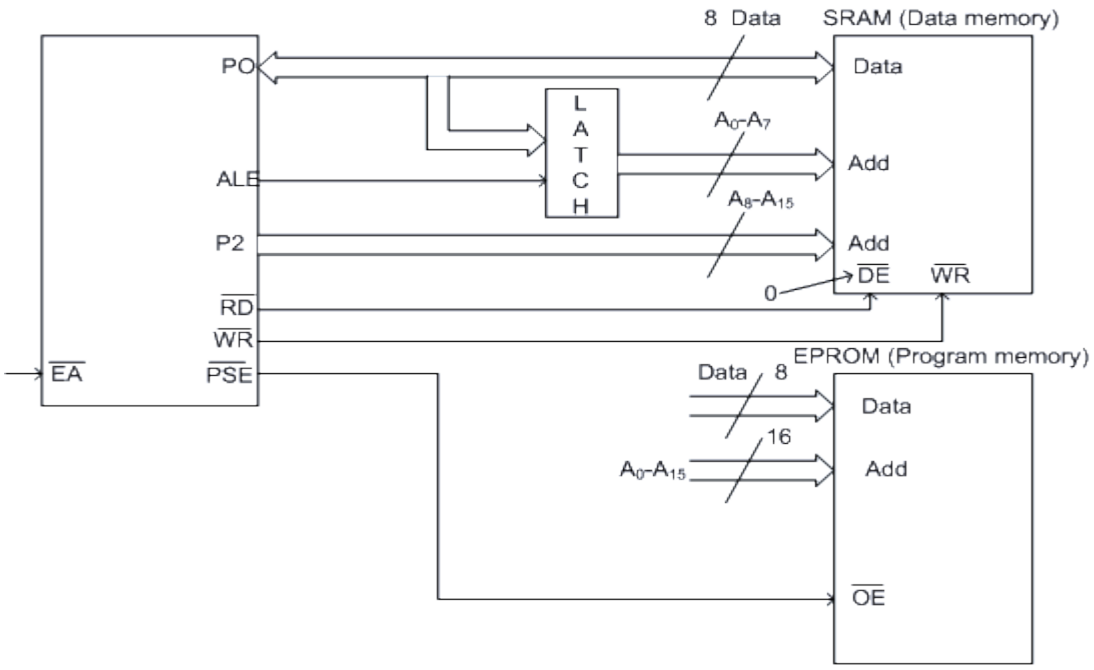

Fig 4.9: Circuit Diagram for Interfacing of External Memory

External program memory is fetched if either of the following two conditions are satisfied.

1. $\overline{EA}$(Enable Address) is low. The microcontroller by default starts searching for program from external program memory.
2. PC is higher than FFFH for 8051 or 1FFFH for 8052.

$\overline{PSEN}$ tells the outside world whether the external memory fetched is program memory or data memory. $\overline{EA}$ is user configurable. $\overline{PSEN}$ is processor controlled.

8051 Addressing Modes

8051 has four addressing modes.

1. Immediate Addressing :

Data is immediately available in the instruction.

For example -

ADD A, #77; Adds 77 (decimal) to A and stores in A

ADD A, #4DH; Adds 4D (hexadecimal) to A and stores in A

MOV DPTR, #1000H; Moves 1000 (hexadecimal) to data pointer

2. Bank Addressing or Register Addressing :

This way of addressing accesses the bytes in the current register bank. Data is available in the register specified in the instruction. The register bank is decided by 2 bits of Processor Status Word (PSW).

For example-

ADD A, R0; Adds content of R0 to A and stores in A

3.. Direct Addressing :
The address of the data is available in the instruction.
For example -

MOV A, 088H; Moves content of SFR TCON (address 088H)to A
4. Register Indirect Addressing :
The address of data is available in the R0 or R1 registers as specified in the instruction.
For example -

MOV A, @R0 moves content of address pointed by R0 to A
External Data Addressing :
Pointer used for external data addressing can be either R0/R1 (256 byte access) or DPTR (64kbyte access).
For example -

MOVX A, @R0; Moves content of 8-bit address pointed by R0 to A

MOVX A, @DPTR; Moves content of 16-bit address pointed by DPTR to A
External Code Addressing :
Sometimes we may want to store non-volatile data into the ROM e.g. look-up tables. Such data may require reading the code memory. This may be done as follows -

MOVC A, @A+DPTR; Moves content of address pointed by A+DPTR to A

MOVC A, @A+PC; Moves content of address pointed by A+PC to A

I/O Port Configuration

Each port of 8051 has bidirectional capability. Port 0 is called 'true bidirectional port' as it floats (tristated) when configured as input. Port-1, 2, 3 are called 'quasi bidirectional port'.
Port-0 Pin Structure
Port -0 has 8 pins (P0.0-P0.7).
The structure of a Port-0 pin is shown in fig 4.10.

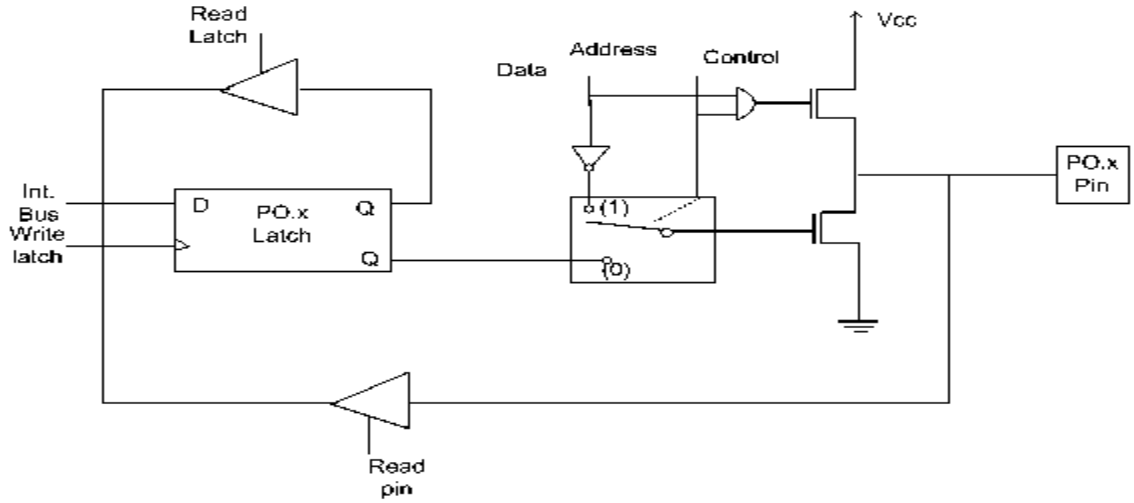

Fig 4.10: Port-0 Structure

Control bits TR1 and TF1 are used by Timer-0 (higher 8 bits) (TH0) in Mode-3 while TR0 and TF0 are available to Timer-0 lower 8 bits(TL0).

Interrupts

8051 provides 5 vectored interrupts. They are -

1. $\overline{INT0}$
2. TF0
3. $\overline{INT1}$
4. TF1
5. RI/TI

Out of these, $\overline{INT0}$ and $\overline{INT1}$ are external interrupts whereas Timer and Serial port interrupts are generated internally. The external interrupts could be negative edge triggered or low level triggered. All these interrupt, when activated, set the corresponding interrupt flags. Except for serial interrupt, the interrupt flags are cleared when the processor branches to the Interrupt Service Routine (ISR). The external interrupt flags are cleared on branching to Interrupt Service Routine (ISR), provided the interrupt is negative edge triggered. For low level triggered external interrupt as well as for serial interrupt, the corresponding flags have to be cleared by software by the programmer.

The schematic representation of the interrupts is as follows -

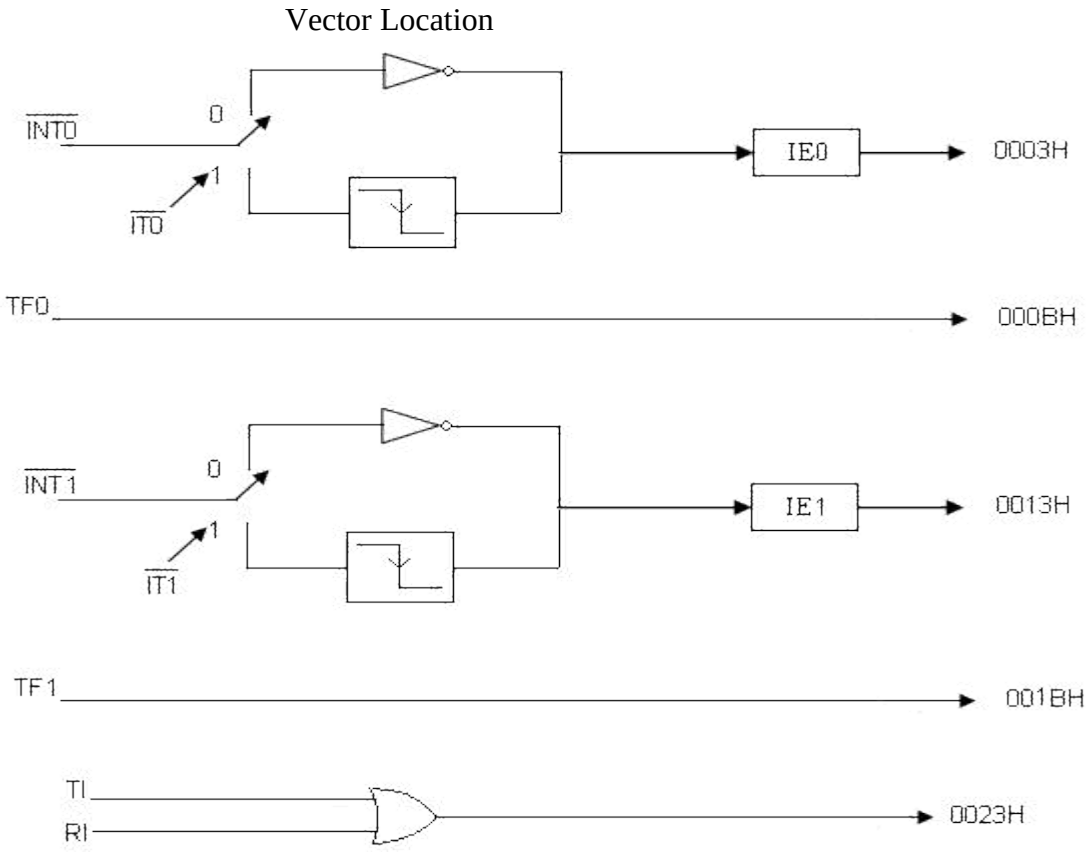

Fig 4.22 8051 Interrupt Details

Each of these interrupts can be individually enabled or disabled by 'setting' or 'clearing' the corresponding bit in the IE (Interrupt Enable Register) SFR. IE contains a global enable bit EA which enables/disables all interrupts at once.

Interrupt Enable register (IE): Address: A8H

7	6	5	4	3	2	1	0
EA	—	ET2	ES	ET1	EX1	ET0	EX0

EX0 ⟶ $\overline{INT0}$ interrupt (External) enable bit

ET0 ⟶ Timer-0 interrupt enable bit

EX1 ⟶ $\overline{INT1}$ interrupt (External) enable bit

ET1 ⟶ Timer-1 interrupt enable bit

ES ⟶ Serial port interrupt enable bit

ET2 ⟶ Timer-2 interrupt enable bit

EA ⟶ Enable/Disable all

Setting '1' ⟶ Enable the corresponding interrupt

Setting '0' ⟶ Disable the corresponding interrupt

Priority level structure:

Each interrupt source can be programmed to have one of the two priority levels by setting (high priority) or clearing (low priority) a bit in the IP (Interrupt Priority) Register . A low priority interrupt can itself be interrupted by a high priority interrupt, but not by another low priority interrupt. If two interrupts of different priority levels are received simultaneously, the request of higher priority level is served. If the requests of the same priority level are received simultaneously, an internal polling sequence determines which request is to be serviced. Thus, within each priority level, there is a second priority level determined by the polling sequence, as follows.

Source	Priority level
IE0	Highest
TF0	↓
IE1	
TF1	
RI+TI	Lowest

Interrupt Priority register (IP)

7	6	5	4	3	2	1	0
—	—	PT2	PS	PT1	PX1	PT0	PX0

'0' ⟶ low priority

'1' ⟶ high priority

Interrupt handling:

The interrupt flags are sampled at P2 of S5 of every instruction cycle (Note that every instruction cycle has six states each consisting of P1 and P2 pulses). The samples are polled

during the next machine cycle (or instruction cycle). If one of the flags was set at S5P2 of the preceding instruction cycle, the polling detects it and the interrupt process generates a long call (LCALL) to the appropriate vector location of the interrupt. The LCALL is generated provided this hardware generated LCALL is not blocked by any one of the following conditions.

1. An interrupt of equal or higher priority level is already in progress.
2. The current polling cycle is not the final cycle in the execution of the instruction in progress.
3. The instruction in progress is RETI or any write to IE or IP registers.

When an interrupt comes and the program is directed to the interrupt vector address, the Program Counter (PC) value of the interrupted program is stored (pushed) on the stack. The required Interrupt Service Routine (ISR) is executed. At the end of the ISR, the instruction RETI returns the value of the PC from the stack and the originally interrupted program is resumed.

Reset is a non-maskable interrupt. A reset is accomplished by holding the RST pin high for at least two machine cycles. On resetting the program starts from 0000H and some flags are modified as follows -

Register	**Value(Hex) on Reset**
PC	0000H
DPTR	0000H
A	00H
B	00H
SP	07H
PSW	00H
Ports P0-3 Latches	FFH
IP	XXX 00000 b
IE	0 XX 00000 b
TCON	00H
TMOD	00H
TH0	00H
TL0	00H
TH1	00H
TL1	00H
SCON	00H
SBUF	XX H
PCON	0 XXXX XXX b

The schematic diagram of the detection and processing of interrupts is given as follows.

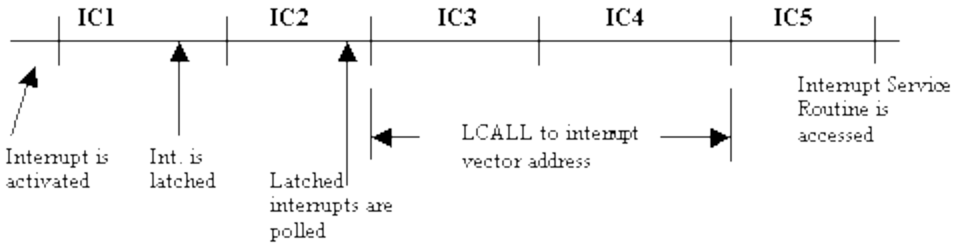

Fig 4.23 Interrupt Handling in 8051

It should be noted that the interrupt which is blocked due to the three conditions mentioned before is not remembered unless the flag that generated interrupt is not still active when the above blocking conditions are removed, i.e. ,every polling cycle is new.

Jump and Call Instructions

There are 3 types of jump instructions. They are:-

1. Relative Jump
2. Short Absolute Jump
3. Long Absolute Jump

Relative Jump

Jump that replaces the PC (program counter) content with a new address that is greater than (the address following the jump instruction by 127 or less) or less than (the address following the jump by 128 or less) is called a relative jump. Schematically, the relative jump can be shown as follows: -

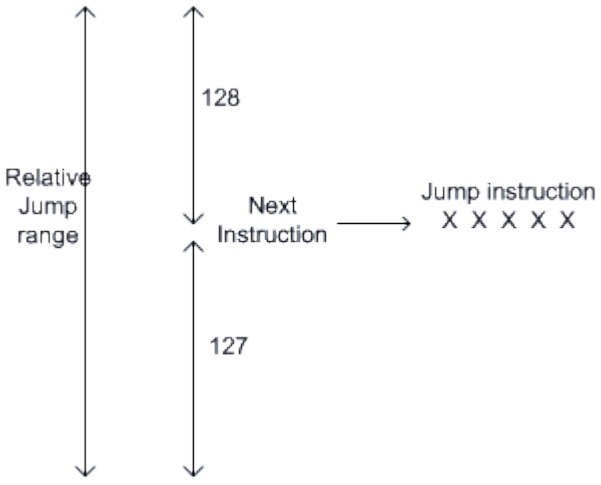

Fig 4.24 Relative Jump

The advantages of the relative jump are as follows:-

1. Only 1 byte of jump address needs to be specified in the 2's complement form, ie. For jumping ahead, the range is 0 to 127 and for jumping back, the range is -

1 to -128.
2. Specifying only one byte reduces the size of the instruction and speeds up program execution.
3. The program with relative jumps can be relocated without reassembling to generate absolute jump addresses.

Disadvantages of the absolute jump: -

1. Short jump range (-128 to 127 from the instruction following the jump instruction)

Instructions that use Relative Jump
SJMP <relative address>

(The remaining relative jumps are conditional jumps)

JC <relative address>
JNC <relative address>
JB bit, <relative address>
JNB bit, <relative address>
JBC bit, <relative address>
CJNE <destination byte>, <source byte>, <relative address>
DJNZ <byte>, <relative address>
JZ <relative address>
JNZ <relative address>

Short Absolute Jump

In this case only 11bits of the absolute jump address are needed. The absolute jump address is calculated in the following manner.
In 8051, 64 kbyte of program memory space is divided into 32 pages of 2 kbyte each. The hexadecimal addresses of the pages are given as follows:-

Page (Hex)	Address (Hex)
00	0000 - 07FF
01	0800 - 0FFF
02	1000 - 17FF
03	1800 - 1FFF
.	.
1E	F000 - F7FF
1F	F800 - FFFF

It can be seen that the upper 5bits of the program counter(PC) hold the page number and the lower 11bits of the PC hold the address within that page. Thus, an absolute address is formed by taking page numbers of the instruction (from the program counter) following the jump and attaching the specified 11bits to it to form the 16-bit address.
Advantage: The instruction length becomes 2 bytes.
However, difficulty is encountered when the next instruction following the jump instruction begins from a fresh page (at X000H or at X800H). This does not give any

problem for the forward jump, but results in an error for the backward jump. In such a case the assembler prompts the user to relocate the program suitably.
Example of short absolute jump: -
ACALL <address 11>
AJMP <address 11>

Long Absolute Jump/Call

Applications that need to access the entire program memory from 0000H to FFFFH use long absolute jump. Since the absolute address has to be specified in the op-code, the instruction length is 3 bytes (except for JMP @ A+DPTR). This jump is not re-locatable.
Example: -

LCALL <address 16>
LJMP <address 16>
JMP @A+DPTR

Serial Interface

The serial port of 8051 is full duplex, i.e., it can transmit and receive simultaneously.

The register SBUF is used to hold the data. The special function register SBUF is physically two registers. One is, write-only and is used to hold data to be transmitted out of the 8051 via TXD. The other is, read-only and holds the received data from external sources via RXD. Both mutually exclusive registers have the same address 099H.

Serial Port Control Register (SCON)

Register SCON controls serial data communication.
Address: 098H (Bit addressable)

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Mode select bits

SM0	SM1	Mode
0	0	Mode 0
0	1	Mode 1
1	0	Mode 2
1	1	Mode 3

SM2: multi processor communication bit
REN: Receive enable bit
TB8: Transmitted bit 8 (Normally we have 0-7 bits transmitted/received)
RB8: Received bit 8
TI: Transmit interrupt flag
RI: Receive interrupt flag

Power Mode control Register

Register PCON controls processor powerdown, sleep modes and serial data bandrate. Only one bit of PCON is used with respect to serial communication. The seventh bit (b7)(SMOD) is used to generate the baud rate of serial communication.

Address: 87H

b7							b0
SMOD	—	—	—	GF1	GF0	PD	IDL

SMOD: Serial baud rate modify bit
GF1: General purpose user flag bit 1
GF0: General purpose user flag bit 0
PD: Power down bit
IDL: Idle mode bit

Data Transmission

Transmission of serial data begins at any time when data is written to SBUF. Pin P3.1 (Alternate function bit TXD) is used to transmit data to the serial data network. TI is set to 1 when data has been transmitted. This signifies that SBUF is empty so that another byte can be sent.

Data Reception

Reception of serial data begins if the receive enable bit is set to 1 for all modes. Pin P3.0 (Alternate function bit RXD) is used to receive data from the serial data network. Receive interrupt flag, RI, is set after the data has been received in all modes. The data gets stored in SBUF register from where it can be read.

Serial Data Transmission Modes:

Mode-0**:** In this mode, the serial port works like a shift register and the data transmission works synchronously with a clock frequency of f_{osc} /12. Serial data is received and transmitted through RXD. 8 bits are transmitted/ received aty a time. Pin TXD outputs the shift clock pulses of frequency f_{osc} /12, which is connected to the external circuitry for synchronization. The shift frequency or baud rate is always 1/12 of the oscillator frequency.

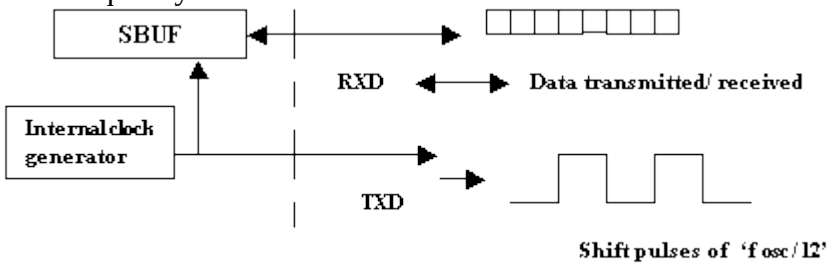

Fig 4.25 Data transmission/reception in Mode-0

Mode-1 (standard UART mode) **:**

In mode-1, the serial port functions as a standard Universal Asynchronous Receiver Transmitter (UART) mode. 10 bits are transmitted through TXD or received through RXD. The 10 bits consist of one start bit (which is usually '0'), 8 data bits (LSB is sent first/received first), and a stop bit (which is usually '1'). Once received, the stop bit goes into RB8 in the special function register SCON. The baud rate is variable.

The following figure shows the way the bits are transmitted/ received.

The schematic diagram for 'Power down' mode and 'Idle' mode is given as follows:

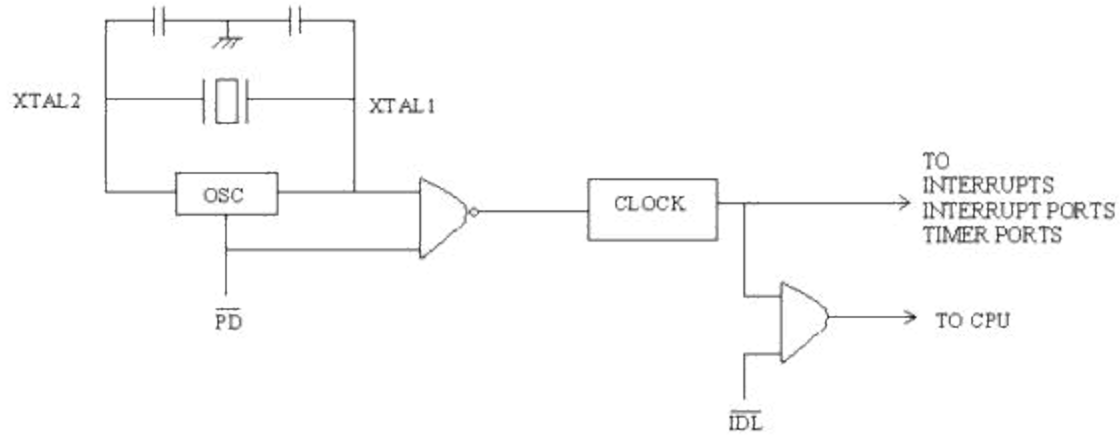

Fig 4.28 Schematic diagram for Power Down and Idle mode implementation

Idle Mode

Idle mode is entered by setting IDL bit to 1 (i.e., $\overline{IDL}$ =0). The clock signal is gated off to CPU, but not to the interrupt, timer and serial port functions. The CPU status is preserved entirely. SP, PC, PSW, Accumulator and other registers maintain their data during IDLE mode. The port pins hold their logical states they had at the time Idle was initiated. ALE and $\overline{PSEN}$ are held at logic high levels.

Ways to exit Idle Mode:

1. Activation of any enabled interrupt will clear PCON.0 bit and hence the Idle Mode is exited. The program goes to the Interrupt Service Routine (ISR). After RETI is executed at the end of the ISR, the next instruction will start from the one following the instruction that enabled Idle Mode.
2. A hardware reset exits the idle mode. The CPU starts from the instruction following the instruction that invoked the 'Idle' mode.

Power Down Mode:

The Power down Mode is entered by setting the PD bit to 1. The internal clock to the entire microcontroller is stopped (frozen). However, the program is not dead. The Power down Mode is exited (PCON.1 is cleared to 0) by Hardware Reset only. The CPU starts from the next instruction where the Power down Mode was invoked. Port values are not changed/ overwritten in power down mode. V_{cc} can be reduced to as low as 2V in PowerDown mode. However, V_{cc} has to be restored to normal value before PowerDown mode is exited.

8051 Instructions

8051 has about 111 instructions. These can be grouped into the following categories

1. Arithmetic Instructions
2. Logical Instructions
3. Data Transfer instructions
4. Boolean Variable Instructions
5. Program Branching Instructions

The following nomenclatures for register, data, address and variables are used while write instructions.
A: Accumulator

B: "B" register

C: Carry bit

Rn: Register R0 - R7 of the currently selected register bank

Direct: 8-bit internal direct address for data. The data could be in lower 128bytes of RAM (00 - 7FH) or it could be in the special function register (80 - FFH).

@Ri: 8-bit external or internal RAM address available in register R0 or R1. This is used for indirect addressing mode.

#data8: Immediate 8-bit data available in the instruction.

#data16: Immediate 16-bit data available in the instruction.

Addr11: 11-bit destination address for short absolute jump. Used by instructions AJMP & ACALL. Jump range is 2 kbyte (one page).

Addr16: 16-bit destination address for long call or long jump.

Rel: 2's complement 8-bit offset (one - byte) used for short jump (SJMP) and all conditional jumps.

bit: Directly addressed bit in internal RAM or SFR

Arithmetic Instructions

Mnemonics	Description	Bytes	Instruction Cycles
ADD A, Rn	A ⟵A + Rn	1	1
ADD A, direct	A ⟵A + (direct)	2	1
ADD A, @Ri	A ⟵A + @Ri	1	1
ADD A, #data	A ⟵A + data	2	1
ADDC A, Rn	A ⟵A + Rn + C	1	1
ADDC A, direct	A ⟵ A + (direct) + C	2	1
ADDC A, @Ri	A ⟵A + @Ri + C	1	1
ADDC A, #data	A ⟵A + data + C	2	1
DA A	Decimal adjust accumulator	1	1
DIV AB	Divide A by B A ⟵quotient B ⟵remainder	1	4
DEC A	A ⟵A -1	1	1
DEC Rn	Rn ⟵Rn - 1	1	1
DEC direct	(direct) ⟵(direct) - 1	2	1

DEC @Ri	@Ri ⟵@Ri - 1	1	1
INC A	A ⟵A+1	1	1
INC Rn	Rn ⟵Rn + 1	1	1
INC direct	(direct) ⟵(direct) + 1	2	1
INC @Ri	@Ri ⟵@Ri +1	1	1
INC DPTR	DPTR ⟵DPTR +1	1	2
MUL AB	Multiply A by B A ⟵low byte (A*B) B ⟵high byte (A* B)	1	4
SUBB A, Rn	A ⟵A - Rn - C	1	1
SUBB A, direct	A ⟵A - (direct) - C	2	1
SUBB A, @Ri	A ⟵A - @Ri - C	1	1
SUBB A, #data	A ⟵A - data - C	2	1

Logical Instructions

Mnemonics	Description	Bytes	Instruction Cycles
ANL A, Rn	A ⟵A AND Rn	1	1
ANL A, direct	A ⟵A AND (direct)	2	1
ANL A, @Ri	A ⟵A AND @Ri	1	1
ANL A, #data	A ⟵A AND data	2	1
ANL direct, A	(direct) ⟵(direct) AND A	2	1
ANL direct, #data	(direct) ⟵(direct) AND data	3	2
CLR A	A⟵ 00H	1	1
CPL A	A⟵A	1	1
ORL A, Rn	A ⟵A OR Rn	1	1
ORL A, direct	A ⟵A OR (direct)	1	1
ORL A, @Ri	A ⟵A OR @Ri	2	1
ORL A, #data	A ⟵A OR data	1	1
ORL direct, A	(direct) ⟵(direct) OR A	2	1
ORL direct, #data	(direct) ⟵(direct) OR data	3	2
RL A	Rotate accumulator left	1	1
RLC A	Rotate accumulator left through carry	1	1
RR A	Rotate accumulator right	1	1
RRC A	Rotate accumulator right through carry	1	1
SWAP A	Swap nibbles within Acumulator	1	1
XRL A, Rn	A ⟵A EXOR Rn	1	1
XRL A, direct	A ⟵A EXOR (direct)	1	1
XRL A, @Ri	A ⟵A EXOR @Ri	2	1
XRL A, #data	A ⟵A EXOR data	1	1

XRL direct, A	(direct) ⟵(direct) EXOR A	2	1
XRL direct, #data	(direct) ⟵(direct) EXOR data	3	2

Data Transfer Instructions

Mnemonics	Description	Bytes	Instruction Cycles
MOV A, Rn	A ⟵Rn	1	1
MOV A, direct	A ⟵(direct)	2	1
MOV A, @Ri	A ⟵@Ri	1	1
MOV A, #data	A ⟵data	2	1
MOV Rn, A	Rn ⟵A	1	1
MOV Rn, direct	Rn ⟵(direct)	2	2
MOV Rn, #data	Rn ⟵data	2	1
MOV direct, A	(direct) ⟵A	2	1
MOV direct, Rn	(direct) ⟵Rn	2	2
MOV direct1, direct2	(direct1) ⟵(direct2)	3	2
MOV direct, @Ri	(direct)⟵ @Ri	2	2
MOV direct, #data	(direct) ⟵#data	3	2
MOV @Ri, A	@Ri ⟵A	1	1
MOV @Ri, direct	@Ri ⟵(direct)	2	2
MOV @Ri, #data	@Ri ⟵data	2	1
MOV DPTR, #data16	DPTR ⟵data16	3	2
MOVC A, @A+DPTR	A ⟵Code byte pointed by A + DPTR	1	2
MOVC A, @A+PC	A ⟵Code byte pointed by A + PC	1	2
MOVC A, @Ri	A ⟵Code byte pointed by Ri 8-bit address)	1	2
MOVX A, @DPTR	A ⟵External data pointed by DPTR	1	2
MOVX @Ri, A	@Ri ⟵A (External data - 8bit address)	1	2
MOVX @DPTR, A	@DPTR ⟵A(External data - 16bit address)	1	2
PUSH direct	(SP) ⟵(direct)	2	2

POP direct	(direct) ⟵(SP)	2	2
XCH Rn	Exchange A with Rn	1	1
XCH direct	Exchange A with direct byte	2	1
XCH @Ri	Exchange A with indirect RAM	1	1
XCHD A, @Ri	Exchange least significant nibble of A with that of indirect RAM	1	1

Boolean Variable Instructions

Mnemonics	Description	Bytes	Instruction Cycles
CLR C	C-bit ⟵0	1	1
CLR bit	bit ⟵0	2	1
SET C	C ⟵1	1	1
SET bit	bit ⟵1	2	1
CPL C	C⟵ $\overline{\text{C-bit}}$	1	1
CPL bit	bit ⟵ $\overline{\text{bit}}$	2	1
ANL C, /bit	C ⟵C . $\overline{\text{bit}}$	2	1
ANL C, bit	C ⟵C. bit	2	1
ORL C, /bit	C⟵ C + $\overline{\text{bit}}$	2	1
ORL C, bit	C ⟵C + bit	2	1
MOV C, bit	C⟵ bit	2	1
MOV bit, C	bit ⟵C	2	2

Program Branching Instructions

Mnemonics	Description	Bytes	Instruction Cycles
ACALL addr11	PC + 2 ⟶(SP) ; addr 11⟶ PC	2	2
AJMP addr11	Addr11 ⟶PC	2	2
CJNE A, direct, rel	Compare with A, jump (PC + rel) if not equal	3	2
CJNE A, #data, rel	Compare with A, jump (PC + rel) if not equal	3	2
CJNE Rn, #data, rel	Compare with Rn, jump (PC + rel) if not equal	3	2
CJNE @Ri, #data, rel	Compare with @Ri A, jump (PC + rel) if not equal	3	2
DJNZ Rn, rel	Decrement Rn, jump if not zero	2	2
DJNZ direct, rel	Decrement (direct), jump if not zero	3	2
JC rel	Jump (PC + rel) if C bit = 1	2	2
JNC rel	Jump (PC + rel) if C bit = 0	2	2
JB bit, rel	Jump (PC + rel) if bit = 1	3	2
JNB bit, rel	Jump (PC + rel) if bit = 0	3	2
JBC bit, rel	Jump (PC + rel) if bit = 1	3	2
JMP @A+DPTR	A+DPTR ⟶PC	1	2

QUESTIONS:

1. Differentiate between microprocessors and microcontrollers.
2. What is a special function register?
3. Which port of 8051 is used as address/data bus?
4. What is function of RS1 and RS0 bits in the PSW of the 8051?
5. What is the address range of the bit-addressable memory of the 8051?
6. Write note on memory organization in the 8051.
7. Explain the stack operation in the 8051.
8. Where are the registers R0 – R7 located in the 8051?
9. Give one example each for one-byte, two-byte and three-byte instructions of the 8051.
10. When the instruction DJNZ useful?
11. Write a program to multiply two 8-bit numbers in the internal RAM and store the result in the external RAM.
12. Write a program to shift a 4-digit BCD number left by one digit. Assume that the data is stored in 30H and 31H.
13. Write a program to reverse the bits in a byte.
14. Write a program to find the biggest number in a block of data stored in the memory locations 70H – 7FH.
15. Write a program to generate a square wave of 10 KHz on the LSB of port 1 i.e. P1.0, using a timer.