1. Introduction to 8051 Microcontroller

Microcontroller is a single chip microcomputer which consists of CPU, Memory, I/O ports, timers and other peripherals. The difference between microprocessor and microcontroller is microprocessor is a single integrated CPU whereas microcontroller is single chip microcomputer. The world leaders of manufacturing of microprocessor and microcontroller are Intel, Motorola, IBM, Cyrix etc. Here we have to focus on microcontroller 8051.

In 1981 Intel Corporation introduced an 8 bit microcontroller called 8051.this microcontroller had 128 bytes of RAM, 4K bytes of on-chip ROM, two timers, one serial port and four ports (each 8bit wide) all on a single chip. It is an 8 bit processor means it can process 8 bit of data at a time. It has total of four I/O ports, each 8 bit wide.

Features of 8051

Feature	Quantity
ROM	4K bytes
RAM	128bytes
Timer	2
I/O pins	32
Serial Port	1
Interrupt sources	6

2. Architecture of 8051

Fig 4.1 shows a simplified architecture for the internal Hardware. Fig 4.2 shows an overview of the internal hardware architecture of the 8051/8031 microcontrollers.

The CPU has the controlled and sequencing logic circuits with signals as in a microprocessor.

The MCU has, besides the CPU, ROM, Interrupt control circuit, internal timing devices (timers T0, T1), serial interface (SI), RAM and special function registers (SFRs). It has four ports P0, P1, P2 and P3 as shown in Fig. 4.1. The overview block diagram of 8051 is depicted in Fig.4.2.

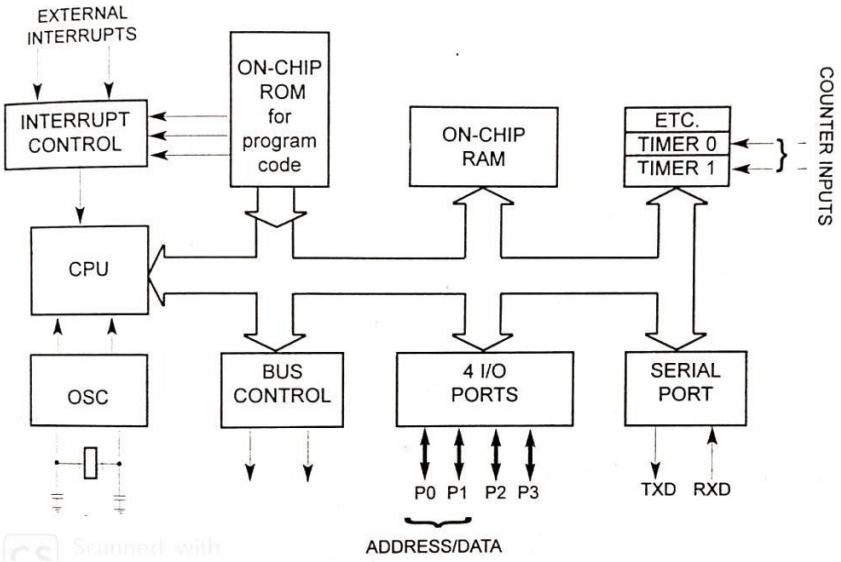


Fig. 4.1 Simplified architecture of 8051

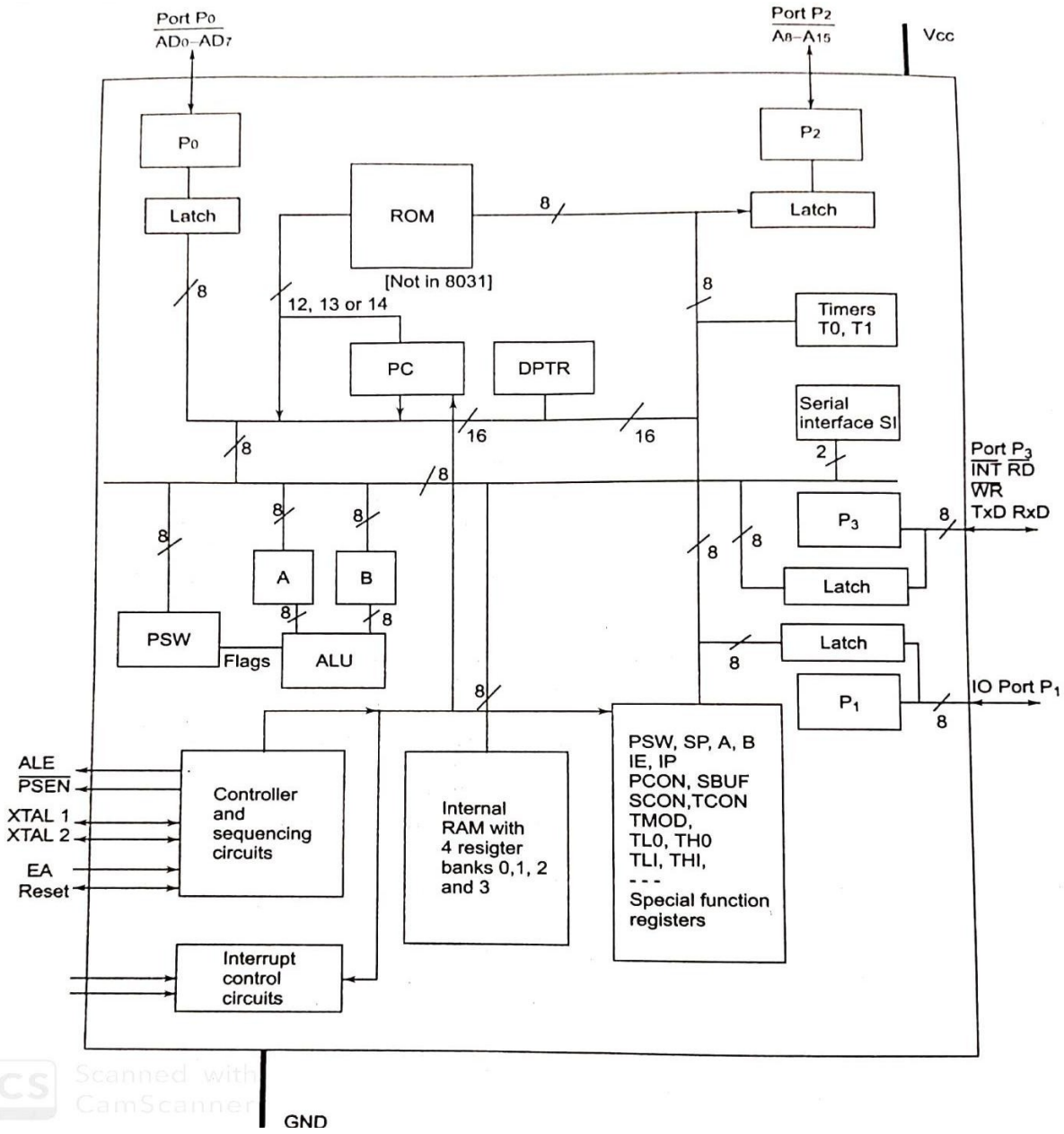


Fig.4.2 Overview (Block diagram) of 8051

Description of Sub units in the hardware architecture and meaning of the symbols

PC- Program Counter

A 16 bit register to hold the program memory address of the instruction being currently fetched. Increments continuously to point to the next instruction, unless there is change in the program flow path.

DPTR- Data Pointer register

A 16-bit register to hold the external data memory address of the data being currently fetched or to be fetched in indirect addressing mode.

A-Accumulator

An 8-bit register to save an operand for an ALU or data transfer operation and is also used to accumulate result after an ALU operation.

B- B register

An 8-bit register to save a second operand for the ALU and also accumulate the result after ALU operation for multiplication or division.

ALU- Arithmetic logic unit

A unit to perform an arithmetic and logical operation at an instance as per the instruction to be executed and give result.

PSW- Processor Status Word

A register to save the bits of different flags.

P0- Port P0

An 8-bit port for the I/Os in a single chip mode and for the data bus-cum- lower order address in the expanded mode.

P2- Port2

An 8-bit port for the I/Os in a single chip mode and for the higher order address in the expanded mode

P1- Port1

An 8-bit port for the I/Os in a single chip mode and a few device operations related bits in certain 8051 family variants in the expanded mode.

P3- Port3

An 8-bit port for the I/Os in a single chip mode and the serial interface (SI) bits , timer T0 and T1 inputs, Interrupts INT0 and INT1 inputs , $\overline{RD}$ and $\overline{WR}$ for the memory read-write in the expanded mode.

SI- Serial Interface Device

Serial device for full duplex UART serial I/O operations through the set of two pins of P3, RxD and TxD and for the half duplex synchronous communication of the bits through the same set of pins, DATA and CLOCK.

T0 and T1- Timers T0 and T1

Timing devices in 8051 family using four registers TH1, TH0, TL1, and TL0.

SFRs- Special Function Registers

All registers the SP, PSW, A, B, IE, IP, SCON, TCON, SMOD, SBUF, PCON, , TL0, TH0, TL1, TH1 are called SFRs

ROM- Read only Program memory

Masked ROM EPROM or flash EEPROM of 4kB in 8051 classic family.

Internal RAM- Internal Random Access Memory

For read and write the 128 B memory is indirectly and directly addressable in address space.

Register banks- Four set of registers

Four register banks each of 8 registers and these are also part of the internal RAM.

XTAL1 and XTAL2 – Pins to the Crystal

Pins to the crystal in the oscillator circuit, usually 12 MHz

$\overline{EA}$ **- External Enable**

To enable use of external memory addresses to external ROM.

RST- Reset Pin

Reset circuit input and also reset few output cycles to the external peripheral devices to let processor reset and synchronize with devices.

$\overline{INT}$ **0 and** $\overline{INT}$ **1- Interrupt pins**

Active low two external interrupts.

VCC and GND- Voltage supply pi and ground pin

For 5 V supply and ground connections respectively.

$\overline{PSEN}$ **- Program Store Enable**

Active low when reading the external program memory bytes

$\overline{RD}$ **-Read**

Active low when reading the byte from external data memory.

$\overline{WR}$ **- Write**

Active low when writing the byte to external data memory

3. Pin Configuration

Fig 4.3 shows 40 pin signals in an 8051 series microcontroller. It shows the I/O pins, P0.0 to P0.7, P1.0 to P1.7, P2.0 to P2.7 and P3.0 to P3.7. It shows other remaining 8 pins, V_{DD}, V_{SS}, XTAL1 and XTAL2, RST, ALE, $\overline{EA}$ and $\overline{PSEN}$.

Vcc - Pin 40 provides supply voltage to the chip. The voltage source is +5V

GND- Pin 20 is the ground.

XTAL1 and XTAL2- 8051 has an on-chip oscillator but requires an external clock to run it. Most upon a quartz crystal oscillator is connected to inputs XTAL1 (pin 19 and XTAL@ (pin-18) The quartz crystal oscillator connected also needs two capacitors of 30 pF. If frequency source other than crystal oscillator such as TTL oscillator will be connected to XTAL1 and XTAL2 is left unconnected.

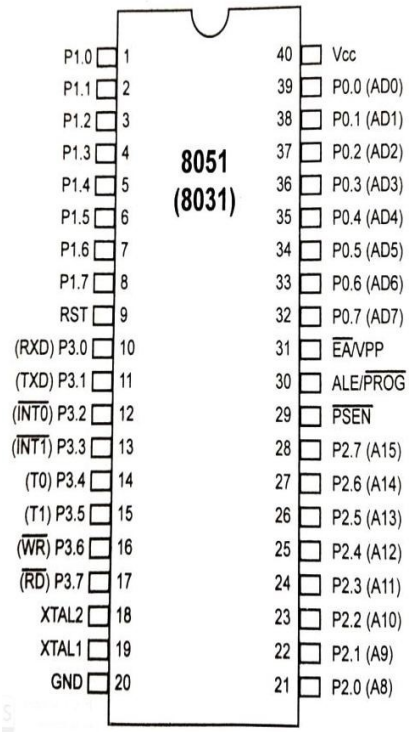


Fig. 4.3 8051 Pin diagram

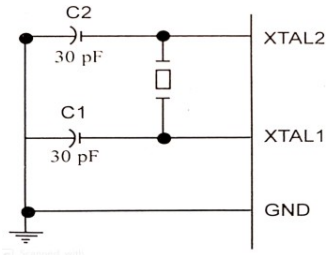


Fig. 4.4 XTAL connection to 8051

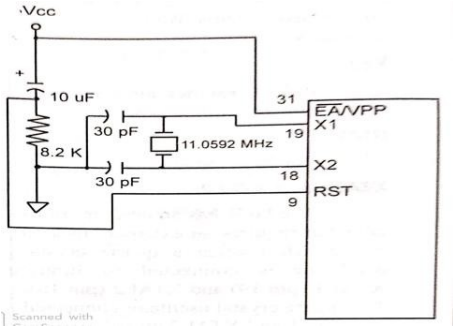


Fig. 4.5 Power-On RESET circuit

RST (I/P)- Pin 9 is the RESET pin and is active high (normally low). Upon applying high pulse to this pin the microcontroller will reset and terminate all activities. This often referred to as power on reset. Once it is activated the contents of all registers become zero except the content of SP which is 07H.

$\overline{\text{EA}}$ **(External Access)** - This pin is connected to V_{CC} for those have on-chip ROM otherwise it is grounded incase 8031 and 8032. Because in case of 8031 and 8032 there is no on-chip ROM.

$\overline{\text{PSEN}}$ **(o/p) (Program Store Enable)-** In case of 8031 based system in which an external ROM holds the program code . To read the code this pin is connected to $\overline{\text{OE}}$ pin of ROM chip.

AlE (o/p) (address Latch enable)- When 8051is connected to external memory, both address and data are transferred through port 0 pins. ALE signal is active high used to demultiplex address/data bus.

P0, P1, P2 and P3 are explained in port section.

4. Memory Organization

The 8051 micro controller has a total of 128 bytes of RAM. The 128 bytes of RAM inside the 8051 are assigned addresses 00H to 7FH and divided into three different groups as follows.

1. A total of 32 bytes from location s 00H to 1FH are set aside for register banks and the stacks.
2. A total of 16 bytes from locations 20H to 2FH are set aside for bit addressable read/write memory.
3. A total of 80 bytes from locations 30H to 7FH are used for read and write storage, or what is normally called a scratch pad. These 80 locations of RAM are widely used for the purpose of storing data and parameters by 8051 programmers.

Register banks in the 8051

As mentioned, a total of 32 bytes of RAM are set aside for the register banks an stack. These 32 bytes are divided into 4 banks of registers in which each bank has 8 registers, R0-R7. RAM locations from 0 to 7 are set aside for bank 0 of R0-R7 where R0 is RAM location 0 , R2 is location 2 and so on. The second bank of registers R0-R7 start RAM location 08 and goes to location 1FH. The third bank of R0-R7 starts at memory location 10 H and goes to location 17H. finally RAM location 18H to 1FH are set aside for the fourth bank of R0-R7. The following shows how 32 bytes are allocated into 4 banks.

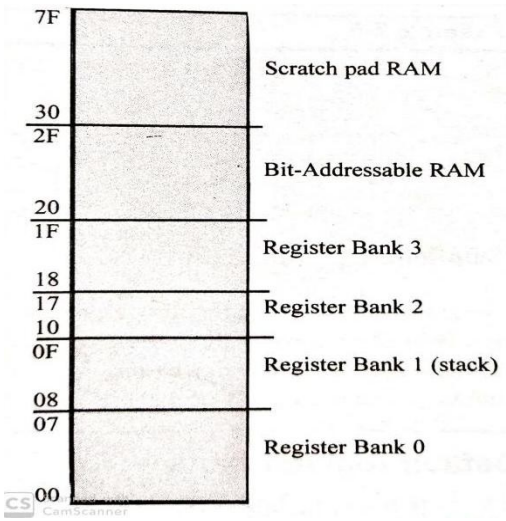


Fig. 4.6 RAM allocation in the 8051

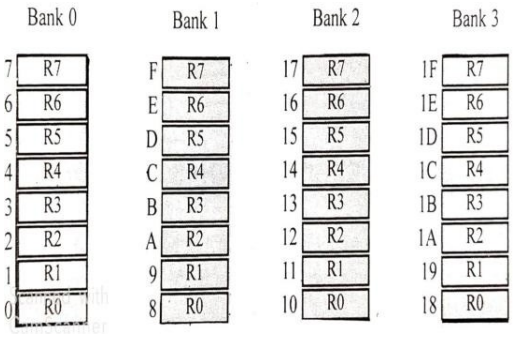


Fig. 4.7 RAM Allocation in the 8051

External Program Memory

Fig.4.8 shows a layout of the external code memory addresses in the classic 8051 architecture.

1. When the the $\overline{EA}$ =0 at RESET, the PC (MCU program counter) starts from 0x0000 and accesses the external addresses from the memory. Memory addresses are between 0x0000 and 0xFFFF.
2. When the $\overline{EA}$ =1 at RESET, the PC starts from 0x0000 for banks0 and 1 and accesses the internal addresses and the 0x1000 onwards from the external addresses from the memory.

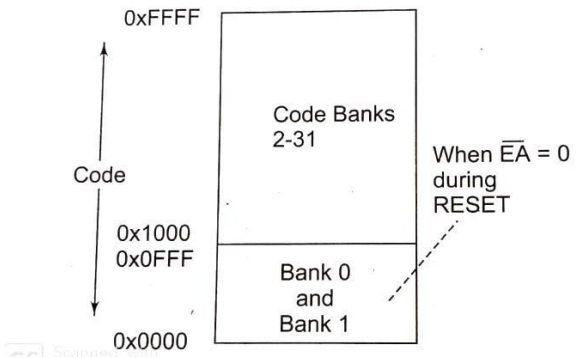


Fig. 4.8 Code Memory (Program memory)

External Data Memory

Fig. 4.9 shows a layout of the external data (X-DATA) memory addresses in the classic 8051 architecture. It can be accessed through the indirect addressing mode used.

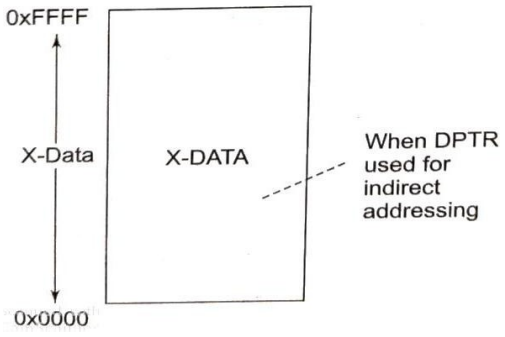


Fig. 4.9 Memory for X-Data in classic 8051

5. Special Function Registers (SFR)

For a programmer, the SFRs are at the directly addressable space special registers. These can be accessed by their names or by their addresses. The SFRs have addresses between 80H and FFH. These addresses are above 80H, since the addresses 00 to 7FH are addresses of RAM memory inside the 8051.Not all the address space of 80 to FF is used by the SFR. The unused locations 80H to FFH are reserved and must not be used by the 8051 programmer. The meaning of each symbol is enlisted in Table 4.1.

Table 4.1 Special Function Register (SFR) Address.

Symbol	Name	Address
ACC*	Accumulator	0E0H
B*	B-register	0F0H
PSW*	Program Status Word	0D0H
SP	Stack Pointer	81H
DPTR	Data Pointer 2 bytes	
	DPL lower byte	82H
	DPH higher byte	83H
P0*	Port0	80H
P1*	Port1	90H
P2*	Port2	0A0H
P3*	Port3	0B0H
IP*	Interrupt Priority Control	0B8H
IE*	Interrupt Enable Control	0A8H
TMOD	Timer /counter mode control	89H
TCON*	Timer/counter control	88H
T2CON*	Timer/counter 2 control	0C8H
T2MOD	Timer /counter mode control	0C9H
TH0	Timer/counter0 high byte	8CH
TL0	Timer/counter0 low byte	8AH
TH1	Timer/counter 1 high byte	8DH
TL1	Timer/counter 1 low byte	8BH
TH2	Timer/counter 2 high byte	0CDH
TL2	Timer/counter 2 low byte	0CCH
RCAP2H	T/C2 capture register high byte	0CBH
RCAP2L	T/C2 capture register high byte	0CAH
SCON*	Serial control	98H
SBUF	Serial data buffer	99H
PCON8	Power control	87H
* indicate Bit addressable		

6. Port Operation

The four ports P0, P1, P2 and P3 each use 8 pins, making them 8-bit ports. All the ports upon RESET are configured as output, ready to be used as output ports. To use any of these ports as an input port , it must be programmed. The port structure is depicted in Fig. 4.10

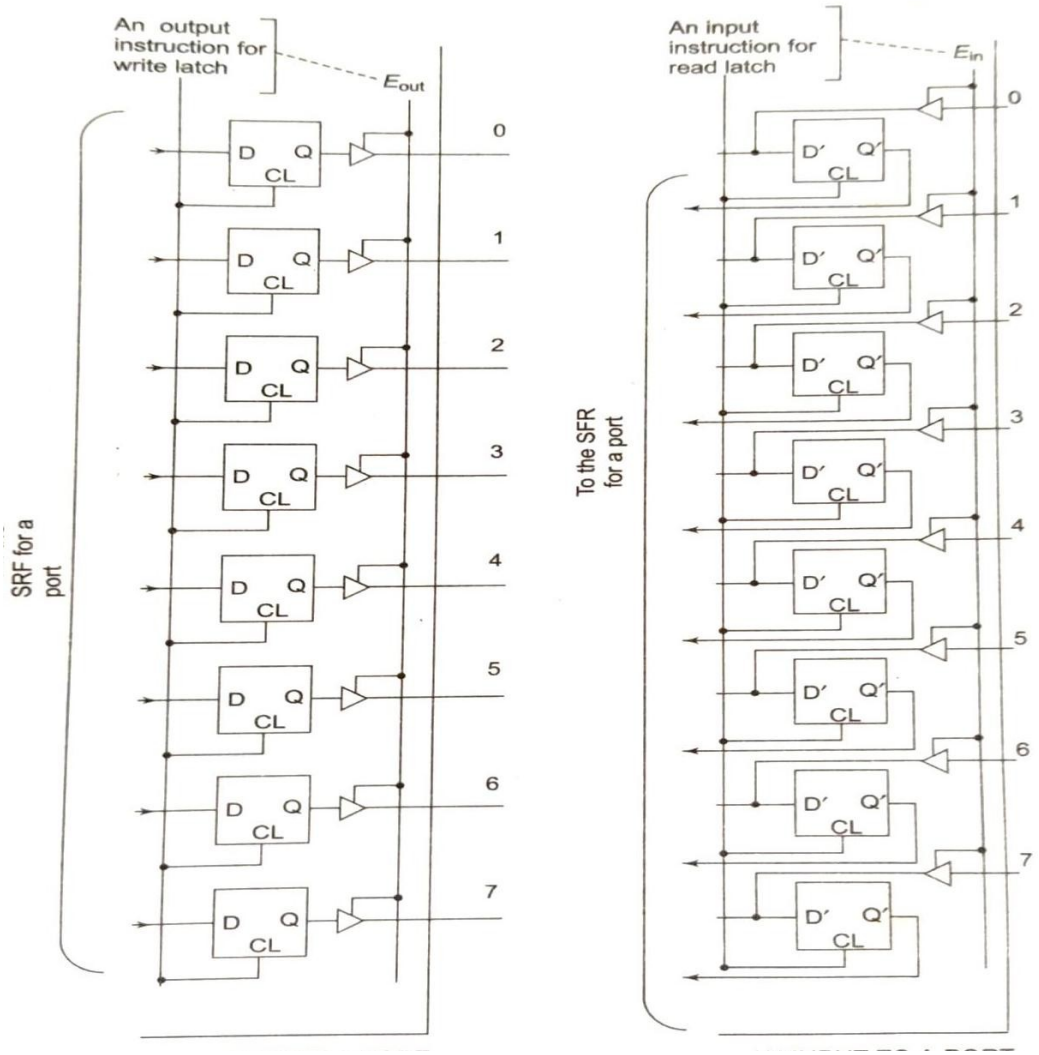


Fig.4.10 Port Structure

Port 0

It can be used for input or output. It occupies total of 8 pins (pins 32-39). To use the pins of port 0 as both input and out ports, each pin must be connected externally to a 10 K ohm pull-up resistor. P0 is an open drain unlike P1, P2 and P3. With external pull-up resistors connected upon reset, port0 is configured as an output port.

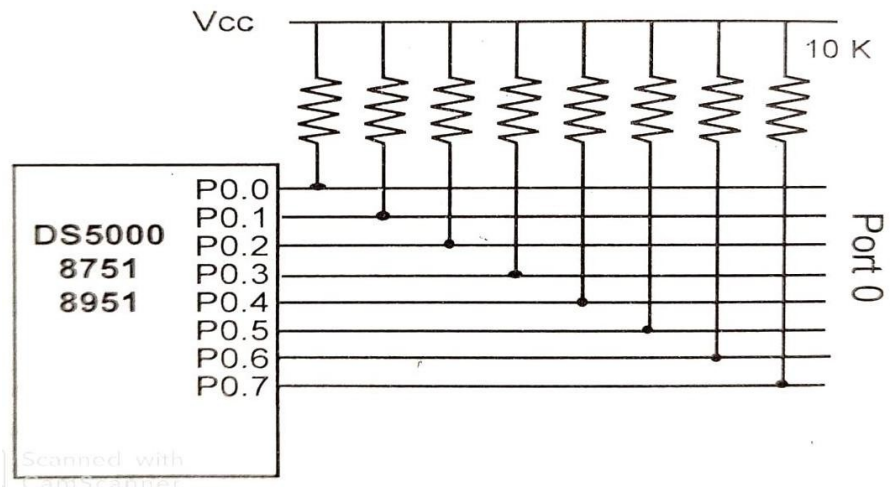


Fig. 4.11 Port 0 with pull up Resistors

With resistors connected to port 0 , in order to make it as input the port must be programmed by writing 1 to all the bits. In the following code.

```
        MOV     A, #0FFH
        MOV     P0, A
BACK:   MOVA, P0
        MOV     P1, A
        SJMP    BACK.
```

Port 1

Port 1 occupies a total of 8 pins (pins 1 through 8) . It can be used as input or output. In contrast to Port 0 , this port does not need any pull-up resistors since it already has pull-up resistors internally. Upon reset port 1 is configured as an output port. To make Port 1 an input port it must be programmed as such by writing 1 to all its bits.

Port 2

Port 2 occupies a total of 8 pins (pins 21 through 28). It can be used as input or output. Just like P1, port 2 does not need any pull-up resistors since it already has pull-up resistors internally. Upon reset, port 2 is configured as an output port. To make port 2 as input, it must programmed

as such by writing 1 to all its bits. The dual role of port 2 is also accomplished by providing higher byte address through A8-A15 to access the external memory.

Port 3

Port 3 occupies a total of 8 pins, pin 10 through 17. It can be used as input or output. P3 does not need any pull-up resistors , the same as P1 and P2. Although Port 3 is configured as an output port upon reset, Port 3 has additional function of providing some extremely important signals such as interrupts. Table depicts the alternate functions of port 2

Table 4.2 Port 3 alternate functions

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

P3.0 and P3.1 are used for the RxD and TxD serial communication signals. P3.2 and P3.3 are used for external interrupts. Bits P3.4 and P3.5 are used for timers 0 and 1. Bits P3.6 and P3.7 are used to provide $\overline{\text{WR}}$ and $\overline{\text{RD}}$ signals for external memories in 8051 based system.

7. Memory interfacing

7.1 Semiconductor memory

In the design of all microprocessor based system , Semiconductor memory are used as primary storage for code and data. It can be in units of K bits, M bits and so on. Semiconductor memories are conducted directly to the CPU and is also called as primary memory. The widely used semiconductor memories are ROM and RAM.

Characteristics of Semiconductor Memory

Memory capacity- The number of bits that a semiconductor memory chip can store is called chip capacity.

Memory organization- Memory chips are organized into number of locations within the IC. Each location hold 1 bit, 4bits, 8bits or even 16 bits, depending on how it is designed internally.

1. A memory chip contains 2^x locations where x is the number of address pins.
2. Each location contains y bits, where y is the number of data pins on the chip.
3. The entire chip will contain 2^x x y bits, where x is the number of address pins and y is the number of data pins

Speed- One of the most important characteristics of a memory chip is the speed at which its data can be accessed.

ROM (Read-Only-Memory)- It is a type of memory that does not loss its contents when the power is turned off. For this reason ROM is called volatile memory. There are different types of read-only- memory such as PROM, EPROM, EEPROM, Flash EPROM and mask ROM.

PROM- It refers to the kind of ROM that the user can burn information into it. That's why it is called as user-programmable memory. For every bit of the PROM, there exists a fuse. So it is programmed by blowing of fuses. It is also referred to as OTP (one –time programmable)

EPROM (Erasable Programmable ROM)- In EPROM, one can program the memory chip and erase it thousands of times. A widely used EPROM is called UV-EPROM. The content of UV-EPROM is erased when it is exposed to ultra violet light. Its erase time is near about 20 minutes.

EEPROM (Electrically Erasable Programmable ROM)- Its desired contents are erased by electrically.

Flash memory EPROM- This memory has become popular user-programmable memory chip, due to the process of erasure of the entire contents takes less than a second. As the erasure method is electrical sometimes it is called as Flash EEPROM.

Mask ROM- Mask ROM refers to kind of ROM in which the contents are programmed by the IC manufacturer. It is not a user-programmable ROM.

RAM (Random Access Memory)- It is called volatile memory since cutting of the power to the IC will result in the loss of data. Sometimes it is called as read and write memory (RAWM). There are three types of RAM: Static RAM (SRAM), NV-RAM (Nonvolatile RAM) and dynamic RAM (DRAM).

NV-RAM (Nonvolatile RAM)-This RAM is nonvolatile. Like other RAMs it allows the CPU to read write to it, but when the power is turned off, the contents are not lost. To retain its content every NV-RAM chip internally is made of the following components.

1. It uses extremely power-efficient. SRAM cells built out of CMOS
2. It uses an internal lithium battery as back energy source.
3. It uses an intelligent control circuitry. The main job of internal circuitry to monitor the V_{cc} pin constantly to detect the loss of external power supply. If the power to the Vcc pin falls the below out-of-tolerance condition, the control circuitry switches automatically to its internal power source, lithium battery.

DRAM (Dynamic RAM)-The use of a capacitor as a means to store data cuts down the number of transistors needed to build up the cell; however it requires constant refreshing due to leakage. This is in contrast to SRAM whose cells are made of flip-lops. The use of capacitor as storage cells in DRAM results in much smaller net memory size.

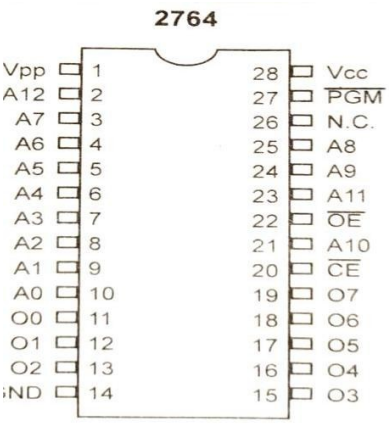


Fig. 4.12 2764 ROM 8Kx8

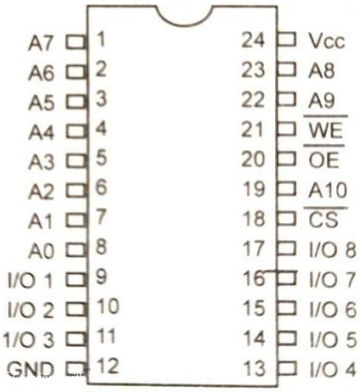


Fig. 4.13 2Kx8 SRAM pins

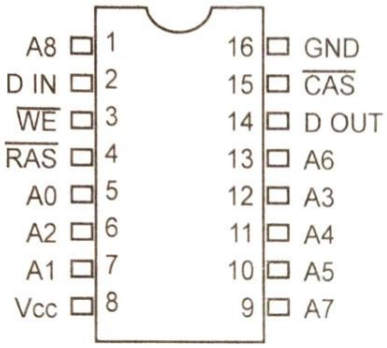


Fig. 4.14 256Kx1 DRAM

7.2 Memory Address Decoding

The job of the decoding circuitry to locate the selected memory block that CPU has access to desired data in memory chip. Memory chips have one more pins called CS (chip select) which must be activated for the memory contents to be accessed. Sometimes the chip select is also referred to as Chip Enable (CE).

Following points are required for interfacing the memory to the CPU.

1. The data bus of the CPU is connected directly to data pins of the memory chip.
2. Control signals RD (read) and WR write from the CPU are connected to the OE (output enable) and WE (write enable) pins of memory chips respectively.
3. In case of the address buses, while lower bits of the addresses from the CPU are connected directly to the address pins of the memory chips and upper address pins are used to activate the CS or CE pin of the memory chip. The CS or CE pin along with RD/WR allows the flow of data in or out of the memory chip.

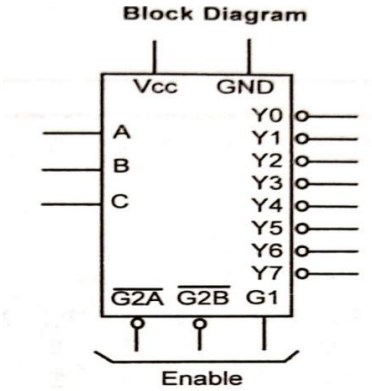


Fig. 4.15 74LS138 Decoder

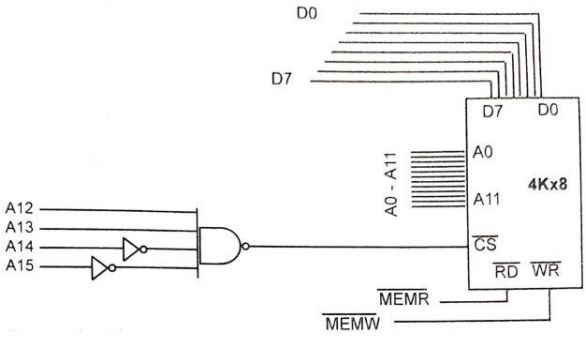


Fig. 4.16 logic Gate as Decoder

6.3 Interfacing with External ROM/RAM as Program and Data Memory

For interfacing to external ROM some pins have important role that to be discussed here.

$\overline{EA}$ -When this pin is connected to Vcc, that indicates the program code is stored in the microcontroller on-chip ROM. For external ROM access tis pin is grounded.

P0 and P2 role in providing addresses- In 8051 P0 and P2 provides the 16-bit address to access external memory. Of these ports P0 provides the lower 8 bit addresses A0-A7, and P2 provides the upper 8 bit addresses A8-A15. More importantly, P0 is also used to provide 8 bit- data bus D0-D7. In other words P0.0- P0.7 are used for both address and Data is called as address/data multiplexing. The sharing of this bus is accomplished by ALE (address latch enable.) Pin.
When ALE=0, the 8051 uses P0 for the data path and when ALE=1, it is used for address path.

$\overline{PSEN}$ **(program store enable)-** It is an output signal must be connected to OE pin of a ROM containing the program code. When $\overline{EA}$ pin is connected to ground the 8051 fetches opcode from external ROM by using $\overline{PSEN}$.

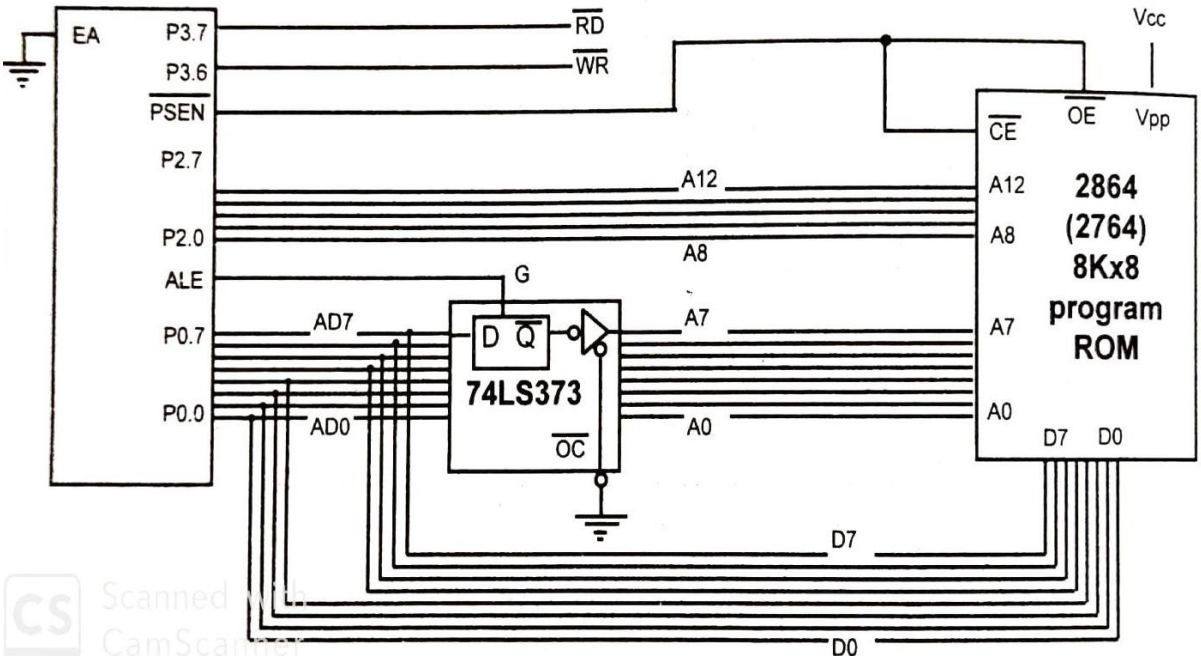


Fig. 4.17 Interfacing of ROM to 8051 as program memory

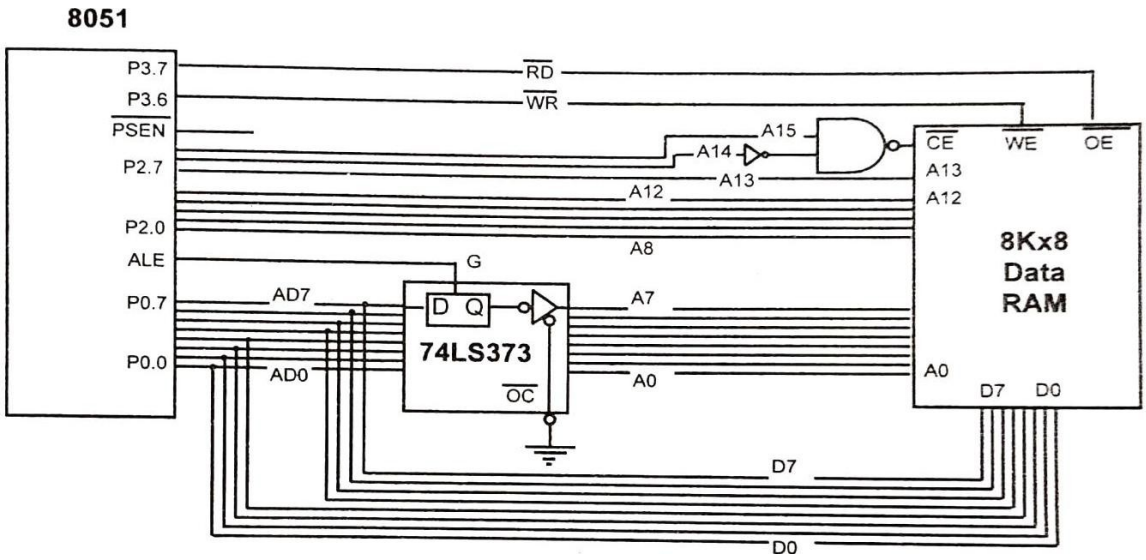


Fig. 4.18 Interfacing of ROM as Data Memory

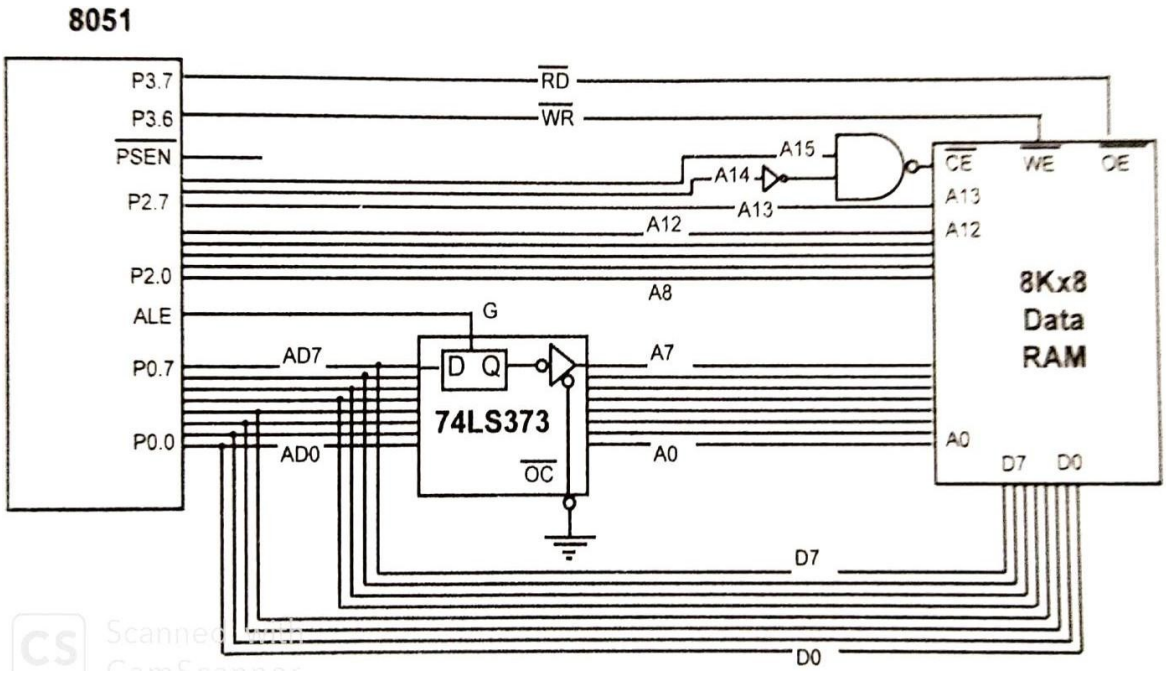


Fig. 4.19 Interfacing with Data RAM

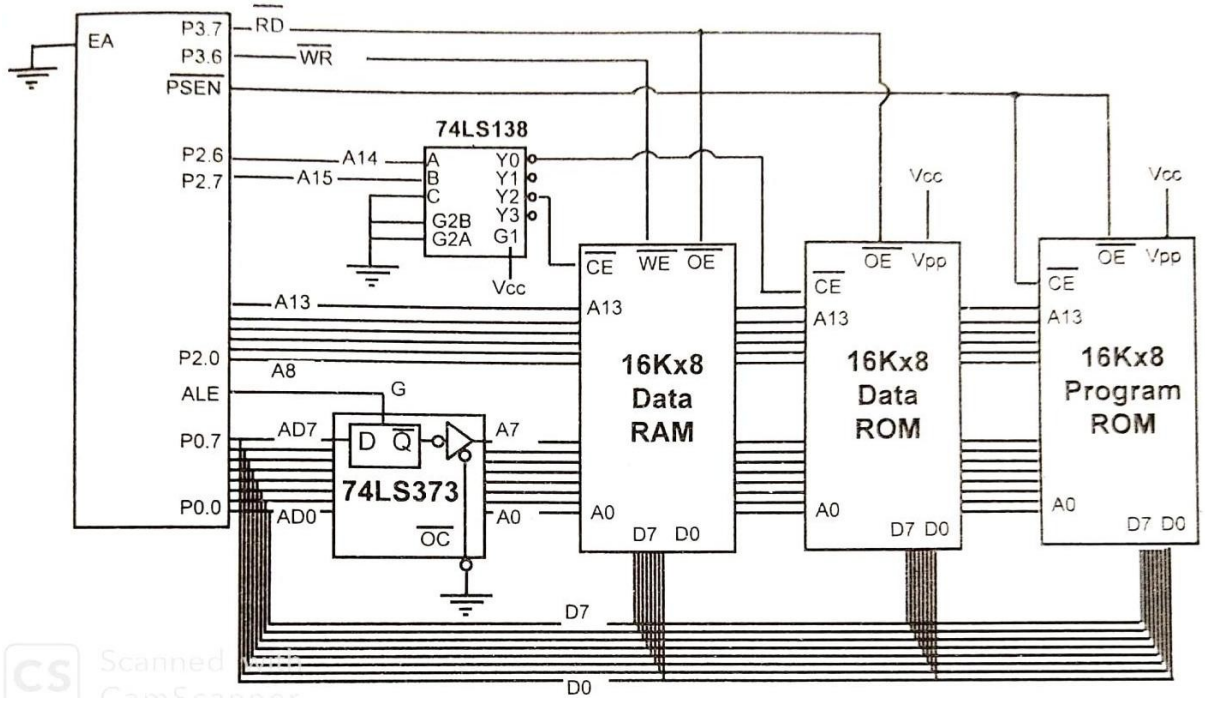


Fig. 4.20 Interfacing with Data RAM Data ROM and Program ROM

8. Interrupt

A single microcontroller can serve several devices. There are two ways to do that: interrupts or polling. In the Interrupt method, whenever any device needs its service, the device notifies the microcontroller by sending an interrupt signal. Once the interrupt is accepted the microcontroller serves the device by executing an interrupt service routine (ISR). In polling method the microcontroller continuously monitor the status of a give device, when the condition is met it performs the service. This polling method is not efficient because it has to monitor all times the status of devices in round-robin fashion and priority assignment is not possible.

Interrupt Service Routine

For every interrupt, there must be an Interrupt service routine (ISR), Interrupt handler. For every interrupt, there is a fixed location in memory that holds the address of its ISR. The group of memory locations set aside to hold the addresses of ISRs is called the interrupt vector table.

Steps in executing an Interrupt

Once an interrupt is activated, microcontroller performs the following steps.

1. It finishes the instruction it is executing and save the address of the next instruction (PC) on the stack.
2. It also saves the the current status of all the interrupts internally.
3. It jumps to a fixed location in memory called the interrupt vector table that holds the address of ISR.
4. The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it. It starts to execute the ISR until it reaches last instruction of subroutine RETI (return from the interrupt).
5. Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted. First it gets PC address from the stack by popping the top two bytes of the stack into the PC

Six Interrupts of 8051

The six interrupts in the 8051 are allocated as follows

1. Reset- when the reset pin is activated, the 8051 jumps to address location 0000. This is power-up reset.

2. Two interrupts are set aside for the timers: one for timer 0 and one for timer 1. Memory locations 000BH and 001BH in the interrupt vector table belongs to timer 0 and timer 1.respectively.
3. Two interrupts are set aside for hardware external hardware interrupts, Pin numbers 12 (P3.2) and 13(P3.3) in port 3 are for the external hardware interrupts INT0 and INT1, respectively. These external interrupts are also referred to as EX1 and EX2. Memory location 0003H and 0013H in the interrupt vector table are assigned to INT0 and INT1 respectively.
4. Serial communication has a single interrupt that belongs to both receive and transfer. The interrupt vector table location 0023H belongs to this interrupt.

Interrupt Vector Table for 8051

From the table it has been observed that only three bytes of ROM space is assigned to the reset pin. They are ROM address locations 0,1 and 2.

Table 4.2 Interrupt vector addresses

Interrupt	**ROM Location(Hex)**	**Pin**
Reset	0000	9
External hardware interrupt (INT0)	0003	12
Timer 0 interrupt (TF0)	000B	13
External hardware interrupt (INT1)	0013	
Timer 1 interrupt (TF1)	001B	
Serial COM interrupt (RI and TI)	0023	

Enabling and disabling an interrupt

Upon reset all interrupts are disabled. The interrupts must be enabled by software. There is a register IE (interrupt enable) that is responsible for enabling and disabling the interrupts.

To enable an interrupt following steps should be followed.

1. Bit D7of IE register (EA) must be set high to allow the rest of register to take effect.
2. If EA=1 , interrupts are enabled and will be responded to if their corresponding bits in IE are high. If EA=0, no interrupt will be responded to, even if the associated bit in the IE register is high.

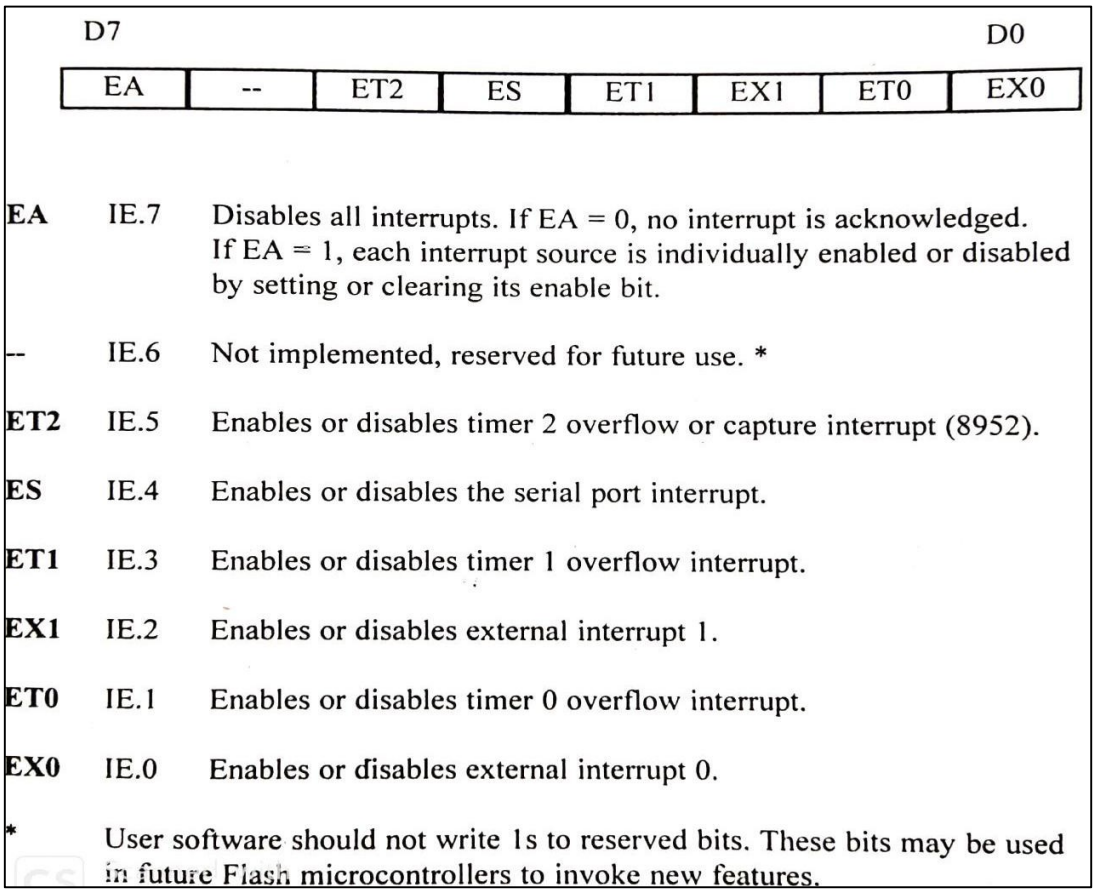


Fig. 4.21 Interrupt Enable Register

Interrupt Priority in the 8051

When the 8051 is powered up, the priorities are assigned, that are enlisted in table.

Table 4.3 Interrupt priority

Highest to Lowest priority	
External Interrupt 0	INT0
Timer Interrupt 0	TF0
External Interrupt 1	INT1
Timer Interrupt 1	TF1
Serial Communication	RI-TI

9. Programmer's Model

The CPU registers are used to store the data temporarily. The information may be data to be processed or address pointing the data to be fetched. The majority of registers are 8 bits. The 8-bit registers are shown in the diagram from MSB (most significant bit) D7 to the LSB (least significant bit) D0. The most widely used registers of 8051 are A (accumulator), B, R0, R1, R2, R3, R4, R5, R6, R7, DPTR (data pointer), and PC (program counter). All these registers are 8 bits except DPTR and the program counter. The accumulator is used to hold one operand before execution and hold the result after execution. The program counter points to the address of next instruction to be fetched. It is a auto increment register. As the size of program counter is 16 bit. 8051 can access the program addresses from 0000H-FFFFH. When 8051 is powered-up the program counter contents will be 0000H. This means that it expects the first opcode to be stored at ROM address 0000H. For this reason in the 8051 system, the first opcode must be burned memory location 0000H of program ROM since this is where it looks for the first instruction when it is booted.

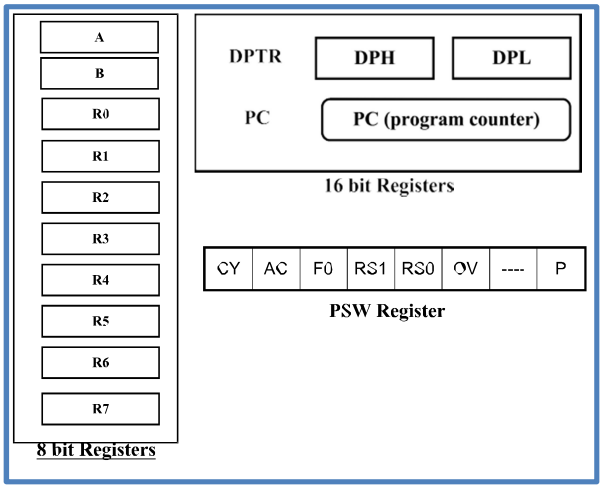


Fig. 4.22 Programmer's model

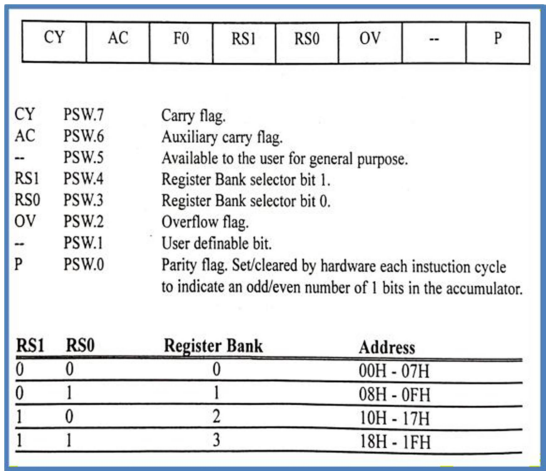


Fig. 4.23 PSW register

PSW (program status word register)

The program status word register (PSW) is an 8-bit register. It is also referred as Flag register. Although this register is size of 8-bits, only 6bits are used by 8051. Two unused bits are user definable flags. Other 4 bits are called as conditional flags such as CY (carry), AC (auxiliary carry), P(parity) and OV(overflow).In this register the bits PSW.3 and PSW.4 are designated as RS0 and RS1 and used to select the banks. PSW.5 and PSW.1 bits are general purpose status flags and can be used by the programmer for any purpose.

10. Operand addressing

An addressing mode is a method of specifying the data source or destination in an instruction. There are 5 types of addressing modes is supported by 8051.

1. Register
2. Immediate
3. Direct (memory related)
4. Register Indirect (memory related)
5. Index register addressing

Register addressing mode

This addressing mode involves the use of registers to hold the data to be manipulated.

Examples:

MOV A, R0 ; Copy the contents of R0 int A

ADD A, R7 ; Add the contents of R7 to contents of A and the result is stored in A

Immediate addressing mode

In this addressing mode immediate data is specified in instruction as a source operand.

Examples:

MOV B, #40H ; load 40H into B register

MOV DPTR, #2000H ; load 2000H into DPTR

Direct addressing mode

As we know the on-chip RAM of 8051 is 128 byte, it can be accessed through memory address from 00H to FF H. The allocations of 128 bytes are as follows.

1. RAM location 00H-1FH are assigned to register banks and stack
2. RAM location 20H-2FH is set aside as bit-addressable space to save single bit data.
3. RAM location 30H-7F is available as place to save bite-sized data.

Although the entire 128 bytes of RAM can be accessed through direct addressing mode, it is most often used to access RAM location 30H-7FH. This is due to fact that register banks are accessed through their names.

Examples:

MOV R4, 70H ; move the contents of RAM location 70H to R4.

MOV 56H, A ; save the content of A in RAM location 56H

PUSH 05 ; push R5 onto the stack

Register indirect addressing mode

In this mode the address (of 8bits) is indirectly specified in the instruction by the contents of pointer. This addressing mode so called because the source operand is from the address specified indirectly by another register in the instruction. The limitation is that only R0 and R1 register can be used in 8051 for indirect addressing. SFRs are directly accessible.

Examples

MOV R1, #55H ; load pointer R1=55H

MOV A, @R1 ; the content of pointer is transferred to A

Index registers addressing

Suppose we need to access external data RAM and external code space of on-chip ROM 16 bit address must be required. In this case we have to use DPTR. This mode is widely used in accessing data elements of look-up table entries in the program ROM space of 8051.

Examples;

MOV DPTR, #0200H ; load DPTR with 0200

CLR A ; clear accumulator

MOVC A,@A+DPTR ; Move the content 0200 location into A

11. Instruction set

The instruction set of 8051 can be classified into following group.

1. Data Transfer Instructions
2. Arithmetic Instructions
3. Logic Instructions
4. Boolean Variable manipulation Instructions
5. Program flow control (Processor and Machine control) Instructions
6. Interrupt flow Control instruction

12.1 Data Transfer Instruction

Three types of the data transfer can be done by move instruction. First type is transfer within the internal RAM and SFRs, second type is transfer using code memory area (CODE) and the third is using the external data memory X-DATA).

MOV instruction

A MOV instruction means move (copy) the bits from one source to a destination.

Table 4.4 MOV instructions within the registers, internal RAM and SFRs in 8051

Instruction (Mnemonic)	Action	Addressing	Length in bytes	cycles
MOV A, Rn	Move Rn into A	Register	1	1
MOV Rn, A	Move into Rn from A	Register	1	1
MOV A, #data	Move immediate 8-bit data into A	Immediate	2	1
MOV Rn, #data	Move into Rn the data.	immediate	2	1
MOV A, direct	Move byte at the direct address into A	Direct	2	1
MOV Rn, direct	Move from direct address into Rn	Direct	2	2
MOV direct, A	Move byte to the direct address form A	Direct	2	1
MOV direct, Rn	Move a byte to the direct address from Rn	Direct	2	2
M OV direct, direct	Move byte to the direct address from the direct address	Direct	3	2
MOV direct, #data	Move immediate data byte to the direct address	Immediate	3	2
MOV a,@Ri	Move into A the byte from the address pointed by Ri	Indirect	2	2
MOV @Ri, A	Move A into address pointed by Ri	Indirect	1	1
MOV direct, @Ri	Move into direct address from address pointed by Ri	indirect	1	1
MOV @Ri, direct	Move from the direct address to the address poined by ri	Indirect	2	2
MOV @Ri, #data	Move data ino address pointed by Ri	immediate	2	2
MOV DPTR, data16	Mov e16 bit dat	immediate	3	2

MOVC-type Instruction

It moves the 8-bit code from one source at the program memory (internal and external) to the register A destination.

Table 4.5 MOVC Instructions for transfer from the program memory area address code or constant to accumulator in 8051

Instruction	Action	Addressing	Length in bytes	Cycles
MOVC A, @A+DPTR	Moves the code or constant into A the byte from the program memory address pointed	Indirect	1	2

	by hypothetical addition of DPTR with the A itself.			
MOVC A, @A+PC	Move the code or constant into A the byte from the program memory address pointed by hypothetical addition of PC with the A itself	Indirect	1	2

MOX-type Instructions

A MOVX instruction means move (copy) the 8-bit data into A and from A using the external data memory address using DPTR or Ri as the pointer

Table 4.6 MOVX instruction

Instruction	Action	Addressing	Length in bytes	Cycles
MOVX A, @DPTR	Move the external data byte (X-DATA) into A from the data memory address pointed by DPTR	Indirect	1	2
MOVX @DPTR,A	Move into the external data memory from A to the address pointed by DPTR	Indirect	1	2
MOVX A,@Ri	Move the external data byte into a from the memory address pointed by Ri	Indirect	1	2
MOVX @Ri, A	Move into the external data memory from A to the memory address pointed by Ri	Indirect	1	2

Table 4.7 PUSH and POP instructions for using the Stack Area employing SP

Instruction	Action	Addressing	Length in bytes	Cycles
PUSH direct	Move byte from a direct internal RAM or SFR into the stack after first incrementing the stack pointer by 1	Direct	2	2
POP direct	Move byte to a direct internal RAM or SFR into the stack and then decrement the stack pointer by 1.	Direct	2	2

XCH-type instructions

An XCH instruction is for exchanging the A register with a source using the register (direct or indirect addresing0 mode.

Table 4.8 XCH and XCHD instruction

Instruction	Action	Addressing	Length in bytes	cycles
XCH A@Ri	Exchange byte at A with the address pointed by Ri	Indirect	1	2
XCH A,Rn	Exchange byte at A with the register Rn	Register	1	2
XCH A, direct	Exchange byte at A with the byte at a direct address.	Direct	1	1
XCHD A,@Ri	Exchange lower hex-digits of the bytes at A with the address pointed by Ri	Indirect	1	2

12.2 Arithmetic Instruction

These instructions include 8 bit addition, subtraction, increment, decrement, multiply and division instruction.

Table 4.9 Arithmetic ADD, SUB,MUL, DIV, INC and DEC instruction s in 8051

Instruction	Action	Addressing	Flags affected	Length (bytes)	Cycles
ADD A,Rn	Add Rn into A	Register	C,AC,OV	1	1
ADD A, direct	Add the byte at the direct address into A	Direct	C,AC,OV	2	1
ADD A, @Ri	Add the byte from the address pointed by the Ri into A	Indirect	C,AC,OV	1	1
ADD A, #data	Add immediate data byte to the A	Immediate	C,AC,OV	2	1
ADDC A, Rn	Add CF(carry) bit and Rn into A	Register	C,AC,OV	1	1
ADDC A, direct	Add CF bit and byte at the direct address ito A	Direct	C,AC,OV	2	1
ADDC A @Ri	Add CF bit and the byte from the address pointed by the Ri	Indirect	C,AC,OV	1	1
ADDC A, #data	Add CF bit and immediate data byte to the A	Immediate	C,AC,OV	**2**	**1**
SBBB A,Rn	Subtract borrow at CF bit and Rn into A	Rgister	C,AC,OV	**1**	**1**
SBBB A, direct	Subtract borrow at CF bit and byte at the direct address into A	Direct	C,AC,OV	**2**	**1**
SBBB A, @Ri	Subtract borrow at C bit and byte at the byte from the address pointed	Indirect	C,AC,OV	**1**	**1**

	by the Ri into A				
SBBB A, #data	Subtract borrow at CF bit and immediate data byte into A	Immediate	C,AC,OV	**2**	**1**
INC A	Increment	Register	None	1	1
INC Rn	Increment Rn	Register	None	1	1
INC direct	Increment byte at the direct address	Direct	None	2	1
INC @Ri	Increment the byte at the address pointed by Ri	Indirect	None	1	1
DEC A	Decrement A	Register	None	1	1
DEC Rn	Decrement Rn	Register	None	1	1
DEC direct	Decrement byte at the direct address	Direct	None	2	1
DEC @Ri	Decrement the byte at the address pointed by the Ri	Indirect	None	1	1
MUL AB	Multiply A and B Result MSB in B and LSB in A	Register	OV	1	4
DIV AB	Divide A (Numerator) and B(denominator) Remainder in B Quotient in A	Register	OV	1	4
DAA	Decimal adjust accumulator	Register	C	1	1

12.3 Logical Instruction

Table gives features of 8-bit AND, OR and XOR instruction. These instructions have 4 addressing modes such as register, immediate, direct and indirect.

Table 4.10 ANL, ORL XRL instruction

Instruction	**Action**	**Addressing**	**Length in bytes**	**Cycles**
ANL A, Rn	AND Rn into A	Register	1	1
ANL A, direct	AND byte at the direct address into A	Direct	2	1
ANL A, @Ri	AND into the byte from the address pointed by the Ri	Indirect	1	1
ANL A, #data	AND immediate data byte into A	immediate	2	1
ANL direct, A	AND A into byte at the direct address	Direct	2	1
ANL direct, #data	AND immediate byte into byte at the direct address	Direct	3	2
ORL A, Rn	OR Rn into A	Register	1	1
ORL A, direct	OR byte at the direct address into A	Direct	2	1
ORL A, @Ri	OR into the byte from the address pointed by Ri	Indirect	1	1
ORL A, #data	OR immediate data byte to the A	immediate	2	1
ORL direct, A	OR A into byte at the direct address	Direct	2	1
ORL direct,#data	OR immediate byte into byte at the	Direct	3	2

	direct address			
XRL A, Rn	XOR Rn into A	Register	1	1
XRL A, direct	XOR byte at the direct address into A	Direct	2	1
XRL A, @Ri	XOR the byte at the address pointed by Ri into A	Indirect	1	1
XRL A, #data	XOR immediate data byte to the A	immediate	2	1
XRL direct, A	XOR A into byte at the direct address	Direct	2	1
XRL direct, #data	XOR immediate byte into byte at the direct address	Direct	3	2

12.4 Boolean Variable manipulation Instructions

These are also called as Boolean processing instruction.

Table 4.11 MOV, CLR, CPL,SETB,ANL, and ORL Boolean Processing Instruction

Instruction	Action	Addressing	Length (bytes)	Cycles
MOV C, bit	Move bit into CF	Direct bit addressing	2	1
MOV bit, C	Move CF into the bit	Direct bit addressing	2	2
CLR C	Clear CF	PSW Register CF bit addressing	1	1
CLR bit	Clear bit	Direct bit addressing	2	1
CPL C	Complement CF	PSW Register CF bit addressing	1	1
CPL bit	Complement bit	Direct bit addressing	2	1
SETB C	Set CF=1	PSW Register CF bit addressing	1	1
SETB bit	Set bit =1	Direct bit addressing	2	1
ANL C,bit	AND between CF and bit, place the result in CF	Direct bit addressing	2	2
ANL C, $\overline{bit}$	AND between CF and , place the result in C	Direct bit addressing	2	2
ORL C,bit	OR between CF and bit, place the result in C	Direct bit addressing	2	2
ORL C, $\overline{bit}$	OR between CF and $\overline{bit}$, place the result in C	Direct bit addressing	2	2

12.5 Control Transfer Instruction

In the main program other sub programs may be called to perform a particular task. When a sub program is called the processor will jump to a new address where this program is available and

it has to accomplish program flow control transfer with help of JUMP and CALL instruction when some condition met.

Table 4.12 Delay-Cycle (NOP) instruction (No operation)

Instruction	Action	Addressing	Length in bytes	Cycles
NOP	No operation, PC gets the address of next instruction on incrementing at NOP.		1	1

Long, Absolute and Short Jump

8051 has three jump instructions: Long- it jumps to 16-bit address, Absolute- it jumps within 2 K bytes and Short- it jumps to address within 128 bytes above or below the present address.

Table 4.13 Long, absolute and short jump instructions

Instruction	Action	Addressing	Length in bytes	Cycles
LJMP addr16	Jump to the next address given by two bytes in the instruction	Direct 16 bit address	3	2
AJMP addr11	Jump to the next address	Direct 11-bit address	2	2
SJMP *rel*	Jump in the range between -128 and +127 from the address of next instruction	Direct 8-bit	2	2
JMP @A+DPTR	Jump in the next address given by addition of 8-bits of A with 16-bits of DPTR	Indirect 16-bit relative addreess		

Table 4.14 Conditional Short Relative Jumps

Instruction	Action	Addressing	Length in bytes	Cycles
JNZ rel	Jump to a relative address if a is not zero	Relative(offset)	2	2
JZ rel	Jump to a relative address if A is zero	Relative(offset)	2	2
JNC rel	Jump to a relative address if CF is not 1	Relative(offset)	2	2
JC rel	Jump to a relative address if CF=1	Relative(offset)	2	2
JB bit, rel	Jump to a relative address if addressed bit 1 (bit not set)	Relative(offset)	2	2
JNB bit,rel	Jump to a relative address if addressed bit 0 (bit not set)	Relative(offset)	2	2
JBC bit, rel	Jump to a relative address if addressed bit 1(bit set) and reset	Relative(offset)	2	2

	carry (make CF=0)			

Decrement and Conditional jump on Zero

Table 4.15 Instruction for decrement and then jump in program-loops in 8051

Instruction	Action	Addressing	Length in bytes	Cycles
DJNZ Rn, Rel	Decrement Rn and jump if Rn is still not zero.	Relative (offset)	2	2
DJNZ direct, Rel	Decrement byte at the direct and jump if byte is still not zero	Relative (offset)	2	2

Jump after comparison

Table 4.16 Compare then conditional jump after comparison

Instruction	Action	Addressing	Flag affected	Length in bytes	Cycles
CJNE A, #data, rel	Compare A and immediate data and jump if both are not equal.	Relative (offset)	C	3	2
CJNE Rn, #data, rel	Compare Rn and immediate data and jump if both are not equal.	Relative (offset	C	3	2
CJNE A, direct, rel	Compare the bytes at A and direct and jump if both are not equal	Relative (offset	C	3	2
CJNE @Ri, #data, rel	Compare byte from the address pointed by Ri and immediate data and jump if both are not equal	Relative (offset)	C	3	2

Call to a Routine

`Table 4.17 Long, absolute call and return instruction

Instruction	**Action**	**Addressing**	**Length in bytes**	**Cycles**
LCALL addr16	Call to the next address given by two bytes in the instruction	Direct 16-bit address	3	2
ACALL addr11	Call the next address given by 11 bits in	Direct 11	2	**2**

	the instruction.	bit address		
RET	Return to PC the saved PCL and PCH from the stack.	Stack address	1	**2**

12.6 Interrupt Control Flow (RETI instruction)

Table 4.18 RETI instruction

Instruction	Action	Addressing	Length In bytes	cycles
RETI	Return into PC the saved PCL and	Stack adddress	1	2

12. Programming

While the CPU can work only in binary, it can do so at a very high speed, however, it is quite tedious and slow for humans to deal with 0s and 1s in order to program the computer. A program that consists of 0s and 1s is called machine language. In the early days of the computer programmers coded programs in machine language. Although the hexadecimal system was used as a more efficient way to represent binary numbers, the process of working in machine code was still cumbersome for humans. Eventually, assembly language were developed which provided **mnemonics** for the machine code instructions. Plus other features which made programming faster and less prone to error. Assembly language is referred to as low level language because it deals directly with internal structure of CPU. Programmer needs assembler to convert the assembly language to machine language for execution purpose. Assembly language consists mnemonics optionally followed by one or two operands.

Programs

P1. Write an ALP (Assembly Language Program) to find the sum of values and store the result in A (lower byte and in R7 (higher byte). Assume that RAM locations 40-44 have the following values.

40=(7B), 41=(EC), 42=(C4), 43=(5B), 44=(30)

Solution:

```
        MOV     R0, #40H        ; load pointer
        MOV     R2, #05H        ; load counter
        CLR     A               ; A=0
        MOV     R7, A           ; clear R7
AGAIN:  ADD     A, @R0          ; add the byte pointer
        JNC     NEXT            ; if CY=0 it can jump to NEXT label
        INC     R7              ; increment counter
NEXT:   INC     R0              ; increment pointer
        DJNZ    R2,AGAIN        ; repeat until R2is zero
HERE:   SJMP    HERE
```

P2. Assume that 5 BCD data items are stored in RAM locations starting a 40H as shown below. Write an ALP to find the sum of all numbers. The result must be in BCD.

Solution:

```
        MOV     R0, #40H        ; load pointer
        MOV     R2, #05H        ; load counter
        CLR     A               ; A=0
        MOV     R7, A           ; clear R7
AGAIN:  ADD     A, @R0          ; add the byte pointer
        DA      A
        JNC     NEXT            ; if CY=0 it can jump to NEXT label
        INC     R7              ; increment counter
NEXT:   INC     R0              ; increment pointer
```

```
        DJNZ    R2,AGAIN        ; repeat until R2is zero
HERE:   SJMP    HERE
```

P3. Write an ALP to get hex data in the range of 00-FFH from port 1 and convert it to decimal. Save the digits in R7, R6 and R5, where the least significant digit in R7.

```
        MOV     A, #0FFH
        MOV     P1, A           ; make an P1 an input port
        MOV     A1, P1          ; read data from P1
        MOV     B, #0AH         ;  move 0AH to register b
        DIV     AB              ; divide by the contents of A by B
        MOV     R7, B           ; Save lower digit in R7 register
        MOV     B , #0AH        ;
        DIV     AB              ;
        MOV     R6, B           ; save the next digit
        MOV     R5, A           ; save the last digit
HERE:   SJMP    HERE
```

P4. Read and test P1 to see whether it has the value 45H. if it does send 99H to P2; otherwise, it stays cleared.

Solution:

```
        MOV     P2, 00H         ; clear P2
        MOV     P1, #0FFH       ; make P1 an input port
        MOV     R3, #45H        ; R3=45H
        MOV     A, P1           ; read P1
        XRL     A, R3           ;
```

```
        JNZ     EXIT
        MOV     P2, #99H
EXIT:   ……
```

P5. Find the 2's complement of the value 78 H

Solution:

```
        MOV     A, #78H         ; A=85H
        CPL     A               ; make 1's complement a
        ADD     A, #01H         ; make 2's complemt
HERE:   SJMP    HERE
```

P6. Write an ALP to determine if register A contains the value 99H, if so, make R1=FFH otherwise make R1=0.

Solution:

```
        MOV     R1, #00H        ; clear R1
        CJNE    A, #99H, NEXT   ; if A is not equal 99H then jump
        MOV     R1, #0FFH       ;  make R1=FFH
NEXT    …..
```

P7. Assume that P1 is an input port connected to a temperature sensor. Write an ALP to read the temperature and test it for the value 75. According to the rest result, place the temperature value into the registers indicated by the following.

If T=75 then A=75

If T<75 then R1=T

If T>75 then R2=T

Solution:

```
        MOV     P1, # 0FFH          ; make P1 an input port
        MOV     A, P1               ; read P1 port, temperature
        CJNE    A, #75, OVER        ; jump if  A is not equal 75
        SJMP    EXIT
OVER:   JNC     NEXT                ; if CY=0, then A>75
        MOV     R1, A               ; if CY=1, A<75
        SJMP    EXIT                ; Exit
NEXT:   MOV     R2, A
EXIT    ……
```

P8. Write an ALP that finds the number of 1s in a given byte 97H.

Solution:

```
        MOV     R1, #00H            ; clear R1
        MOV     R7, 08H             ; Counter=08
        MOV     A, 97H
AGAIN:  RLC     A                   ; rotate through CY once
        JNC     NEXT                ; check for CY
        INC     R1                  ; if CY=1 then increment R1
NEXT:   DJNZ    R7, AGAIN           ; go through 8times
HERE:   SJMP    HERE
```

P9. Assume that register a has packed BCD 29H, write an ALP to convert packed BCD to ASCII numbers and place them in R2 and R6.

Solution:

```
        MOV     A, #29H         ; A=29H, packed BCD
        MOV     R2, A           ; keep a copy of BCD data in R2
        ANL     A, #0FH         ; mask the upper nibble (A=09)
        ORL     A, #30H         ; make it an ASCII, A=39H
        MOV     A, R6           ; save in R6
        MOV     A, R2           ; A=29H
        ANL     A, #0F0H        ; mask the lower nibble
        RR      A               ; rotae right
        RR      A               ; rotae right
        RR      A               ; rotate right
        RR      A               ; rotate right
        ORL     A, #30 H        ; A=32H
        MOV     R2, A           ; save the ASCII character in R2
HERE:   SJMP    HERE
```

P10. Write an ALP to create a square wave of 50% duty cycle on bit 0 of port 1.

Solution:

```
HERE:   SETB    P1.0            ; set to high bit 0of port 1
        LCALL   DELAY           ; call the delay subroutine
        CLR     P1.0            ; p1.0=0
        LCALL   DELAY
        SJMP    HERE
```

Embedded Systems

P11. Assume that the bit P2.2 is used to control the outdoor light and bit P2.5 to control the light inside the building. Write an ALP to turn on outside light and to turn the inside one.

Solution:

```
SETB    C           ; CY=1
ORL     C, P2.2     ; CY=P2.2
MOV     P2.2, C     ;  turn  it “on” if not already “on”
CLR     C           ; CY=0
ANL     C, P2.5     ; CY=P2.5 ANDed with CY
MOV     P2.5, C     ; turn it off if not already off.
```

\

Embedded Systems

1. Introduction to Embedded Systems

What is Embedded System? (DEC2016, March-2017.)

An Electronic/Electro mechanical system which is designed to perform a specific function and is a combination of both hardware and firmware (Software)

E.g. Electronic Toys, Mobile Handsets, Washing Machines, Air Conditioners, Automotive Control Units, Set Top Box, DVD Player etc…

Embedded Systems are:

- Unique in character and behavior
- With specialized hardware and software

Embedded Systems Vs General Computing Systems: (March-2017)

General Purpose Computing System	Embedded System
A system which is a combination of generic hardware and General Purpose Operating System for executing a variety of applications	A system which is a combination of special purpose hardware and embedded OS for executing a specific set of applications
Contain a General Purpose Operating System (GPOS)	May or may not contain an operating system for functioning
Applications are alterable (programmable) by user (It is possible for the end user to re-install the Operating System, and add or remove user applications)	The firmware of the embedded system is pre-programmed and it is non-alterable by end-user
Performance is the key deciding factor on the selection of the system. Always „Faster is Better"	Application specific requirements (like performance, power requirements, memory usage etc) are the key deciding factors
Less/not at all tailored towards reduced operating power requirements, options for different levels of power management.	Highly tailored to take advantage of the power saving modes supported by hardware and Operating System
Response requirements are not time critical	For certain category of embedded systems like mission critical systems, the response time requirement is highly critical
Need not be deterministic in execution behavior	Execution behavior is deterministic for certain type of embedded systems like „Hard Real Time" systems

History of Embedded Systems:

- First Recognized Modern Embedded System: Apollo Guidance Computer (AGC) developed by Charles Stark Draper at the MIT Instrumentation Laboratory.
  - It has two modules
  - 1.Command module(CM) 2.Lunar Excursion module(LEM)
  - RAM size 256 , 1K ,2K words
  - ROM size 4K,10K,36K words
  - Clock frequency is 1.024MHz
  - 5000 ,3-input RTL NOR gates are used
  - User interface is DSKY(display/Keyboard)

- First Mass Produced Embedded System: *Autonetics **D-17** Guidance comp* *missile*

Classification of Embedded Systems: (March-2017)

- *Based on Generation*
- **Based on Complexity & Performance Requirements**
- **Based on deterministic behavior**
- **Based on Triggering**

1. Embedded Systems - Classification based on Generation

First Generation: The early embedded systems built around 8-bit microprocessors like 8085 and Z80 and 4-bit microcontrollers

EX. stepper motor control units, Digital Telephone Keypads etc.

- **Second Generation**: Embedded Systems built around 16-bit microprocessors and 8 or 16-bit microcontrollers, following the first generation embedded systems

  EX.SCADA, Data Acquisition Systems etc.

- **Third Generation:** Embedded Systems built around high performance 16/32 bit Microprocessors/controllers, Application Specific Instruction set processors like Digital Signal Processors (DSPs), and Application Specific Integrated Circuits (ASICs).The instruction set is complex and powerful.

  EX. Robotics, industrial process control, networking etc.

- **Fourth Generation:** Embedded Systems built around System on Chips (SoCs), Re-configurable processors and multicore processors. It brings high performance, tight integration and miniaturization into the embedded device market

  EX Smart phone devices, MIDs etc.

2. Embedded Systems - Classification based on Complexity & Performance

- **Small Scale:** The embedded systems built around low performance and low cost 8 or 16 bit microprocessors/ microcontrollers. It is suitable for simple applications and where performance is not time critical. It may or may not contain OS.

- **Medium Scale**: Embedded Systems built around medium performance, low cost 16 or 32 bit microprocessors / microcontrollers or DSPs. These are slightly complex in hardware and firmware. It may contain GPOS/RTOS.

- **Large Scale/Complex:** Embedded Systems built around high performance 32 or 64 bit RISC processors/controllers, RSoC or multi-core processors and PLD. It requires complex hardware and software. These system may contain multiple processors/controllers and co-units/hardware accelerators for offloading the processing requirements from the main processor. It contains RTOS for scheduling, prioritization and management.

3. Embedded Systems - Classification Based on deterministic behavior: It is applicable for Real Time systems. The application/task execution behavior for an embedded system can be either deterministic or non-deterministic

These are classified in to two types

1. **Soft Real time Systems:** Missing a deadline may not be critical and can be tolerated to a certain degree

2. **Hard Real time systems:** Missing a program/task execution time deadline can have catastrophic consequences (financial, human loss of life, etc.)

4. *Embedded Systems - Classification Based on Triggering: These are classified into two types*

1. **Event Triggered :** Activities within the system (e.g., task run-times) are dynamic and depend upon occurrence of different events **.**

2. **Time triggered:** Activities within the system follow a statically computed schedule (i.e., they are allocated time slots during which they can take place) and thus by nature are predictable.

Major Application Areas of Embedded Systems:

- **Consumer Electronics:** Camcorders, Cameras etc.
- **Household Appliances:** Television, DVD players, washing machine, Fridge, Microwave Oven etc.
- **Home Automation and Security Systems:** Air conditioners, sprinklers, Intruder detection alarms, Closed Circuit Television Cameras, Fire alarms etc.
- **Automotive Industry:** Anti-lock breaking systems (ABS), Engine Control, Ignition Systems, Automatic Navigation Systems etc.
- **Telecom:** Cellular Telephones, Telephone switches, Handset Multimedia Applications etc.
- **Computer Peripherals:** Printers, Scanners, Fax machines etc.
- **Computer Networking Systems:** Network Routers, Switches, Hubs, Firewalls etc.
- **Health Care:** Different Kinds of Scanners, EEG, ECG Machines etc.
- **Measurement & Instrumentation:** Digital multi meters, Digital CROs, Logic Analyzers PLC systems etc.
- **Banking & Retail:** Automatic Teller Machines (ATM) and Currency counters, Point of Sales (POS)
- **Card Readers:** Barcode, Smart Card Readers, Hand held Devices etc.

Purpose of Embedded Systems: (DEC2016)

Each Embedded Systems is designed to serve the purpose of any one or a combination of the following tasks.

- Data Collection/Storage/Representation
- Data Communication
- Data (Signal) Processing
- Monitoring
- Control
- Application Specific User Interface

1. Data Collection/Storage/Representation:-

- Performs acquisition of data from the external world.
- The collected data can be either analog or digital
- Data collection is usually done for storage, analysis, manipulation and transmission
- The collected data may be stored directly in the system or may be transmitted to some other systems or it may be processed by the system or it may be deleted instantly after giving a meaningful representation

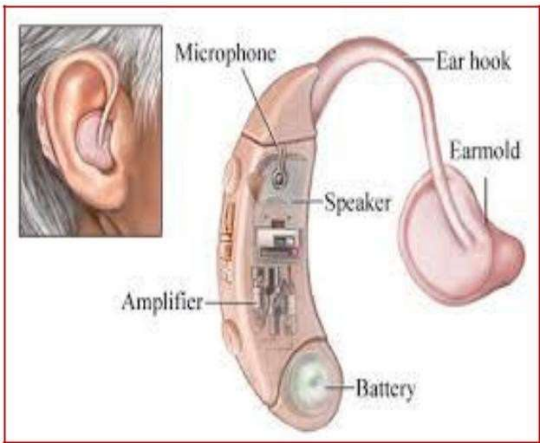
2. Data Communication:-

- Embedded Data communication systems are deployed in applications ranging from complex satellite communication systems to simple home networking systems
- Embedded Data communication systems are dedicated for data communication
- The data communication can happen through a wired interface (like Ethernet, RS-232C/USB/IEEE1394 etc)
- or wireless interface (like Wi-Fi, GSM,/GPRS, Bluetooth, ZigBee etc)

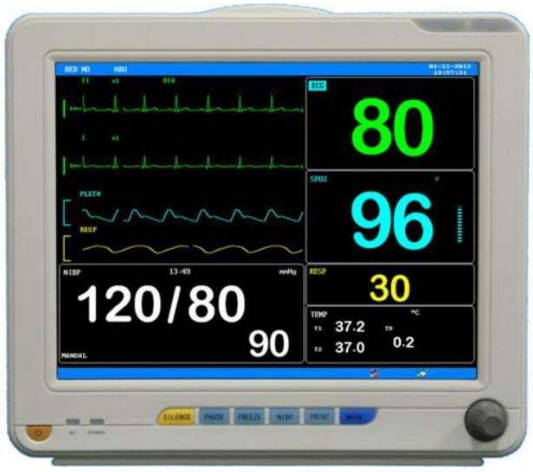
- Network hubs, Routers, switches, Modems etc are typical examples for dedicated data transmission embedded systems

3. Data (Signal) Processing:-

- Embedded systems with Signal processing functionalities are employed in applications demanding signal processing like Speech coding, synthesis, audio video codec, transmission applications etc
- Computational intensive systems
- Employs Digital Signal Processors (DSPs)



4. Monitoring:-



- Embedded systems coming under this category are specifically designed for monitoring purpose
- They are used for determining the state of some variables using input sensors
- They cannot impose control over variables.
- Electro Cardiogram (ECG) machine for monitoring the heart beat of a patient is a typical example for this
- The sensors used in ECG are the different Electrodes connected to the patient‟s body
- Measuring instruments like Digital CRO, Digital Multi meter, Logic Analyzer etc used in Control & Instrumentation applications are also examples of embedded systems for monitoring purpose

5. Control:-

- Embedded systems with control functionalities are used for imposing control over some variables according to the changes in input variables
- Embedded system with control functionality contains both sensors and actuators
- Sensors are connected to the input port for capturing the changes in environmental variable or measuring variable
- The actuators connected to the output port are controlled according to the changes in input variable to put an impact on the controlling variable to bring the controlled variable to the specified range
- Air conditioner for controlling room temperature is a typical example for embedded system with „Control‟ functionality
- Air conditioner contains a room temperature sensing element (sensor) which may be a thermistor and a handheld unit for setting up (feeding) the desired temperature
- The air compressor unit acts as the actuator. The compressor is controlled according to the current room temperature and the desired temperature set by the end user.

6. Application Specific User Interface:-

- Embedded systems which are designed for a specific application
- Contains Application Specific User interface (rather than general standard UI) like key board, Display units etc
- Aimed at a specific target group of users
- Mobile handsets, Control units in industrial applications etc are examples

Characteristics of Embedded systems: *(DEC2016, March-2017)*

Embedded systems possess certain specific characteristics and these are unique to each Embedded system.

1. Application and domain specific
2. Reactive and Real Time
3. Operates in harsh environments
4. Distributed
5. Small Size and weight
6. Power concerns
7. Single-functioned
8. Complex functionality
9. Tightly-constrained
10. Safety-critical

1. *Application and Domain Specific:-*

- Each E.S has certain functions to perform and they are developed in such a manner to do the intended functions only.
- They cannot be used for any other purpose.
- Ex – The embedded control units of the microwave oven cannot be replaced with AC‟S embedded control unit because the embedded control units of microwave oven and AC are specifically designed to perform certain specific tasks.

2. Reactive and Real Time:-

- E.S are in constant interaction with the real world through sensors and user-defined input devices which are connected to the input port of the system.
- Any changes in the real world are captured by the sensors or input devices in real time and the control algorithm running inside the unit reacts in a designed manner to bring the controlled output variables to the desired level.
- E.S produce changes in output in response to the changes in the input, so they are referred as reactive systems.
- Real Time system operation means the timing behavior of the system should be deterministic ie the system should respond to requests in a known amount of time.
- Example – E.S which are mission critical like flight control systems, Antilock Brake Systems (**ABS**) etc are Real Time systems.

3. *Operates in Harsh Environment :–*

- The design of E.S should take care of the operating conditions of the area where the system is going to implement.
- Ex – If the system needs to be deployed in a high temperature zone, then all the components used in the system should be of high temperature grade.
- Also proper shock absorption techniques should be provided to systems which are going to be commissioned in places subject to high shock.

4. *Distributed: –*

- It means that embedded systems may be a part of a larger system.
- Many numbers of such distributed embedded systems form a single large embedded control unit.
- Ex – Automatic vending machine. It contains a card reader, a vending unit etc. Each of them are independent embedded units but they work together to perform the overall vending function.

5. *Small Size and Weight:-*

- Product aesthetics (size, weight, shape, style, etc) is an important factor in choosing a product.
- It is convenient to handle a compact device than a bulky product.
- In embedded domain compactness is a significant deciding factor.

6. **Power Concerns:-**

- Power management is another important factor that needs to be considered in designing embedded systems.
- E.S should be designed in such a way as to minimize the heat dissipation by the system.

7. **Single-functioned:-** Dedicated to perform a single function

8. **Complex functionality: -** We have to run sophisticated algorithms or multiple algorithms in some applications.

9. *Tightly-constrained:-*

- Low cost, low power, small, fast, etc

10. *Safety-critical:-*

- Must not endanger human life and the environment

Quality Attributes of Embedded System: Quality attributes are the non-functional requirements that need to be documented properly in any system design.**(DEC16,March-2017)**

Quality attributes can be classified as

I. *Operational quality attributes*

II. Non-operational quality attributes.

I. Operational Quality Attributes: The operational quality attributes represent the relevant quality attributes related to the embedded system when it is in the operational mode or online mode.

Operational Quality Attributes are:

1. **Response :-**

- It is the measure of quickness of the system.
- It tells how fast the system is tracking the changes in input variables.
- Most of the E.S demands fast response which should be almost real time.

Ex – Flight control application.

2. *Throughput :-*

- It deals with the efficiency of a system.
- It can be defined as the rate of production or operation of a defined process over a stated period of time.
- The rates can be expressed in terms of products, batches produced or any other meaningful measurements.
- Ex – In case of card reader throughput means how many transactions the reader can perform in a minute or in an hour or in a day.
- Throughput is generally measured in terms of "Benchmark".
- A Benchmark is a reference point by which something can be measured

3. *Reliability :-*

- It is a measure of how much we can rely upon the proper functioning of the system.
- Mean Time Between Failure (MTBF) and Mean Time To Repair (MTTR) are the terms used in determining system reliability.
- MTBF gives the frequency of failures in hours/weeks/months.
- MTTR specifies how long the system is allowed to be out of order following a failure.
- For embedded system with critical application need, it should be of the order of minutes.

4. *Maintainability:-*

- It deals with support and maintenance to the end user or client in case of technical issues and product failure or on the basis of a routine system checkup.
- Reliability and maintainability are complementary to each other.
- A more reliable system means a system with less corrective maintainability requirements and vice versa.
- Maintainability can be broadly classified into two categories

  1. Scheduled or Periodic maintenance (Preventive maintenance)

  2. Corrective maintenance to unexpected failures

5. *Security:-*

- Confidentiality, Integrity and availability are the three major measures of information security.
- Confidentiality deals with protection of data and application from unauthorized disclosure.
- Integrity deals with the protection of data and application from unauthorized modification.
- Availability deals with protection of data and application from unauthorized users.

6. *Safety :-*

- Safety deals with the possible damages that can happen to the operator, public and the environment due to the breakdown of an Embedded System.
- The breakdown of an embedded system may occur due to a hardware failure or a firmware failure.
- Safety analysis is a must in product engineering to evaluate the anticipated damages and determine the best course of action to bring down the consequences of damage to an acceptable level.

II. Non-Operational Quality Attributes: The quality attributes that needs to be addressed for the product not on the basis of operational aspects are grouped under this category.

1. **Testability and Debug-ability:-**

- Testability deals with how easily one can test the design, application and by which means it can be done.
- For an E.S testability is applicable to both the embedded hardware and firmware.
- Embedded hardware testing ensures that the peripherals and total hardware functions in the desired manner, whereas firmware testing ensures that the firmware is functioning in the expected way.
- Debug-ability is a means of debugging the product from unexpected behavior in the system
- Debug-ability is two level process
- 1.Hardware level 2.software level
- **1. Hardware level:** It is used for finding the issues created by hardware problems.
- **2. Software level:** It is employed for finding the errors created by the flaws in the software.

2. Evolvability :-

- It is a term which is closely related to Biology.
- It is referred as the non-heritable variation.
- For an embedded system evolvability refers to the ease with which the embedded product can be modified to take advantage of new firmware or hardware technologies.

3. Portability:-

- It is the measure of system independence.
- An embedded product is said to be portable if the product is capable of functioning in various environments, target processors and embedded operating systems.
- „Porting" represents the migration of embedded firmware written for one target processor to a different target processor.

4. Time-to-Prototype and Market:-

- It is the time elapsed between the conceptualization of a product and the time at which the product is ready for selling.
- The commercial embedded product market is highly competitive and time to market the product is critical factor in the success of commercial embedded product.
- There may be multiple players in embedded industry who develop products of the same category (like mobile phone).

5. Per Unit Cost and Revenue:-

- Cost is a factor which is closely monitored by both end user and product manufacturer.
- Cost is highly sensitive factor for commercial products
- Any failure to position the cost of a commercial product at a nominal rate may lead to the failure of the product in the market.
- Proper market study and cost benefit analysis should be carried out before taking a decision on the per-unit cost of the embedded product.
- The ultimate aim of the product is to generate marginal profit so the budget and total cost should be properly balanced to provide a marginal profit.

SUMMARY

1. An embedded system is an electronic/electromechanical system designed to perform a specific function and is a combination of both hardware and firmware (software).
2. A general purpose computing system is a combination of generic hardware and general purpose operating system for executing a variety of applications, whereas an embedded
3. System is a combination of special purpose hardware and embedded OS/firmware for executing a specific set of applications.
4. Apollo Guidance Computer (AGC) is the first recognized modern embedded system and Autonetics D-17, the guidance computer for the Minuteman-I missile, was the first mass produced embedded system.
5. Based on the complexity and performance requirements, embedded systems are classified into small-scale, medium-scale and large-scale/complex.
6. The presences of embedded system vary from simple electronic system toys to complex flight and missile control systems.
7. Embedded systems are designed to serve the purpose of any one or combination of data collection/storage/representation, data processing, monitoring, control or application specific user interface.

Wearable devices refer to embedded systems which are incorporated into accessories and apparels. It envisions the bonding of embedded technology in our day to day lives.