EE 6502-MICROPROCESSOR AND MICROCONTROLLER
UNIT-I 8085 PROCESSORS

Introduction

8085 Microprocessor

- A CPU built into a single LSI/VLSI chip is called a microprocessor.
- A digital computer using microprocessor as its CPU is called a microcomputer.

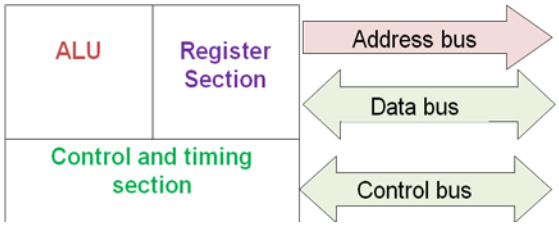


Fig: Internal structure and basic operation of microprocessor

****

Draw and explain the architecture of 8085.(Nov 2013, Dec 2014,Dec 2015,June 2016,Dec 2016, Dec 2018)

****

- The Intel 8085 is an 8-bit microprocessor introduced by Intel in 1977.
- The 8085 microprocessor is an 8-bit processor available as a 40-pin IC package and uses +5 V for power.
- It can run at a maximum frequency of 3 MHz.
- Its data bus width is 8-bit and address bus width is 16-bit, thus it can address 216 = 64 KB of memory.
- The internal architecture of 8085 is shown in figure.

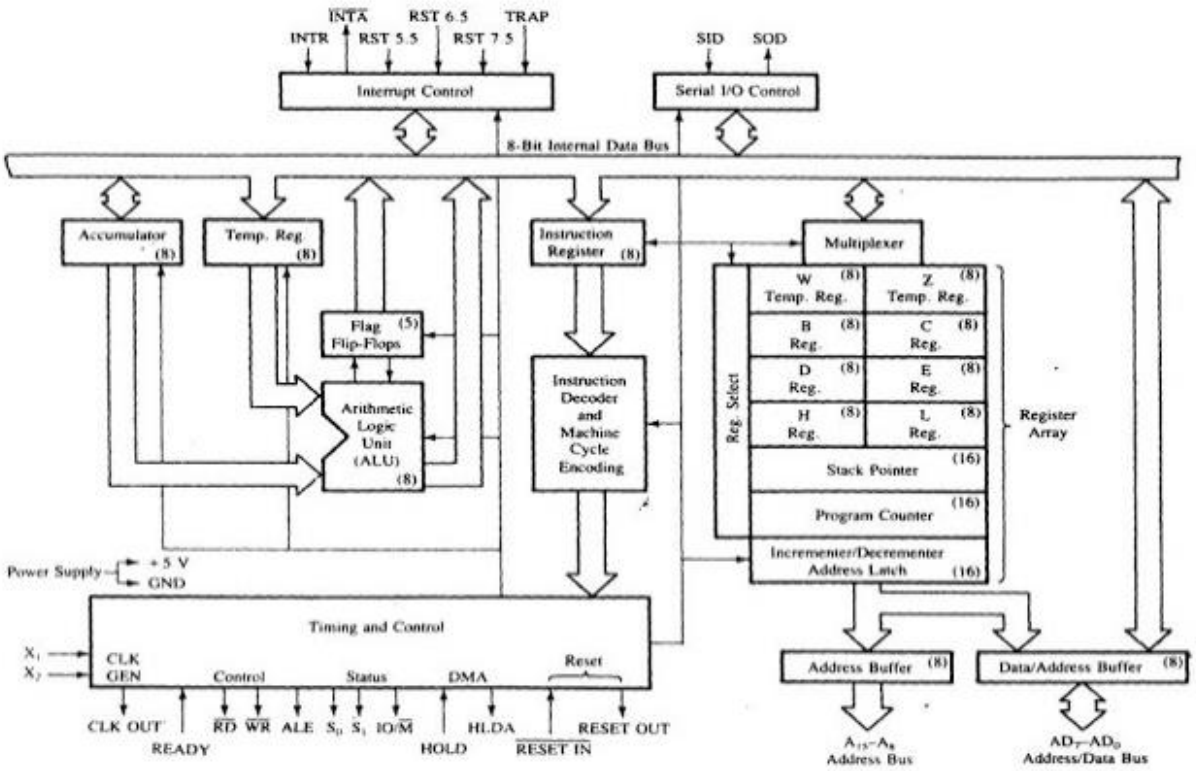


Figure: Block diagram of 8085 architecture

1. Timing and Control unit

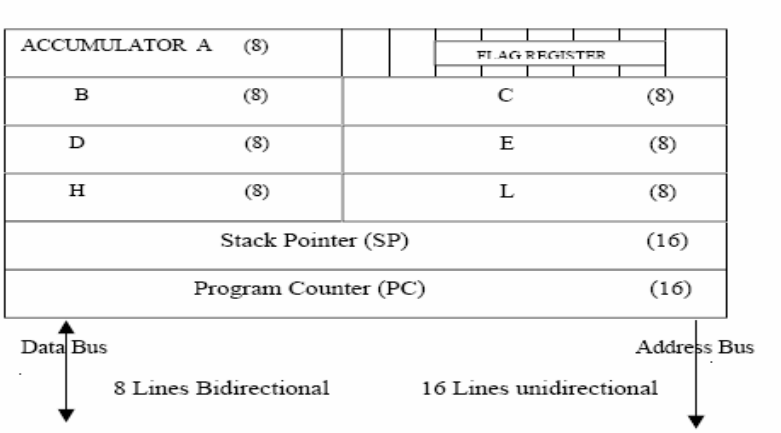
- This unit synchronizes all the microprocessor operations with the clock and generates control signals necessary for communicate between microprocessor and peripherals.
- The RD.WR signals are sync pulses indicating the availability of data on data bus.

2. Arithmetic Logic Unit

- The ALU performs the actual numerical and logic operation such as 'add', 'subtract', 'AND', 'OR', etc.
- Uses data from memory and from Accumulator to perform arithmetic.
- Always stores result of operation in Accumulator.

3. Register Array

- The 8085 includes six registers, one accumulator and one flag register. In addition, it has two 16-bit registers: stack pointer and program counter.



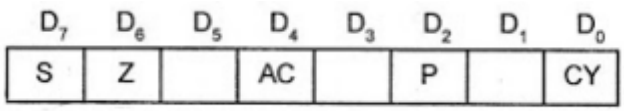
- The 8085 has six general-purpose registers to store 8-bit data; these are identified as B, C, D, E, H, and L
- They can be combined as register pairs - BC, DE, and HL - to perform some 16-bit operations
- The programmer can use these registers to store or copy data into the registers by using data copy instructions
- The HL register pair is also used to address memory locations
- **The accumulator** is an 8-bit register that is a part of ALU. This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.
- **Program Counter** - Deals with sequencing the execution of instructions. Acts as a memory pointer.
- **Stack Pointer** – Points to a memory location in R/W memory, called the stack.

4. Instruction register and Decoder

- **Instruction register**
  - It is an 8-bit register that temporarily stores the current instruction of a program. Latest instruction sent here from memory prior to execution.

- Decoder then takes instruction and decodes or interprets the instruction. Decoded instruction then passed to next stage.

➢ **Flag register**



- The ALU includes five flip-flops, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers.
- They are called Zero (Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags.
- **CY-Carry flag**. If the sum in the accumulator is larger than eight bits, the flip-flop uses to indicate a carry called the Carry flag (CY) is set to one.
- **Z-Zero flag** .When an arithmetic operation results in zero, the flip-flop called the Zero (Z) flag is set to one.
- **AC-auxillary carry flag**. In arithmetic operations, when a carry is generated by Digit D4 and passed to D5 ,the AC Flag is set.
- **P-Parity flag-** This flag is set when the result of the instruction has odd number of 1's.
- **S-Sign flag-**In arithmetic operation with signed number, the D7 bit is reserved for indicating the sign.
    - ❖ If D7 bit=1, the sign flag is set, and it indicates (-ve) number.
    - ❖ If D7 bit=0, it indicates it is a (+ ve) number.
- The combination of the flag register and the accumulator is called Program Status Word (PSW) and PSW is the 16-bit unit for stack operation.

5. System bus

- **Data Bus:**
    - ❖ Data bus carries data in binary form between microprocessor and other external units such as memory.
    - ❖ Data bus is bidirectional in nature.
    - ❖ The data bus width of 8085 microprocessor is 8-bit.
- **Address Bus:**
    - ❖ The address bus carries addresses and is one way bus from microprocessor to the memory or other devices. 8085 microprocessor contain 16-bit address bus and are generally identified as A0 - A15.
    - ❖ The higher order address lines (A8 – A15) are unidirectional and the lower order lines (A0 – A7) are multiplexed (time-shared) with the eight data bits (D0 – D7) and hence, they are bidirectional.

- **Control Bus:** Control buses are various lines which have specific functions for coordinating and controlling microprocessor operations. Ex.read/Write controlline

6. Interrupt control

- Interrupt is a signal ,which suspends the routine what the MP is doing, brings the control to perform the subroutines,completes it and returns to main routine.E.g. INTR,TRAP,RST 7.5,RST 6.5 ,RST 5.5

7. Serial I/O control

- It is used for serial data transmission and data bits are sent one bit at a time. The two signals used are
    - ❖ SID(serial input data) and
    - ❖ SOD (serial output data).

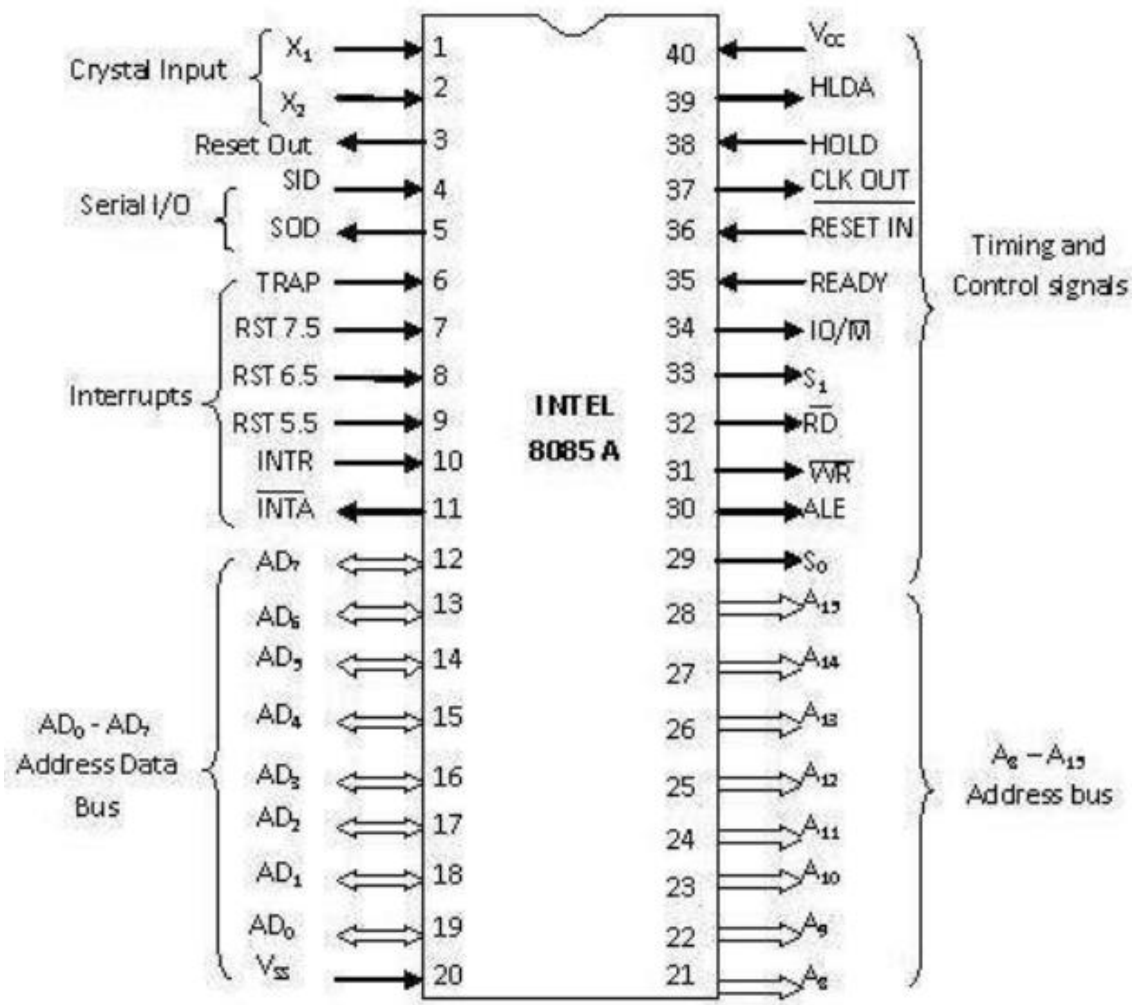
****

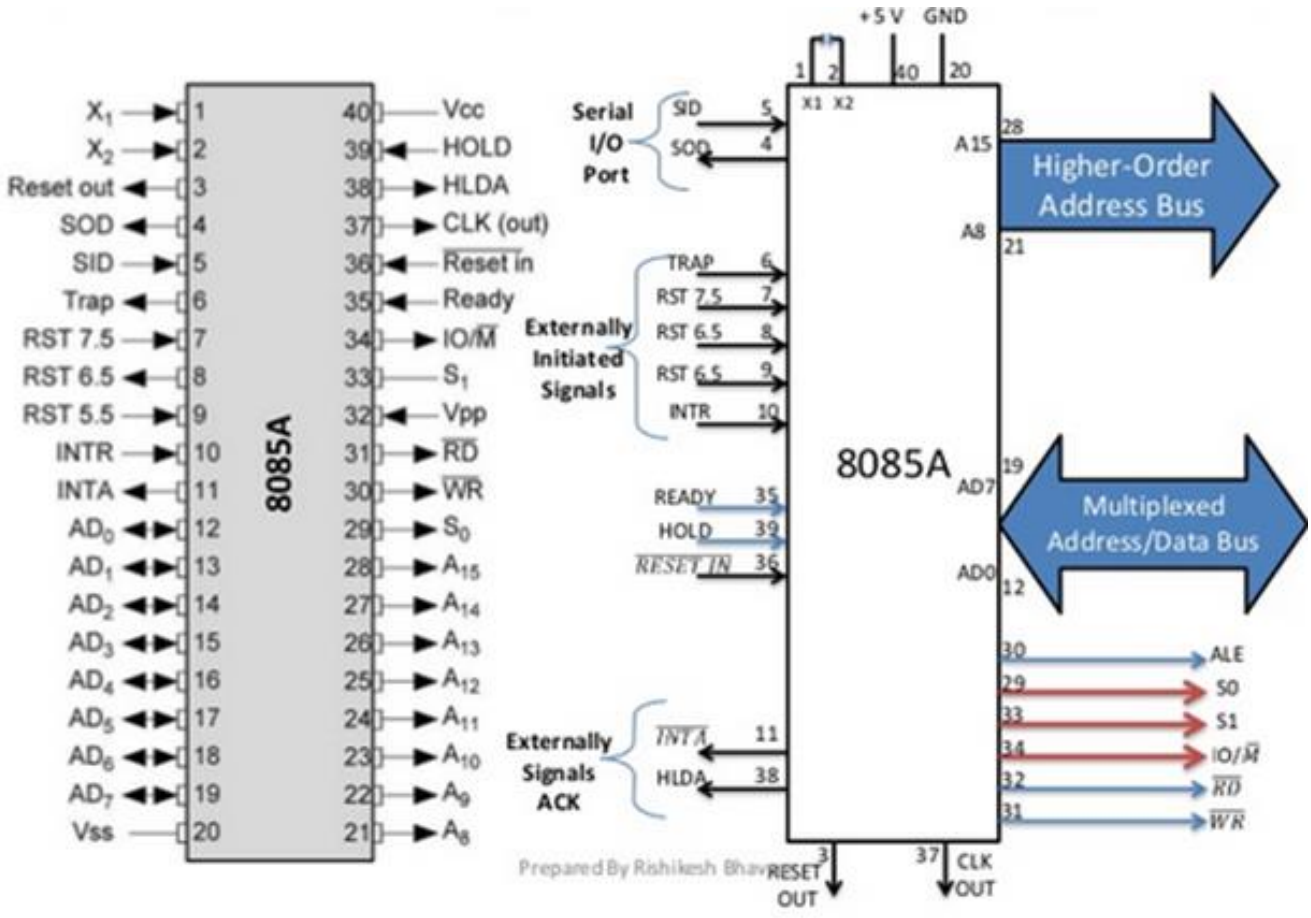
Draw and explain pin diagram and functional block diagram of 8085.(Dec 2014, April 2018)
Explain 8085 with signal diagram.

****

8085 Pin diagram

Figure shows 8085 pin configuration and functional 8085 Microprocessor Pin Diagram respectively. The signals of 8085 can be classified into seven groups according to their functions.





a)Pin configuration **b)Functional pin diagram**

- **Data and Address bus**
  - ❖ **$A_8 – A_{15}$ Address bus -** it carries the most significant 8-bits of memory I/O address.
  - ❖ **AD_7-AD_0,(Multiplexed Address/data bus).**It carries the least significant 8-bit address and data bus. These pins serve the dual purpose of transmitting lower order address and data byte. During 1st clock cycle, these pins act as lower half of address. In remaining clock cycles, these pins act as data bus

- **Control and status signals**

  These signals are used to identify the nature of operation. There are 3 control signal and 3 status signals.
  - ✓ **Three control signals are RD, WR & ALE.**
  - ✓ **RD** − This signal indicates that the selected IO or memory device is to be read and is ready for accepting data available on the data bus.
  - ✓ **WR** − This signal indicates that the data on the data bus is to be written into a selected memory or IO location.
  - ✓ **ALE** − It is a positive going pulse generated when a new operation is started by the microprocessor. When the pulse goes high, it indicates address. When the pulse goes down it indicates data.

✓ **Three status signals are IO/M, S_0 & S_1.**

❖ **S1 & S0**

S_0 and S_1 are called Status Pins. They tell the current operation which is in progress in 8085.

S_0	S_1	Operation
0	0	Halt
0	1	Write
1	0	Read
1	1	Opcode Fetch

❖ **IO/M**

This signal is used to differentiate between I/O and Memory operations, i.e.

- ✓ when it is high indicates I/O operation and
- ✓ when it is low then it indicates memory operation.

These signals are used to identify the type of current operation.

Control and Status Signals

Machine Cycle	IO/M	S1	S0	Control signals
Opcode Fetch	0	1	1	RD=0
Memory Read	0	1	0	RD=0
Memory Write	0	0	1	WR=0
I/O Read	1	1	0	RD=0
I/O Write	1	0	1	WR=0
Interrupt Acknowledge	1	1	1	INTA=0
Halt	Z	0	0	RD, WR=z and INTA=1
Hold	Z	X	X	RD, WR=z and INTA=1
Reset	Z	X	X	RD, WR=z and INTA=1

- **Power supply**

  There are 2 power supply signals **V_{CC} & V_{SS}**.

  - V_{CC} indicates +5v power supply and
  - V_{SS} indicates ground signal.

- **Clock signals**

  There are 3 clock signals, i.e. X1, X2, CLK OUT.

  - **X1, X2** − A crystal (RC, LC N/W) is connected at these two pins and is used to set frequency of the internal clock generator. This frequency is internally divided by 2.
  - **CLK OUT** − This signal is used as the system clock for devices connected with the microprocessor.

- **Interrupts & externally initiated signals**

  Interrupts are the signals generated by external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR. We will discuss interrupts in detail in interrupts section.

  - **INTA** − It is an interrupt acknowledgment signal.
  - **RESET IN** − This signal is used to reset the microprocessor by setting the program counter to zero.
  - **RESET OUT** − This signal is used to reset all the connected devices when the microprocessor is reset.
  - **READY** − This signal indicates that the device is ready to send or receive data. If READY is low, then the CPU has to wait for READY to go high.
  - **HOLD** − This signal indicates that another master is requesting the use of the address and data buses.
  - **HLDA (HOLD Acknowledge)** − It indicates that the CPU has received the HOLD request and it will relinquish the bus in the next clock cycle. HLDA is set to low after the HOLD signal is removed.

- **Serial I/O signals**

  There are 2 serial signals, i.e. SID and SOD and these signals are used for serial communication.

  - **SOD** (Serial output data line) − The output SOD is set/reset as specified by the SIM instruction.
  - **SID** (Serial input data line) − The data on this line is loaded into accumulator whenever a RIM instruction is executed.

- **Five Hardware Interrupts in 8085**
  - ❖ TRAP is a non-maskable interrupt
  - ❖ RST 7.5 is an edge triggered interrupt.
  - ❖ RST 6.5 is a maskable and level triggered interrupt
  - ❖ RST 5.5 is a maskable and level triggered interrupt
  - ❖ INTR is a non-vectored interrupt

****

Explain the memory organization of 8085

****

3. Memory Organization (R/W Memory):

Memory is an essential component of a microcomputer system. It stores binary instructions and data for the microprocessor. There are two types of memory: Read/Write Memory(R/WM) and Read-only Memory (ROM). The 8085 has 16 address lines. That means it can address upto 2^{16}=64 Kbytes.

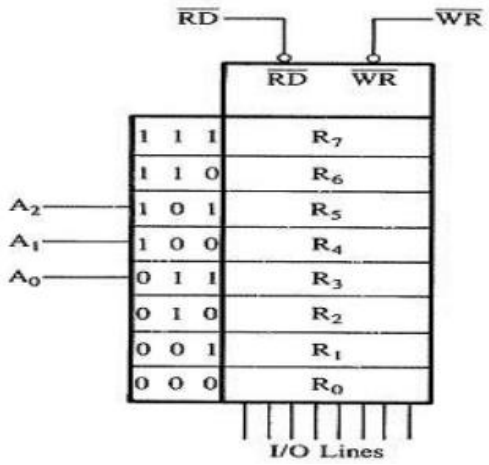


Fig: Memory chip with 8 registers

To communicate with memory, the MPU should be able to:

1. identifies the memory location (with address)
2. Generates timing and control signal
3. Data transfer takes place.

The MPU uses the CS line to select the chip, and the R/W line to control data flow.

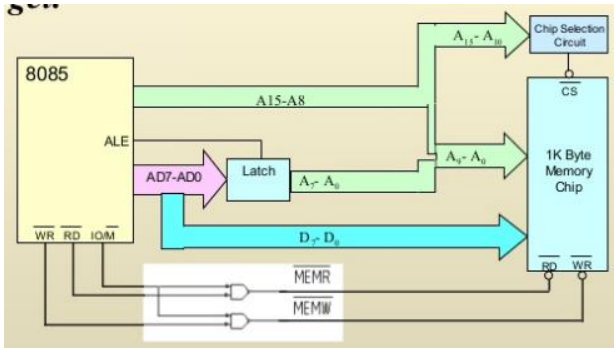


Fig: Interfacing 8085 with R/W memory

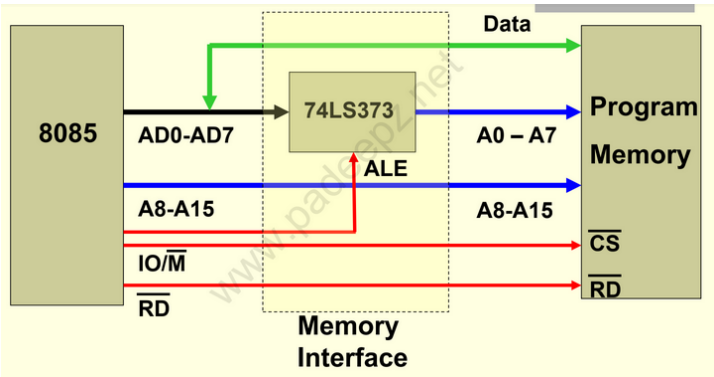
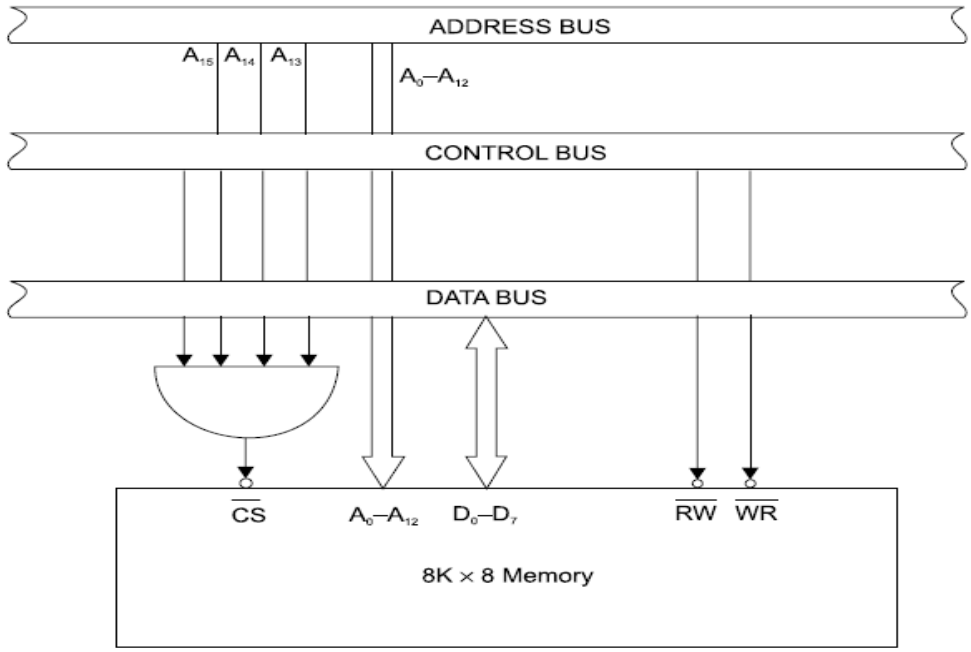
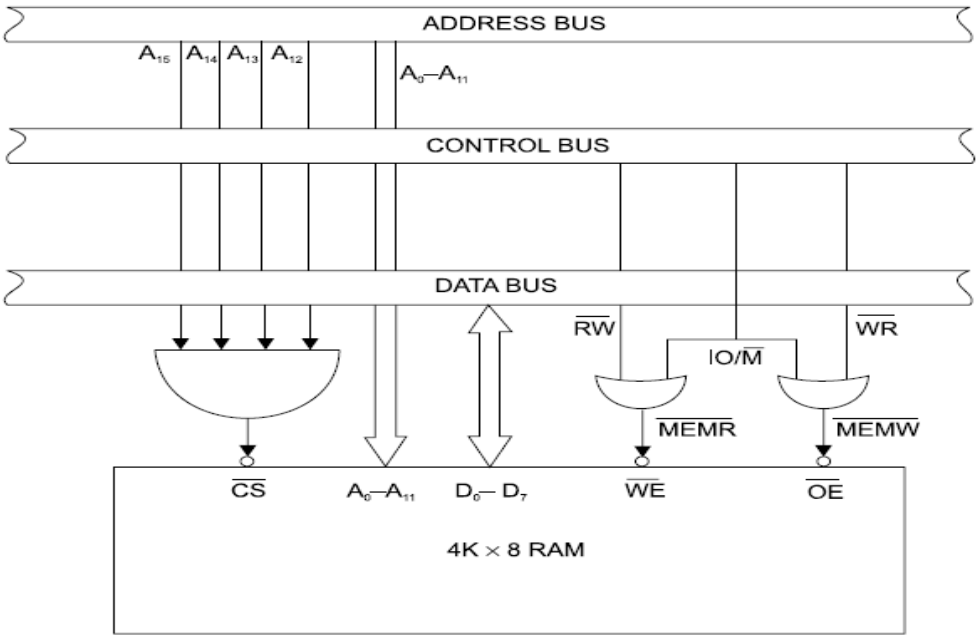


Fig: Interfacing 8085 with ROM



Interfacing 8K byte RAM with microprocessor

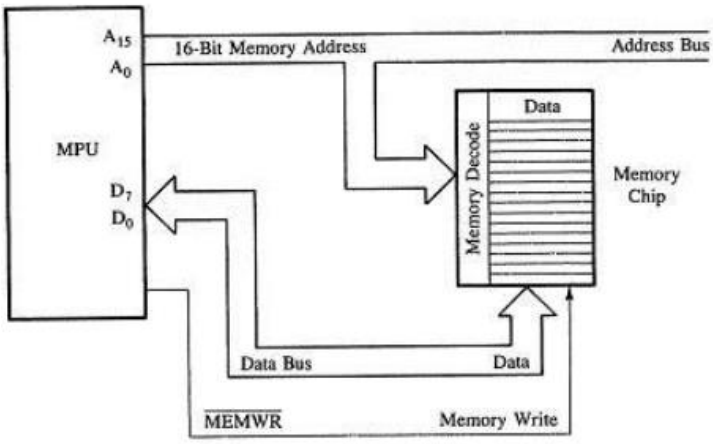


Interfacing 4K byte RAM with microprocessor

How the MPU writes into and Read from memory?

- **To write a byte into a memory location from 8085 MPU**
  1. Places the 16-bit address on the address bus of the memory location where a byte is to be stored This address is decoded to select the memory chip, and the memory register is identified.
  2. Places the byte on the data bus.
  3. Sends the control signal ($\overline{MEMW}$) to enable the input buffers of the memory and then stores the byte.

- **To read from memory, the steps are similar**.
  1. The MPU places the 16-bit address on the address bus and sends the control signal $\overline{MEMR}$ to enable the output buffer of the memory chip.
  2. The interfacing logic of the memory chip decodes the address and selects the appropriate memory register.
  3. The memory chip places the data byte on the data bus, and the MPU reads the data byte.



Memory Write Operation

****

Explain I/O Interfacing in 8085 .

****

4. 8085 Interfacing with I/O devices

There are various communication devices like the keyboard, mouse, printer, etc. So, we need to interface the keyboard and other devices with the microprocessor by using latches and buffers. This type of interfacing is known as I/O interfacing.

Microprocessor needs to Identify I/O devices with binary number. I/O devices can be interfaced in three steps.

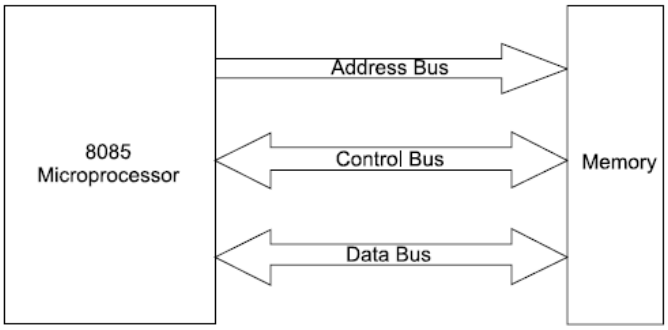
1. Identify the memory location (with address).
2. Generate timing and control signal.
3. Data transfer takes place.

The techniques used for I/O interfacing are

1. Memory mapped I/O
2. I/O mapped I/O or Peripheral mapped I/O

1. Memory-Mapped I/O

- In memory mapped I/O, each device has an address just like a memory location.
- The memory map (64K) is shared between I/O device and system memory.
- Instructions STA/.LDA and MOV are used for data transfer.
- Device is identified by 16-bit address (Space ranges from 0000H –FFFFH).

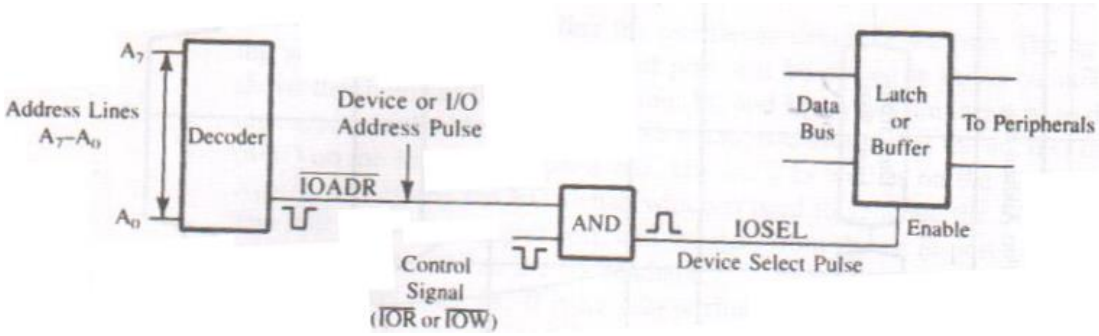


Memory mapped I/O scheme

2. Peripheral Mapped I/O

- 256 input device and 256. output device can be connected.
- It has separate numbering scheme for I/O devices.
- Instructions IN & OUT are used for data transfer.
- Device is identified by 8-bit address (Space ranges from 00H –FFH).

Device Selection and Data Transfer



Data transfer steps

❖ **For data transfer from input device to processor the following operations are performed**

1. The MPU places an 8-bit device address on address bus then decoded.
2. The MPU sends a control signal (IOR or IOW) to enable the I/O device.
3. Data are placed on the data bus for transfer.

❖ **Data transfer from processor to output device the following operations are performed.**

To send data to O/P device :-

1. The MPU places the device address (output port no.) on the address bus.

2. The MPU places data on data bus.
3. The MPU enables the output device using the control signal (IOW).
4. The O/P device latches and displays data (if O/P = LED). The other peripherals that are not enabled remain in a high impedance state called (tri-state).

Comparison between Memory mapped I/O and Peripheral mapped I/O

Memory mapped I/O	I/O mapped I/O
It uses 16 bit address	It uses 8 bit address
Memory related instructions MOV,LDA STAX are used	I/O related instructions IN and OUT are used
Data transfer between any register and I/O	Data transfer between accumulator and I/O
More hardware is needed to decode16 bit address	Less hardware is needed to decode 8 bit address
Memory map 64K is shared	256 I/P devices & 256 O/P devices are shared.

****

Write short notes on Data transfer concept

****

4.2. Data Transfer Concepts

- The 8085 microprocessor is a parallel device. That means it transfers eight bits of data simultaneously over eight data lines (parallel I/O mode).
- However in many situations, the parallel I/O mode is either impractical or impossible. For example, parallel data communication over a long distance becomes very expensive.
- Similarly, parallel data communication is not possible with devices such as CRT terminal or Cassette tape etc

Types of Data transfer scheme

Data transfer between microprocessor to memory and microprocessor to I/O devices is explained in the following ways
The data transfer can be classified into

1. **Parallel data transfer**
2. **Serial data transfer**

I. Parallel data transfer

Parallel data transfer scheme is faster than serial I/O transfer. in parallel data transfer 8-bit data send all together with 8 parallel wire. It is further divided into

- Programmed I/O
- Interrupt I/O
- DMA

Programmed I/O:

- ✓ Here the processor has to check whether the I/O device is ready or not through the Ready signal of the I/O device.
- ✓ If the ready signal is high then it will send the data to the I/O device.
- ✓ Otherwise it will continuously check the Ready signal.
- ✓ The processor is busy in checking the Ready signal.
- ✓ The drawback is wastage of time.

Interrupt I/O:

- ✓ In this method the I/O device will interrupt the Processor through the INTR signal to indicate to the processor that it is ready to accept the next data.
- ✓ Then the processor will send the INTA signal.
- ✓ Then the processor stops its normal execution and start transferring the data to the I/O device.

DMA:

- ✓ Using DMA I/O device can directly transfer the data to the Memory using the Address and Data buses of Processor.

II. Serial data Transfer

- ✓ Some of the external I/0 devices receive only the serial data. Normally serial communication is used in the Multiprocessor environment.
- ✓ In serial I/O mode transfer a single bit of data on a single line at a time. For serial I/O data transmission mode, 8-bit parallel word is converted to a stream of eight serial bit using parallel-to-serial converter.
- ✓ Similarly, in serial reception of data, the microprocessor receives a stream of 8-bit one by one which are then converted to 8- bit parallel word using serial-to-parallel converter.8051 has two pins for serial communication.
  - ❖ SID- Serial Input data.
  - ❖ SOD-Serial Output data

****

Explain the interrupt structure of 8085.(Dec 2013, April 2015,Dec 2015,June 2014,June 2016)
Explain the interrupts of 8085 with its types with interrupt service routine.(April 2018,Dec 2018)

****

5. Interrupts in 8085

5.1. Interrupt

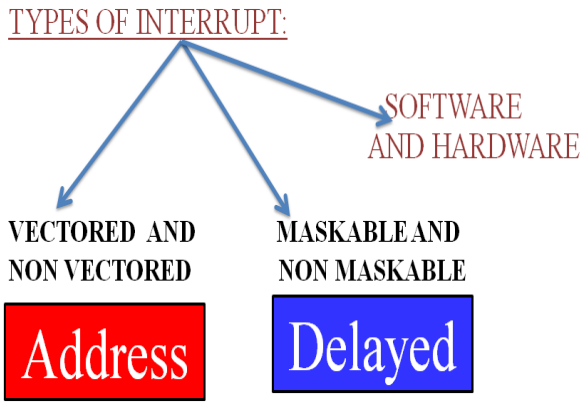
Definition:

1. Interrupt is the mechanism by which the processor is made to transfer control from its current program execution to another program having higher priority. The interrupt signal may be given to the processor by any external peripheral device

2. Interrupts are the signals generated by the external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR.

5.2. Types of interrupt

Interrupt are classified into following groups based on their parameter.



Vector and Non-Vector interrupt

❖ **Vector interrupt** − In this type of interrupt, the interrupt address is known to the processor .For example: RST7.5, RST6.5, RST5.5, TRAP.

- The address to which program control is transferred are

• Name	Vectored address
• TRAP	0024 (4.5 X0008)
• RST 5.5	002C (5.5 X 0008)
• RST 6.5	0034 (6.5 X 0008)
• RST 7.5	003C (7.5 X 0008)

❖ **Non-Vector interrupt** − In this type of interrupt, the interrupt address is not known to the processor so, the interrupt address needs to be sent externally by the device to perform interrupts. **For example:** INTR.

Maskable and Non-Maskable interrupt

- **Maskable interrupt** − In this type of interrupt, we can disable the interrupt by writing some instructions into the program. **For example:** RST7.5, RST6.5, RST5.5.
- **Non-Maskable interrupt** − In this type of interrupt, we cannot disable the interrupt by writing some instructions into the program. **For example:** TRAP.

The 'EI' instruction is a one byte instruction and is used to Enable the maskable interrupts.

The 'DI' instruction is a one byte instruction and is used to Disable the maskable interrupts

Software and Hardware Interrupt

- **Software interrupt** − In this type of interrupt, the programmer has to add the instructions into the program to execute the interrupt. There are 8 software interrupts in 8085, i.e. RST0, RST1, RST2, RST3, RST4, RST5, RST6, and RST7.
- **Hardware interrupt** − There are 5 interrupt pins in 8085 used as hardware interrupts, i.e. TRAP, RST7.5, RST6.5, RST5.5, INTA.

Note − NTA is not an interrupt, it is used by the microprocessor for sending acknowledgement. TRAP has the highest priority, then RST7.5 and so on.

5.3. Priority of interrupt

Interrupt	Priority
TRAP	1
RST 7.5	2
RST 6.5	3
RST 5.5	4
INTR	5

5.4. Interrupt Service Routine (ISR)

A small program or a routine that when executed, services the corresponding interrupting source is called an ISR.

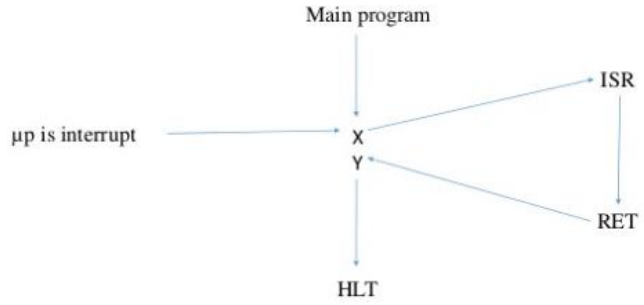


Figure:Interrupt service routine procedure

1. It allows the external devices to interrupt the normal program execution of the microprocessor.

2. When microprocessor receives interrupt signal, it temporarily stops current program and starts executing new program indicated by the interrupt signal.

3. Interrupt signals are generated by external peripheral devices like keyboard, sensors, printers etc.

4. After execution of the new program, microprocessor returns back to the previous program.

5.5. Interrupt structure of 8085

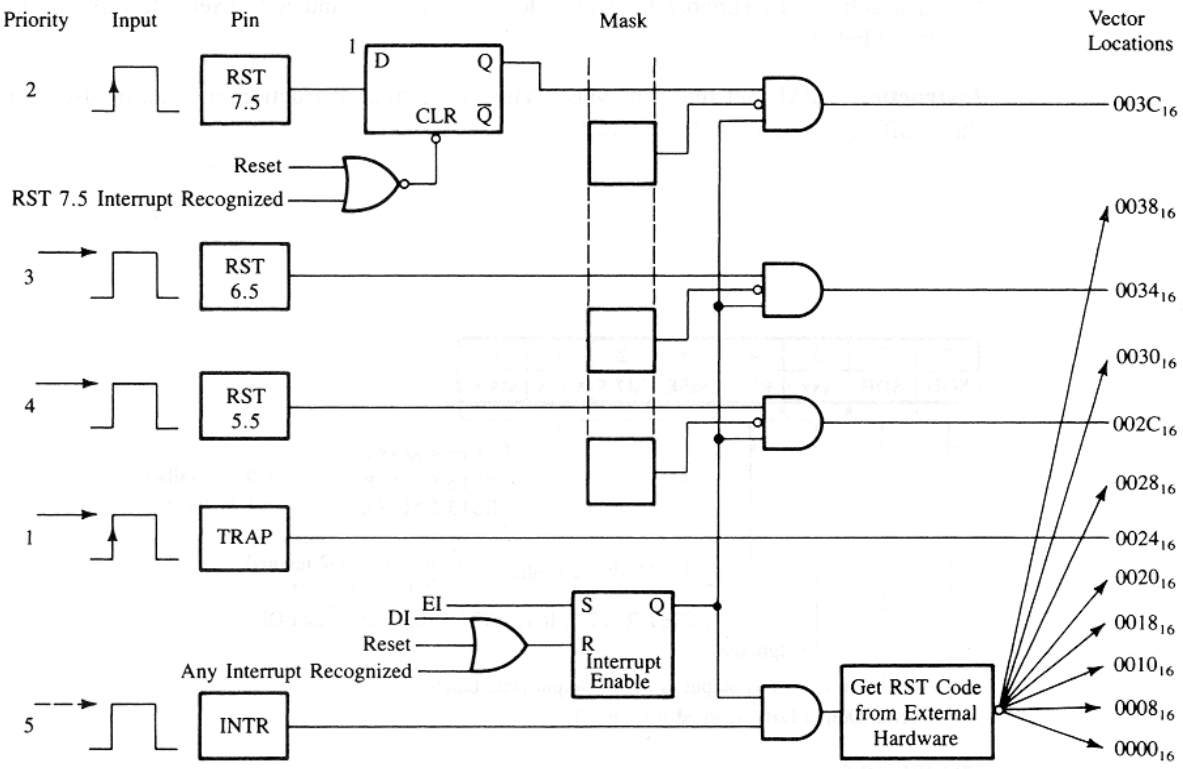


Figure: Interrupt structure of 8085

TRAP

It is a non-maskable interrupt, having the highest priority among all interrupts. Bydefault, it is enabled until it gets acknowledged. In case of failure, it executes as ISR and sends the data to backup memory. This interrupt transfers the control to the location 0024H.

RST7.5

It is a maskable interrupt, having the second highest priority among all interrupts. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 003CH address.

RST 6.5

It is a maskable interrupt, having the third highest priority among all interrupts. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 0034H address.

RST 5.5

It is a maskable interrupt. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 002CH address.

INTR

It is a maskable interrupt, having the lowest priority among all interrupts. It can be disabled by resetting the microprocessor.

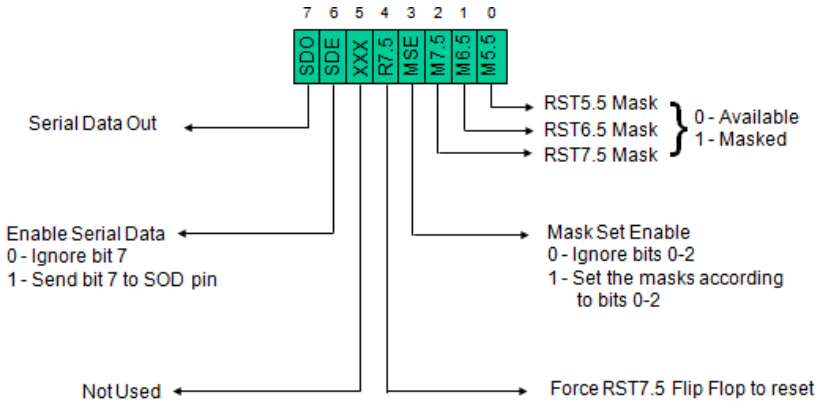
When **INTR signal goes high**, the following events can occur −

- The microprocessor checks the status of INTR signal during the execution of each instruction.
- When the INTR signal is high, then the microprocessor completes its current instruction and sends active low interrupt acknowledge signal.
- When instructions are received, then the microprocessor saves the address of the next instruction on stack and executes the received instruction.
- The Interrupt Enable flip flop is manipulated using the EI/DI instructions.
- The individual masks for RST 5.5, RST 6.5 and RST 7.5 are manipulated using the SIM instruction.

This instruction takes the bit pattern in the Accumulator and applies it to the interrupt mask enabling and disabling the specific interrupts

SIM Instruction:

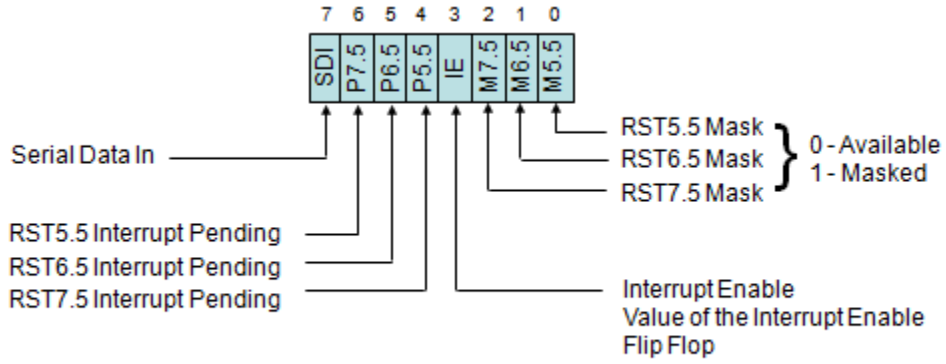
The SIM instruction is used to mask or unmask RST hardware interrupts. When executed, the SIM instruction reads the content of accumulator and accordingly mask or unmask the interrupts. The format of control word to be stored in the accumulator before executing SIM instruction is as shown in Fig.



RIM Instruction:

RIM instruction is used to read the status of the interrupt mask bits. When RIM instruction is executed, the accumulator is loaded with the current status of the interrupt masks and the pending interrupts. The format and the meaning of the data stored in the accumulator after execution of RIM instruction is shown in Fig

- If the mask bit is 0, the interrupt is available.
- If the mask bit is 1, the interrupt is masked



Ex: Write an assembly language program to enables all the interrupts in 8085 after reset.

```
EI          : Enable interrupts
MVI A, 08H  : Unmask the interrupts
SIM         : Set the mask and unmask using SIM instruction
```

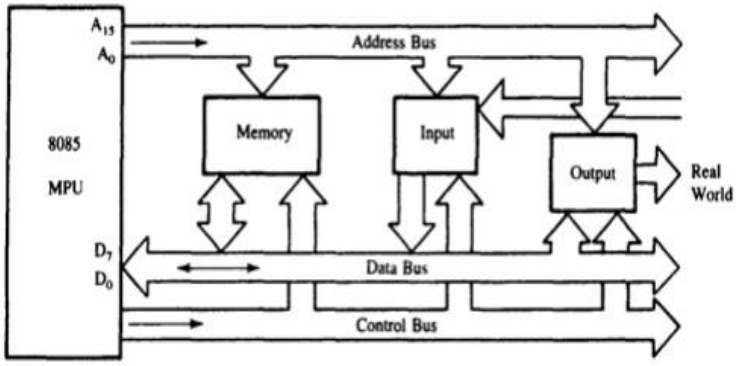
****

Basic operations of Microprocessor

The microprocessor performs primarily four operations:

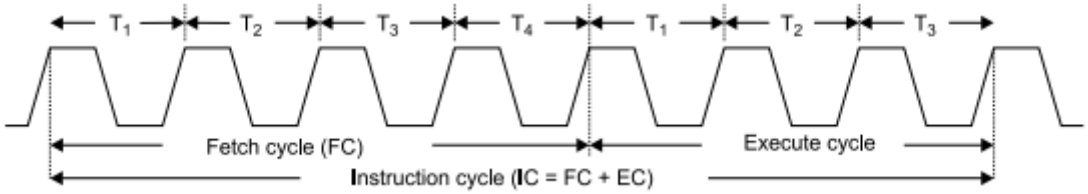
I. Memory Read: Reads data (or instruction) from memory.
II. Memory Write: Writes data (or instruction) into memory.
III. I/O Read: Accepts data from input device.
IV. I/O Write: Sends data to output device.

The 8085 processor performs these functions using address bus, data bus and control bus as shown in Fig



Timing Diagram

- Timing Diagram is a graphical representation.
- It represents the execution time taken by each instruction in a graphical format.
- The execution time is represented in T-states.



1. **Instruction cycle:** this term is defined as the number of steps required by the CPU to complete the entire process ie. Fetching and execution of one instruction. The fetch and execute cycles are carried out in synchronization with the clock.

2. **Machine cycle:** It is the time required by the microprocessor to complete the operation of accessing the memory devices or I/O devices. In machine cycle various operations like opcode fetch, memory read, memory write, I/O read, I/O write are performed.
3. **T-state:** Each clock cycle is called as T-states.

MACHINE CYCLES OF 8085

The 8085 microprocessor has 5 basic machine cycles. They are

1. Opcode fetch cycle (4T)
2. Memory read cycle (3 T)
3. Memory write cycle (3 T)
4. I/O read cycle (3 T)
5. I/O write cycle (3 T)

Opcode fetch

- The microprocessor requires instructions to perform any particular action.
- In order to perform these actions microprocessor utilizes Opcode which is a part of an instruction which provides detail (ie.which operation µp needs to perform) to microprocessor.

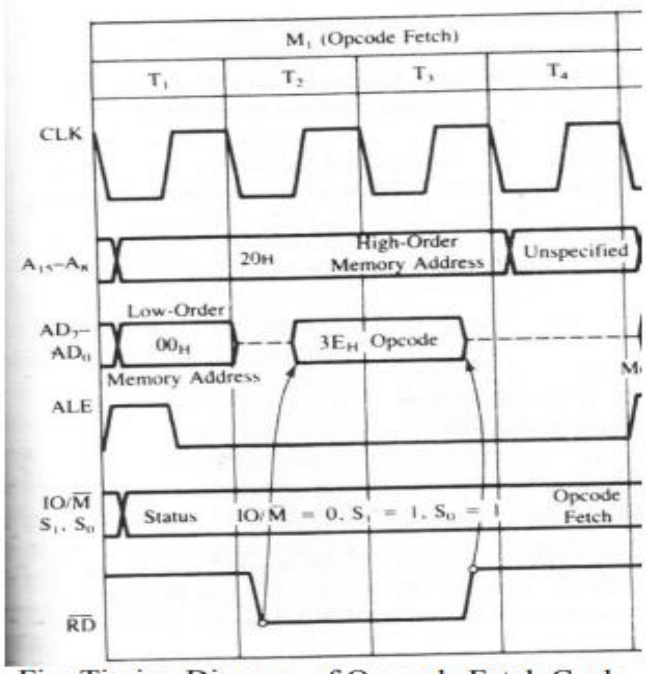
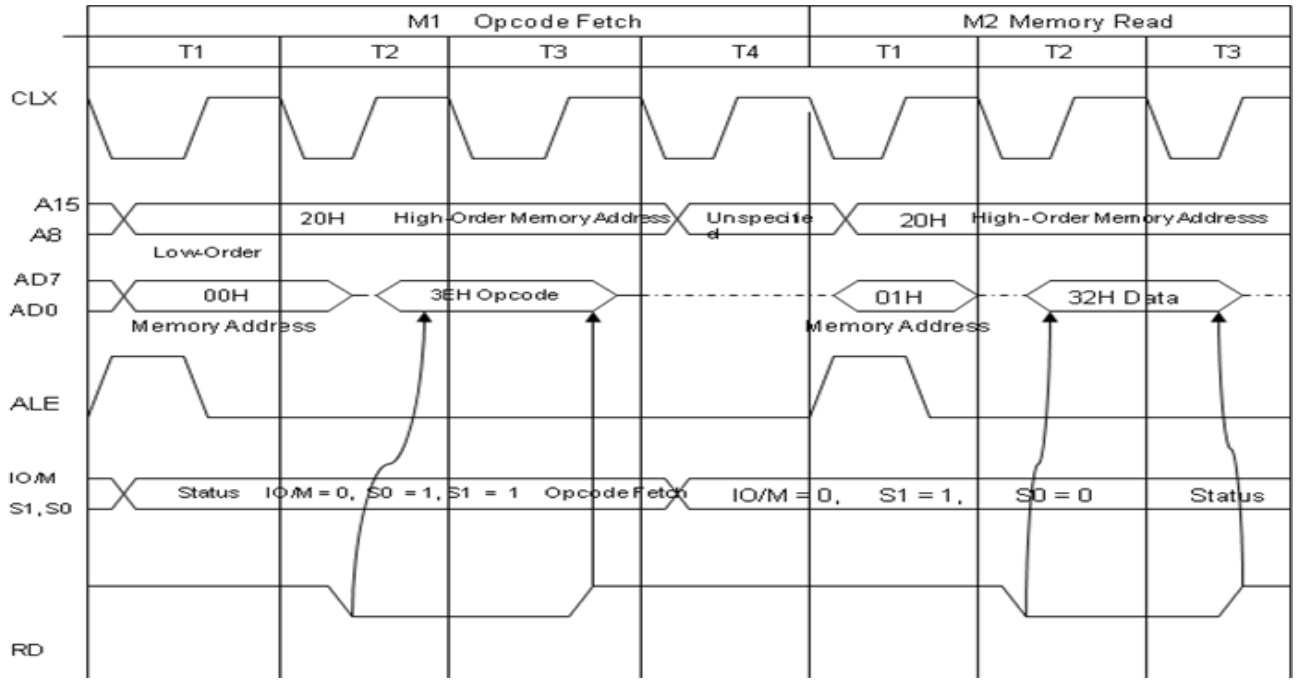


Fig: Timing Diagram of Op-code Fetch Cycle

Memory Read

- For example MVI A,32 (April 2018)

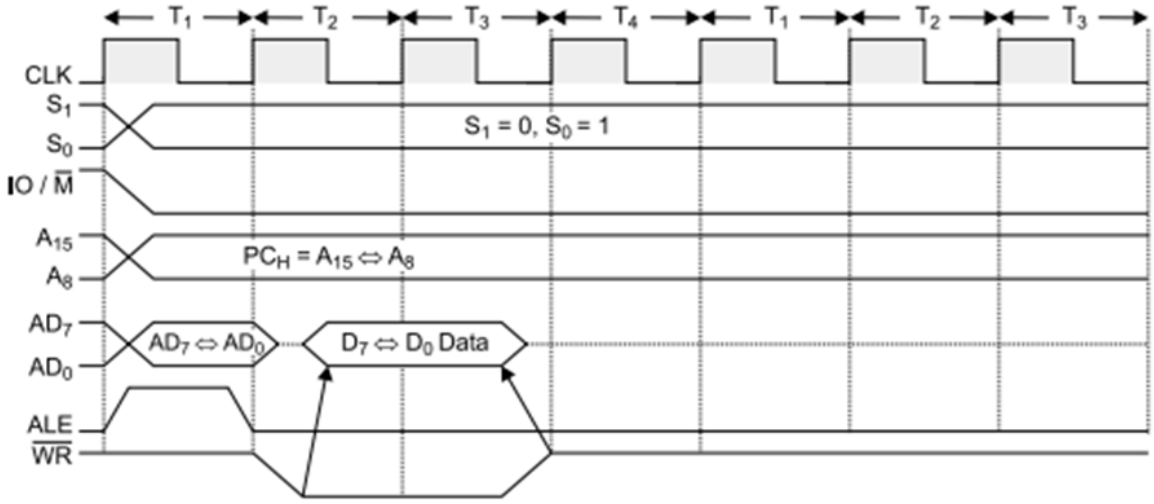


- The total cycle consists of 7 T states and 2 machine cycles.opcode fetch and memory read.

- At the end of opcode fetch the PC is incremented thus the address is 2001H and the instruction decoder has 3EH.Now the operand is to be read from the memory to Register A.
- The 2nd m/c cycle are similar to first 3 states of opcode exept the status signal(S0=0 and S1=1)

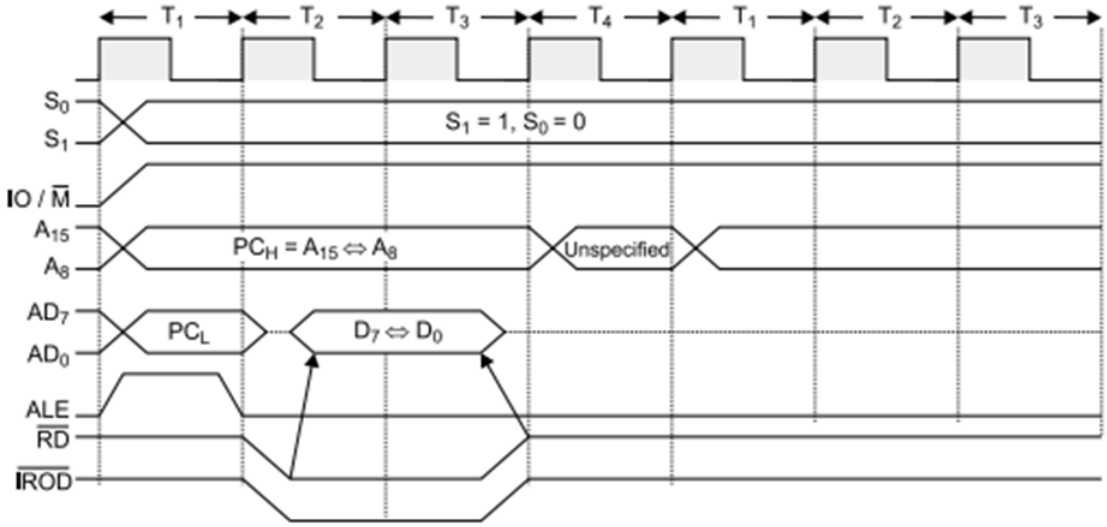
Memory Write

- The memory write machine cycle is executed by the processor to write a data byte in a memory location.
- The processor takes, 3T states to execute this machine cycle



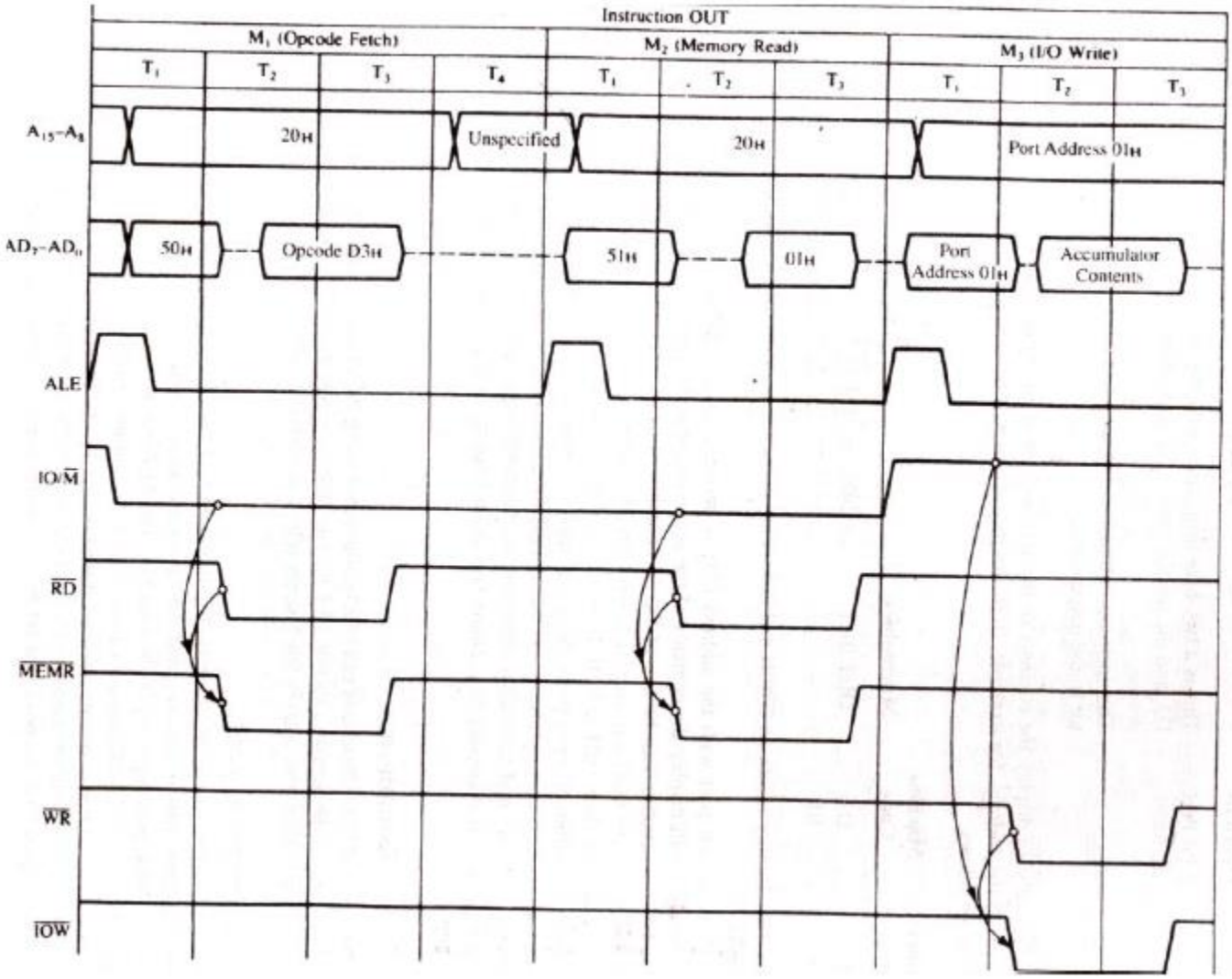
I/O read

- The I/O Read cycle is executed by the processor to read a data byte from I/O port or from the peripheral.
- The processor takes 3T states to execute this machine cycle.
- The IN instruction uses this machine cycle during the execution.

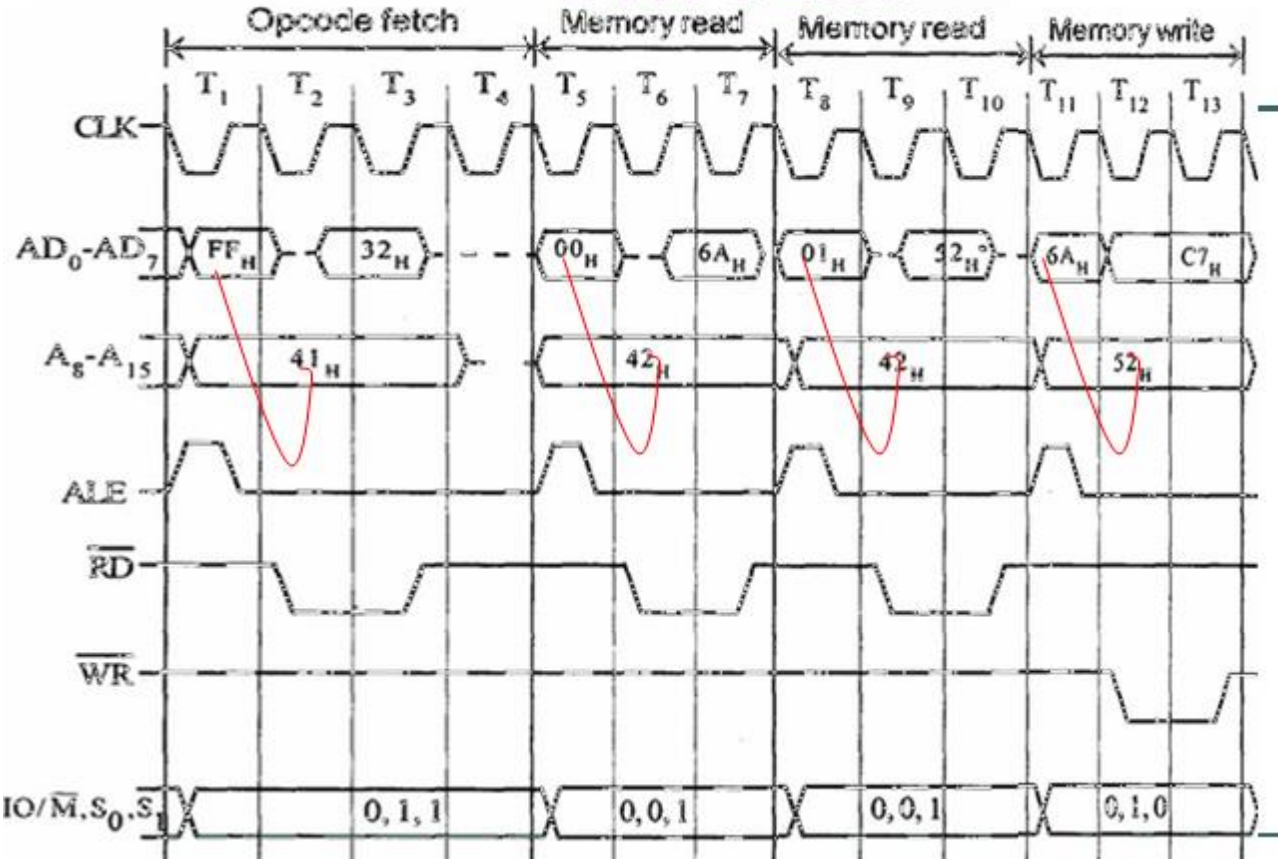


I/O write

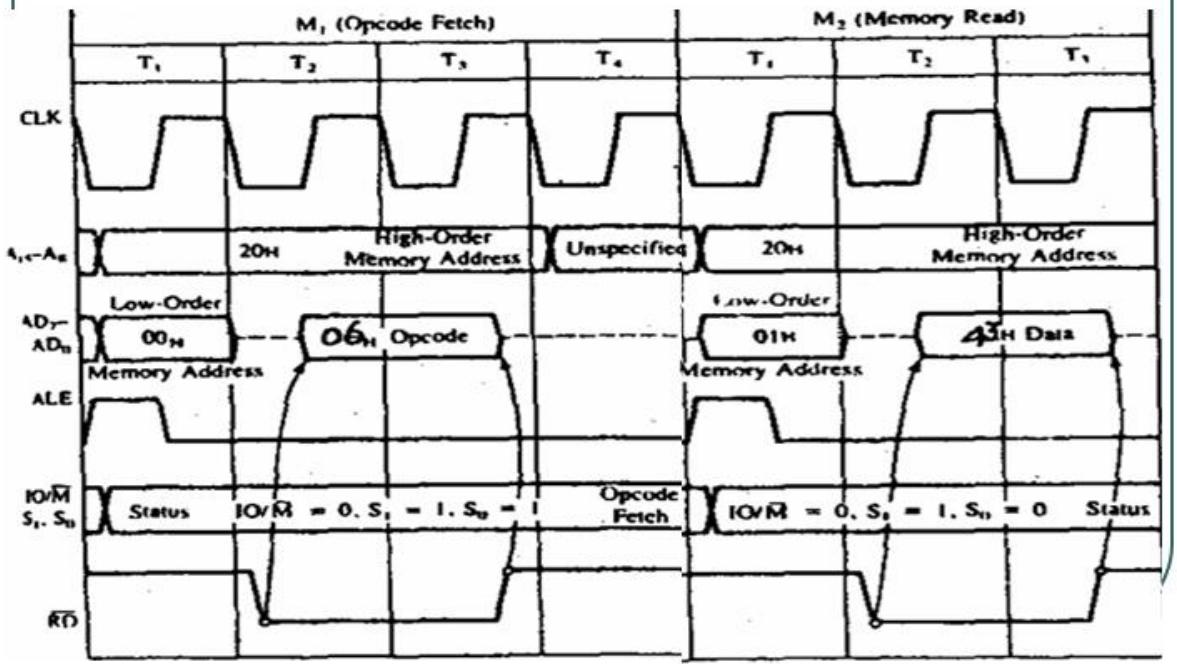
- Example OUT 01H stored at memory location 2050.The opcode of the instruction is D3
- The processor takes, 10T states to complete the cycle.It requires 3 machine cycle opcode fetch,memory read and memory write.



STA 526A



MVI B, 43



UNIT II
PROGRAMMING OF 8085 PROCESSOR

Instruction -format and addressing modes – Assembly language format – Data transfer, data manipulation& control instructions – Programming: Loop structure with counting & Indexing – Look up table - Subroutine instructions – stack

****

8085 Instruction Format

1. Instruction:

- It is a command given to the microprocessor to perform given task on specified data.Each instruction has two parts viz. task to be performed known as operation code or **opcode** and second is the data to be operated upon known as **operand**.
- The Operand can be used in many different ways e.g. 8 bit data or 16 bit data or internal register or memory location or 8 bit or 16 bit address.
- 8085 Instructions can be classified based on the size they occupy in memory or by the functions they perform. Figure shows the classification of the instructions.

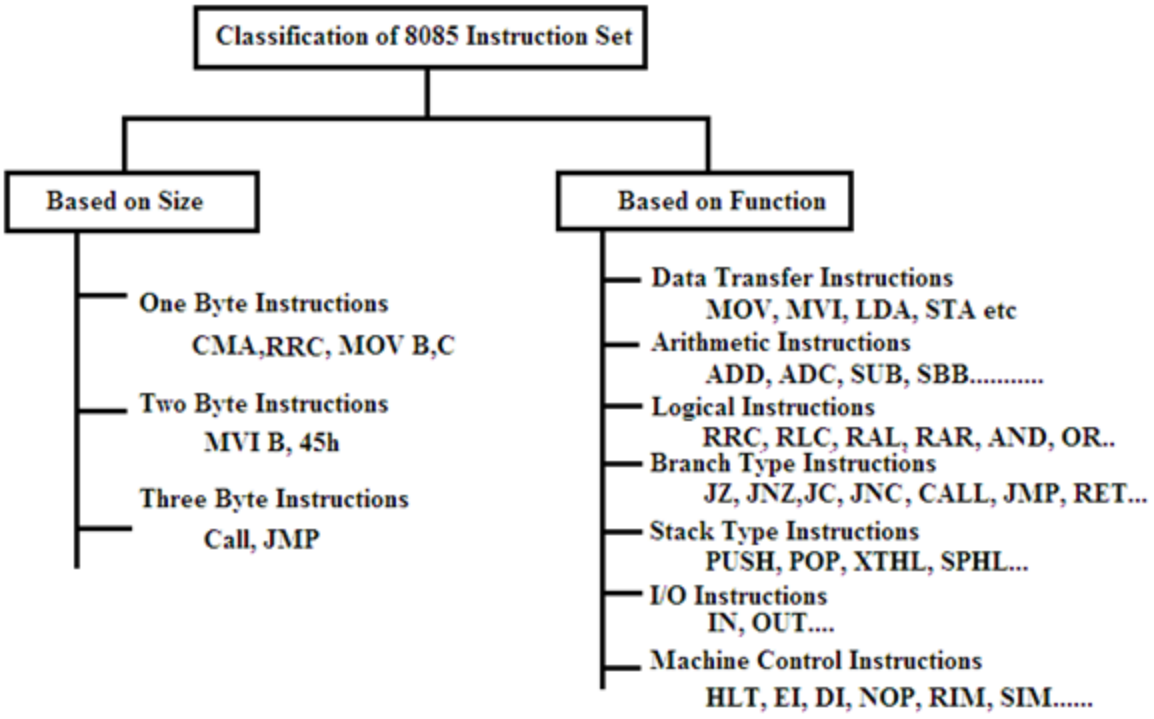


Fig: Classification of Instruction Set of 8085

****

Discuss in detail about the 8085 instruction set and explaining about the various types of operation.[December 2013,April 2011,June 2016][April 2018]

****

1.1.Instruction Format

Based on the size, the instructions can be classified are as follows

One byte Instructions:

These instructions are of one byte in size and hence occupy one memory location in RAM. Examples are CMA, RLC, RRC, RAL, RAR, STC, CMC etc. These instructions do not require any operand to be specified with the instructions; instead the operand is implied in the instructions.

One Byte Instructions				
Task	**Opcode**	**Operand**	**Binary Code**	**Hex code**
Add the contents of register B to contents of accumulator	ADD	B	1000 0000	80H
Copy contents of accumulator in register C	MOV	C,A	01001111	4FH

Two Byte Instruction:

These instructions of two byte (16-bits) in size and hence will occupy two memory locations in RAM. Examples of such instructions are MVI C, 0A;

Two Byte Instructions				
Task	**Opcode**	**Operand**	**Binary Code**	**Hex code**
Load 8 bit data byte in the accumulator	MVI	A, data	0011 1110, data	3E, data

Three Byte Instructions:

These are of three byte in size and hence occupy three locations in memory (RAM). Examples of such instructions are CALL, JMP etc.

Three Byte Instructions				
Task	**Opcode**	**Operand**	**Binary Code**	**Hex code**
Transfer the program sequence to memory location 2085H.	JMP	2085H	11000011 10000101 00100000	C3 85 20

****

Explain the classification of Instruction set with example.
Explain logical instruction with example. (December 2015)

****

2. Instruction Set Classification

An **instruction** is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions, called the **instruction set**, determines what functions the microprocessor can perform.

These instructions can be classified into the following five functional categories:

- Data transfer (copy) operations,
- Arithmetic operations,
- Logical operations,
- Branching operations, and
- Machine-control operations.

1 Data Transfer Croup

The data transfer instructions move data between registers or between memory and registers.

- MOV Move
- MVI Move Immediate
- LDA Load Accumulator Directly from Memory
- STA Store Accumulator Directly in Memory
- LHLD Load H & L Registers Directly from Memory
- SHLD Store H & L Registers Directly in Memory

An 'X' in the name of a data transfer instruction implies that it deals with a register pair (16-bits);

- LXI Load Register Pair with Immediate data
- LDAX Load Accumulator from Address in Register Pair
- STAX Store Accumulator in Address in Register Pair
- XCHG Exchange H & L with D & E
- XTHL Exchange Top of Stack with H & L

2 Arithmetic Group

The arithmetic instructions add, subtract, increment, or decrement data in registers or memory.

- ADD Add to Accumulator
- ADI Add Immediate Data to Accumulator
- ADC Add to Accumulator Using Carry Flag
- ACI Add immediate data to Accumulator Using Carry
- SUB Subtract from Accumulator
- SUI Subtract Immediate Data from Accumulator

- SBB Subtract from Accumulator Using Borrow (Carry) Flag
- SBI Subtract Immediate from Accumulator Using Borrow (Carry) Flag
- INR Increment Specified Byte by One
- DCR Decrement Specified Byte by One
- INX Increment Register Pair by One
- DCX Decrement Register Pair by One
- DAD Double Register Add; Add Content of Register Pair to H & L Register Pair

3 Logical Group

This group performs logical (Boolean) operations on data in registers and memory and on condition flags. The logical AND, OR, and Exclusive OR instructions enable you to set specific bits in the accumulator ON or OFF.

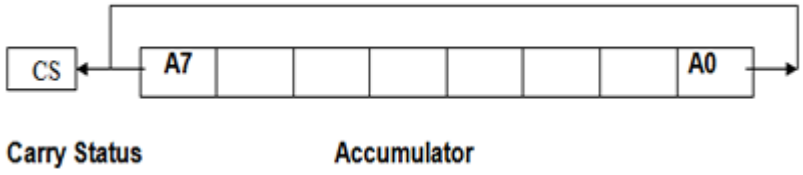
- ANA Logical AND with Accumulator
- ANI Logical AND with Accumulator Using Immediate Data
- ORA Logical OR with Accumulator
- OR Logical OR with Accumulator Using Immediate Data
- XRA Exclusive Logical OR with Accumulator
- XRI Exclusive OR Using Immediate Data

The Compare instructions compare the content of an 8-bit value with the contents of the accumulator;

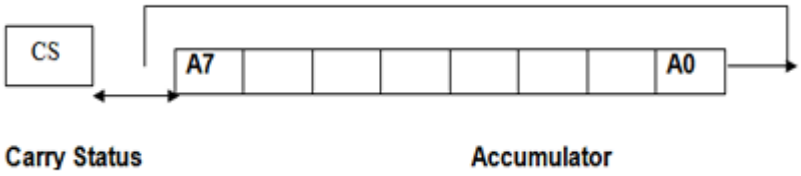
- CMP Compare
- CPI Compare Using Immediate Data

The rotate instructions shift the contents of the accumulator one bit position to the left or right:

- RLC Rotate Accumulator Left



- RRC Rotate Accumulator Right



- RAL Rotate Left Through Carry
- RAR Rotate Right Through Carry

Complement and carry flag instructions:

- CMA Complement Accumulator
- CMC Complement Carry Flag
- STC Set Carry Flag

4 Branch Group

The branching instructions alter normal sequential program flow, either unconditionally or conditionally. The **unconditional branching instructions** are as follows:

- JMP Jump
- CALL Call
- RET Return

Conditional branching instructions examine the status of one of four condition flags to determine whether the specified branch is to be executed. The conditions that may be specified are as follows:

- NZ Not Zero (Z = 0)
- Z Zero (Z = 1)
- NC No Carry (C = 0)
- C Carry (C = 1)
- PO Parity Odd (P = 0)
- PE Parity Even (P = 1)
- P Plus (S = 0)
- M Minus (S = 1)

Thus, the conditional branching instructions are specified as follows:

Jumps	**Calls**	**Returns**
INC	CNC	RNC (No Carry)
JNZ	CNZ	RNZ (Not Zero)
JM	CM	RM (Minus)
JP0	CPO	RPO (Parity Odd)
JM	CM	RM (Minus)
JPE	CPE	RPE (Parity Even)
JP0	CPO	RPO (Parity Odd)

Two other instructions can affect a branch by replacing the contents or the program counter:

- PCHL Move H & L to Program Counter
- RST Special Restart Instruction Used with Interrupts

5 .Stack Instructions

The following instructions affect the Stack and/or Stack Pointer

- PUSH Push Two bytes of Data onto the Stack
- POP Pop Two Bytes of Data off the Stack
- XTHL Exchange Top of Stack with H & L
- SPHL Move content of H & L to Stack Pointer

6 .I/0 instructions

- IN Initiate Input Operation
- OUT Initiate Output Operation

7 Machine Control instructions

- EI Enable Interrupt System
- DI Disable Interrupt System
- HLT Halt
- NOP No Operation

****

Define addressing mode.Write the types of addressing modes with example.(December 2014)(December 2015) (April 2015)(April 2018)(Dec 2018)

****

3. Addressing Modes

Various ways of specifying the operands or various formats for specifying the operands is called addressing mode

- **Implicit addressing**

  This mode doesn't require any operand; the data is specified by the opcode itself.

  - CMA – Complement the contents of accumulator

- **Immediate addressing**

  In this mode, the 8/16-bit data is specified in the instruction itself as one of its operand.

  - MVI B, 05H means 05 is copied into register B.
  - ADI 06H

- **Direct addressing** –

  In this mode, the data is directly copied from the given address to the register.

  - STA 2400H, IN 02H
  - STA 2400H means the data at address 2400 is copied to register A.

- **Register addressing**

  In this mode, the data is copied from one register to another.

  - MOV A, B
  - ADD B

- **Register indirect addressing**

  In this mode, the data is transferred from one register to another by using the address pointed by the register.

  - LDAX B,
  - STAX D

  LDAX B means data is transferred from the memory address pointed by the register pair BC to the register A.

****

Write short notes on STACK and Subroutine

****

4. STACK:

Stack is an area of memory identified by the programmer for temporary storage of information.

- Stack is a LIFO structure.
- Stack normally grows backwards into memory-the programmer defines the bottom of the stack and the stack grows up into reducing address range.
- Stack is defined by setting the SP(stack pointer) register.LXI SP,FFFFH
- The size of the stack is limited only by the available memory.
- Information is saved on the stack by PUSHing it on.
- Information is retrieved from the stack by POPing it off.
- The 8085 provides two instructions:PUSH and POP for storing information on the stack and retrieving it back.
- Both PUSH and POP work with register pairs only.

PUSH instruction

EX: PUSH B

Steps to be followed for PUSH B one byte instruction

- Decrement SP

- Copy the contents of register B to the memory location pointed by SP.
- Decrement SP
- Copy the contents of register C to the memory location pointed to by SP.

POP instruction:

EX: POP D

Steps to be followed for POP D one byte instruction

- Copy the contents of the memory location pointed to by the SP to register E.
- Increment SP
- Copy the contents of the memory location pointed to by the SP to register D.
- Increment SP

Operation of the stack:

- During pushing, the stack operates in a ' decrement then store' style-The stack pointer is decremented first, then the information is placed on the stack
- During popping, the stack operates in a "use then increment' style.-The information is retrieved from the top of the stack and then the pointer is incremented.
- The SP pointer always points to the "top of the stack"

LIFO:

The order of PUSHs and POPs must be opposite of each other in order to retrieve information back into its original location.

```
PUSH B
PUSH D
.....
POP D
POP B
```

Reversing the order of the POP instructions will result in the exchange of the contents of BC and DE

PSW register pair:

The 8085 recognizes one additional register pair called the PSW(PROGRAM STATUS WORD). This register pair is made up of the accumulator and the flag registers.

****

Explain the use of lookup table in 8085.

****

5. Lookup table

- A lookup table is an array that replaces runtime computation with a simpler array indexing operation.
- The savings in terms of processing time can be significant, since retrieving a value from memory is often faster than undergoing an 'expensive' computation or input/output operation.
- The tables may be pre-calculated and stored in static program storage, calculated (or "pre-fetched") as part of a program's initialization phase (memorization), or even stored in hardware in application-specific platforms.
- Lookup tables are also used extensively to validate input values by matching against a list of valid (or invalid) items in an array and, in some programming languages, may include pointer functions (or offsets to labels) to process the matching input.

Example of look up table Algorithm

1. Initialize HL pair to point Look up table
2. Get the data
3. Check whether the given input is less than 9
4. If yes go to next step else halt the program
5. Add the desired address with the accumulator content
6. Store the result

Program:

```
        LXI H,5000  ;Initialsie Look up table address
        LDA 5050    ;Get the data
        CPI 0A      ;Check input > 9
        JC AFTER    ;if yes error
        MVI A,FF    ;Error Indication
        STA 5051
        HLT
AFTER:  MOV C,A     ;Add the desired Address
        MVI B,00
        DAD B
```

```
        MOV A,M
        STA 5051    ;Store the result
        HLT         ;Terminate the program
```

LOOK UP TABLE:

```
5000  01
5001  04
5002  09
5003  16
5004  25
5005  36
5006  49
5007  64
5008  81
```

RESULT:

Input: Data : 05H in memory location 5050

Output: Data : 25H (Square of 5) in memory location 5051

Input: Data : 11H in memory location 5050

Output: Data : FFH (Error Indication) in memory location 5051

****

6. Sample 8085 Assembly Programs

Example-1: Write assembly program to add two numbers. (December 2014)

```
MVI D, 8CH
MVI C, 6EH
MOV A, C
ADD D
OUT PORT1
HLT
```

Example-2: Write assembly program to multiply a number by 8
Multiply by 2 is equivalent to shifting.

```
MVI A, 40H
RLC
RLC
RLC
OUT PORT1
HLT
```

Example-2: Write assembly program to multiply two 8 bit number

```
        MVI B, 07H
```

```
        MVI C, 06H
        XRA A
GO      ADD B
        DCR C
        JNZ GO
        STA 4A00
        HLT
```

Example-3: Write assembly program to find greatest between the two numbers.

```
MVI B, 30H
MVI C, 40H
MOV A, B
CMP C
JZ EQU
JC GRT
OUT PORT1
HLT
EQU: MVI A, 01H
OUT PORT1
HLT
GRT: MOV A, C
OUT PORT1
HLT
```

Write an assembly language program for to generate Fibannoci series using subroutines(Dec 2014)

```
MVI A,00
STA 8000
MVI A,01
STA 8001
MVI B,08
LXI H,8000
BACK: MOV A,M
INX  H
ADD M
INX H
MOV M,A
DCR B
DCX H
JNZ BACK
HLT
```

Write an assembly language program for to multiply two 16 bit numbers. (June 2016)

```
        LXIH, 0000
        LXIB, 1CD2
        LXI SP,01AD

        LXI D, 0000
MUL     DAD SP
        JNC DOWN
```

```
        INX D
DOWN    DCX B
        MOV A,B
        ORA C
        JNZ MUL
        SHLD   4A00
        XCH G
        SHLD  4A02
        HLT
```

Programming using subroutine Instructions
Generation of Square waveform using DAC

ADDRESS	LABEL	MNEMONICS	OPCODE
	START	MVI A,00H	
		OUT C8	
		CALL DELAY	
		MVI A,FF	
		OUT C8	
		CALL DELAY	
		JMP START	
		MVI B,05H	
		MVI C,FF	
	DELAY	DCR C	
	L2	JNZ L1	
	L1	DCR B	
		JNL L2	
		RET	

Programming using Loop structure with Counting and Indexing

(i) 16 bit Multiplication

		INX B	
	L2	DCX D	
		MOV A,E	
		ORA D	
		JNZ L1	
		SHLD 4204	
		MOV L,C	
		MOV H,B	
		SHLD 4206	
		HLT	

ADDRESS	LABEL	MNEMONICS	OPCODE
	START	LHLD 4200	
		SPHL	
		LHLD 4202	
		XCHG	
		LXI H,0000	
	L1	LXI B,0000	
		DAD SP	
		JNC L2	

(ii)Finding the maximum number in the given array

ADDRESS	LABEL	MNEMONICS	OPCODE
	START	LDA 4500	
		MOV C, A	
		LXI H, 4501	
	L2	MOV A, M	
	L3	DCR C	
		INX H	
		JZ L1	
		CMP M	
		JC L2	
		JMP L3	
	L1	STA 4520	
		HLT	

Develop an algorithm and 8085 assembly language program to sort 100 byte type data. Explain the instruction used in the program.(Dec 2018)

```
L3      MVI B,00
        LXI H,4200
        MOV C,M
        INX H
        DCR C
L2      MOV A,M
        INX H
        CMP M
        JC      L1
        MOV D,M
        MOV M,A
        DCX H
        MOV  M,D
        MVI B,01
L1      DCR C
        JNZ     L2
        DCR B
        JZ    L3
        HLT
```

INPUT

4200 99

4201 data1

…….

4300 data 100

&&&&&&&&&&&&&&&&&&&&&&&&&&&&

UNIT III

MICROCONTROLLER

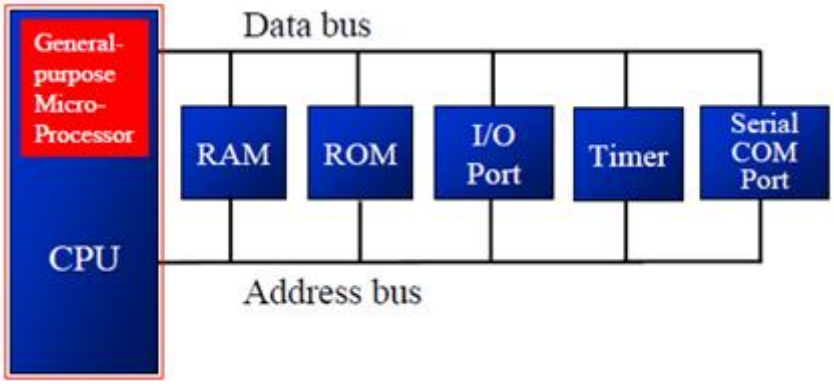
Hardware Architecture, pinouts – Functional Building Blocks of Processor – Memory organization – I/O ports and data transfer concepts– Timing Diagram – Interrupts- Data Transfer, Manipulation, Control Algorithms& I/O instructions, Comparison to Programming concepts with 8085.

INTRODUCTION:

General-purpose microprocessor contains

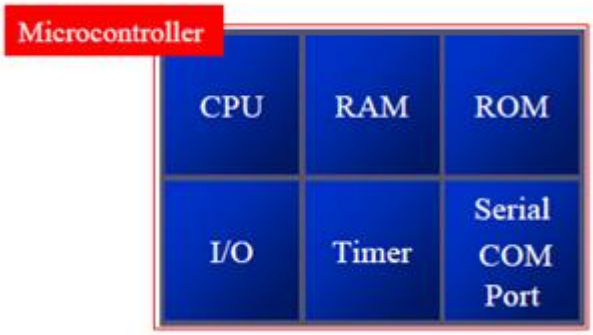
- No RAM
- No ROM
- No I/O ports

It must add RAM, ROM, I/O ports, and timers externally to make them functional.It makes the system bulkier and much more expensive.It has the advantage of versatility on the amount of RAM, ROM, and I/O ports



Microcontroller has

- CPU (microprocessor)
- RAM
- ROM
- I/O ports
- Timer
- ADC and other peripherals



The fixed amount of on-chip ROM, RAM, and number of I/O ports makes them ideal for many applications in which cost and space are critical. In many applications, the space it takes, the power it consumes, and the price per unit are much more critical considerations than the computing power.

What is a microcontroller?

A device which contains the microprocessor with integrated peripherals like memory, serial ports, parallel ports, timer/counter, interrupt controller, data acquisition interfaces like ADC, DAC is called microcontroller

Comparison between Microprocessor and Micro controller

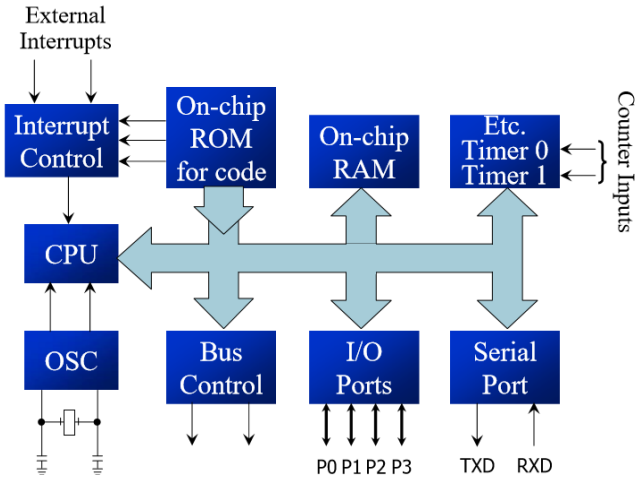
Microprocessor	Microcontroller
1 Microprocessor contains ALU, general purpose registers, stack pointer, program counter, clock timing circuit and interrupt circuit	1.Microcontroller contains the circuitry of microprocessor and in addition ithas built- in ROM, RAM, I/Odevices, timers and counters.
2 It has many instructions to move data between memory and CPU.	2. It has one or two instructions to movedata between memory and CPU.
3 It has one or twobithandlinginstructions.	3. It has many bit handling instructions.
4 Access times for memory and I/O devices are more	4.Less access times for built-in memoryand I/O devices
5 Microprocessor based system requires more hardware	5.Microcontroller based system requiresless hardware reducing PCB size andincreasing the reliability.

****

Draw and Explain of architecture of 8051 microcontroller. (Nov 2010/May 2010,May 2015) (MAY/JUNE 2016),(NOV/DEC 2014),(APRIL/MAY 2017)(April 2015)(April 2018)

****

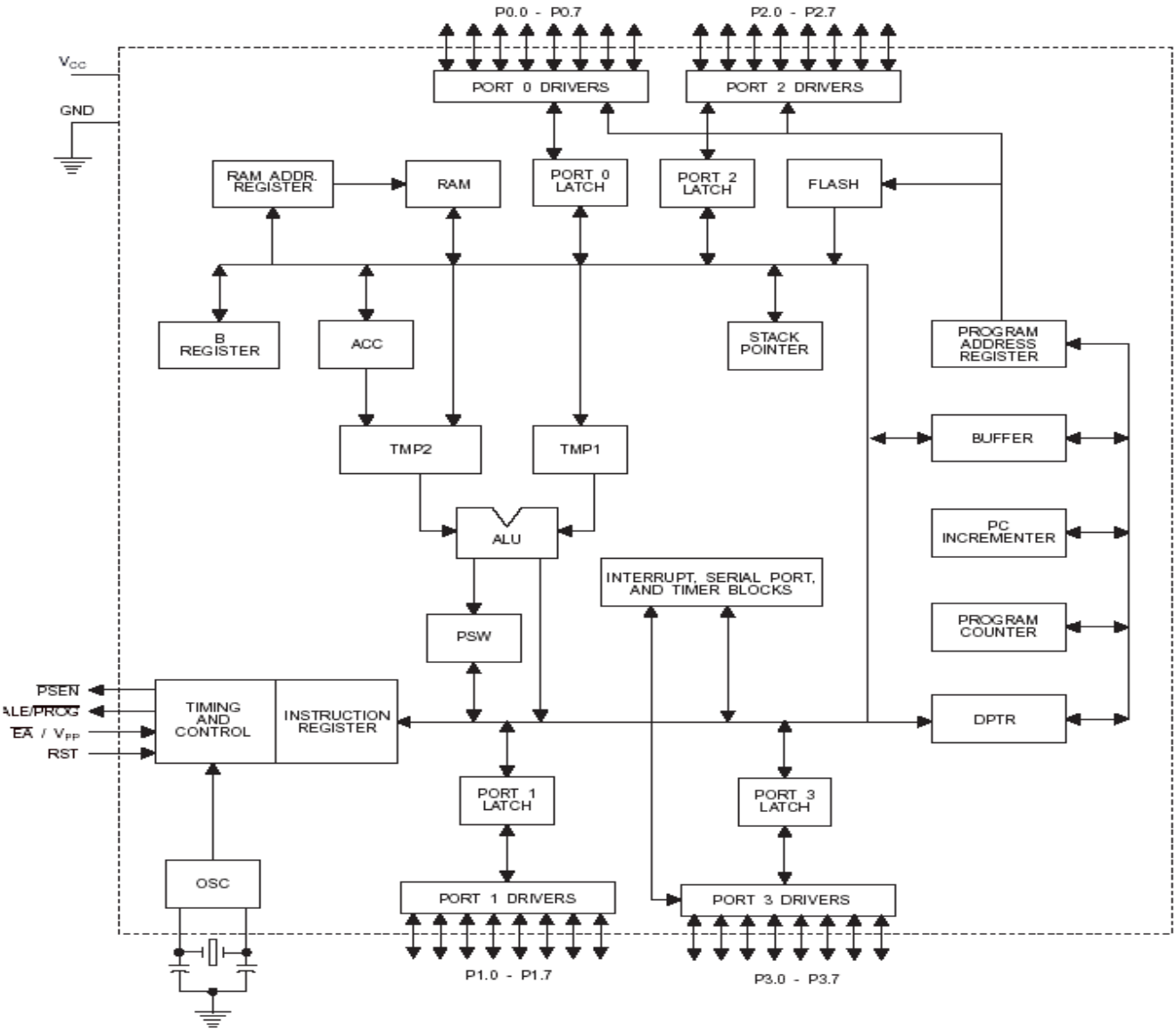
ARCHITECTURE & BLOCK DIAGRAM OF 8051 MICROCONTROLLER



In 1981 ,intel corporation introduced an 8 bit microcontroller called the 8051.The 8051 is an 8-bit processor. The CPU can work on only 8 bits of data at a time. The features of 8051 are

Feature	8051	8052	8031
ROM (on-chip program space in bytes)	4K	8K	0K
RAM (bytes)	128	256	128
Timers	2	3	2
I/O pins	32	32	32
Serial port	1	1	1
Interrupt sources	6	8	6

Block Diagram

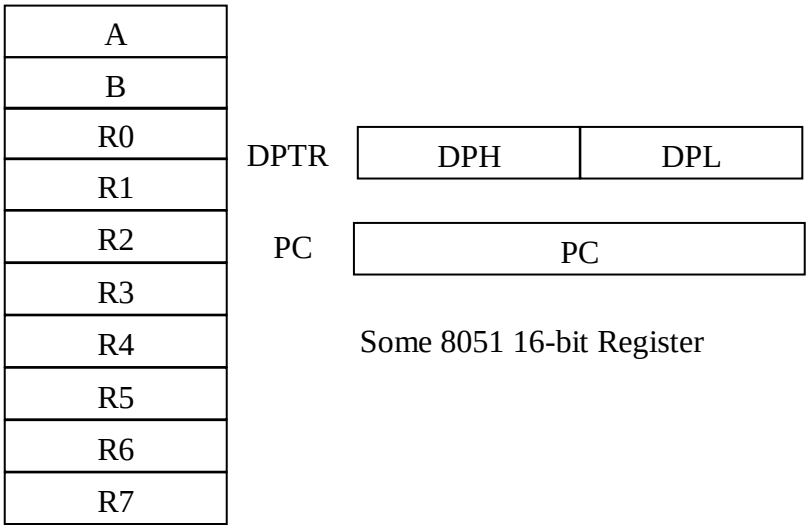


1. Registers of 8051

The most widely used registers are

- A (Accumulator) for all arithmetic and logic instructions
- B, R0, R1, R2, R3, R4, R5, R6, R7
- DPTR (data pointer),
- PC (program counter)

- **The R registers**: The "R" registers are a set of eight registers that are named R0, R1, etc. up to R7. These registers are used as auxiliary registers in many operations. The "R" registers are also used to temporarily store values.
- **A and B Registers :** The A and B registers are special function registers which hold the results of many arithmetic and logical operations of 8051.
  - The A register is also called the **Accumulator** and as it's name suggests, is used as a general register to accumulate the results of a large number of instructions.
  - By default, it is used for all mathematical operations and also data transfer operations between CPU and any external memory



Some 8051 16-bit Register

Some 8-bit Registers of the 8051

All the above registers are 8 bits except DPTR and PC.

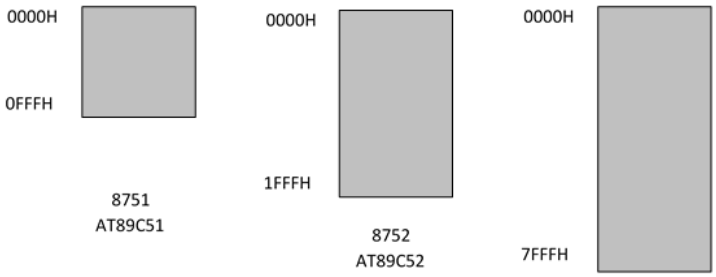
Program Counter (PC):

- 8051 has a 16-bit program counter.
- The program counter always points to the address of the next instruction to be executed. After execution of one instruction the program counter is incremented to point to the address of the next instruction to be executed.

Data Pointer Register (DPTR):

- It is a 16-bit register which is the only user-accessible.
- DPTR, as the name suggests, is used to point to data.
- When the 8051 accesses external memory it will access external memory at the address indicated by DPTR.
- This DPTR can also be used as two 8-registers DPH and DPL.

2. ROM memory map in 8051 family



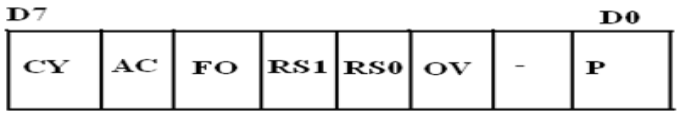
- The microcontroller wakes up at memory address 0000 when it is powered up.
- When 8051 is powered up, the PC has the value of 0000 in it. This means that it expects the first opcode to be stored at ROM address 0000H.
- For this reason, in 8051 system, the first opcode must be burned into memory location 0000H of program ROM.Since this is where it looks for the first instruction when it is booted.
- The first location of on chip ROM of this 8051 has an address of 0000 and the last location has the address of 0FFFH.

3. STACK POINTER (SP)

- The register used to access the stack is called SP (stack pointer) register.
- The stack pointer in the 8051 is only 8 bits wide, which means that it can take value 00 to FFH. When 8051 powered up, the SP register contains value 07.

4. Flag bits and PSW Register:

- The 8051 has an 8-bit Program Status Word register which is also known as Flag register is used to indicate arithmetic conditions and logical conditions such as the carry bits.
- In the 8-bit register, only 6-bits are used by 8051.The two unused bits are user definable bits. In the 6-bits four of them are conditional flags.
- They are Carry –CY, Auxiliary Carry-AC, Parity-P, and Overflow-OV.These flag bits indicate some conditions that resulted after an instruction was executed.



- The bits PSW3 and PSW4 are denoted as RS0 and RS1 and these bits are used to select the bank registers of the RAM location. The meaning of various bits of PSW register is shown below.

CY	PSW.7	Carry Flag
AC	PSW.6	Auxiliary Carry Flag
FO	PSW.5	Flag 0 available for general purpose
RS1	PSW.4	Register Bank select bit 1
RS0	PSW.3	Register bank select bit 0
OV	PSW.2	Overflow flag
---	PSW.1	User definable flag
P	PSW.0	Parity flagset/cleared by hardware.

The selection of the register Banks and their addresses are given below.

RS1	RS0	Register Bank	Address
0	0	0	00H – 07H
0	1	1	08H – 0FH
1	0	2	10H – 17H
1	1	3	18H – 1FH

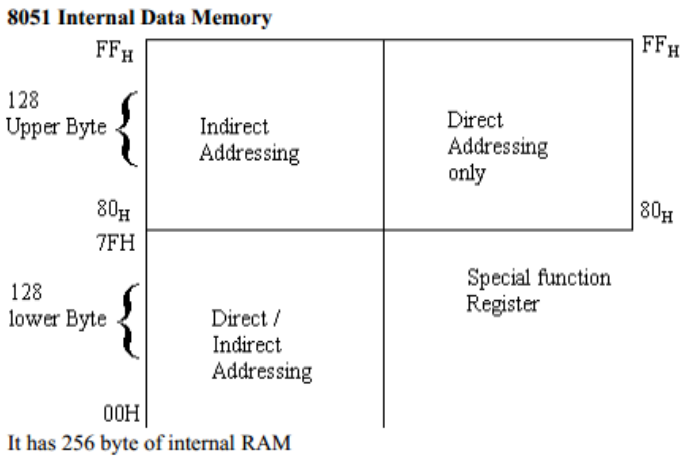
Explain RAM structure of 8051. **[December 2016]**
Draw the memory structure of 8051 microcontroller. **[December 2017]**

---

5. 8051 Register Banks and stack pointer

There are 128 bytes of RAM in the 8051 are assigned addresses 00 to 7FH. The 128 bytes are divided into three different groups as follows:

- 1) A total of 32 bytes from locations 00 to 1F hex are set aside for register banks and the stack.
- 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, called scratch pad



It has 256 byte of internal RAM

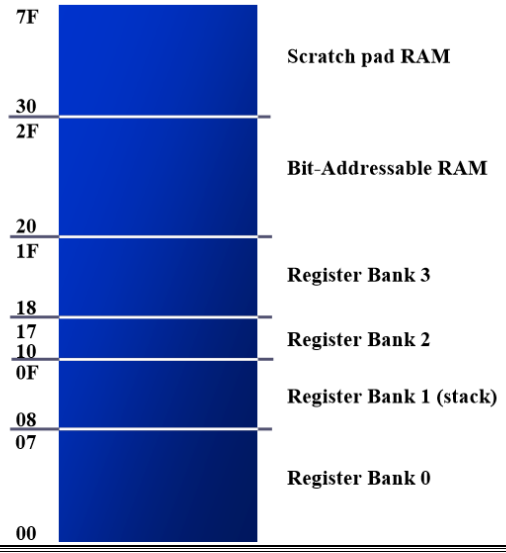


Fig:RAM memory space allocation in the 8051

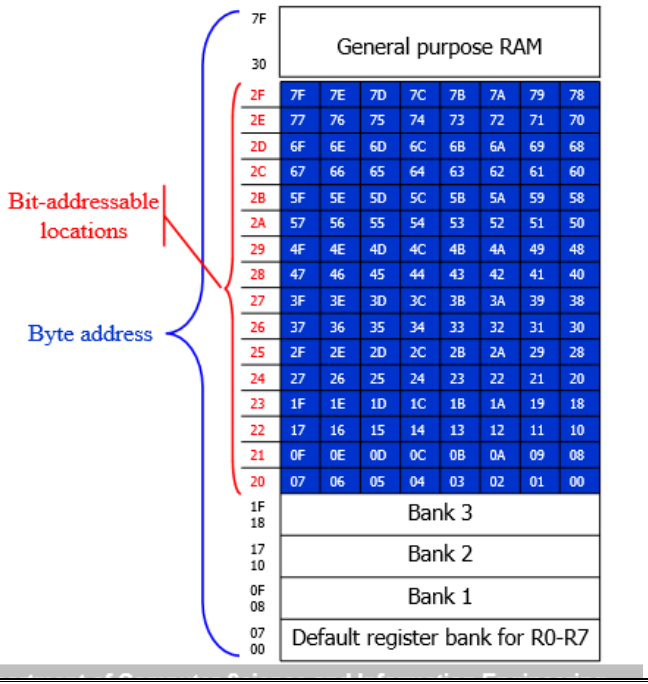
Register bank in 8051

- These 32 bytes are divided into 4 banks of registers in which each bank has 8 registers, R0-R7.
- RAM location from 0 to 7 are set aside for bank 0 of R0-R7 where R0 is RAM location 0, R1 is RAM location 1, R2 is RAM location 2, and so on, until memory location 7 which belongs to R7 of bank 0.
- It is much easier to refer to these RAM locations with names such as R0, R1, and so on, than by their memory locations Register bank 0 is the default when 8051 is powered up.

Bank 0		Bank 1		Bank 2		Bank 3	
7	R7	F	R7	17	R7	1F	R7
6	R6	E	R6	16	R6	1E	R6
5	R5	D	R5	15	R5	1D	R5
4	R4	C	R4	14	R4	1C	R4
3	R3	B	R3	13	R3	1B	R3
2	R2	A	R2	12	R2	1A	R2
1	R1	9	R1	11	R1	19	R1
0	R0	8	R0	10	R0	18	R0

Bit addressable RAM

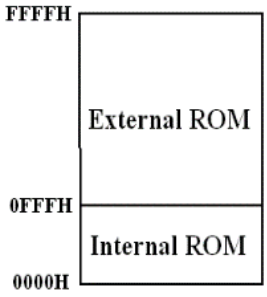
- The bit-addressable RAM locations are 20H to 2FH.
- These 16 bytes provide 128 bits of RAM bit-addressability, since 16 ×8 = 128. 0 to 127 (in decimal) or 00 to 7FH.
- The first byte of internal RAM location 20H has bit address 0 to 7H



- The last byte of 2FH has bit address 78H to 7FH.Internal RAM locations 20-2FH are both byte-addressable and bit addressable.
- Bit address 00-7FH belong to RAM byte addresses 20-2FH.
- Bit address 80-FFH belong to SFR P0, P1, …
- Only registers A, B, PSW, IP, IE, ACC, SCON, and TCON are bit-addressable.
-

While all I/O ports are bit-addressable, In PSW register, two bits are set aside for the selection of the register banks Upon RESET, bank 0 is selected .We can select any other banks using the bit-addressability of the PSW.

Structure of Internal ROM (On –chip ROM):



- The 8051 microcontroller has 4kB of on chip ROM but it can be extended up to 64kB.
- This ROM is also called program memory or code memory.
- The CODE segment is accessed using the program counter (PC) for opcode fetches and by DPTR for data.
- The external ROM is accessed when the EA (active low) pin is connected to ground or the contents of program counter exceeds 0FFFH.
- When the Internal ROM address is exceeded the 8051 automatically fetches the code bytes from the external program memory.

6. STACK in 8051

- The stack is a section of RAM used by the CPU to store information temporarily. This information could be data or an address.
- The register used to access the stack is called the SP (stack pointer) register .The stack pointer in the 8051 is only 8 bit wide, which means that it can take value of 00 to FFH.
- When the 8051 is powered up, the SP register contains value 07.RAM location 08 is the first location begin used for the stack by the 8051.
- The storing of a CPU register in the stack is called a PUSH and loading the contents of the stack back into a CPU register is called a POP.

Pushing on to the Stack

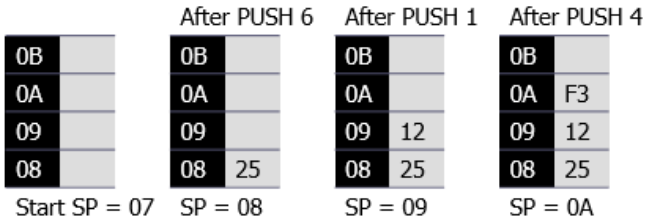
SP is pointing to the last used location of the stack ¾As we push data onto the stack, the SP is incremented byone.

Example 2-8

Show the stack and stack pointer from the following. Assume the default stack area.

```
MOV R6, #25H
MOV R1, #12H
MOV R4, #0F3H
PUSH  6
PUSH  1
PUSH  4
```

Solution:



.

Popping from the stack

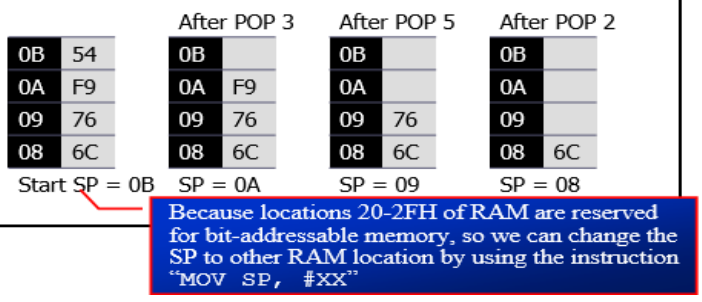
With every pop, the top byte of the stack is copied to the register specified by the instruction and the stack pointer is decremented once.

Example 2-9

Examining the stack, show the contents of the register and SP after execution of the following instructions. All value are in hex.

```
POP     3          ; POP stack into R3
POP     5          ; POP stack into R5
POP     2          ; POP stack into R2
```

Solution:



****

Explain the pin description of 8051 microcontroller. (Dec 2010,June 2013) (Dec 2018)

****

8051 PIN DIAGRAM

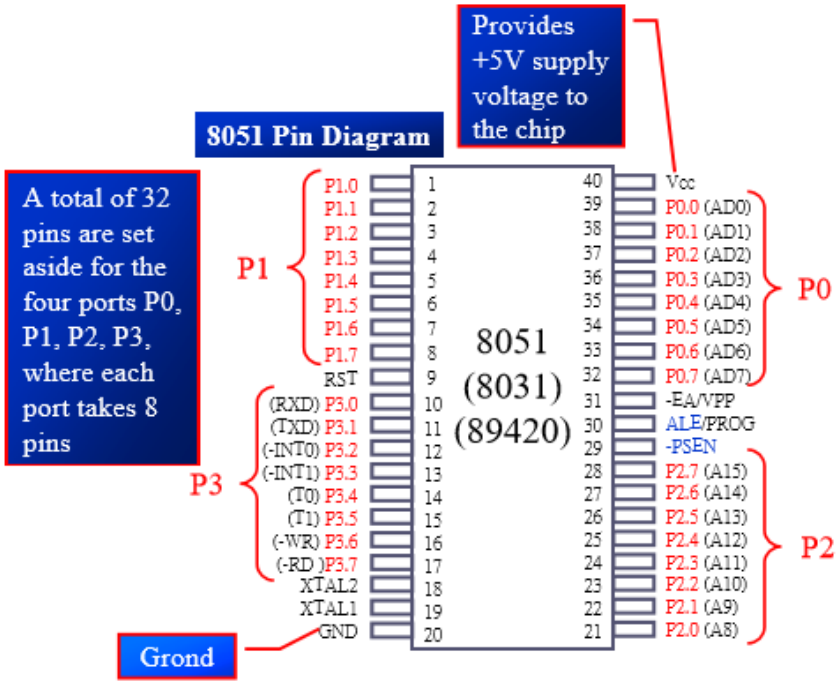
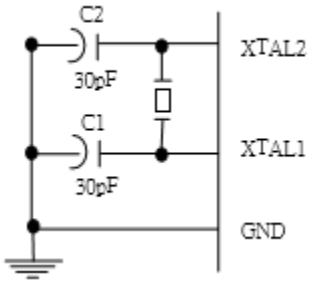


Figure:8051 pin description

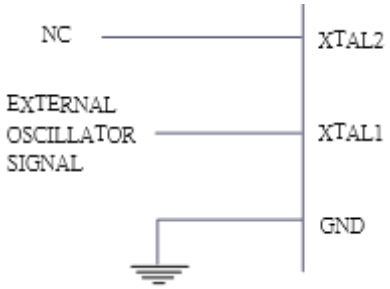
The 8051 family members (e.g, 8751, 89C51, 89C52, DS89C4x0) have 40 pins dedicated for various functions such as I/O, -RD, -WR, address, data, and interrupts .They come in different packages, such as DIP(dual in-line package),QFP(quad flat package), and LLC(leadless chip carrier).

- **V_{cc}** pin 40 provides supply voltage to the chip.The voltage source is $+5V$.
- **GND** Pin 20 is the ground.
- **XTAL1 AND XTAL2 (PIN 19,18)**

  The 8051 has an on-chip oscillator but requires an external clock to run it .A quartz crystal oscillator is connected to inputs XTAL1 (pin19) and XTAL2 (pin18) .The quartz crystal oscillator also needs two capacitors of 30 pFvalue**.**



- If you use a frequency source other than a crystal oscillator, such as a TTL oscillator ¾It will be connected to XTAL1.
- XTAL2 is left unconnected



The speed of 8051 refers to the maximum oscillator frequency connected to XTAL.Ex. A 12-MHz chip must be connected to a crystal with 12 MHz frequency or less. We can observe the frequency on the XTAL2 pin using the oscilloscope

- **EA（pin 31）：external access**
  - There is no on-chip ROM in 8031 and 8032 .so the EA pin is connected to GND to indicate the code is stored externally.
  - EA pin is connected to Vcc because the 8051 family members all come with on-chip ROM to store programs.

- **PSEN（pin 29）：Program store enable**
  - This is an output pin and is connected to the OE pin of the ROM.

- **ALE（pin 30）：Address latch enable**
  - The ALE pin is used for de-multiplexing the address and data bus of Port 0 which provides both address and data

- **RST（pin 9）：Reset**
  - It is a power-on reset.
    - Upon applying a high pulse to RST, the microcontroller will reset and all values in registers will be lost.
  - .

Reset values of some 8051 registers

Register	Reset Value
PC	0000
DPTR	0000
ACC	00
PSW	00
SP	07
B	00
P0-P3	FF

PARELLEL I/O PORTS

I/O port pins

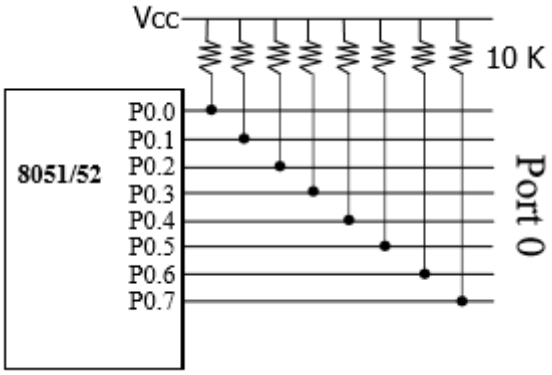
The four ports P0, P1, P2, and P3. Each port uses 8 pins. All I/O pins are bi-directional. All the ports upon RESET are configured as output, ready to be used as output ports. To make the ports as an input port, it must programmed as such by writing 1 to all its bits.

The 8051 has four I/O ports

- Port 0 P0 (P0.0 ~ P0.7)
- Port 1 P1 (P1.0 ~ P1.7)
- Port 2 P2 (P2.0 ~ P2.7)
- Port 3 P3 (P3.0 ~ P3.7)

Port 0

- **It** is also designated as AD0-AD7, allowing it to be used for both address and data.
- When connecting an 8051/31 to an external memory, port 0 provides both address and data.
- The 8051 multiplexes address and data through port 0 to save pins.
- ALE indicates if P0 has address or data
  - When ALE=0, it provides data D0-D7
  - When ALE=1, it has address A0-A7
- It can be used for input or output, each pin must be connected externally to a 10K ohm pull-up resistor .
- This is due to the fact that P0 is an open drain, unlike P1, P2, and P3 .



PORT1

Port 1 occupies total of 8 pins 1 to 8 In contrast to Port 0,this port does not need any pull up resistor, since it has already pull up resistors internally.

PORT2

- In 8051-based systems with no external memory connection Both P1 and P2 are used as simple I/O.
- In 8031/51-based systems with external memory connections,Port 2 must be used along with P0 to provide the 16-bit address for the external memory
- P0 provides the lower 8 bits via A_0 –A_7
- P2 is used for the upper 8 bits of the 16-bit address, designated as A8 –A15, and it cannot be used for I/O.

Port 3

- It can be used as input or output.
- Port 3 does not need any pull-up resistors
- Port 3 has the additional function of providing some extremely important signals.

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

Serial communications

External interrupts

Timers

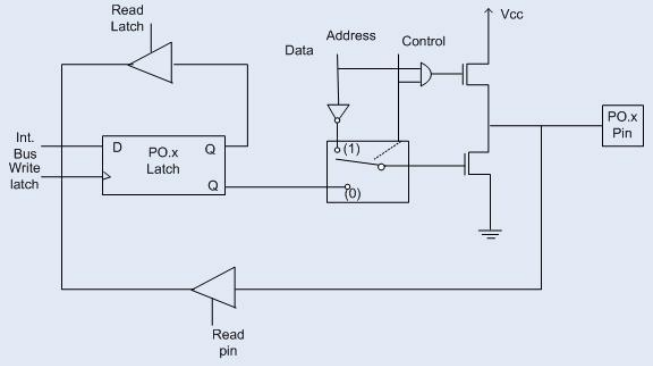
Read/Write signals of external memories

Explain Parallel ports of 8051 with its circuit description in detail. [NOV/DEC 2016, APRIL 2015,April 2018]

****

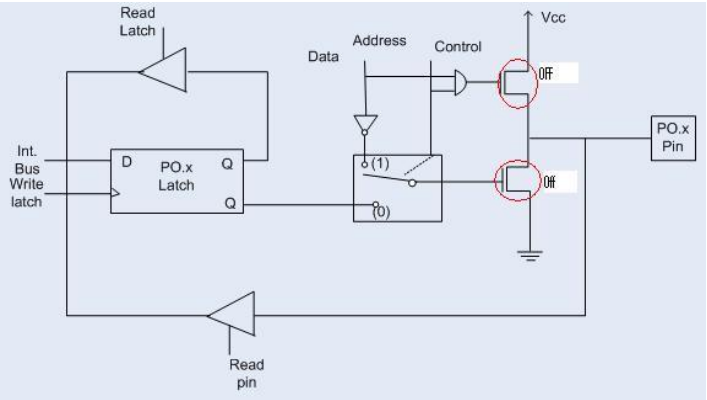
PORT 0

Port-0 can be used as a normal bidirectional I/O port or it can be used for address/data interfacing for accessing external memory. When control is '1', the port is used for address/data interfacing. When the control is '0', the port can be used as a bidirectional I/O port



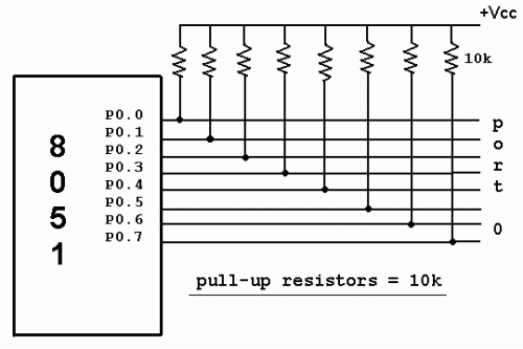
.PORT 0 as an Input Port

Let us assume that control is '0'. When the port is used as an input port, '1' is written to the latch. In this situation both the output MOSFETs are 'off'. Hence the output pin have floats hence whatever data written on pin is directly read by read pin.

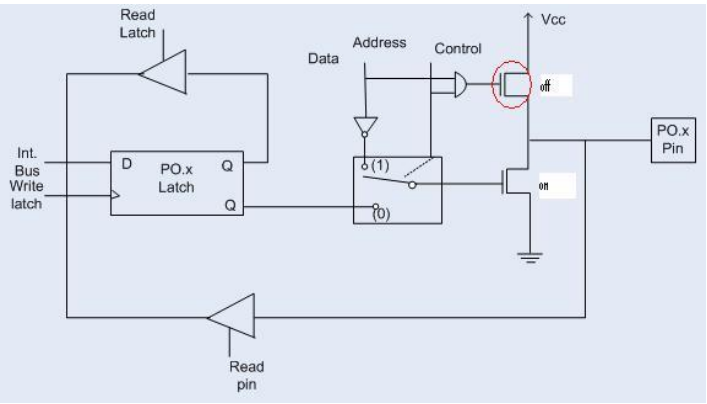


PORT 0 as an Output Port

Suppose we want to write 1 on pin of Port 0, a '1' written to the latch whichturns 'off' the lower FET while due to '0' control signal upper FET also turns off as shown in fig. above. Here we want logic '1' on pin but we getting floating value so to convert that floating value into logic '1' we need to connect the pull up resistor parallel to upper FET. This is the reason **why we needed to connect pull up resistor to port 0 when we want to initialize port 0 as an output port**.



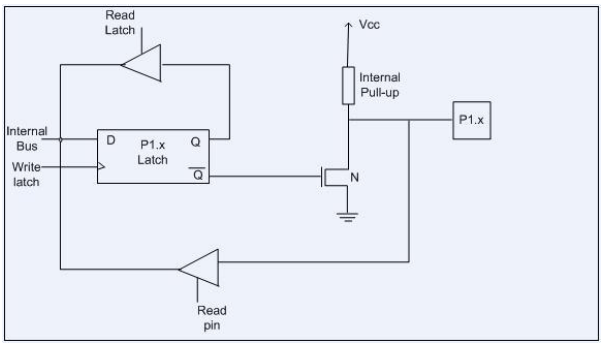
If we want to write '0' on pin of port 0, when '0' is written to the latch, the pin is pulled down by the lower FET. Hence the output becomes zero



When the control is '1', address/data bus controls the output driver FETs. If the address/data bus (internal) is '0', the upper FET is 'off' and the lower FET is 'on'. The output becomes '0'. If the address/data bus is '1', the upper FET is 'on' and the lower FET is 'off'. Hence the output is '1'. Hence for normal address/data interfacing (for external memory access) no pull-up resistors are required.Port-0 latch is written to with 1's when used for external memory access.

PORT 1:

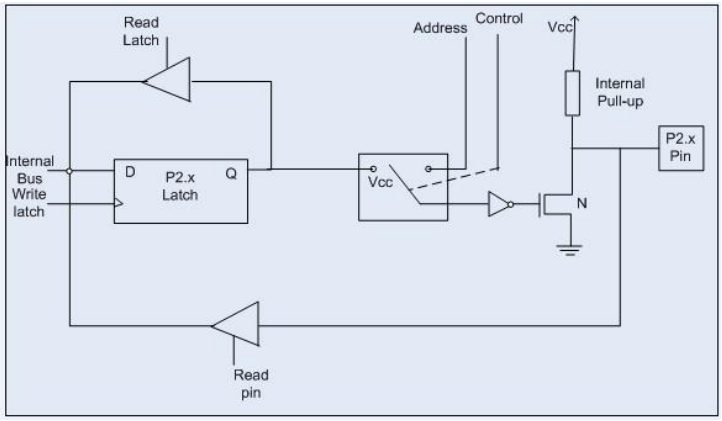
The structure of a port-1 pin is shown in fig below. It has 8 pins (P1.1-P1.7) .



- Port-1 dedicated only for I/O interfacing. When used as output port, not needed to connect additional pull-up resistor like port 0.
- It has provided internally pull-up resistor as shown in fig. below. The pin is pulled up or down through internal pull-up when we want to initialize as an output port.
- To use port-1 as input port, '1' has to be written to the latch. In this input mode when '1' is written to the pin by the external device then it read fine. But when '0' is written to the pin by the external device then the external source must sink current due to internal pull-up.
- If the external device is not able to sink the current the pin voltage may rise, leading to a possible wrong reading.

PORT 2:

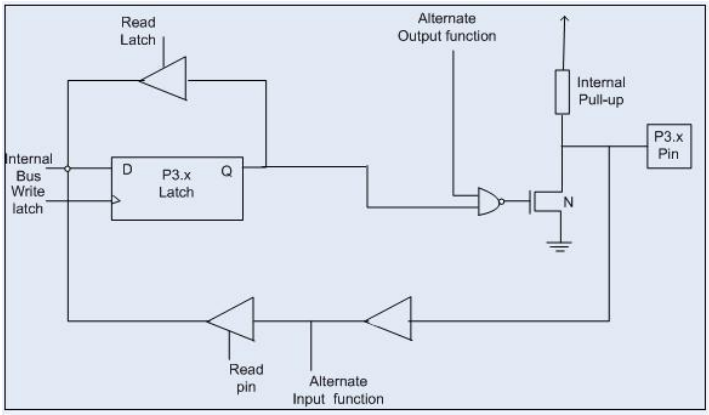
The structure of a port-2 pin is shown in fig. below. It has 8-pins (P2.0-P2.7) .



Port-2 we use for higher external address byte or a normal input/output port. The I/O operation is similar to Port-1. Port-2 latch remains stable when Port-2 pin are used for external memory access. Here again due to internal pull-up there is limited current driving capability.

PORT 3:

Port-3 (P3.0-P3.7) having alternate functions to each pin,The internal structure of a port-3 pin is shown in fig below.



Following are the alternate functions of port 3

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

Serial communications

External interrupts

Timers

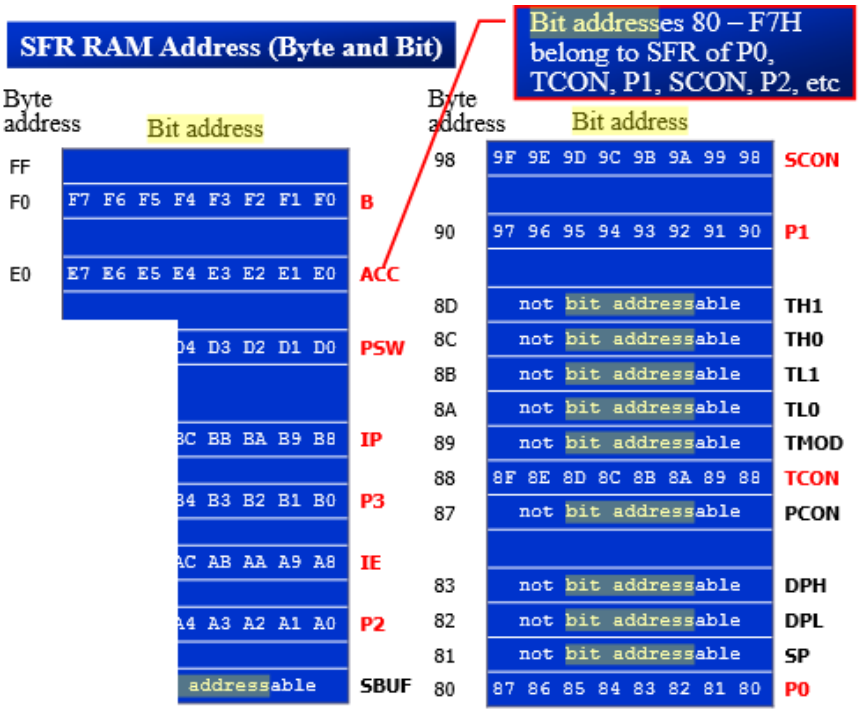
Read/Write signals of external memories

****

Explain special function register of 8051. (Dec 2010)

SPECIAL FUNCTION REGISTER

In 8051 microcontroller there are certain registers which uses the RAM addresses from 80h to FFH and they are meant for certain specific operations. These registers are called Special function registers (SFRs).Some of these registers are bit addressable also.



Special Function Register

The SFR (Special Function Register) can be accessed by their names or by their addresses. Not all the address space of 80 to FF is used by SFR.The unused locations 80H to FFH are reserved and must not be used by the 8051 programmer

The list of SFRs and their functional names are given below. In these SFRs some of them are related to I/O ports (P0,P1,P2 and P3) and some of them are meant for control operations (TCON, SCON, PCON) and remaining are the auxillary SFRs, in the sense that they don't directly configure the 8051

S.No	Symbol		Name of SFR	Address (Hex)
1	ACC*		Accumulator	**0E0**
2	B*		B-Register	**0F0**
3	PSW*		Program Status word register	**0DO**
4	SP		Stack Pointer Register	**81**
5	DPTR	DPL	Data pointer low byte	**82**
		DPH	Data pointer high byte	**83**
6	P0*		Port 0	**80**
	P1*		Port 1	**90**
8	P2*		Port 2	**0A**
9	P3*		Port 3	**0B**
10	IP*		Interrupt Priority control	**0B8**
11	IE*		Interrupt Enable control	**0A8**
12	TMOD		Tmier mode register	**89**
13	TCON*		Timer control register	**88**
14	TH0		Timer 0 Higher byte	**8C**
15	TL0		Timer 0 Lower byte	**8A**
16	TH1		Timer 1Higher byte	**8D**
17	TL1		Timer 1 lower byte	**8B**
18	SCON*		Serial control register	**98**
19	SBUF		Serial buffer register	**99**
20	PCON		Power control register	**87**

****

Explain the various addressing modes of 8051 microcontroller.

****

ADDRESSING MODES OF 8051:

The way in which the data operands are accessed by different instructions is known as the **addressing modes. There are various methods of denoting the data operands in the instruction. The** 8051 microcontroller supports mainly 5 addressing modes. They are

1. Immediate addressing mode
2. Direct Addressing mode
3. Register addressing mode
4. Register Indirect addressing mode
5. Indexed addressing mode

1. Immediate addressing mode :

The addressing mode in which the data operand is a constant and it is a part of the instruction itself is known as immediate addressing mode. Normally the data must be preceded by a # sign. This addressing mode can be used to transfer the data into any of the registers including DPTR.

Example:

MOV A, # 27 H : The data (constant) 27 is moved to the accumulator register

ADD R1,#45 H : Add the constant 45 to the contents of the accumulator

MOV DPTR ,# 8245H :Move the data 8245 into the data pointer register.

MOV P1,#21 H

2. Direct addressing mode:

The addressing mode in which the data operand is in the RAM location (00 -7FH) and the address of the data operand is given in the instruction is known as Direct addressing mode. The direct addressing mode uses the lower 128 bytes of Internal RAM and the SFRs.

Example:

MOV R1, 42H: Move the contents of RAM location 42 into R1 register

MOV 49H,A: Move the contents of the accumulator into the RAM location 49.

ADD A, 56H: Add the contents of the RAM location 56 to the accumulator

3. Register addressing mode:

The addressing mode in which the data operand to be manipulated lies in one of the registers is known as register addressing mode.

Example:

MOV A, R0: Move the contents of the register R0 to the accumulator

ADD A, R6 :Add the contents of R6 register to the accumulator

MOV P1, R2 : Move the contents of the R2 register into port 1

MOV R5, R2 : This is invalid .The data transfer between the registers is not allowed.

4. Register Indirect addressing mode:

The addressing mode in which a register is used as a pointer to the data memory block is known as Register indirect addressing mode.

Example:

MOV A,@ R0 :Move the contents of RAM location whose address is in R0 into **A** (accumulator)

MOV @ R1 , B : Move the contents of B into RAM location whose address is held by R1

When R0 and R1 are used as pointers, they must be preceded by @ sign

One of the advantages of register indirect addressing mode is that it makes accessing the data more dynamic than static as in the case of direct addressing mode.

5.Indexed addressing mode :

This addressing mode is usedin accessing the data elements of lookup table entries located in program ROM space of 8051.

Example : MOVC A,@ A+DPTR

The 16-bit register DPTR and register A are used to form the address of the data element stored in on-chip ROM. Here C denotes code .In this instruction the contents of A are added to the 16-bit DPTR register to form the 16-bit address of the data operand.

Explain the Data Transfer Schemes and its types in detail.

****

DATA TRANSFER SCHEMES

- In a microprocessor-based system, the data transfer takes place between two devices such as microprocessor and memory, microprocessor and I/O devices and memory and I/O devices.
- A microprocessor based system or a computer may have several I/O devices of different speed.
- A slow I/O device cannot transfer data because it takes some time to get ready.
- To solve this problem of speed mismatch between a microprocessor and I/O devices a number of data transfer techniques have been developed.

They are classified into two categories.

1. Programmed data transfer scheme
2. DMA (Direct Memory Access) data transfer scheme

Programmed Data Transfer Scheme

- Programmed data transfer scheme are controlled by the CPU.
- Data are transferred an I/O device to the CPU or vice versa under the control of programs.
- These programs are executed by the CPU when an I/O device is ready to transfer data.
- It is used when small amount of data are to be transferred. It is classified into following three categories.

Synchronous Data Transfer Scheme

- Synchronous means “at the same time”.
- The device which sends data and the device which receives data are synchronized with the same clock.
- The data transfer with I/O devices is performed by executing IN or OUT instructions for I/O mapped I/O devices.

[OR]

- The data transfer with I/O devices is performed by executing memory read/write instruction for memory mapped I/O devices.
- In this type of data transfer, the status of the I/O device i.e., whether it is ready or not, is not examined before data are transferred. Hence, this technique is rarely used for I/O devices.

Asynchronous Data Transfer Scheme

- Asynchronous means “at irregular intervals”.
- The device which sends data and the device which receives data are not synchronized with the same clock.
- This technique of data transfer is used when the speed of an I/O device does not match the speed of the microprocessor and also the timing characteristic of I/O device is not predictable.
- The status of the I/O device i.e., whether the device is ready or not is checked by the microprocessor before the data are transferred.
- If it is not ready, the microprocessor initiates the I/O device to get ready and then continuously checks the status of the I/O device till the I/O device becomes ready to transfer data.

- When I/O device becomes ready, the microprocessor sends instruction to transfer data.
- This method of data transfer is also called **handshaking mode**.
- The microprocessor sends an initiating signal to the I/O device to get ready.
- When I/O device becomes ready it sends signals to the processor to indicate that it is ready. Such signals are called **handshake signals.**

Interrupt Driven Data Transfer Scheme

- In this scheme, the microprocessor initiates an I/O device to get ready and then it executes its main program instead of remaining in a program loop to check the status of the I/O device.
- When the I/O device becomes ready to transfer data, it sends a high signal to the microprocessor through a special input line called an interrupt line.
- In other word, it interrupts the normal processing sequence of the microprocessor.
- On receiving the microprocessor completes the current instruction at hand and then attends the I/O device.
- It saves the contents of the program counter on the stack first and then takes up a subroutine called Interrupt Service Subroutine (ISS).

DMA Transfer Scheme

- DMA transfer scheme is not controlled by the CPU. Data are directly transferred from an I/O device to the memory or vice versa.
- The data transfer is controlled by the I/O device or a DMA controller. It is used when large amount of data are to be transferred.
- DMA data transfer scheme is faster than programmed data transfer scheme.
- It is used to transfer data from mass storage devices such as hard disks, floppy disks etc.,
- It is also used for high-speed printers.

DMA data transfer scheme are of the following two types. **Burst Mode**

- In which the I/O device withdraws the DMA request only after on the data bytes have been transferred is called burst mode of data transfer.
- It is employed by magnetic disk drives.

Cycle Stealing Technique

- In this technique, a long block of data is transferred by a sequence of DMA cycles.
- In this method after transferring one byte or several bytes the I/O device withdraws DMA request.
- This method reduces interference in CPU"s activities.
- The interference can be eliminated completely by designing an interfacing circuitry which can steal bus cycle for DMA data transfer only when the CPU is not using the system bus.

****

****

Explain the various types of instruction set of 8051 microcontroller. (June 2016)(Dec 2015)(Dec 2017)

****

INSTRUCTION SET IN 8051 MICROCONTROLLER:

1. Arithmetic Instructions:

ADD

- 8-bit addition between the accumulator (A) and a second operand.
  - The result is always in the accumulator.
  - The CY flag is set/reset appropriately.
  - ADDC
- 8-bit addition between the accumulator, a second operand and the previous value of the CY flag.
  - Useful for 16-bit addition in two steps.
  - The CY flag is set/reset appropriately.

- DA
  - Decimal adjust the accumulator.
    - Format the accumulator into a proper 2 digit packed BCD number.
    - Operates only on the accumulator.
    - Works only after the ADD instruction.
- SUBB
  - Subtract with Borrow.
    - Subtract an operand and the previous value of the borrow (carry) flag from the accumulator.
      - A ← A - <operand> - CY.
      - The result is always saved in the accumulator.
      - The CY flag is set/reset appropriately.
- INC
  - Increment the operand by one.
    - The operand can be a register, a direct address, an indirect address, the data pointer.
- DEC
  - Decrement the operand by one.
    - The operand can be a register, a direct address, an indirect address.
- MUL AB / DIV AB
  - Multiply A by B and place result in A:B.
  - Divide A by B and place result in A:B.

2. logical instructions in 8051

- ANL / ORL
  - Work on byte sized operands or the CY flag.
    - ANL A, Rn
    - ANL A, direct
    - ANL A, @Ri
    - ANL A, #data
    - ANL direct, A
    - ANL direct, #data
    - ANL C, bit
    - ANL C, /bit
    -

- XRL
  - Works on bytes only.
  - CPL / CLR
  - Complement / Clear.
  - Work on the accumulator or a bit.
    - CLR P1.2
- RL / RLC / RR / RRC
  - Rotate the accumulator.
    - RL and RR without the carry
    - RLC and RRC rotate through the carry.
    - SWAP A
  - Swap the upper and lower nibbles of the accumulator.
  - No compare instruction.
  - Built into conditional branching instructions.

3. <u>Data Transfer Instructions</u>

- MOV
  - 8-bit data transfer for internal RAM and the SFR.
    - MOV A, Rn
    - MOV A, direct
    - MOV A, @Ri
    - MOV A, #data
    - MOV Rn, A
    - MOV Rn, direct
    - MOV Rn, #data
    - MOV direct, A
    - MOV direct, Rn
    - MOV direct, direct
    - MOV direct, @Ri
    - MOV direct, #data
    - MOV @Ri, A
    - MOV @Ri, direct
    - MOV @Ri, #data
- MOV
  - 1-bit data transfer involving the CY flag
    - MOV C, bit
    - MOV bit, C
- MOV
  - 16-bit data transfer involving the DPTR
    - MOV DPTR, #data
- MOVC
  - Move Code Byte
    - Load the accumulator with a byte from program memory.
    - Must use indexed addressing
    - MOVC A, @A+DPTR
    - MOVC A, @A+PC
- MOVX
  - Data transfer between the accumulator and a byte from external data memory.
    - MOVX A, @Ri

        - MOVX A, @DPTR
        - MOVX @Ri, A
        - MOVX @DPTR, A
- PUSH / POP
    - Push and Pop a data byte onto the stack.
    - The data byte is identified by a direct address from the internal RAM locations.
        - PUSH DPL
        - POP 40H
- XCH
    - Exchange accumulator and a byte variable
        - XCH A, Rn
        - XCH A, direct
        - XCH A, @Ri
- XCHD
    - Exchange lower digit of accumulator with the lower digit of the memory location specified.
        - XCHD A, @Ri
        - The lower 4-bits of the accumulator are exchanged with the lower 4-bits of the internal memory location identified indirectly by the index register.
        - The upper 4-bits of each are not modified.

Explain the various bit manipulation instruction in 8051 with example.(Dec 2018)

4. <u>Boolean (or) Bit manipulation instructions in 8051.</u>

- This group of instructions is associated with the single-bit operations of the 8051.
- This group allows manipulating the individual bits of bit addressable registers and memory locations as well as the CY flag.
    - The P, OV, and AC flags cannot be directly altered.
- This group includes:
    - Set, clear, and, or complement, move.
    - Conditional jumps.
- CLR
    - Clear a bit or the CY flag.
        - CLR P1.1
        - CLR C
- SETB
    - Set a bit or the CY flag.
        - SETB A.2
        - SETB C
        - CPL
    - Complement a bit or the CY flag.
        - CPL 40H ; Complement bit 40 of the bit addressable memory
- ORL / ANL
    - OR / AND a bit with the CY flag.
        - ORL C, 20H ; OR bit 20 of bit addressable memory with the CY flag
        - ANL C, /34H ; AND complement of bit 34 of bit addressable memory with the CY flag.
- MOV
    - Data transfer between a bit and the CY flag.
        - MOV C, 3FH ; Copy the CY flag to bit 3F of the bit addressable memory.
        - MOV P1.2, C ; Copy the CY flag to bit 2 of P1.

- JC / JNC
  - Jump to a relative address if CY is set / cleared.
- JB / JNB
  - Jump to a relative address if a bit is set / cleared.
    - JB ACC.2, <label>
- JBC
  - Jump to a relative address if a bit is set and clear the bit.
  - Instructions that are used for signal-bit operations are as following

Single-Bit Instructions

Instruction	Function
SETB bit	Set the bit (bit = 1)
CLR bit	Clear the bit (bit = 0)
CPL bit	Complement the bit (bit = NOT bit)
JB bit, target	Jump to target if bit = 1 (jump if bit)
JNB bit, target	Jump to target if bit = 0 (jump if no bit)
JBC bit, target	Jump to target if bit = 1, clear bit (jump if bit, then clear)

5. Branching instructions in 8051.

- The 8051 provides four different types of unconditional jump instructions:
  - Short Jump – SJMP
    - Uses an 8-bit signed offset relative to the 1st byte of the next instruction.
  - Long Jump – LJMP
    - Uses a 16-bit address.
    - 3 byte instruction capable of referencing any location in the entire 64K of program memory.
  - Absolute Jump – AJMP
    - Uses an 11-bit address.
    - 2 byte instruction
      - The upper 3-bits of the address combine with the 5-bit opcode to form the 1st byte and the lower 8-bits of the address form the 2nd byte.
    - The 11-bit address is substituted for the lower 11-bits of the PC to calculate the 16-bit address of the target.
      - The location referenced must be within the 2K Byte memory page containing the AJMP instruction.
  - Indirect Jump – JMP
    - JMP @A + DPTR
- The 8051 provides 2 forms for the CALL instruction:
  - Absolute Call – ACALL
    - Uses an 11-bit address similar to AJMP
    - The subroutine must be within the same 2K page.
  - Long Call – LCALL
    - Uses a 16-bit address similar to LJMP
    - The subroutine can be anywhere.
  - Both forms push the 16-bit address of the next instruction on the stack and update the stack pointer.
- The 8051 provides 2 forms for the return instruction:
  - Return from subroutine – RET
    - Pop the return address from the stack and continue execution there.

- Return from ISV – RETI
    - Pop the return address from the stack.
    - Restore the interrupt logic to accept additional interrupts at the same priority level as the one just processed.
    - Continue execution at the address retrieved from the stack.
    - The PSW is not automatically restored.

- The 8051 supports 5 different conditional jump instructions.
    - ALL conditional jump instructions use an 8-bit signed offset.
    - Jump on Zero – JZ / JNZ
        - Jump if the A == 0 / A != 0
            - The check is done at the time of the instruction execution.
    - Jump on Carry – JC / JNC
        - Jump if the C flag is set / cleared.
    - Jump on Bit – JB / JNB
        - Jump if the specified bit is set / cleared.
        - Any addressable bit can be specified.
    - Jump if the Bit is set then Clear the bit – JBC
        - Jump if the specified bit is set.
        - Then clear the bit.
- Compare and Jump if Not Equal – CJNE
    - Compare the magnitude of the two operands and jump if they are not equal.
        - The values are considered to be unsigned.
        - The Carry flag is set / cleared appropriately.
        - CJNE A, direct, rel
        - CJNE A, #data, rel
        - CJNE Rn, #data, rel
        - CJNE @Ri, #data, rel
- Decrement and Jump if Not Zero – DJNZ
    - Decrement the first operand by 1 and jump to the location identified by the second operand if the resulting value is not zero.
        - DJNZ Rn, rel
        - DJNZ direct, rel
- No Operation
    - NOP

Basic I/O Instructions

- **IN, OUT, INS** and **OUTS** are the instructions for the transfer of data to and from an I/O device.
- IN and OUT transfer data between an I/O device and the microprocessor's accumulator (AL, AX or EAX).

The I/O address is stored in:

- Register DX as a 16-bit I/O address (variable addressing).
- The byte, D8, immediately following the opcode (fixed address).

```
IN  AL, 19H     ;8-bits are saved to AL from I/O port 19H.
IN  EAX, DX     ;32-bits are saved to EAX.
OUT DX, EAX     ;32-bits are written to port DX from EAX.
OUT 19H, AX     ;16-bits are written to I/O port 0019H.
```

- Only 16-bits (A0 to A15) are decoded.
- Address connections above A15 are undefined for I/O instructions.
- 0000H-03XXH are used for the ISA bus.
- INS and OUTS transfer to I/O devices using ES:DI and DS:SI, respectively.

****

With a neat Diagram explain what is interrupts and types of interrupts in 8051. [NOV/DEC 2016 , MAY/JUNE 2016, APRIL/MAY 2015]

****

Interrupt Programming

An interrupt is an external or internal event that interrupts the microcontroller to inform it that a device needs its service.A single microcontroller can serve several devices by two ways

❖ **Polling**

- The microcontroller continuously monitors the status of a given device
- When the conditions met, it performs the service.
- After that, it moves on to monitor the next device until every one is serviced

Polling can monitor the status of several devices and serve each of them as certain conditions are met The polling method is not efficient, since it wastes much of the microcontroller's time by polling devices that do not need service

ex. JNB TF,target

❖ **Interrupts**

- Whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal
- Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device
- The program which is associated with the interrupt is called the interrupt service routine (ISR) or interrupt handler.

For every interrupt, there must be an interrupt service routine (ISR), or interrupt handler

- When an interrupt is invoked, the microcontroller runs the interrupt service routine.
- 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 **interrupt vector table**

❖ **Steps in executing an interrupt**

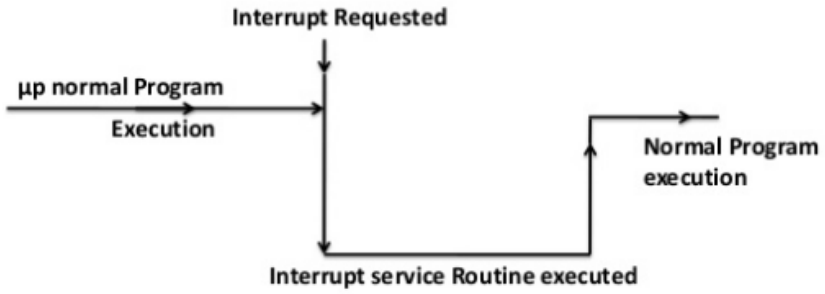


Figure: Interrupt service routine

1. Finish current instruction and saves the address of the next instruction (PC) on the stack

2. It jumps to a fixed location in memory called the interrupt vector table that holds the address of the ISR.

4. It starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine which is RETI (return from interrupt)

5. Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted. Get POP PC from Stack.

6. Then it starts to execute from that address

❖ **Interrupt Sources**

Six interrupts are allocated as follows

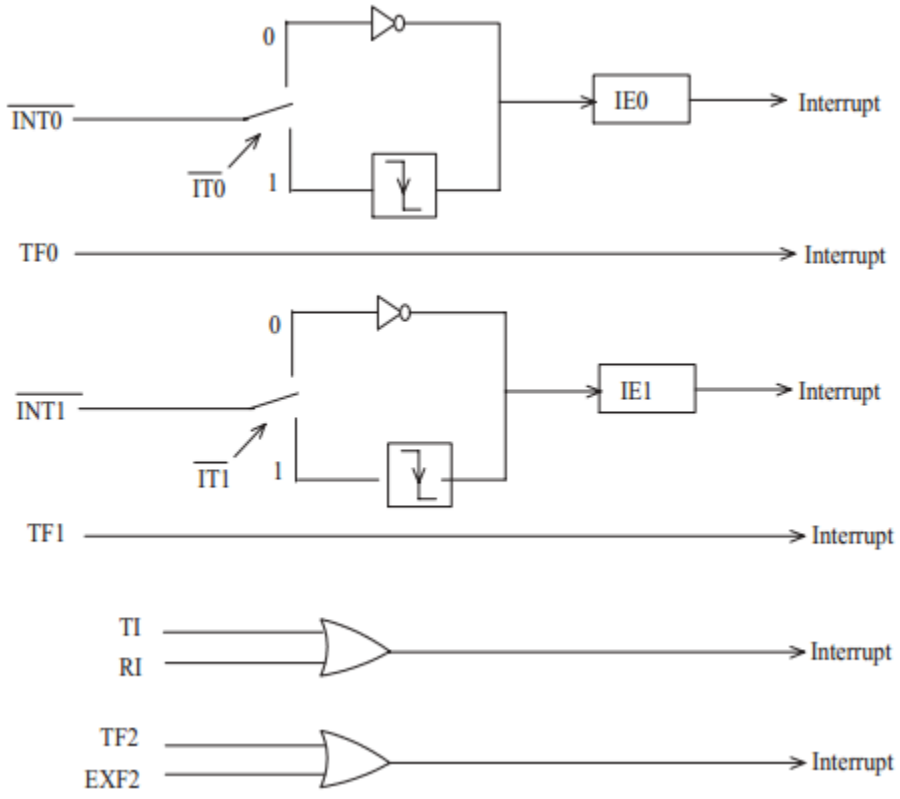


Figure: 8051 Interrupt sources

❖ **Interrupt Vectors**

Interrupt vector table

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

Upon reset, all interrupts are disabled (masked), meaning that none will be responded to by the microcontroller if they are activated

- The interrupts must be enabled by software in order for the microcontroller to respond to them.
- There is a register called IE (interrupt enable) that is responsible for enabling (unmasking) and disabling (masking) the interrupts.

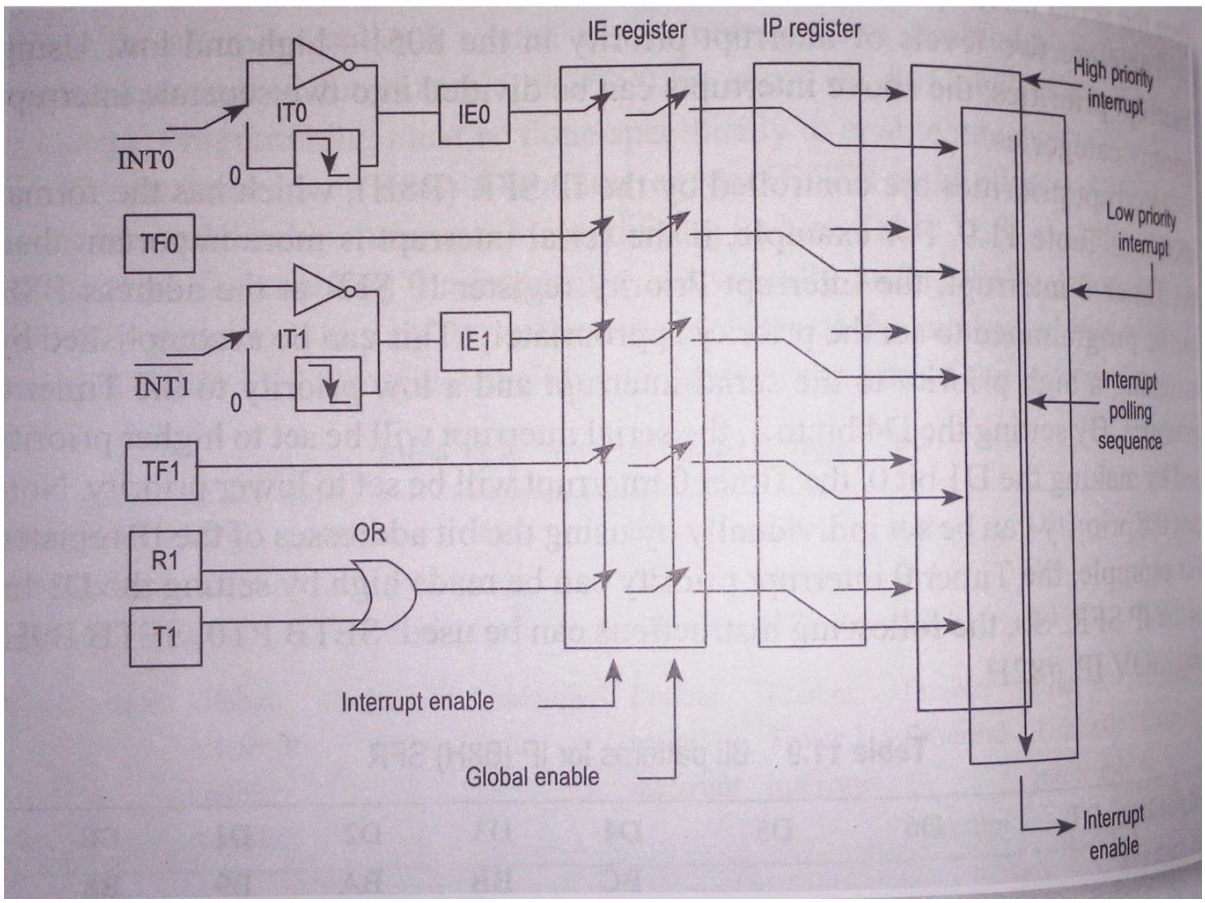


Fig:Interrupt structure of 8051

❖ **Interrupt Related Register**

The various registers associated with interrupts are

- Interrupt Enable (IE)

- Interrupt Priority(IP)
- Timer control TCON)
- Serial control(SCON**)**

1. **Interrupt Enable (IE)Register**(Enabling and Disabling)

EA	-----	ET2	ES	ET1	EX1	ET0	EX0
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

- EX0/EX1 : Enables(1)/disables(0) the external interrupt 0 and the external interrupt 1 on port P3.2 / P3.3
- ET0/ET1 : Enables(1)/disables(0) the Timer0 and Timer1 interrupt via TF0/1
- ES : Enables(1)/disables(0) the serial port interrupt for sending and receiving data
- EA : Enables(1)/disables(0) all interrupts

To enable an interrupt, we take the following steps:

1. Bit D7 of the IE register (EA) must be set to high to allow the rest of register to take effect
2. The value of EA

- 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.

2. **Interrupt Priority(IP)Register**

-	-	PT2	PS	PT1	PX1	PT0	PX0
bit7	bit6	bit5	bit4	bit3	bit2	bit1	

- PS- IP.4- Serial Port Interrupt Priority bit
- PT1- IP.3- Timer 1 Interrupt Priority bit
- PX1- IP.2 External Interrupt 1 Priority bit
- PT0- IP.1 Timer 0 Interrupt Priority bit
- PX0- IP.0 External Interrupt 0 Priority bit

When the 8051 is powered up, the priorities are assigned according to the following

Interrupt Priority Upon Reset

Highest To Lowest Priority	
External Interrupt 0	(INT0)
Timer Interrupt 0	(TF0)
External Interrupt 1	(INT1)
Timer Interrupt 1	(TF1)
Serial Communication	(RI + TI)

We can alter the sequence of interrupt priority by assigning a higher priority to any one of the interrupts by programming a register called IP (interrupt priority)

- To give a higher priority to any of the interrupts, we make the corresponding bit in the IP register high When two or more interrupt bits in the IP register are set to high
- While these interrupts have a higher priority than others, they are serviced according to the sequence of Table.

3. TCON (Timer control register)

It is used to select edge and type of external interrupts EX0 and EX1.

TCON (timer control) register is an 8-bit register. TCON register is a bit-addressable register

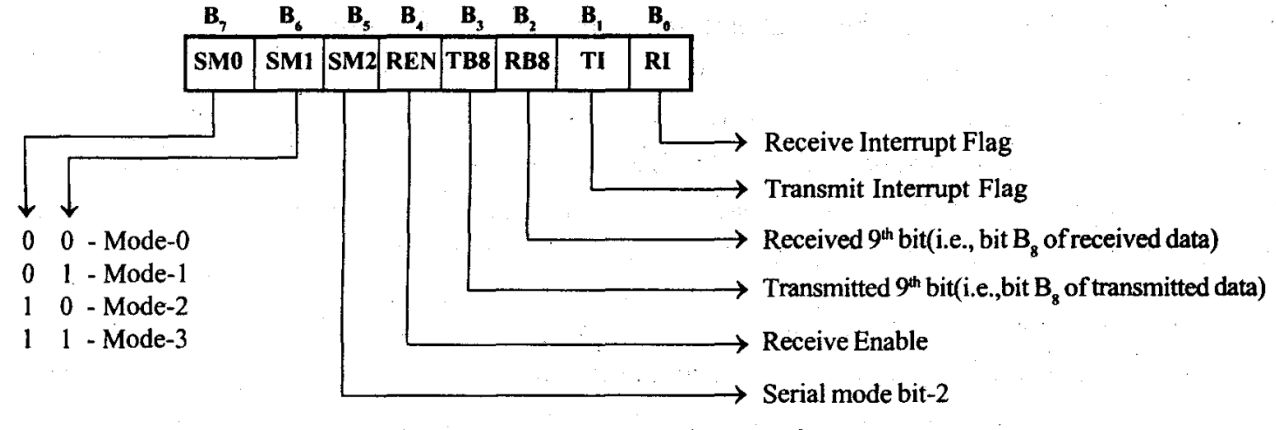
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

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

4. SCON Register(Serial control register)

Used to set RI and TI interrupt flags of serial communication

SM0	SM1	SM2	REN	TB8	RB8	TI	RI



****

<u>Timer Interrupt Programming</u>

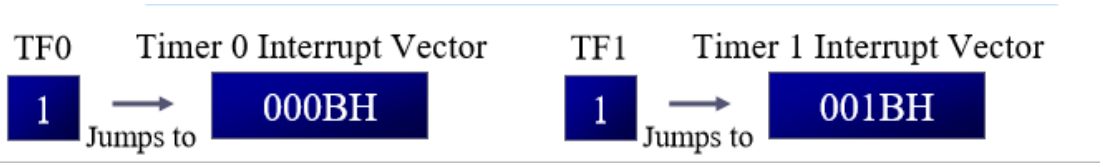
The timer flag (TF) is raised when the timer rolls over

In polling TF, we have to wait until the TF is raised

- The problem with this method is that the microcontroller is tied down while waiting for TF to be raised, and cannot do anything else

Using interrupts solves this problem and, avoids tying down the controller

- If the timer interrupt in the IE register is enabled, whenever the timer rolls over, TF is raised, and the microcontroller is interrupted in whatever it is doing, and jumps to the interrupt vector table to service the ISR.
- In this way, the microcontroller can do other until it is notified that the timer has rolled over.



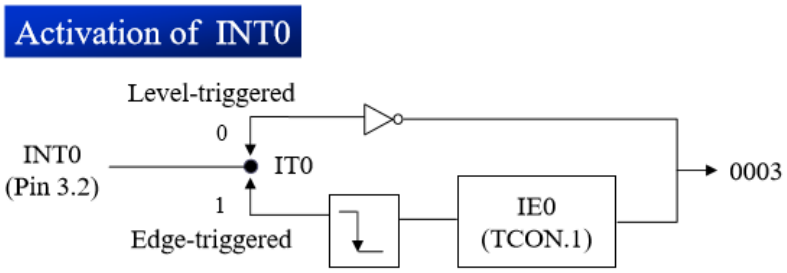
Example 11-4
Write a program to generate a square wave if 50Hz frequency on pin P1.2. This is similar to Example 9-12 except that it uses an interrupt for timer 0. Assume that XTAL=11.0592 MHz

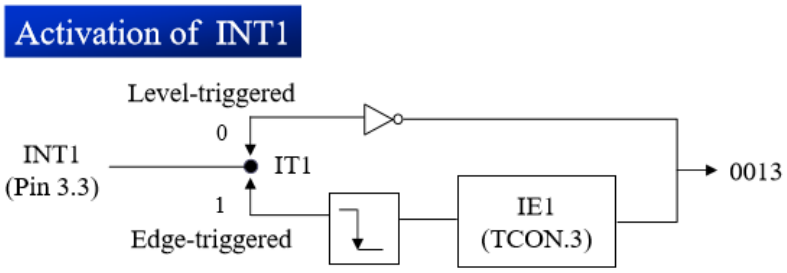
Solution:

```
      ORG  0
      LJMP MAIN
      ORG  000BH  ;ISR for Timer 0
      CPL  P1.2
      MOV  TL0,#00
      MOV  TH0,#0DCH
      RETI
      ORG  30H
;--------main program for initialization
MAIN:MOV  TM0D,#00000001B  ;Timer 0, Mode 1
      MOV  TL0,#00
      MOV  TH0,#0DCH
      MOV  IE,#82H   ;enable Timer 0 interrupt
      SETB TR0
HERE:SJMP HERE
      END
```

The 8051 has two external hardware interrupts

- Pin 12 (P3.2) and pin 13 (P3.3) of the 8051, designated as INT0 and INT1, are used as external hardware interrupts
- The interrupt vector table locations 0003H and 0013H are set aside for INT0 and INT1
- There are two activation levels for the external hardware interrupts
  - Level triggered
  - Edge triggered





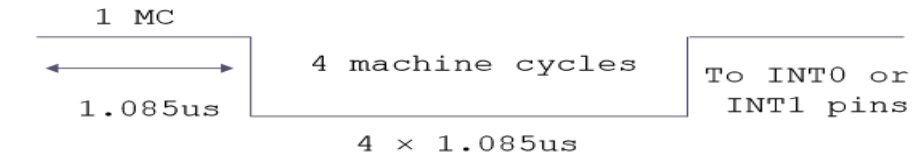
In the level-triggered mode, INT0 and INT1 pins are normally high

- If a low-level signal is applied to them, it triggers the interrupt
- Then the microcontroller stops whatever it is doing and jumps to the interrupt vector table to service that interrupt
- The low-level signal at the INT pin must be removed before the execution of the last instruction of the ISR, RETI; otherwise, another interrupt will be generated
- This is called a level-triggered or level activated interrupt and is the default mode upon reset of the 8051

Pins P3.2 and P3.3 are used for normal I/O unless the INT0 and INT1 bits in the IE register are enabled

- After the hardware interrupts in the IE register are enabled, the controller keeps sampling the INTn pin for a low-level signal once each machine cycle
- According to one manufacturer's data sheet,
  - The pin must be held in a low state until the start of the execution of ISR
  - If the INTnpin is brought back to a logic high before the start of the execution of ISR there will be no interrupt
  - If INTnpin is left at a logic low after the RETI instruction of the ISR, another interrupt will be activated after one instruction is executed
- To ensure the activation of the hardware interrupt at the INTnpin, make sure that the duration of the low-level signal is around 4 machine cycles, but no more .
  - This is due to the fact that the level-triggered interrupt is not latched

- Thus the pin must be held in a low state until the start of the ISR execution



note: On reset, IT0 (TCON.0) and IT1 (TCON.2) are both low, making external interrupt level-triggered

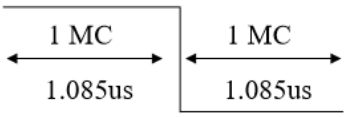
To make INT0 and INT1 edge triggered interrupts, we must program the bits of the TCON register

- The TCON register holds, among other bits, the IT0 and IT1 flag bits that determine level-or edge-triggered mode of the hardware interrupt
- IT0 and IT1 are bits D0 and D2 of the TCON register
- They are also referred to as TCON.0 and TCON.2 since the TCON register is bitaddressable.

In edge-triggered interrupts

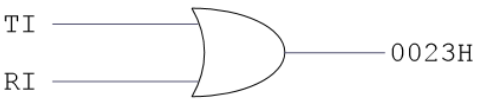
- The external source must be held high for at least one machine cycle, and then held low for at least one machine cycle
- The falling edge of pins INT0 and INT1 are latched by the 8051 and are held by the TCON.1 and TCON.3 bits of TCON register
  - Function as interrupt-in-service flags
  - It indicates that the interrupt is being serviced now and on this INTn pin, and no new interrupt will be responded to until this service is finished

Minimum pulse duration to detect edge-triggered interrupts XTAL=11.0592MHz



In the 8051 there is only one interrupt set aside for serial communication

- This interrupt is used to both send and receive data
- If the interrupt bit in the IE register (IE.4) is enabled, when RI or TI is raised the 8051 gets interrupted and jumps to memory location 0023H to execute the ISR
- In that ISR we must examine the TI and RI flags to see which one caused the interrupt and respond accordingly.



Serial interrupt is invoked by TI or RI flags

Explain Timer modes of 8051 microcontroller.(April 2017)

****

PROGRAMMING TIMERS OF 8051

1. Timer Registers.

The 8051 has two timers/counters, they can be used either as

- Timers are used to generate a time delay or as Event counters to count events happening outside the microcontroller.

Both Timer0 and Timer1 registers are 16 bits wide.

- Since 8051 has an 8-bit architecture, each 16-bits timer is accessed as two separate registers of low byte and high byte. The low byte register is called TL0/TL1 and the high byte register is called TH_0/TH_1.It can be accessed like any other register

For example MOV TL_0,#4FH
MOV R_5, TH_0

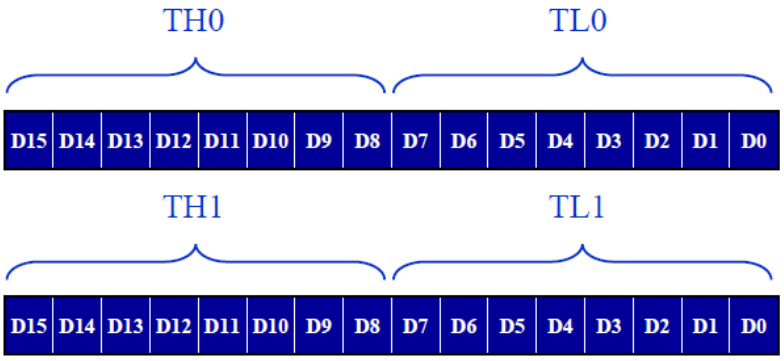


Figure:Timer Registers

2. TMOD (Timer mode Register)

Both timers 0 and 1 use the same register, called TMOD (timer mode), to set the various timer operation modes

- TMOD is an 8-bit register
- The lower 4 bits are for Timer 0
- The upper 4 bits are for Timer 1

In each case,

- The lower 2 bits are used to set the timer mode
- The upper 2 bits to specify the operation

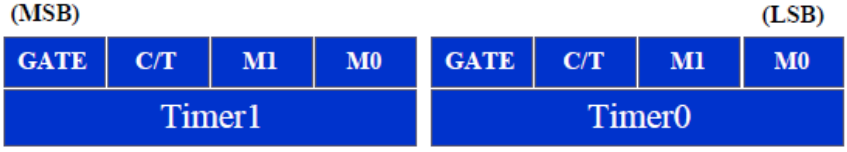


Figure:TMOD Register

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

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

Timers of 8051 do starting and stopping by either software or hardware control

For using software to start and stop the timer where GATE = 0

- The start and stop of the timer are controlled by way of software by the TR (timer start) bits TR0 and TR1
- The SETB instruction starts it, and it is stopped by the CLR instruction.
- These instructions start and stop the timers as long as GATE=0 in the TMOD register
- The hardware way of starting and stopping the timer by an external source is achieved by making GATE=1 in the TMOD register.
- The another register used in timer programming is TCON register.

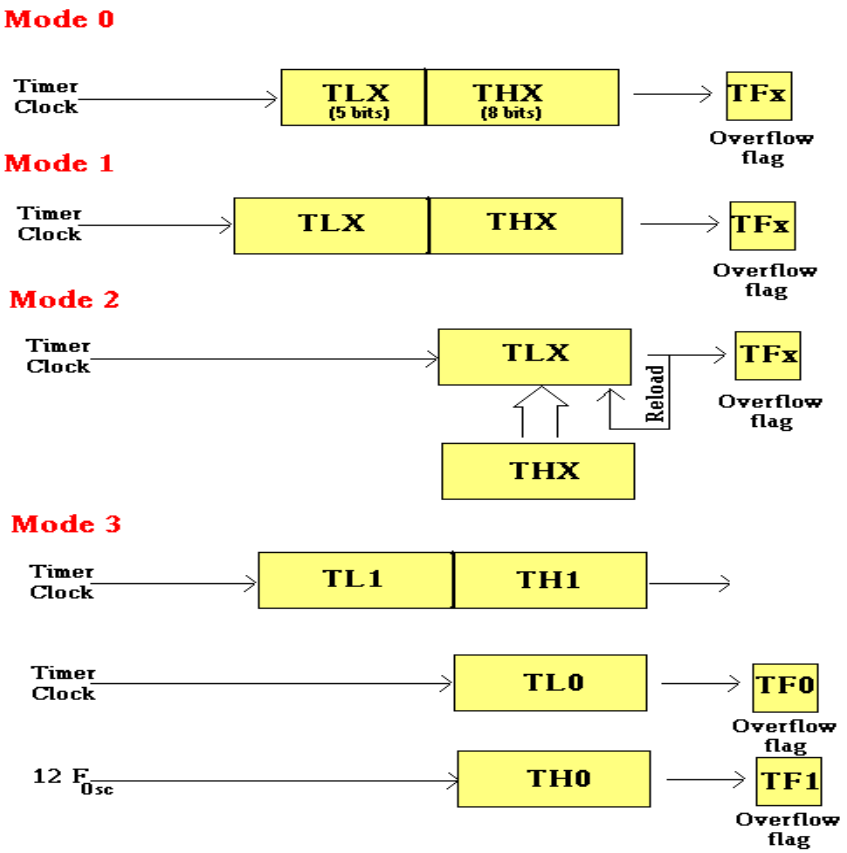
3. TCON (Timer control register)

TCON (timer control) register is an 8-bit register. TCON register is a bit-addressable register

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

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

Modes of operation of 8051 timers



:

MODE 1:16 bit Timer

The following are the characteristics and operations of mode1:

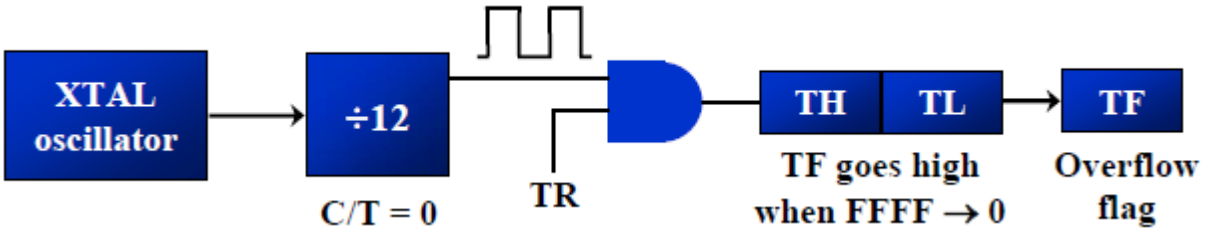
1. It is a 16-bit timer; therefore, it allows value of 0000 to FFFFH to be loaded into the timer's register TL and TH

2. After TH and TL are loaded with a 16-bit initial value, the timer must be started.

- This is done by SETB TR0 for timer 0 and SETB TR1 for timer 1

3. After the timer is started, it starts to count up

- It counts up until it reaches its limit of FFFFH



- When it rolls over from FFFFH to 0000, it sets high a flag bit called TF (timer flag)
- Each timer has its own timer flag: TF0 for timer 0, and TF1 for timer 1. This timer flag can be monitored

- When this timer flag is raised, one option would be to stop the timer with the instructions CLR TR0 or CLR TR1, for timer 0 and timer 1, respectively.
- . After the timer reaches its limit and rolls over, in order to repeat the process TH and TL must be reloaded with the original value, and TF must be reloaded to 0.

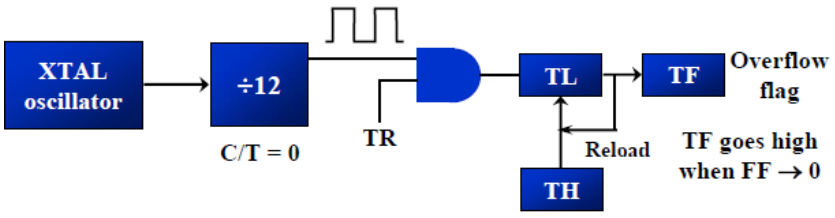
To generate a time delay

1. Load the TMOD value register indicating which timer (timer 0 or timer 1) is to be used and which timer mode (0 or 1) is selected
2. Load registers TL and TH with initial count value
3. Start the timer
4. Keep monitoring the timer flag (TF) with the JNB TFx ,target instruction to see if it is raised
- Get out of the loop when TF becomes high
5. Stop the timer
6. Clear the TF flag for the next round
7. Go back to Step 2 to load TH and TL again.

MODE 2:8 bit Timer Autoreload

The following are the characteristics and operations of mode 2:

1. It is an 8-bit timer; therefore, it allows only values of 00 to FFH to be loaded into the timer's register TH
2. After TH is loaded with the 8-bit value,the 8051 gives a copy of it to TL
   - Then the timer must be started
   - This is done by the instruction SETB TR0 for timer 0 and SETB TR1 for timer 1
3. After the timer is started, it starts to count up by incrementing the TL register
   - It counts up until it reaches its limit of FFH
   - When it rolls over from FFH to 00, it sets high the TF (timer flag)
   - When the TL register rolls from FFH to 0 and TF is set to 1, TL is reloaded automatically with the original value kept by the TH register.
   - To repeat the process, we must simply clear TF and let it go without any need by the programmer to reload the original value.
   - This makes mode 2 an auto-reload, in contrast with mode 1 in which the programmer has to reload TH and TL.

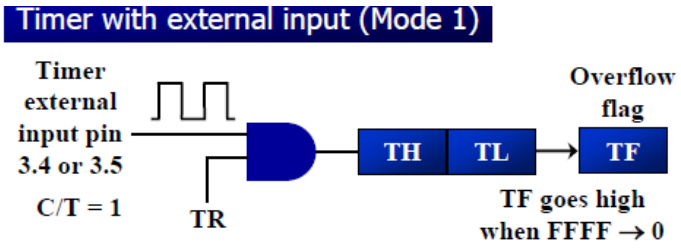


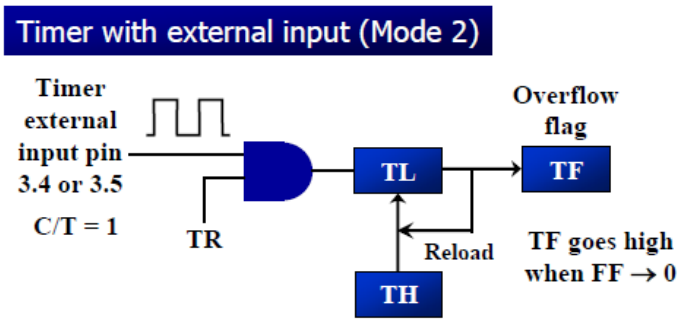
To generate a time delay

1. Load the TMOD value register indicating which timer (timer 0 or timer 1) is to be used, and the timer mode (mode 2) is selected
2. Load the TH registers with the initial count value
3. Start timer
4. Keep monitoring the timer flag (TF) with the JNB TFx, target instruction to see whether it is raised
   - Get out of the loop when TF goes high
5. Clear the TF flag
6. Go back to Step4, since mode 2 is autoreload.

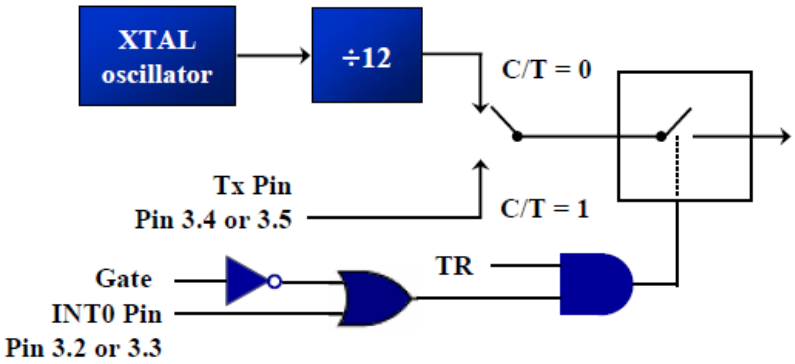
Timers as counters

- Timers can also be used as counters which are used for counting events happening outside the 8051.
- When it is used as a counter, it is a pulse outside of the 8051 that increments the TH, TL register.
- TMOD and TH, TL registers are the same as for the timer discussed previously except the source of the frequency The C/T bit in the TMOD registers decides the source of the clock for the timer.
- When C/T = 1, the timer is used as a counter and gets its pulses from outside the 8051.
- The counter counts up as pulses are fed from pins 14 and 15, these pins are called T0 (timer0 input) and T1 (timer 1 input).





- If GATE = 1, the start and stop of the timer are done externally through pins P3.2 and P3.3 for timers 0 and 1, respectively
- This hardware way allows to start or stop the timer externally at any time via a simple switch



- The frequency for the timer is always 1/12th the frequency of the crystal attached to the 8051, regardless of the 8051 version.

.Explain the serial programming of 8051 with its associated registers.[December 2017]
Explain how to program for sending and receiving data serially using 8051.
SERIAL COMMUNICATION PROGRAMMING

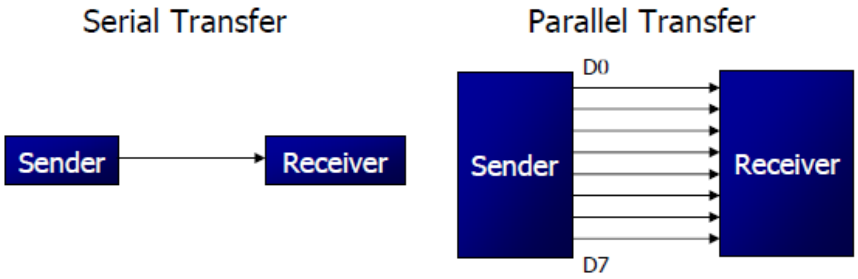
Computers transfer data in two ways:

- **Parallel**

Often 8 or more lines (wire conductors) are used to transfer data to a device that is only a few feet away

- **Serial**

To transfer to a device located many meters away, the serial method is used.The data is sent one bit at a time.



At the transmitting end, the byte of data must be converted to serial bits using parallel-in-serial-out shift register

- At the receiving end, there is a serial in- parallel-out shift register to receive the serial data and pack them into byte.
- When the distance is short, the digital signal can be transferred as it is on a simple wire and requires no modulation.
- If data is to be transferred on the telephone line, it must be converted from 0s and 1s to audio tones. This conversion is performed by a device called a **modem**, "Modulator/demodulator.

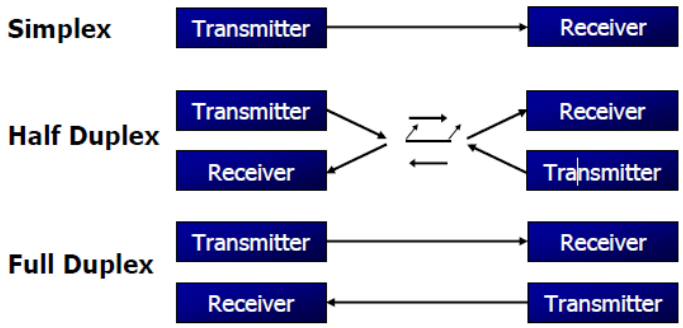
Serial data communication uses two methods

- **Synchronous method** transfers a block of data at a time.
- **Asynchronous method** transfers a single byte at a time.

It is possible to write software to use either of these methods, but the programs can be tedious and long
There are special IC chips made by many manufacturers for serial communications

- **UART** (universal asynchronous Receiver/transmitter)
- **USART** (universal synchronous-asynchronous Receiver-transmitter)
  - If data can be transmitted and received, it is a duplex transmission

- If data transmitted one way a time, it is referred to as half duplex.
- If data can go both ways at a time, it is full duplex.
- This is contrast to simplex transmission.

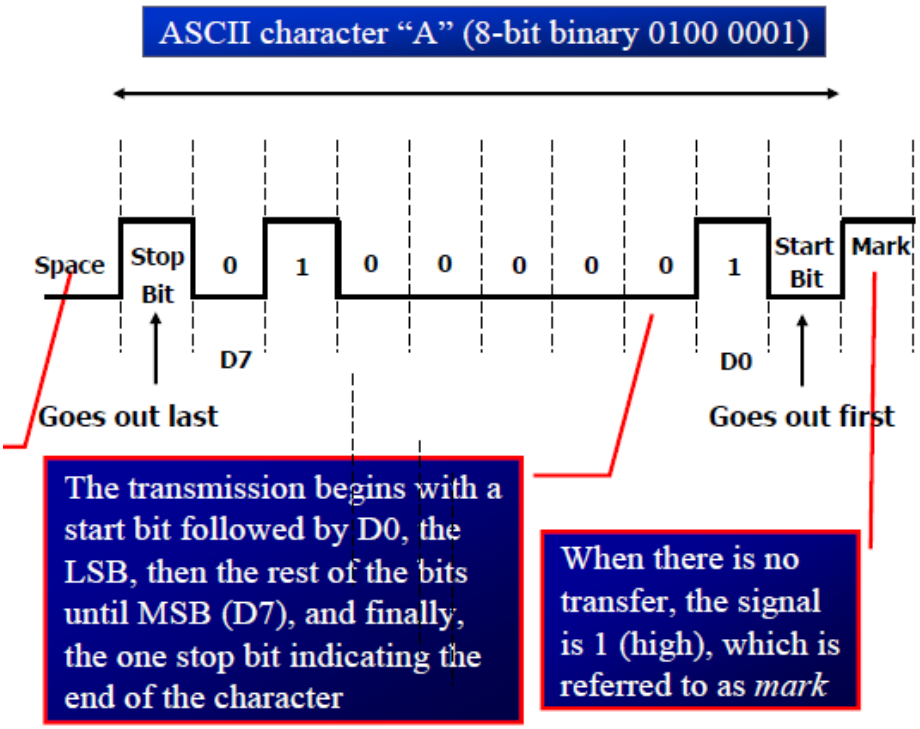


Asynchronous serial data communication is widely used for character-oriented transmissions.

 Each character is placed in between start and stop bits, this is called **framing.**

 Block-oriented data transfers use the synchronous method

 The start bit is always one bit, but the stop bit can be one or two bits .The start bit is always a 0 (low) and the stop bit(s) is 1 (high)



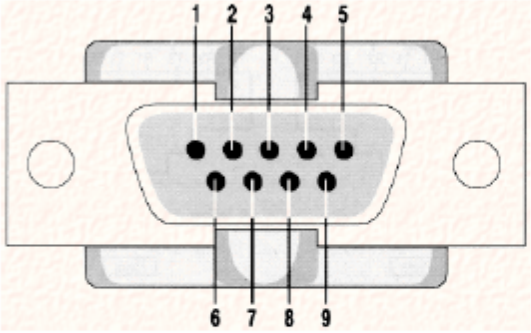
The rate of data transfer in serial data communication is stated in bps (bits per second).Another widely used terminology for bps is baud rate

RS232

It is an interfacing standard RS232 was set by the Electronics Industries Association (EIA) in 1960. The standard was set long before the advent of the TTL logic family, its input and output voltage levels are not TTL compatible

In RS232, a 1 is represented by -3 ~ -25 V,while a 0 bit is +3 ~ +25 V, making -3 to +3 undefined Since not all pins are used in PC cables,IBM introduced the DB-9 version of the serial I/O standard

RS232 Connector DB-9



RS232 DB-9 Pins

Pin	Description
1	Data carrier detect (-DCD)
2	Received data (RxD)
3	Transmitted data (TxD)
4	Data terminal ready (DTR)
5	Signal ground (GND)
6	Data set ready (-DSR)
7	Request to send (-RTS)
8	Clear to send (-CTS)
9	Ring indicator (RI)

Handshake signals of MODEM

DTR (data terminal ready)
When terminal is turned on, it sends out signal DTR to indicate that it is ready for communication

DSR (data set ready)
When DCE is turned on and has gone through the self-test, it assert DSR to indicate that it is ready to communicate

RTS (request to send)
When the DTE device has byte to transmit,it assert RTS to signal the modem that it has a byte of data to transmit

CTS (clear to send)
When the modem has room for storing the data it is to receive, it sends out signal CTS to DTE to indicate that it can receive the data now.

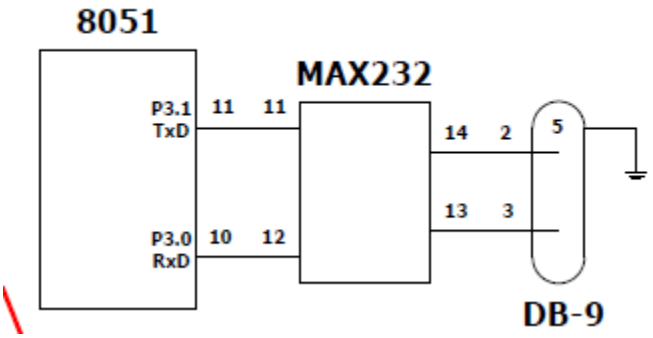
DCD (data carrier detect)
The modem asserts signal DCD to inform the DTE that a valid carrier has been detected and that contact between it and the other modem is established

RI (ring indicator)
An output from the modem and an input to a PC indicates that the telephone is ringing
It goes on and off in synchronous with the ringing sound.

MAX232

MAX232 chip is called as a line driver which is required to convert RS232 voltage levels to TTL levels, and vice versa.

- 8051 has two pins that are used specifically for transferring and receiving data serially.
- These two pins are called TxD and RxD and are part of the port 3 group (P3.0 and P3.1).
- These pins are TTL compatible; therefore, they require a line driver to make them RS232 compatible. We need a line driver (voltage converter) to convert the R232's signals to TTL voltage levels that will be acceptable to 8051's TxD and RxD pins.



A line driver such as the MAX232 chip is required to convert RS232 voltage levels to TTL levels, and vice versa .

- 8051 has two pins that are used specifically for transferring and receiving data serially.
- These two pins are called TxD and RxD and are part of the port 3 group (P3.0 and P3.1).
- These pins are TTL compatible; therefore,they require a line driver to make them RS232. compatible.

SBUF is an 8-bit register used solely for serial communication.

- For a byte data to be transferred via the TxD line, it must be placed in the SBUF register.
- The moment a byte is written into SBUF, it is framed with the start and stop bits and transferred serially via the TxD line.
- SBUF holds the byte of data when it is received by 8051 RxD line.
- When the bits are received serially via RxD, the 8051 deframes it by eliminating the stop and
- start bits, making a byte out of the data received.

SCON is an 8-bit register used to program the start bit, stop bit, and data bits of data framing, among other things

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	SCON.7	Serial port mode specifier
SM1	SCON.6	Serial port mode specifier
SM2	SCON.5	Used for multiprocessor communication
REN	SCON.4	Set/cleared by software to enable/disable reception
TB8	SCON.3	Not widely used
RB8	SCON.2	Not widely used
TI	SCON.1	Transmit interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW
RI	SCON.0	Receive interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW

Note: Make SM2, TB8, and RB8 =0

SM0, SM1

They determine the framing of data by specifying the number of bits per character, and the start and stop bits
This enables the multiprocessing capability of the 8051

SM0	SM1	
0	0	Serial Mode 0
0	**1**	**Serial Mode 1, 8-bit data, 1 stop bit, 1 start bit**
1	0	Serial Mode 2
1	1	Serial Mode 3

Only mode 1 is of interest to us

SM2

When 8051 receives data serially via RxD, it gets rid of the start and stop bits and places the byte in SBUF register

 It raises the RI flag bit to indicate that a byte has been received and should be picked up before it is lost

 RI is raised halfway through the stop bit

In programming the 8051 to transfer character bytes serially

1. TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode 2 (8-bit auto-reload) to set baud rate.
2. The TH1 is loaded with one of the values to set baud rate for serial data transfer
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits
4. TR1 is set to 1 to start timer 1
5. TI is cleared by CLR TI instruction
6. The character byte to be transferred serially is written into SBUF register
7. The TI flag bit is monitored with the use of instruction JNB TI,xx to see if the character has been transferred completely.
8. To transfer the next byte, go to step 5.

Write a program for the 8051 to transfer letter "A" serially at 4800 baud, continuously.

Solution:

```
        MOV  TMOD,#20H  ;timer 1,mode 2(auto reload)
        MOV  TH1,#-6    ;4800 baud rate
        MOV  SCON,#50H  ;8-bit, 1 stop, REN enabled
        SETB TR1        ;start timer 1
AGAIN:  MOV  SBUF,#"A"  ;letter "A" to transfer
HERE:   JNB  TI,HERE    ;wait for the last bit
        CLR  TI         ;clear TI for next char
        SJMP AGAIN      ;keep sending A
```

Write a program for the 8051 to transfer "YES" serially at 9600 baud, 8-bit data, 1 stop bit, do this continuously

Solution:

```
        MOV  TMOD,#20H  ;timer 1,mode 2(auto reload)
        MOV  TH1,#-3    ;9600 baud rate
        MOV  SCON,#50H  ;8-bit, 1 stop, REN enabled
        SETB TR1        ;start timer 1
AGAIN:  MOV  A,#"Y"     ;transfer "Y"
        ACALL TRANS
        MOV  A,#"E"     ;transfer "E"
        ACALL TRANS
        MOV  A,#"S"     ;transfer "S"
        ACALL TRANS
        SJMP AGAIN      ;keep doing it
;serial data transfer subroutine
TRANS:  MOV  SBUF,A     ;load SBUF
HERE:   JNB  TI,HERE    ;wait for the last bit
        CLR  TI         ;get ready for next byte
         RET
```

The steps that 8051 goes through in transmitting a character via TxD

1. The byte character to be transmitted is written into the SBUF register
2. The start bit is transferred
3. The 8-bit character is transferred on bit at a time
4. The stop bit is transferred

- It is during the transfer of the stop bit that 8051 raises the TI flag, indicating that the last character was transmitted

5. By monitoring the TI flag, we make sure that we are not overloading the SBUF

- If we write another byte into the SBUF before TI is raised, the un transmitted portion of the previous byte will be lost

6. After SBUF is loaded with a new byte, the TI flag bit must be forced to 0 by CLR TI in order for this new byte to be transferred

- By checking the TI flag bit, we know whether or not the 8051 is ready to transfer another byte
- It must be noted that TI flag bit is raised by 8051 itself when it finishes data transfer
- It must be cleared by the programmer with instruction CLR TI
- If we write a byte into SBUF before the TI flag bit is raised, we risk the loss of a portion of the byte being transferred
- The TI bit can be checked by
- The instruction JNB TI,xx Using an interrupt

In programming the 8051 to receive character bytes serially

1. TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode 2 (8-bit auto-reload) to set baud rate

2. TH1 is loaded to set baud rate
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits
4. TR1 is set to 1 to start timer 1
5. RI is cleared by CLR RI instruction
6. The RI flag bit is monitored with the use of instruction JNB RI,xx to see if an entire character has been received yet
7. When RI is raised, SBUF has the byte, its contents are moved into a safe place.
8. To receive the next character, go to step 5.

Write a program for the 8051 to receive bytes of data serially, and put them in P1, set the baud rate at 4800, 8-bit data, and 1 stop bit

Solution:

```
        MOV  TMOD,#20H  ;timer 1,mode 2(auto reload)
        MOV  TH1,#-6    ;4800 baud rate
        MOV  SCON,#50H  ;8-bit, 1 stop, REN enabled
        SETB TR1        ;start timer 1
HERE:   JNB  RI,HERE    ;wait for char to come in
        MOV  A,SBUF     ;saving incoming byte in A
        MOV  P1,A       ;send to port 1
        CLR  RI         ;get ready to receive next
                        ;byte
        SJMP HERE       ;keep getting data
```

In receiving bit via its RxD pin, 8051 goes through the following steps.

1. It receives the start bit
   - Indicating that the next bit is the first bit of the character byte it is about to receive
2. The 8-bit character is received one bit at time
3. The stop bit is received
   - When receiving the stop bit 8051 makes RI = 1,indicating that an entire character byte has been received and must be picked up before itgets overwritten by an incoming character raised, we know that a character has been received and is sitting in the SBUF register
   - We copy the SBUF contents to a safe place in some other register or memory before it is lost
5. After the SBUF contents are copied into a safe place, the RI flag bit must be forced to 0 by CLR RI in order to allow the next received character byte to be placed in SBUF.
   - Failure to do this causes loss of the received character.

There are two ways to increase the baud rate of data transfer

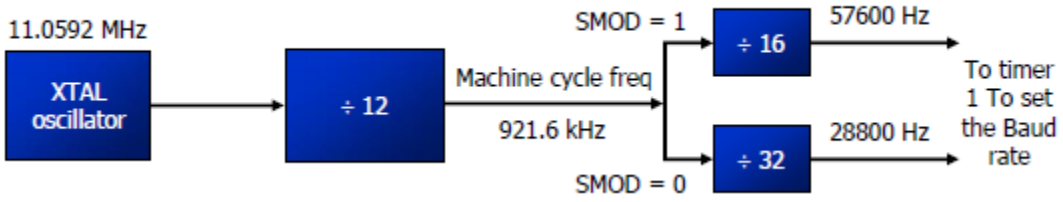
- To use a higher frequency crystal
- To change a bit in the PCON register

PCON

- PCON register is an 8-bit register
- When 8051 is powered up, SMOD is zero. We can set it to high by software and thereby double the baud rate.

SMOD	--	--	--	GF1	GF0	PD	IDL

```
MOV  A,PCON     ;place a copy of PCON in ACC
SETB ACC.7      ;make D7=1
MOV  PCON,A     ;changing any other bits
```



****

Explain the interfacing of external RAM and ROM with 8051.
Write short notes on memory addressing (November 2007,December 2017)
Explain how to access external memory devices in an 8051 based system.

Interfacing to external memory

For 8751/89C51/DS5000-based system,
EA (External access)

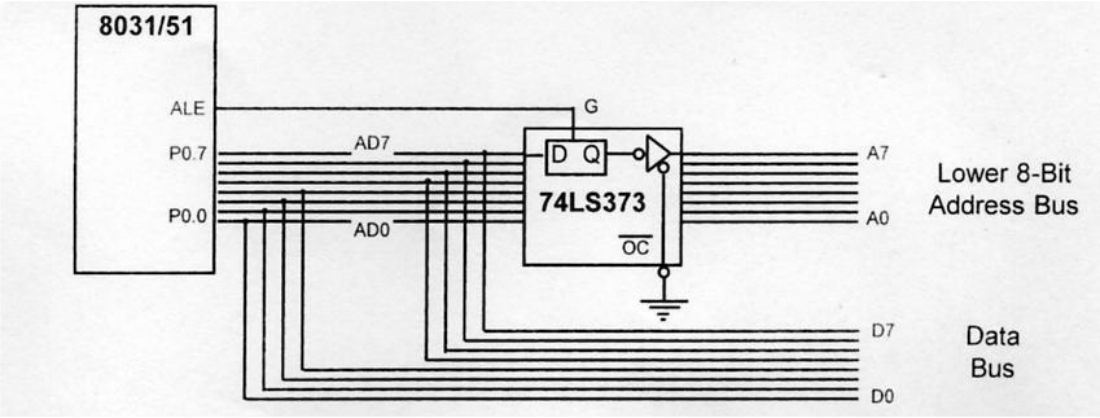
- we connected the **EA** pin to **Vcc** to indicate that the program code is stored in the microcontroller's **on-chip ROM**
- To indicate that the program code is stored in **external ROM**, this pin must be connected to **GND.**

Since the PC (program counter) of the 8031/51 is 16-bit, it is capable of accessing up to 64K bytes of program code.

- In the 8031/51, port 0 and port 2 provide the 16-bit address to access external memory.
- P0 provides the lower 8 bit address A0 – A7, and P2 provides the upper 8 bit address A8 – A15
- P0 is also used to provide the 8-bit data bus D0 – D7.
- P0.0 – P0.7 are used for both the address and data paths using address/data multiplexing.

ALE (**address latch enable)** pin is an output pin for 8031/51

- ALE = 0, P0 is used for data path.
- ALE = 1, P0 is used for address path 74LS373 D Latch.
- To extract the address from the P0 pins we connect P0 to a 74LS373 and use the ALE pin to latch the address



PSEN (program store enable) signal is an output signal for the 8031/51 microcontroller and must be connected to the OE pin of a ROM containing the program code

 It is important to emphasize the role of EA and PSEN when connecting the 8031/51 to external ROM

 When the EA pin is connected to GND, the 8031/51 fetches opcode from external ROM by using PSEN

The connection of the PSEN pin to the OE pin of ROM

 In systems based on the 8751/89C51 DS5000 where EA is connected to Vcc,these chips do not activate the PSEN pin

 This indicates that the on-chip ROM contains program code.

Connection to External Program ROM

We use RD to connect the 8031/51 to external ROM containing data. For the ROM containing the program code, PSEN is used to fetch the code.

The 8051 has 128K bytes of address space

 64K bytes are set aside for program code

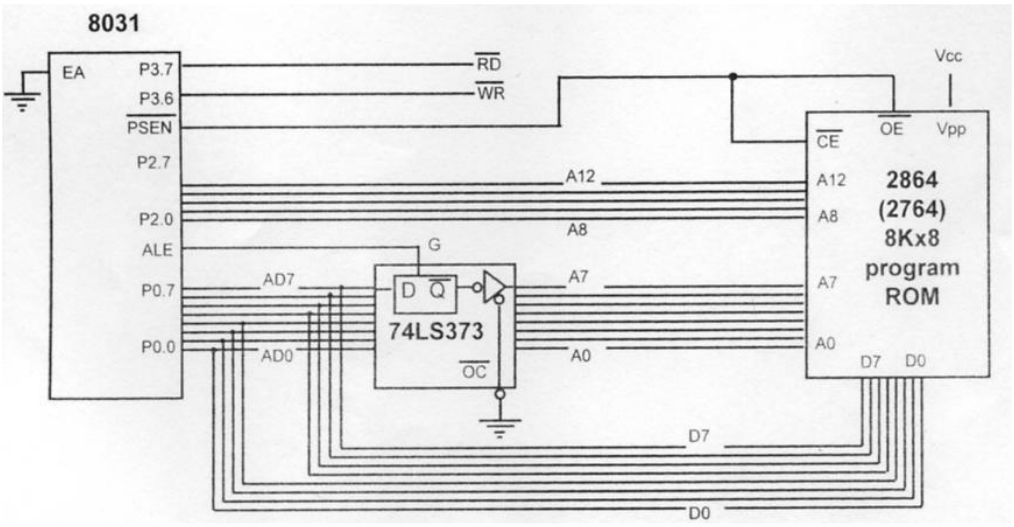
 Program space is accessed using the program counter (PC) to locate and fetch instructions

 In some example we placed data in the code space and used the instruction MOVC A,@A+DPTR to get data, where C stands for code

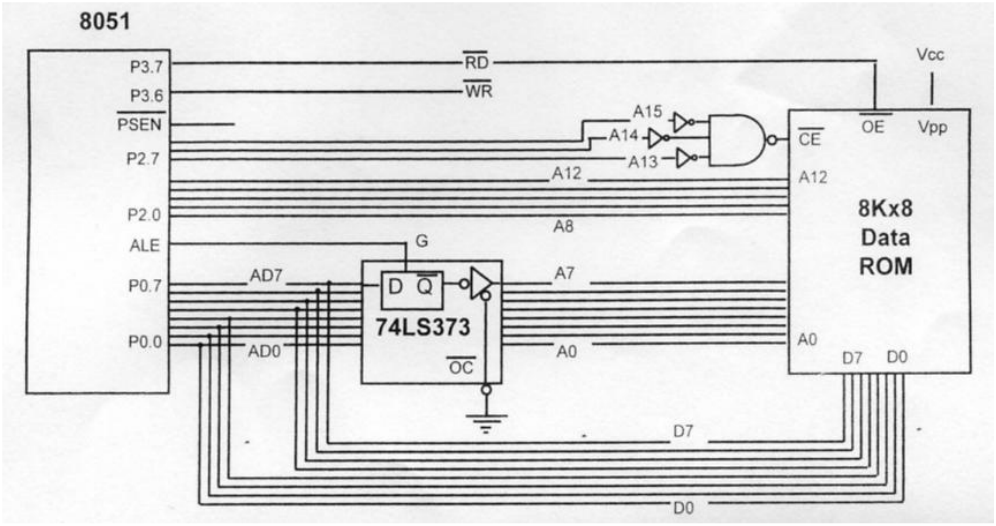
 The other 64K bytes are set aside for data. The data memory space is accessed using the DPTR register and an instruction called MOVX ,where X stands for external – The data memory space must be implemented externally.

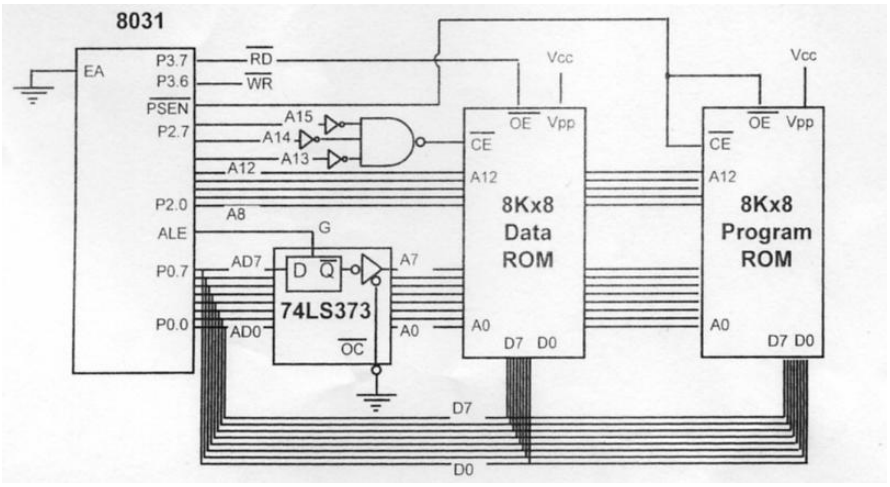
We use RD to connect the 8031/51 to external ROM containing data. For the ROM containing the program code, PSEN is used to fetch the code.

Connection to External program ROM



Connection to External Data ROM

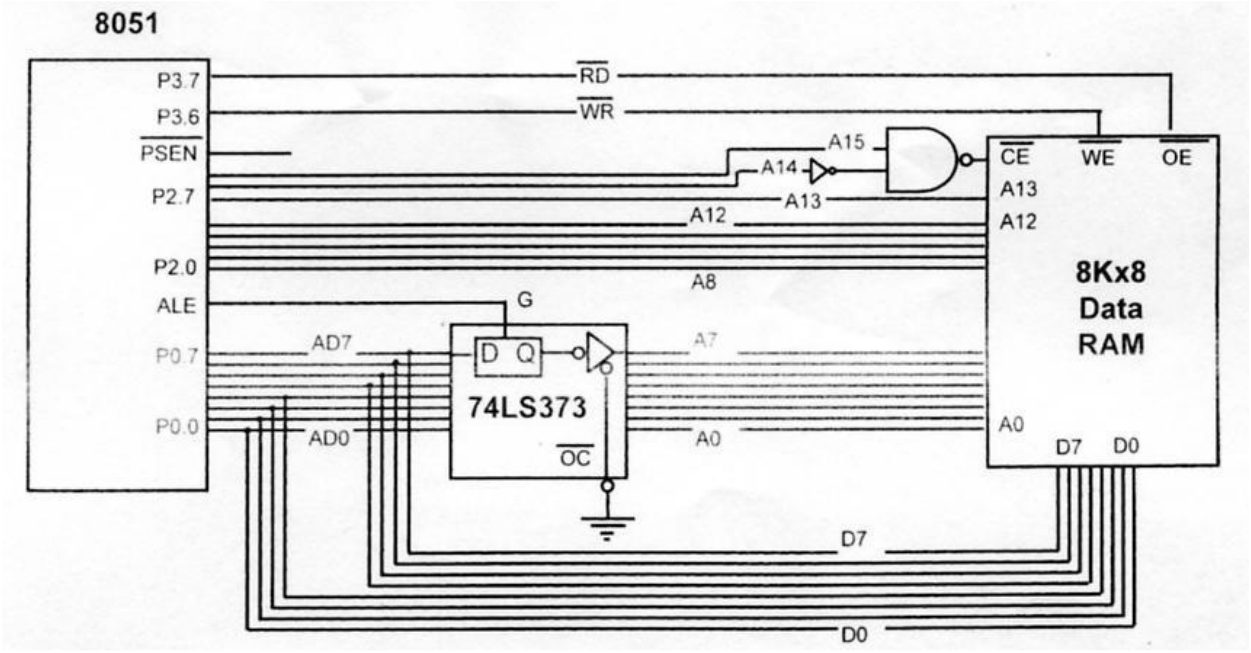




Interfacing external RAM with 8051

MOVX is a widely used instruction allowing access to external data memory space
 To bring externally stored data into the CPU, we use the instruction MOVX A,@DPTR

To connect the 8051 to an external SRAM, we must use both RD (P3.7) and WR (P3.6)



 In writing data to external data RAM, we use the instruction MOVX @DPTR,A

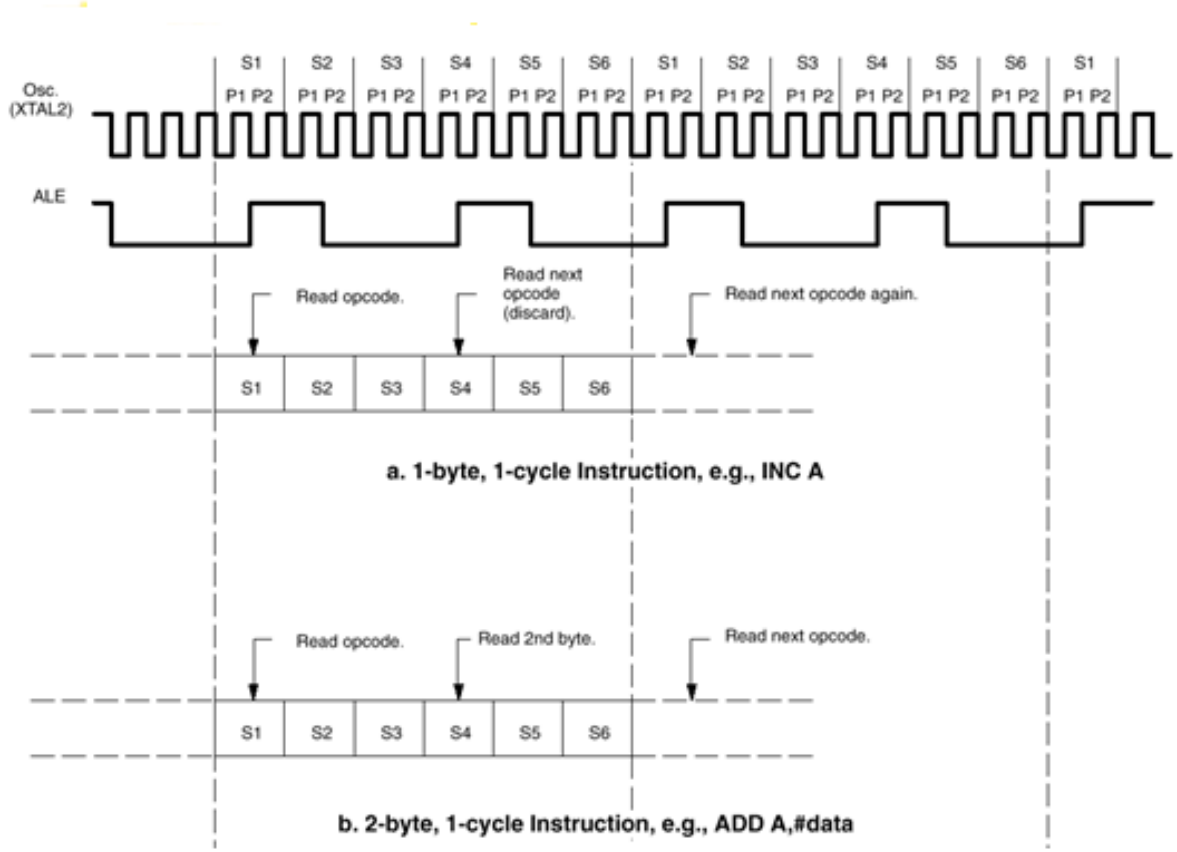
****

Timing Diagram

Instruction Timings

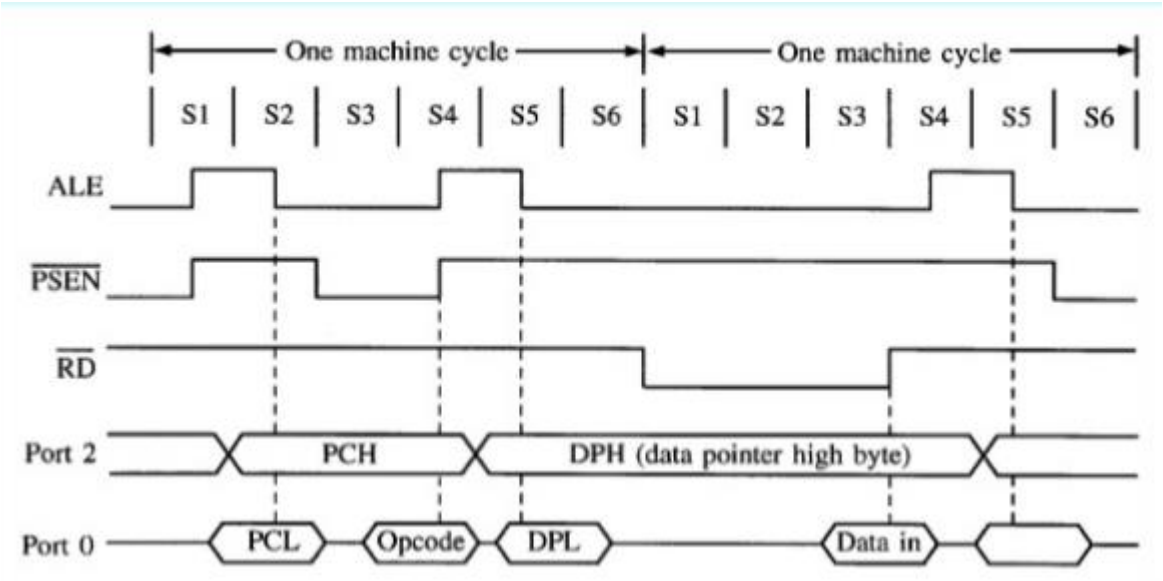
- One "machine cycle" = 6 states (S1 - S6)
- One state = 2 clock cycles
- One "machine cycle" = 12 clock cycles
- Instructions take 1 - 4 cycles
  - ✓ e.g. 1 cycle instructions: ADD, MOV, SETB, NOP
  - ✓ e.g. 2 cycle instructions: JMP, JZ
  - ✓ 4 cycle instructions: MUL, DIV

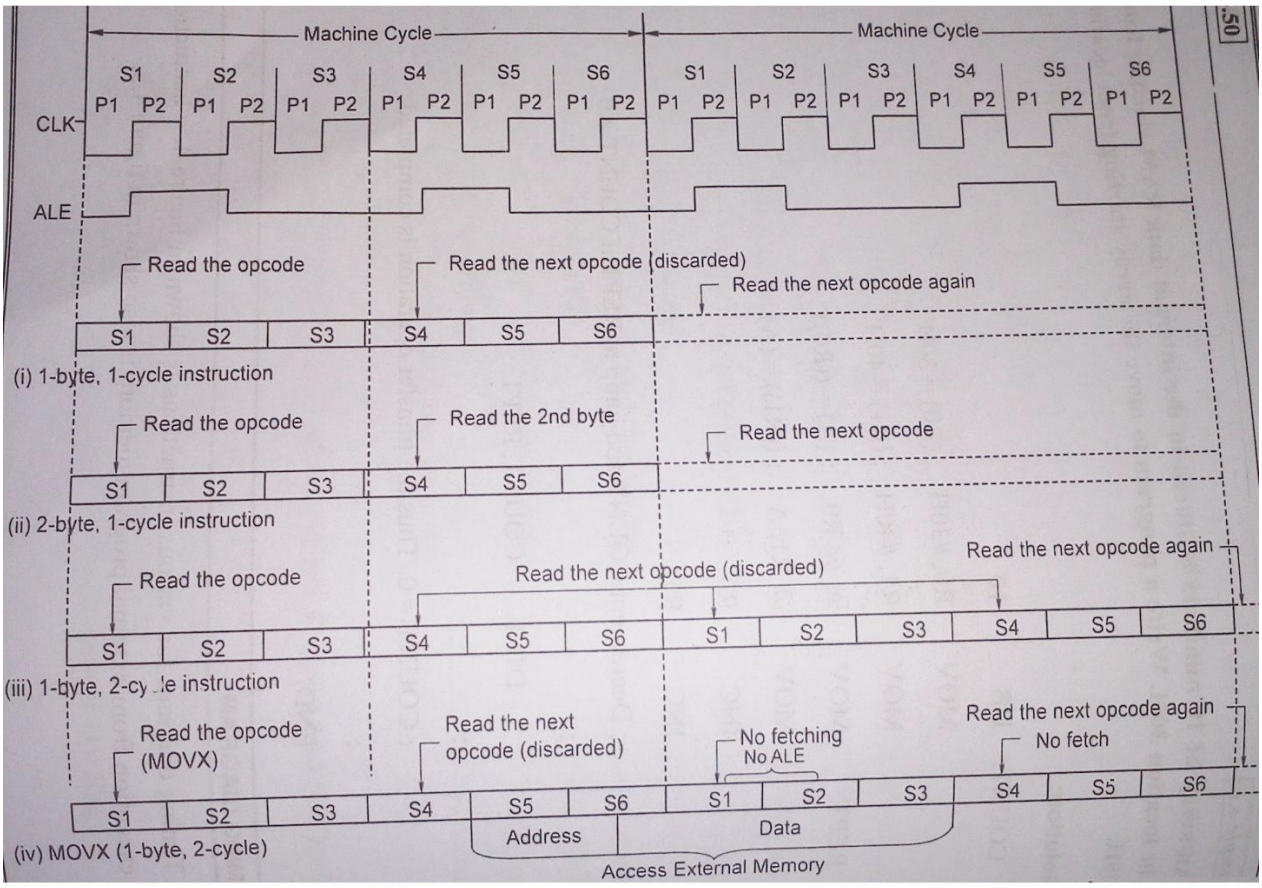
Instruction Timing



Describe the timing diagram of external data memory read cycle of 8051.(Dec 2018)

MOVX





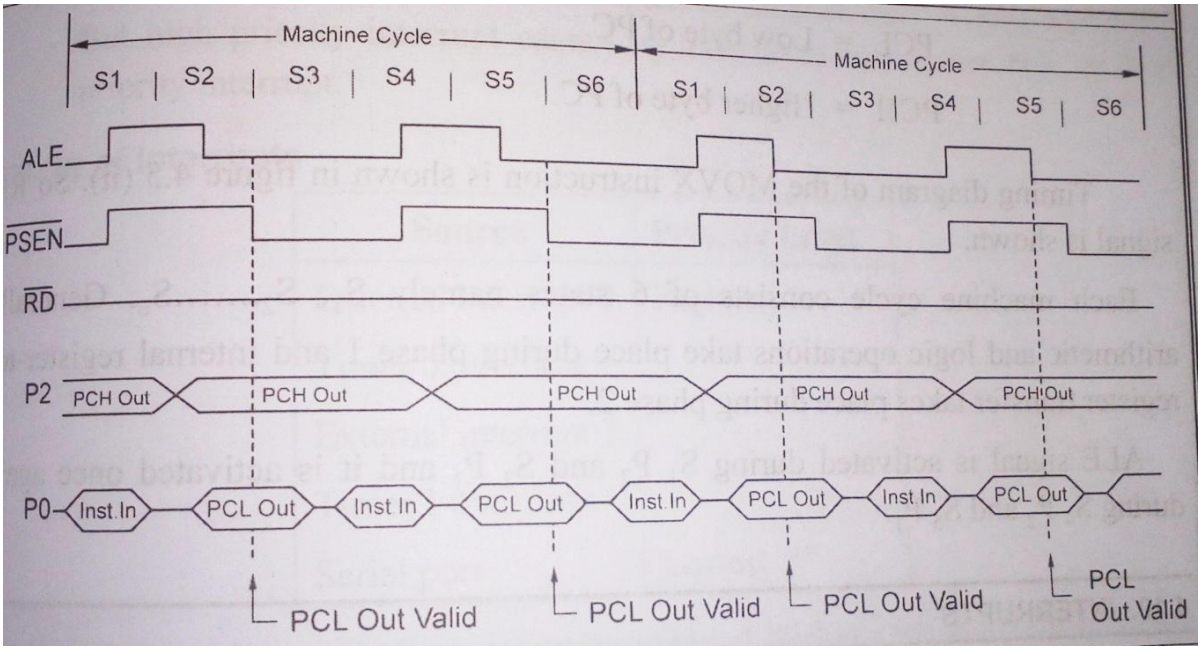
P_0 = Port 0

P_2 = Port2

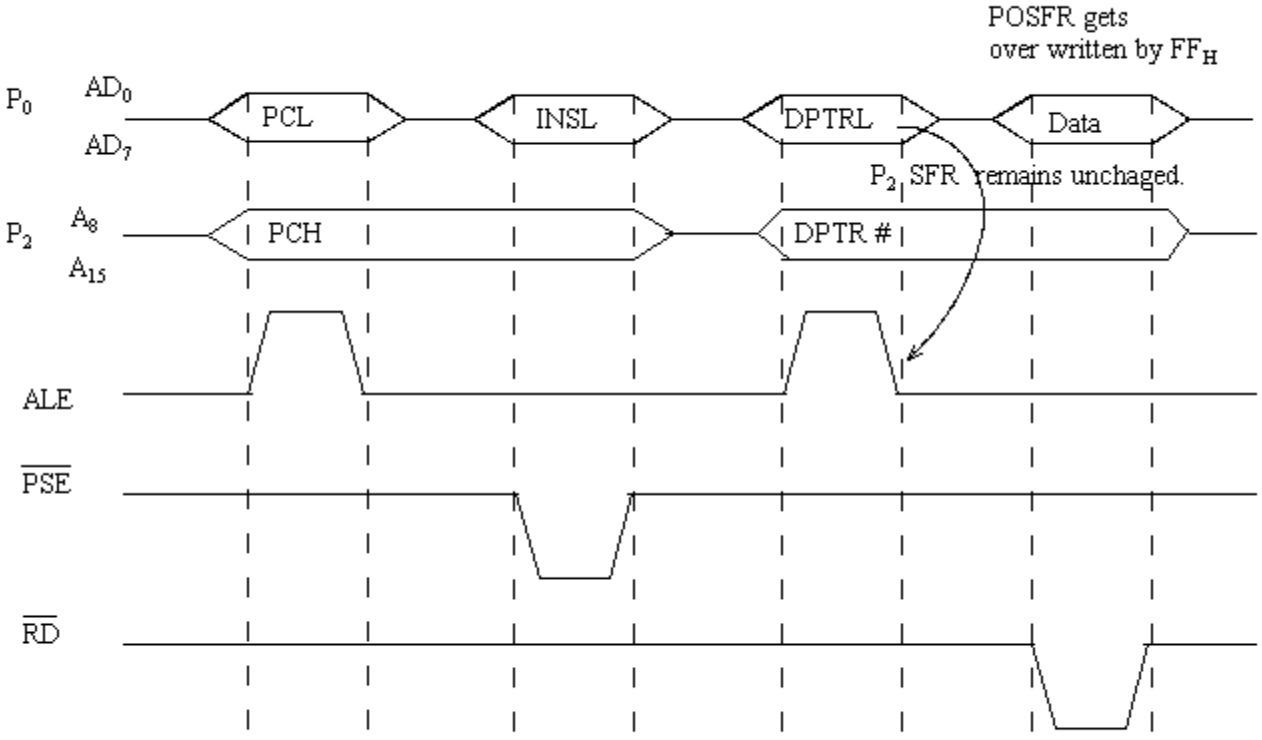
PCL = Low byte of PC

PCH = Higher byte of PC

- Timing diagram of the MOVX instruction is shown above.
- Each machine cycle consists of 6 states namely S_1, S_2 … S_6. Generally arithmetic and logic operations take place during phase 1 and internal register-to-register transfer takes place during phase 2.ALE signal is activated during S_1 P_2 and S_2 P_1 and it is activated once again during S_4 P_2 and S_5 P_1.



How MOV A,@DPTR works



For 8-bit Memory address access, P_2 Pins o/p the SFR register contents and helps in memory pages.

The higher order 8-bit address is taken the address available in the P_2 SFSR and the lower order 8-bit address is the data available in register RO.

****

SAMPLE PROGRAMS:

1. Add two 8-bit numbers

```
MOV A, #30H          ; (A)□30
ADD A, #50H          ; (A)□(A) + 50H
```

2. Add two 16- bit numbers

```
MOV DPRT, #2040H    ; (DPTR)□2040H (16 bit number)
MOV A, #2BH         ; (A)□2BH (lower byte of second 16 bit number) MOV A,
#20H          ; (B)□20H (Higher byte of second 16 bit number) ADD A, DPL
; Add lower bytes
MOV DPL, A                  ; Save result of lower byte addition MOV
A, B                        ; Get higher byte of second number in A
ADD A, DPH                  ; Add higher bytes with any carry from lower byte
addition MOV DPH, A         ; Save result of higher byte addition
```

3. Division two 8-bit numbers

```
MOV A, #90                  ; Get the first number in A
MOV B, #20                  ; Get the second number in B
DIV A, B                    ; A+B, Remainder in B and Quotient in A
```

4. Multiply two 8-bit numbers

```
MOV A, #8F                  ; Get the first number in A
MOV B, #79                  ; Get the second number in B
MUL A, B                    ; A x B, Higher byte of result in B and lower byte of result in A
```

5. To add two 16 bit BCD numbers

```
MOV DPTR, #1234H            ; Load first number
MOV R0, #20H                ; Load lower byte of second number
MOV R1, #30H                ; Load higher byte of second number
MOV A, R0                   ; Get the lower byte of second number
ADD A, DPL                  ; add two lower bytes
DA A                        ; Adjust result to valid BCD
MOV DPL, A                  ; Store the sum of lower bytes
MOV A, R1                   ; Get the higher byte of second number
ADDC A, DPH                 ; Add two higher bytes considering carry of lower byte
addition DA A               ; Adjust result to valid BCD
MOV DPH, A                  ; Store the sum of higher bytes
```

6. To find the sum of 10 numbers stored in the array

```
MOV DPTR, #2200H            ; Initialize memory pointer
MOVX A, @DPTR               ; Get the count
MOV R0, A                   ; Initialize the iteration counter
INC DPTR                    ; Initialize pointer to array of numbers
MOV R1, #00                 ; Result = 0
BACK: MOVX A, @DPTR         ; get the number
ADD A, R1                   ; A□Result + A
MOV R1, A                   ; Result □A
INC DPTR                    ; Increment the array pointer
DJNZ R0, BACK               ; Decrement iteration count if not zero
repeat MOV DPRT, #2300H     ; Initialize memory pointer
MOV A, R1                   ; Get the result
MOVX @DPTR, A               ; Store the result
```

****

UNIT IV
PERIPHERAL INTERFACING

****

Study on need, Architecture, configuration and interfacing, with ICs: 8255, 8259, 8254, 8237, 8251, 8279, - A/D and D/A converters &Interfacing with 8085& 8051.

****

1. Introduction:

Data Transfer Techniques

Data transfer may take place between two devices.
For e.g.

- Microprocessor and memory
- Microprocessor and I/O device
- Memory and I/O device

Classification of data transfer techniques

1. Programmed data transfer
2. Direct Memory Access (DMA)

Programmed data transfer

- Data is transferred from the I/O device to the microprocessor or memory.
- Data is transferred under the control of the program stored in the program memory of the microprocessor based system.
- This technique of data transfer is normally used if the size of data to be transferred is small.

Programmed data transfer techniques is classified as

- Parallel data transfer
- Serial data transfer
- Synchronous data transfer
- Asynchronous data transfer
- Interrupt Initiated data transfer

Comparison of Parallel/ Serial

Parallel	Serial
8 bits of data tranferred at a time 9 lines required to connect two devices Advantageous over small distances	Only 1 bit of data is transferred at a time 2 lines required to connect two devices Advantageous over long distances

Comparison of Asynchronous/ Synchronous Data Transfer techniques

Asynchronous	Synchronous
✵ Used to transfer one character at a time ✵ Start and stop bits are used with each character ✵ Speed is less ✵ Transmitter and receiver can use two separate clock inputs	✵ Used to transfer a block of characters at a time ✵ No start/ stop bits are used ✵ Speed is high ✵ Transmitter and receiver share a common clock

❖ **Interrupt Initiated Data Transfer**

- Microprocessor initiates the interrupt mechanism and starts executing the main program.
- I/O device informs the microprocessor that it is ready by generating an interrupt signal.
- Microprocessor services the interrupt by completing the data transfer.

- **DMA Controlled Data Transfer**
  - DMA stands for Direct Memory Access
  - used when large amount of data is to be transferred
  - Microprocessor does not participate in this type of data transfer
  - Data is transferred directly between an I/O device and memory or vice-versa
  - Data transfer is controlled by an I/O device or a DMA controller
  - DMA data transfer is fast as compared to programmed data transfer

****

Describe the internal block diagram of 8255 / PPI .(December 2010) (April 2018)(December 2017) Explain the functioning of 8255 programmable peripheral interface and its modes. [April/May 2017, May/June 2016, April/May 2015, Nov/Dec2015,April/May 2011,May/June 2014,May/June 2013,Nov/Dec 2013,May/June 2009]

****

. **2.<u>Parallel communication interface (8255)</u>(Programmable peripheral interface)**

Definition:

The 8255 is a general purpose programmable I/O device used for parallel data transfer. It can be programmed to transfer data under various conditions, from simple I/O to Interrupt I/O. It is flexible, versatile and economical when multiple I/O ports are required.

Functional Block Diagram

The 8255 consists of four sections namely,

- **Data bus buffer**
- **Read/write control logic**
- **Group A control**
- **Group B control**

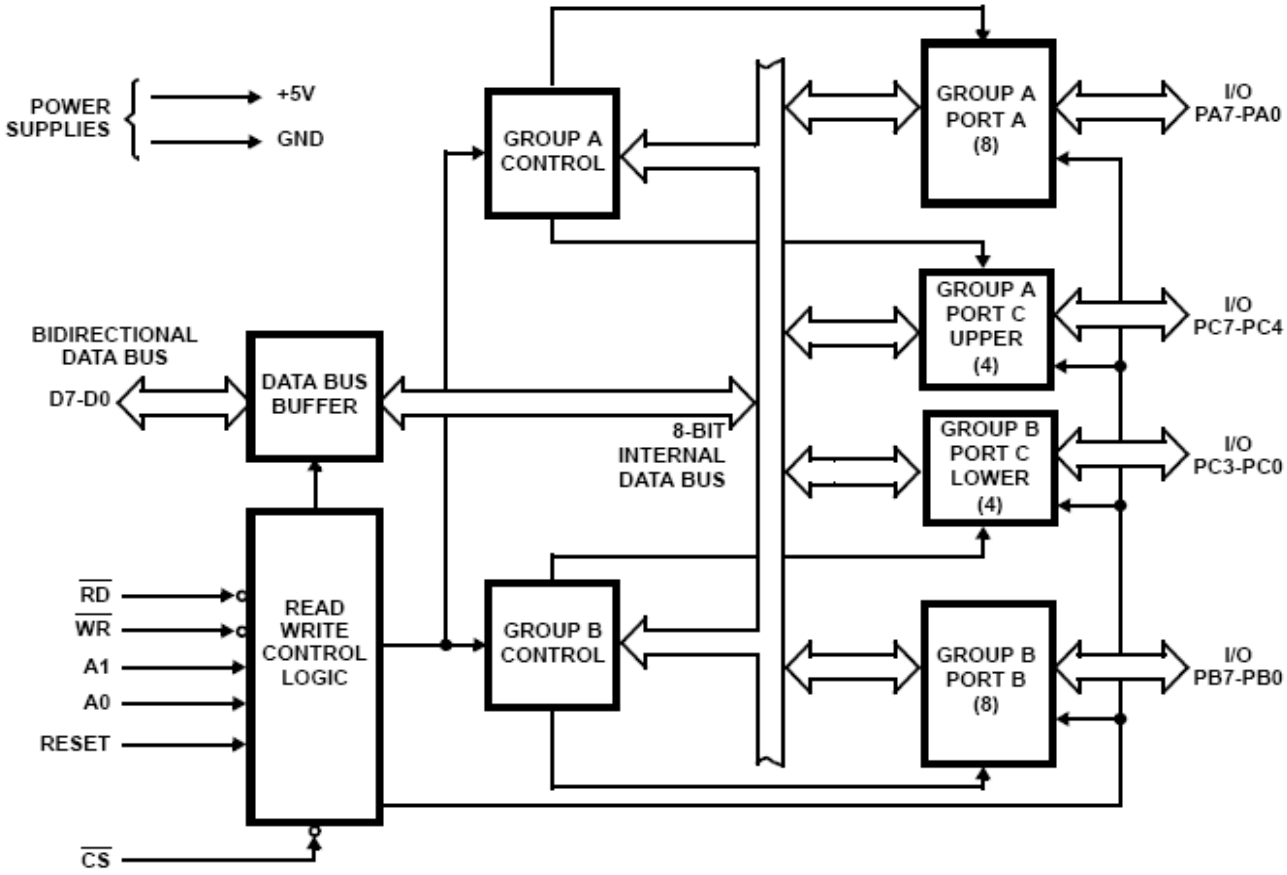


Figure. Block diagram of 8255 Programmable Peripheral interface.

DATA BUS BUFFER:

- This is a tri-state, bi-directional data bus used to interface the internal data bus of 8255A to the system data bus of 8085.
- Using IN or OUT instructions, CPU can read or write the data from/to the data bus buffer.
- It can also be used to transfer control words and status information between CPU and 8255A.
-

READ/WRITE CONTROL LOGIC:

- This block controls the chip detection (CS), read (RD) and write (WR) operations.
- It consists of A_0 and A_1 signals which are generally connected to the CPU address lines A_0 and A_1 respectively.
- When CS (Chip select) signal goes low, different values of A_0 and A_1 select one of I/O ports or control register.

Chip Select Lines $\overline{CS}$								Hex Address	Port
A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0		
1	0	0	0	0	0	0	0	80H	A
1	0	0	0	0	0	0	1	81H	B
1	0	0	0	0	0	1	0	82H	C
1	0	0	0	0	0	1	1	83H	Control Register

Group A and Group B control:

- Group A and B get the Control Signal from CPU and send the command to the individual control blocks.
- Group A send the control signal to port A and Port C (Upper) PC7-PC4.
- Group B send the control signal to port B and Port C (Lower) PC3-PC0.

PORT A:

- This is a 8-bit buffered I/O latch.
- It can be programmed by mode 0 , mode 1, mode 2 **.**

PORT B:

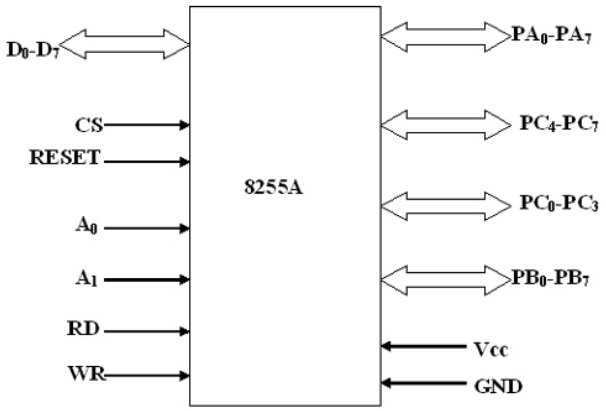
- This is a 8-bit buffer I/O latch.
- It can be programmed by mode 0 and mode 1.

PORT C:

- The eight bit ports of PORT C can be used as individual bits or be grouped into two 4 bit ports. C_{upper} (C_u) and C_{Lower} (C_L).The functions of these ports are defined by writing a control word in the control register.

Functions of Pin:

The signal description of 8255 is briefly presented as follows



Signals of 8255

- It has 24 I/O programmable pins which can be grouped into three 8 bit parallel ports of Port A, Port B and Port C.
- It is TTL compatible.

PA_7-PA_0: These are eight port A lines that acts as either latched output or buffered input lines depending upon the control word loaded into the control word register.

PC_7-PC_4 : Upper nibble of port C lines. They may act as either output latches or input buffers lines. This port also can be used for generation of handshake lines in mode 1 or mode 2.

PC_3-PC_0 : These are the lower port C lines, other details are the same as PC7-PC4 lines.

PB_0-PB_7 : These are the eight port B lines which are used as latched output lines or buffered input lines in the same way as port A.

A_1-A_0: These are the address input lines and are driven by the microprocessor. These address lines A_1-A_0 are used for addressing any one of the four registers, i.e. three ports and a control word register as given in table below.

$\overline{CS}$	A1	A0	Selects
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control register
1	x	x	8255 is not selected

$\overline{RD}$: This is the input line driven by the microprocessor and should be low to indicate read operation to 8255.

$\overline{WR}$: This is an input line driven by the microprocessor. A low on this line indicates write operation.

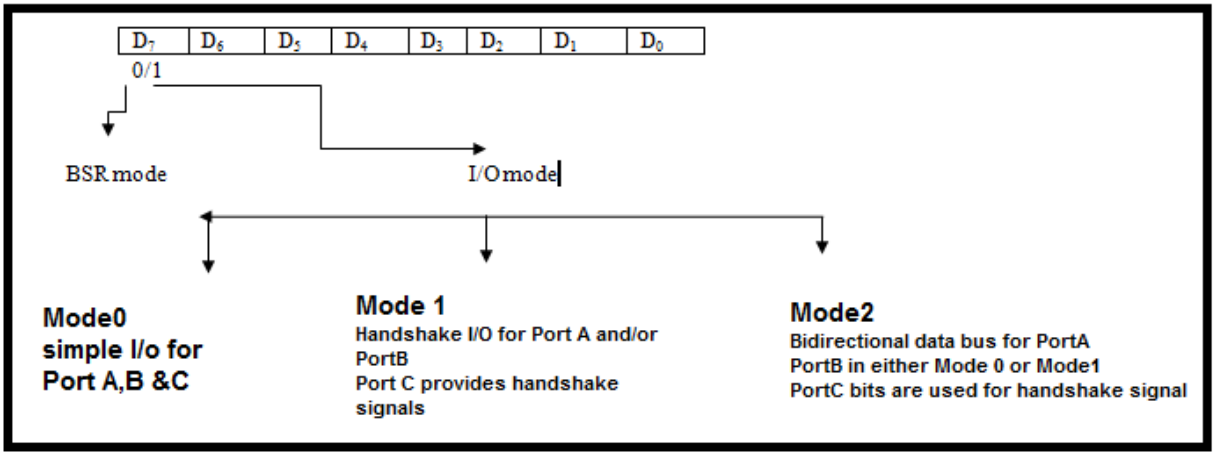
$\overline{CS}$: This is a chip select line. If this line goes low, it enables the 8255 to respond to RD and WR signals, otherwise RD and WR signal are neglected • In case of 8086 systems, if the 8255 is to be interfaced with lower order data bus,the A_0 and A_1 pins of 8255 are connected with A_1 and A_2 respectively.

D_0-D_7 : These are the data bus lines those carry data or control word to/from the microprocessor.

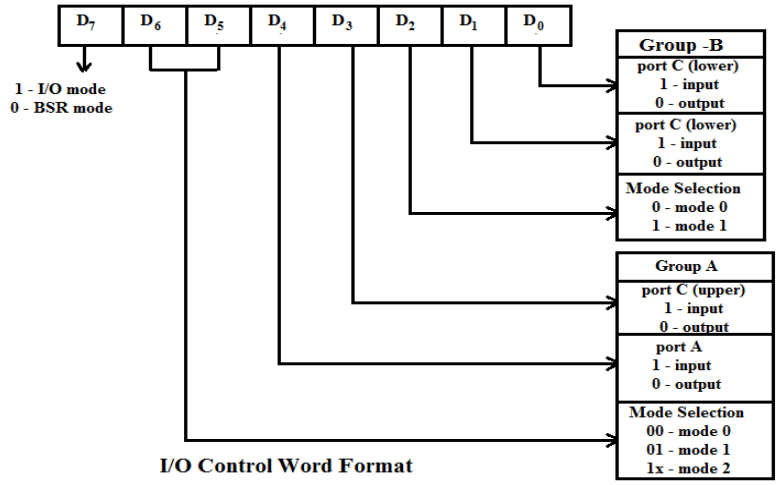
RESET: A logic high on this line clears the control word register of 8255. All ports are set as input ports by default after reset.

OPERATING MODES OF 8255

- BSR mode
- I/O mode



CONTROL WORD FORMATS:



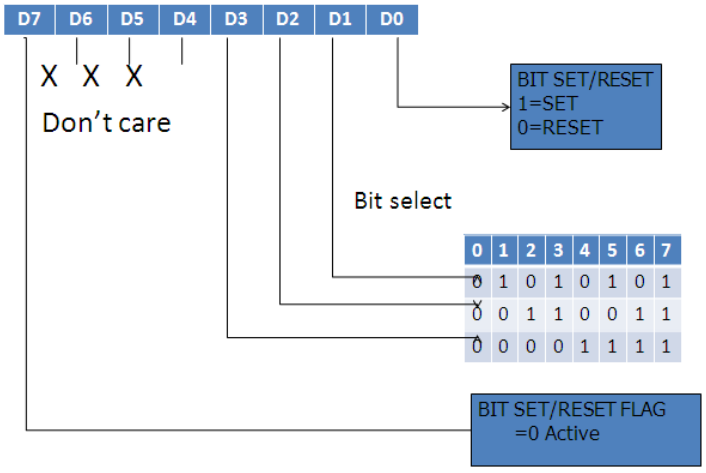
D7 bit of control word decides the type of mode.

- If it is '1'. I/O mode is selected.
- If it is '0'. BSR mode is selected

a) BSR (BIT SET/RESET) mode:

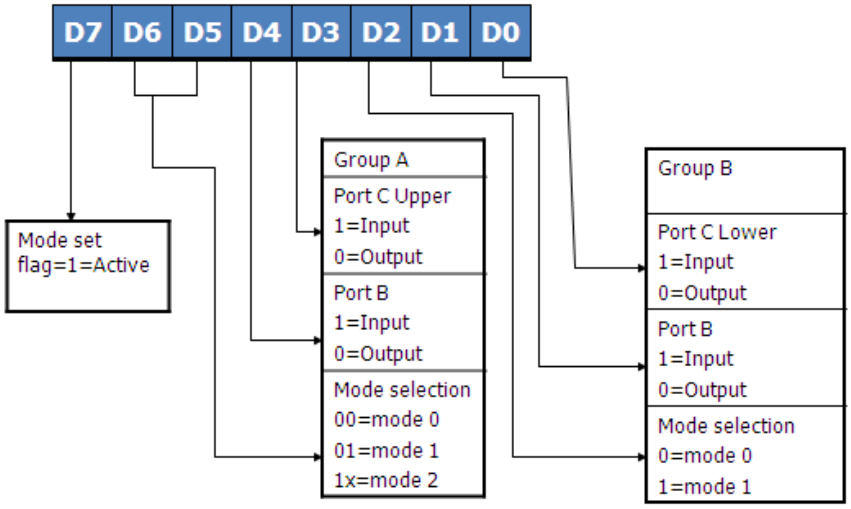
- The PORT C can be Set or Reset by sending OUT instruction to the CONTROL registers.
- In BSR mode individual bits of Port C can be used for applications such as on/off switch.
- The control word sets or reset one bit at a time.

- This is bit set/reset control word format.



b) FOR I/O MODE

The mode format for I/O as shown in figure



The I/O mode is divided into three modes mode 0, mode 1, and mode 2 given below

- **Mode 0** – Basic I/O mode
- **Mode 1** – strobbed I/O mode
- **Mode 2** – Bidirectional data transfer mode

Steps to communicate with peripherals through the 8255 .They are

1. Determine the addresses of Port A,B and C and of the control register according to the chip select logic and address line A0 and A1.

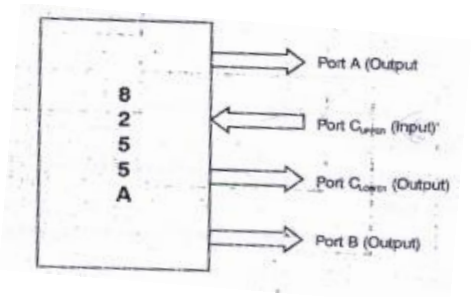
2. Write a control word in the control register

3. Write I/O instructions to communicate with peripherals through ports A, B and C.control word does not alter any previously transmitted control word with bit D_7=1.Thus the I/O operations of Port A and Port B are not affected by a BSR control word.

I/O MODES:

1) MODE 0 (Simple input / Output):

In this mode , port A, port B are used as two simple 8 bit I/O ports and port C as two 4 bit ports used as individually (Simply).



- **Features of mode 0 are:**
  - Any port can be input or output
  - Outputs are latched
  - Inputs are not latched

2) MODE 1: (Input/output with Hand shake)

In this mode, input or output is transferred by hand shaking Signals. The handshaking signals are exchanged between the microprocessor and peripheral prior to data transfer.

- **Features of mode 1**

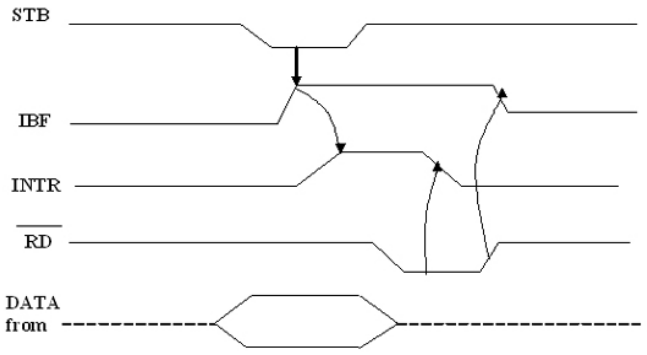
1. Two ports (A and B) function as 8 bit I/O ports .They can be configured either as input or output ports.
2. Each port uses 3 lines from port C as handshake signals. The remaining 2 lines of PORT C can be used for simple I/O operations.

2.1)Input control signal (Mode 1):

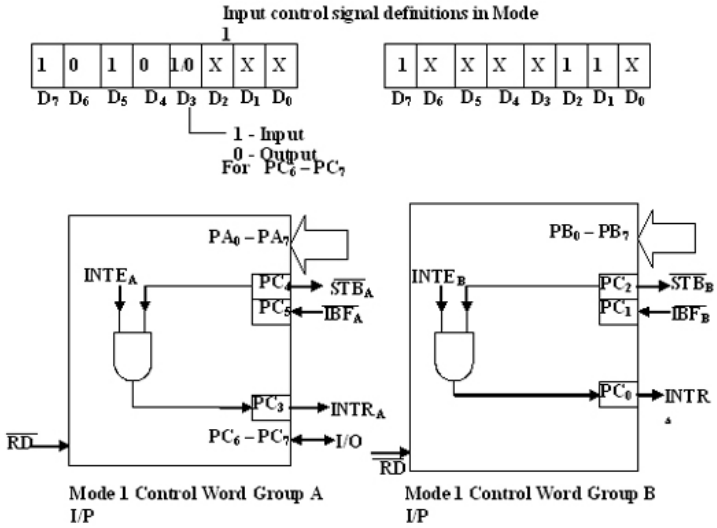
- **STB (Strobe input**) – If this lines falls to logic low level, the data available at 8-bit input port is loaded into input latches.

- **IBF (Input buffer full)** If this signal rises to logic 1, it indicates that data has been loaded into latches, i.e. it works as an acknowledgement. IBF is set by a low on STB and is reset by the rising edge of RD input.

- **INTR (Interrupt request)** This active high output signal can be used to interrupt the CPU. whenever an input device requests the service. INTR is set by a high STB pin and a high at IBF pin.
  - **INTE** is an internal flag that can be controlled by the bit set/reset mode of either PC_4(INTEA) or PC_2(INTEB) as shown in fig
  - **INTR** is reset by a falling edge of RD input. Thus an external input device can be request the service of the processor by putting the data on the bus and sending the strobe signal.

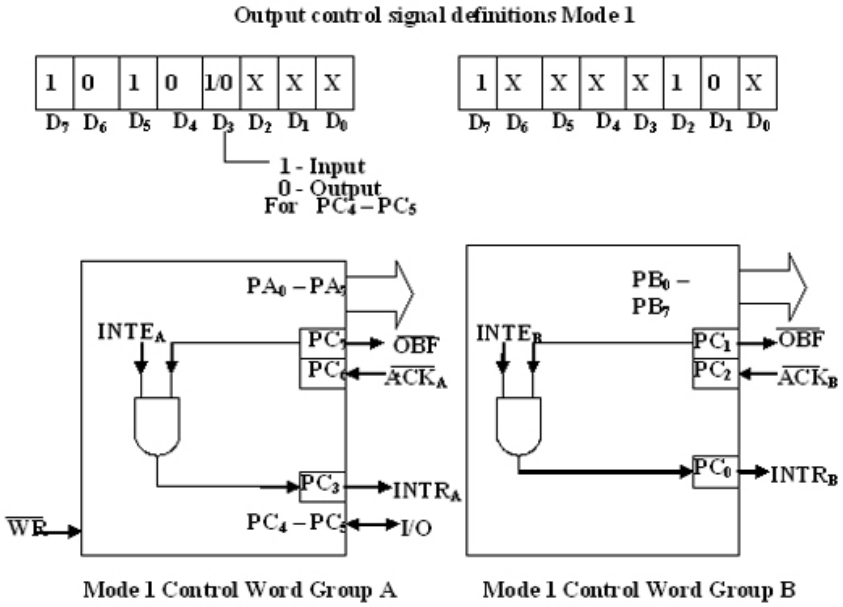


Mode 1 Strobed Input Data Transfer



2.2) Output control signal(Mode 1) :

- **OBF (Output buffer full)** – This status signal, whenever falls to low, indicates that CPU has written data to the specified output port. The OBF flip-flop will be set by a rising edge of WR signal and reset by a low going edge at the ACK input.
- **ACK (Acknowledge input**) – ACK signal acts as an acknowledgement to be given by an output device. ACK signal, whenever low, informs the CPU that the data transferred by the CPU to the output device through the port is received by the output device.
- **INTR (Interrupt request**) – Thus an output signal that can be used to interrupt the CPU when an output device acknowledges the data received from the CPU.INTR is set when ACK, OBF and INTE are 1. It is reset by a falling edge on WR input. The INTEA and INTEB flags are controlled by the bit set-reset mode of PC6 and PC2 respectively.

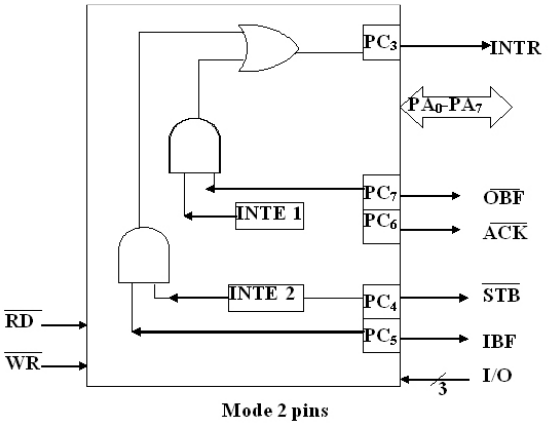


3) MODE 2 :bi-directional I/O data transfer:

- **Features of Mode2**

- In this mode, Port A can be configured as the bidirectional port and Port B is either in Mode 0 or Mode 1.
- Port A uses 5 signals from Port C as handshake signals for data transfer .The remaining 3 signals from Port C can be used either as simple I/O or as handshake for Port B.

.



Mode 2 pins

****

3. <u>Serial Communication Interface (8251)</u>

(Programmable Communication Interface)

Introduction:

The basic concepts concerning the serial I/O mode can be classified into the following categories

1. Interfacing Requirements
2. Alphanumeric codes
3. Transmission format
4. Error checks in data communication
5. Data communication over telephone lines

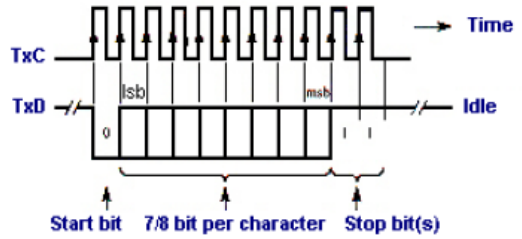
- **Interfacing Requirements**

<u>Serial I/O Interfacing</u>

- The MPU selects the peripheral through chip select and uses the control signals .Read to receive data and write to transmit data.
- To communicate with alphabetic letters and decimal numbers of the computer into binary, we use ASCII code of 7 bit 00H to 7FH is assigned to a letter, a decimal number, a symbol or a machine command.

[30 – 39H ⟶ 0 to 9]
[41H– 5AH ⟶ A to Z]
[21H – 2FH ⟶ for various symbols]
[00 - 1FH ⟶ for machine commands]

- **Transmission format**



  - In **synchronous format**, receiver and a transmitter are synchronized with the same clock and a block of character is transmitted along with the synchronization information. This format is generally used for high speed transmission (more than 20 Kbits/second)
  - The **asynchronous format** is character oriented. Each character carries the information of the start and stop bits. Transmission starts with one start bit(low) followed by a character , and one or two stop bits (high) .This is also known as **framing**. It is used in low speed transmission less than 20Kbits/second.

- **Communication Modes**

  According to the direction and simultaneity of data flow, it is classified as

  Simplex - Data are transmitted in only one direction.

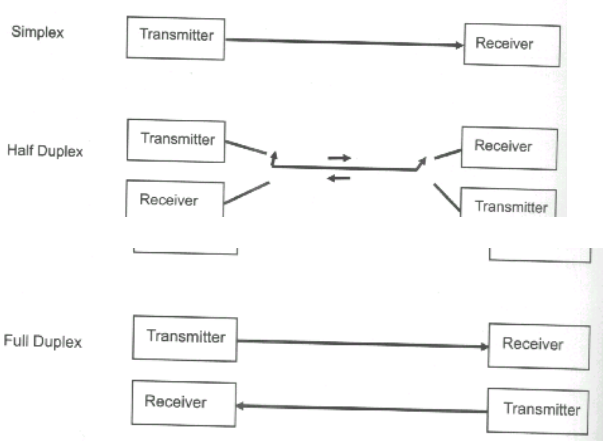
  Example: Transmission from a microcomputer to a printer.

  Duplex - Data flow in both direction

  - **Half Duplex** - If the transmission goes one way at a time it is called half duplex.
  - **Full Duplex** – If both transmitting and receiving signals goes simultaneously, it is called full duplex. Example: Transmission between computers.

- **Rate of transmission**

  The rate at which the bits are transmitted is called bits/second or Baud rate. For example 1200 baud = 1200 bits/second. It indicates1200 bits are transmitted in a second. For 1 bit it takes 1/1200 =0.83 ms.



****

Explain USART (8251) serial communication interface with its functional block diagram.(April 2018)(June 2016)

****

Programmable Communication Interface 8251 (USART)/ **(Programmable serial interface)**

Definition:

The 8251 is a programmable USART (Universal Synchronous Asynchronous Receiver Transmitter) is designed for Synchronous and Asynchronous serial communication; The 8251 receives parallel data from the CPU and transmits serial data after conversion. This device also receives serial data from the outside and transmits parallel data to the CPU after conversion.

The block diagram of 8251 includes five sections:

- Read /Write control logic
- Transmitter section
- Receiver Section
- Data bus buffer
- Modem control

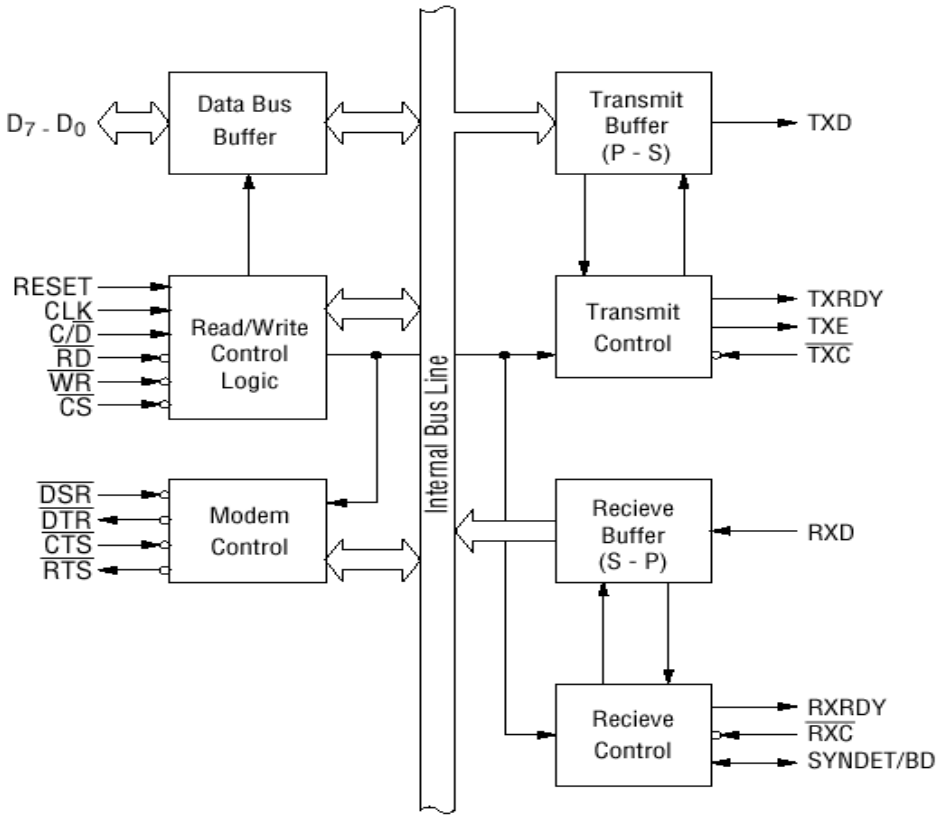


Figure:8251 USART Serial communication Interface

- The control logic interfaces the chip with the MPU determines the functions of the chip according to the control word in its register and monitors the data flow.

- The transmitter section converts a parallel word received from the MPU into a serial bits and

transmits them over TxD line to a peripheral.

- The receiver section receives serial bits from a peripheral converts them into parallel word and transfers the word to the MPU.

- The MODEM control is used to establish data communication through modems over telephone lines.

1. Transmitter section

- The transmitter accepts parallel data from the MPU and converts them into serial data.
- It has two registers, A buffer register to hold eight bits and an output register to convert eight bits into a stream of serial bits.
- The MPU writes a byte in the buffer register whenever the output register is empty, the contents of the buffer register are transferred to the output register.
- This section transmits data on the TxD pin with the appropriate framing bits(start & stop).3 output and 1 input signal are associated with transmitter section.

TXD (Transmit Data)

This is an output terminal for transmitting data from which serial-converted data is sent out.

TXRDY (Transmitter Ready)

This is an output terminal during high indicates that the 8251is ready to accept a transmitted data character. It can be used either to interrupt the MPU or to indicate the status.

TXEMPTY (Transmitter Empty)

This signal at logic 1 indicates that the 8251 has transmitted all the characters and the output register is empty. It is reset when a byte is transferred from the buffer to the output register.

TXC (Transmitter clock)

This is a clock input signal which determines the transfer speed of transmitted data. In "synchronous mode," the baud rate will be the same as the frequency of TXC. In "asynchronous mode", it is possible to select the baud rate factor by mode instruction. It can be 1,16 or 64 the baud.

2. Receiver Section

- The receiver accepts serial data on the RxD line from a peripheral and converts them into parallel data.
- The section has two registers, the receiver input register and the buffer register.
- When RxD line goes low, the control logic assumes it is a start bit, waits for half a bit time and samples the line again. If the line is still low, the i/p register accepts the following bits, forms a character and loads it into the buffer register subsequently. The parallel byte is transferred to the Microprocessor when requested.

RXD (Receive Data)

The bits are received serially on this line and converted into a parallel byte in the receiver input register.

RXC (Receiver clock)

This clock signal controls the rate at which bits are received by the USART .In asynchronous mode, the clock can be set to 1,16 and 64 times the baud.

RXRDY (Receiver Ready)

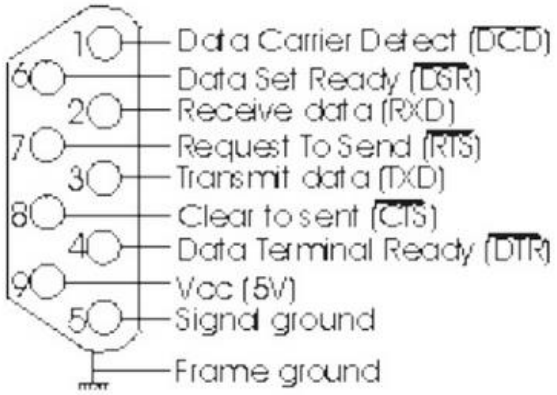
This output signal goes high when USART has a character in the buffer register and is ready to transfer it to the MPU .This line can be used either to indicate the status or to interrupt the MPU.

SYNDET/BD (Input or output terminal)

This is a terminal whose function changes according to mode. In "internal synchronous mode." this terminal is at high level, if sync characters are received and synchronized.

In "asynchronous mode," this is an output terminal which generates "high level" output upon the detection of a "break" character if receiver data contains a "low-level" space between the stop bits of two continuous characters.

3. MODEM Control



.

1. $\overline{DSR}$ **(Data set ready)**

   This is an input port for MODEM interface. This is normally used to check if the Data set is ready when communicating with a modem.

2. $\overline{DTR}$ **(Data terminal ready)**

   This is an output port for MODEM interface. It is used to indicate that the device is ready to accept data when the 8251 is communicating with a modem.

3. $\overline{CTS}$ **(Clear to send)**

   This is an input terminal for MODEM interface which is used for controlling a transmit circuit. The terminal controls data transmission if the device is set in "Tx Enable" status by a command. Data is transmittable if the terminal is at low level.

4. $\overline{RTS}$ **(Request to send data)**

   This is an output port for MODEM interface. It is used to indicate the MODEM that the receiver is ready to receive a data byte from the MODEM.

4.Read/Write control logic and Registers

This section includes R/W control logic,

- six input signals,
- control logic and
- 3 buffer registers: Data register, control register and status register.

Control Register

The 16 bit register for a control word consists of two independent bytes. The first byte is called the mode instruction and the second byte is called the command instruction. This register can be accessed as an output port when the C/D pin is high**.**

Status Register

This input register checks the ready status of a peripheral. This register is addressed as an input port when the C /D is high .It has the same port address as the control register.

Data Buffer

This bidirectional register can be addressed as an input port and an output port when C/D pin is low.

The input signals to the control logic are as follows.

There are two types of control word.

1. Mode instruction (setting of function)
2. Command (setting of operation)

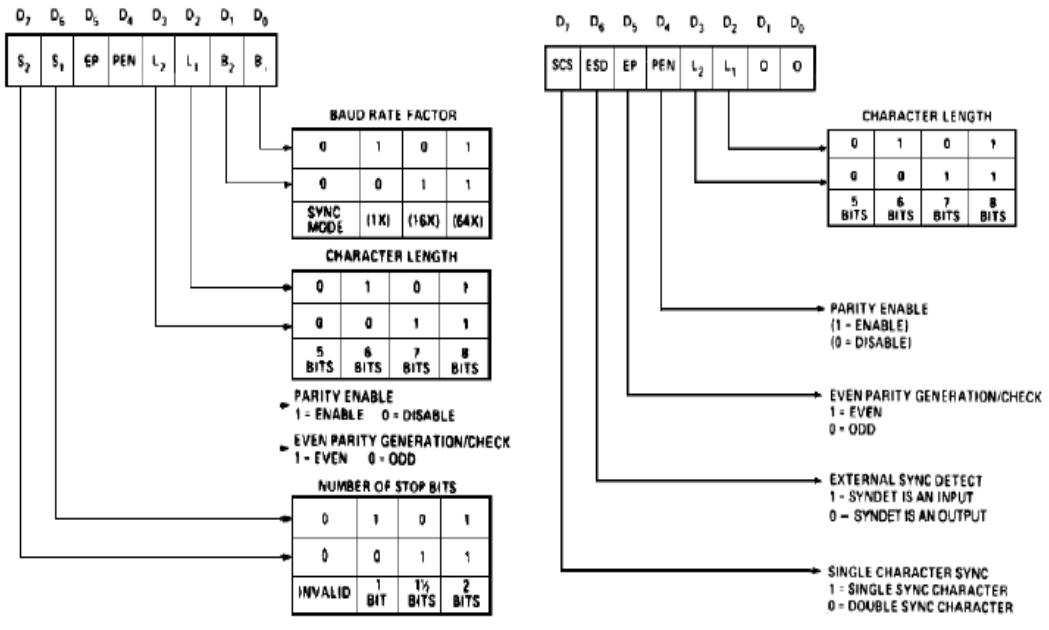
1) Mode Instruction

Mode instruction is used for setting the function of the 8251. The writing of a control word after resetting will be recognized as a "mode instruction."

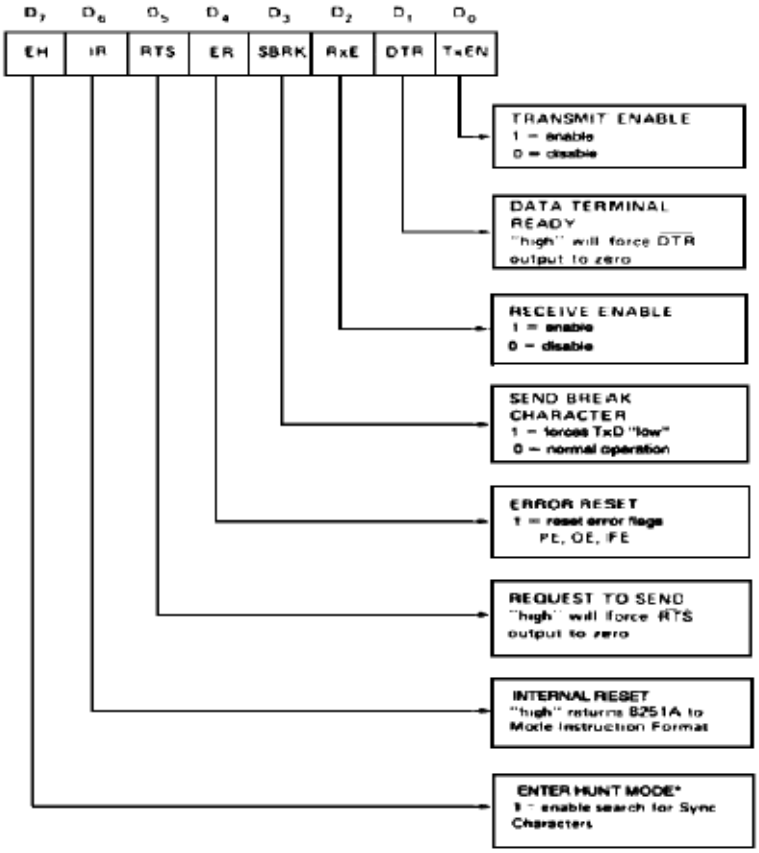
8251 Mode word

Items set by mode instruction are as follows:

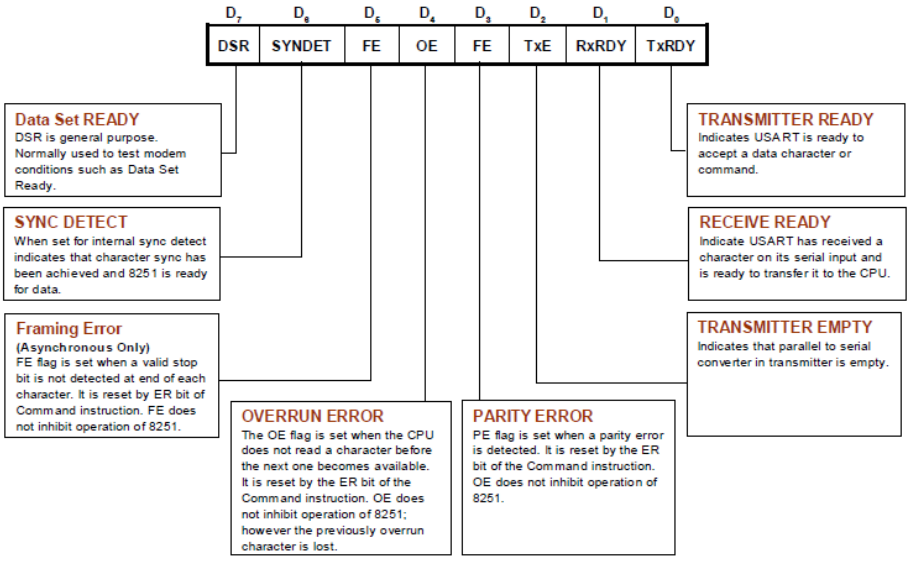
- Synchronous/asynchronous mode
- Stop bit length (asynchronous mode)
- Character length
- Parity bit
- Baud rate factor (asynchronous mode)
- Internal/external synchronization (synchronous mode)
- Number of synchronous characters (Synchronous mode)



8251 command word



8251 status word

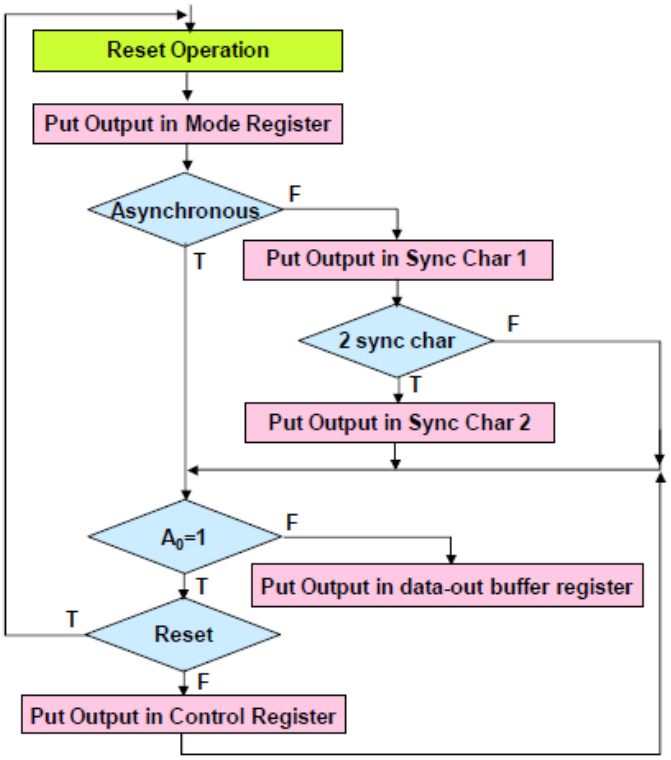


Pin Description

❖ **Initializing the 8251**

- To implement serial communication, the MPU must inform 8251 of all details such as mode, baud , stop bits, parity etc.,
- Therefore prior to data transfer ,a set of control words must be loaded into the 16 bit control register of the 8251.The MP must check the readiness of a peripheral by reading the status register.

- The control words are divided into two formats: **Mode word** and **command word.**
- The mode word specifies the general characteristics of operation (such as baud, parity, number of stop bits) .The command word enables data transmission and/or reception and the status word provide the information concerning register status and transmission errors.



****

Draw the block diagram of 8279 Keyboard/Display controller and explain how to interface the Hex Key Pad and 7-segment LEDs using 8279. (December 2017)

****

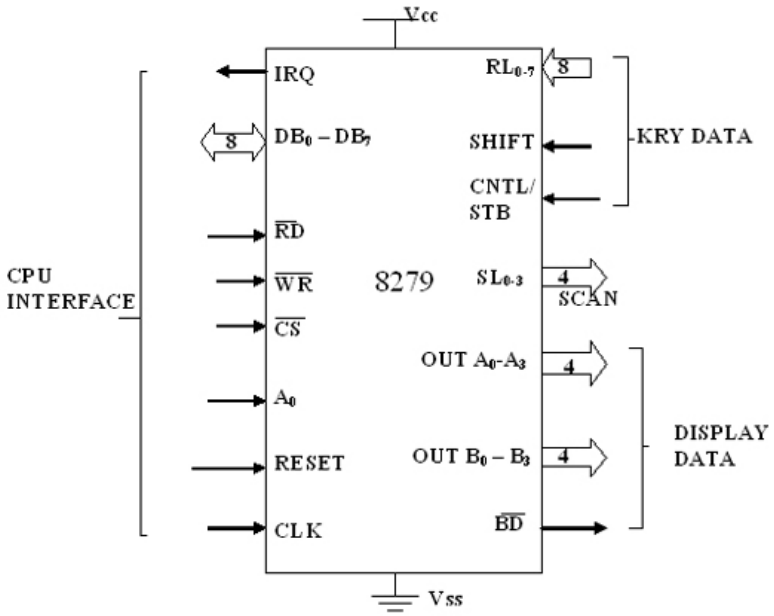
4.Programmable Keyboard/Display controller – 8279

Definition:

A 8279 is a general purpose keyboard display controller that simultaneously drives the display of a system and interferes a keyboard with the CPU, leaving it free for routine task.

- ✓ The keyboard is interfaced either in interrupt mode or polled mode.
- ✓ In the Interrupt mode, the processor is requested service only if any key is pressed, otherwise the CPU can proceed with its main task.
- ✓ In the Polled mode, the CPU periodically reads an internal flag of 8279 to check for a key pressed.

Basic Description of the 8279



DATA BUS (D7-D0)

All data and commands between the microprocessor and 8279 are transmitted on these lines.

$\overline{RD}$ **(read):**

Microprocessor reads the data/ status from 8279.

$\overline{WR}$ **(write):**

Microprocessor writes the data to 8279

A0:

A high signal on this line indicates that the word is a command or status. A low signal indicates the data.

RESET:

High signal in this pin resets the 8279. After being reset, the 8279 is placed in the following modes

- 16 x 8 – bit character display – left entry
- Two key lock out

$\overline{CS}$ **(Chip Select):**

A low signal on this input pin enables the communication between 8279 and the microprocessor.

IRQ (Interrupt Request):

- The interrupt line goes low with each FIFO/sensor RAM reads and returns high if there still information in the RAM

SL_0-SL_3:

- The scan lines which are used to scan the key switch or sensor matrix and the displays digits. These lines can be either encoded (1 of 16) or decoded (1 of 4)

RL_0-RL_7:

- Input return lines which are connected to the scan lines through the keys or sensor switches. They have active internal pull-ups to keep them high serve as an 8- bit input in the strobed input mode.

SHIFT:

- It has an active internal pull-up to keep it high until a switch closure pulls it low.

CNTL/STB:

- For keyboard mode, this line is used as a control input and stored like status on a key closure.
- The line is also the strobed line to enter the data into the FIFO in the strobed input.

OUT A_0 – OUT A_3, OUT B_0 – OUT B_3:

- These two ports are the outputs for the 16x4 display refresh registers. These two ports may also be considered as one 8 – bit port.
- The two 4 – bit ports may be blanked independently.

BD:

This output is used to blank the display digit switching or by a display banking command.

****

Explain the working Principle of 8279 Keyboard/Display Controller.(April 2010)(June 2016) (December 2016)(May 2015)(December 2015)

****

Keyboard/Display Controller (8279)

A 8279 is a general purpose keyboard display controller that simultaneously drives the display of a system and interferes a keyboard with the CPU, leaving it free for routine task.

Functional block diagram of 8279

It consists of four sections

- Keyboard section
- Scan section
- Display section
- CPU Interface section

CPU INTERFACE SECTION:

- This section has bi-directional data buffer (DB0 –DB7), I/O control lines (RD, WR, CS, A0) and Interrupt Request lines (IRQ).
- The A_0 signal determines whether transmit/receive control word or data is used.
- An active high in line IRQ is generated to interrupt the microprocessor whenever the data is available.

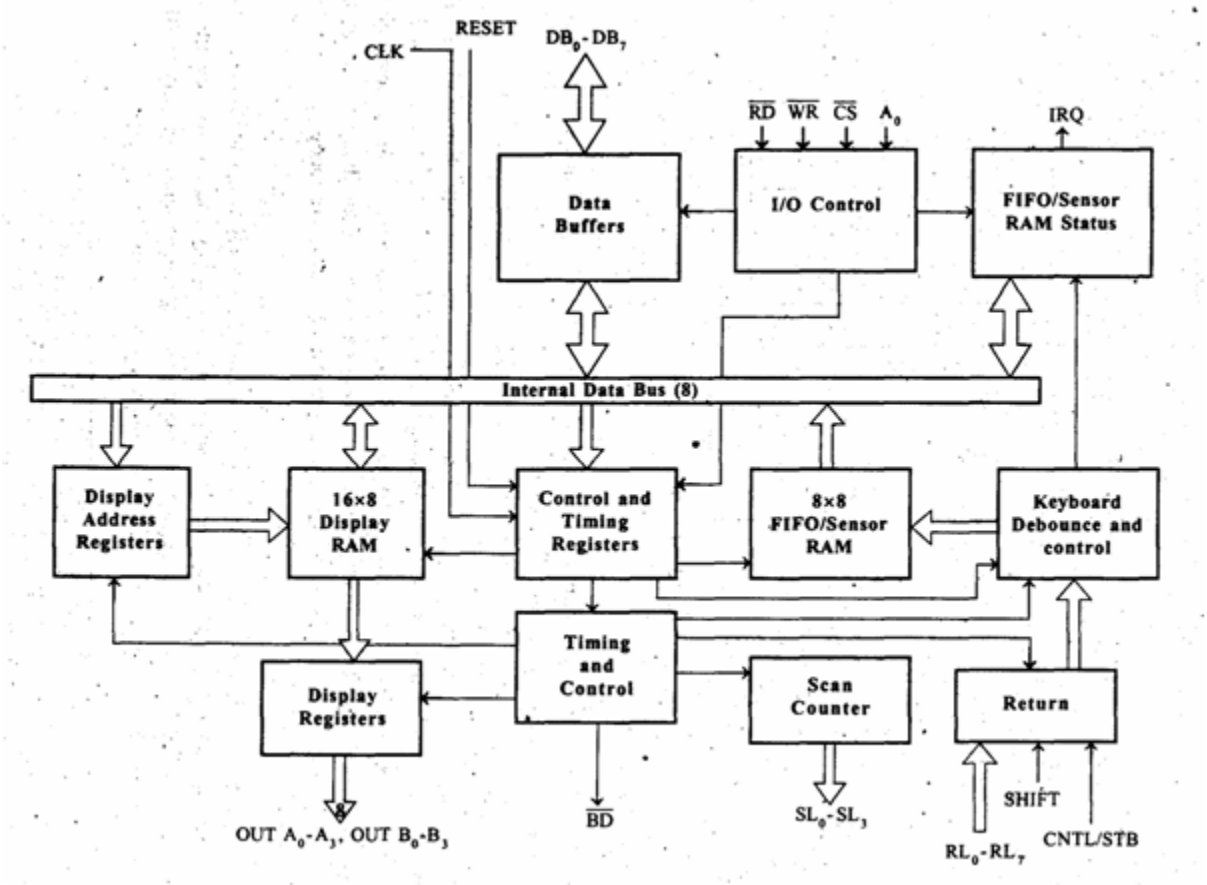


Figure:8279 keyboard /Display Interface

- It consists of
  a) A set of four scan lines and eight return lines for interfacing keyboards
  b) A set of eight output lines for interfacing display.

A0	RD	WR	Operation
0	0	0	MPU writes the data is 8279
0	0	1	MPU reads the data from 8279
1	1	0	MPU writes control word to 8279
1	0	1	MPU read status word from 8279

KEYBOARD SECTION:

- This section has keyboard debounce and control, 8X8 FIFO/sensor RAM, 8 return lines (RL0 – RL7) and CNTL/STB and shift lines.
- In the keyboard debounce and control unit, keys are automatically debounced and the keyboard can be operated in two modes.
  - **Two keys lock out**
  - **N – key roll over**
- The 8X8 FIFO/sensor RAM consists of 8 registers that are used to store eight keyboard entries.

- The return lines (RL0-RL7) are connected to eight columns of keyboard.
- The status of shift and CNTL/STB lines are stored along with the key closure.

SCAN SECTION:

- This section has scan counter and four scan lines (SL0 – SL3).
- These lines are decoded by 4 to 16 decoder to generate 16 scan lines.
- Generally SL0 – SL3 are connected with the rows of a matrix keyboard.

DISPLAY SECTION:

- This section has two groups of outputs lines A0 – A3 and B0 – B3. These lines are used to send data to display drivers.
- BD line is used blank the display. It also has 16X8 displays RAM.
- The display address register holds the address of the word currently being written or read by the CPU to or from the display RAM.
- The contents of the registers are automatically updated by 8279 to accept the next data entry by CPU.

Modes of operations of 8279

1. **Input (Keyboard) modes**
2. **Output (Display) modes**

1. Keyboard modes

❖ **Scanned keyboard mode with N key rollover**

In this mode, each key depression is treated independently. When a key is pressed, the debounce circuit waits for 2 keyboards scans and then checks whether the key is still depressed. If it is still depressed, the code is entered in FIFO RAM

❖ **Scanned keyboard mode with 2 key lock out.**

It prevents 2 keys from being recognized if pressed simultaneously. If two keys are pressed within a debounce cycle (simultaneously),no key is recognized till one of them remains closed, and the other is released. The last key that remains depressed is considered as single valid key depression.

2. Display modes

❖ **Left entry mode**

The data is entered from the left side of the display unit.

❖ **Right entry mode**

The first entry to be displayed is entered on the rightmost display.

Programming the Keyboard Interface :

- Before any keystroke is detected, the 8279 must be programmed
- The first 3 bits of the number sent to the control port (11H) select one of the 8 different control words.

Command Words of 8279

Keyboard Display mode set

The format of the command word is to select different modes of operation of 8279

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
0	0	D	D	D	K	K	K	1

D_7	D_6	D_5	*Function*	*Purpose*
0	0	0	Mode set	Selects the number of display positions, type of key scan...
0	0	1	Clock	Programs internal clk, sets scan and debounce times.
0	1	0	Read FIFO	Selects type of FIFO read and address of the read.
0	1	1	Read Display	Selects type of display read and address of the read.
1	0	0	Write Display	Selects type of write and the address of the write.
1	0	1	Display write inhibit	Allows half-bytes to be blanked.
1	1	0	Clear	Clears the display or FIFO
1	1	1	End interrupt	Clears the IRQ signal to the microprocessor.

D	D	Display modes
0	0	Eight 8-bit character Left entry
0	1	Sixteen 8-bit character left entry
1	0	Eight 8-bit character Right entry
1	1	Sixteen 8-bit character Right entry

K	K	K	Keyboard modes
0	0	0	Encoded Scan, 2 key lockout (Default after reset)
0	0	1	Decoded Scan, 2 key lockout
0	1	0	Encoded Scan, N- key Roll over
0	1	1	Decoded Scan, N- key Roll over
1	0	0	Encode Scan, N- key Roll over
1	0	1	Decoded Scan, N- key Roll over
1	1	0	Strobed Input Encoded Scan
1	1	1	Strobed Input Decoded Scan

Control Word Description

a)000DDMMM

***Mode set*: Opcode 000.**

DD sets displays mode.

MMM sets keyboard mode

DD field selects either**:** 8- or 16-digit display Whether new data are entered to the rightmost or leftmost display position.

b) Programmable clock (<u>001PPPPP)</u>

The clock for operation of 8279 is obtained by dividing the external clock input signal by a programmable constant called prescaler**.**The clock command word programs the internal clock driver**.**

The code PPPPP, is a prescalar that divides the clock input pin (CLK) to achieve the desired operating frequency, e.g. 100 KHz requires 01010_2

(c)Read FIFO/Sensor RAM(<u>010 AI X AAA)</u>

The read FIFO control word selects the address (AAA) of a keystroke from the FIFO buffer (000 to 111).X - don't care and AI selects auto-increm(ent for the address

d) Read Display RAM(<u>011 AI AAAA)</u>

This command enables a programmer to read the display RAM data.

The display read control word selects the 4 bit address AAAA points to the 16 byte display RAM positions that is to be read.

e) Write Display RAM(<u>100 AI AAAA)</u>

The display write control word selects the 4 bit address AAAA points to the 16 byte display RAM positions that is to be written.. Display. Z selects auto-increment so subsequent writes go to subsequent display positions.

f) Display with inhibit blanking (<u>1010WWBB)</u>

The display write inhibit control word inhibits writing to either the leftmost 4 bits of the display (left W) or rightmost 4 bits (right W).BB works similarly except that they blank (turn off) half of the output pins.

g) Clear Display RAM (<u>1100CCFA)</u>

The clear control word clears the display, FIFO or both Bit F clears FIFO and the display RAM status, and sets address pointer to 000.

If CC are 00 or 01, all display RAM locations become 00000000.

If CC is 10, --> 00100000, if CC is 11, --> 11111111.

h) End Interrupt/Error mode set(<u>1110E000)</u>

End of Interrupt control word is issued to clear IRQ pin to zero in sensor matrix mode

- Clock must be programmed first. If 3.0 MHz drives CLK input, PPPPP is programmed to 30 or 11110_2.
- Keyboard type is programmed next. The previous example illustrates an encoded keyboard, external decoder used to drive matrix.
- Program the operation of the FIFO.Once programmed never reprogrammed done, until a procedure is needed to read prior keyboard codes .

****

Draw and explain the functional block diagram of 8254 Timer and its command word format.[May/June 2016, Nov/Dec 2016, May/June 2013, May/June 2009][Dec 2017]

Explain the blocks diagram and modes of the 8254 timer. {Nov/Dec 2015 .Dec 2012,June 2014]

****

5. Programmable Interval Timer (8254/8253)

Definition:

The 8254 is a programmable interval timer/counter is used for the generation of accurate time delays , controlling real-time events such as real-time clock, events counter, and motor speed and direction control under software control.

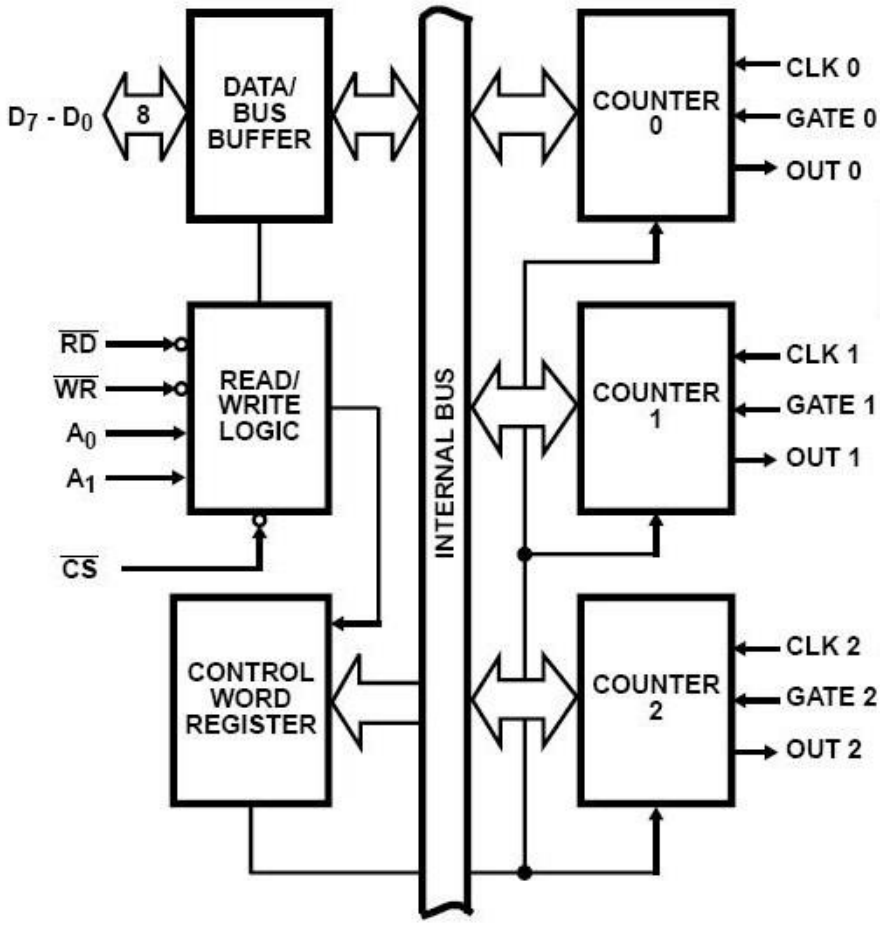
After the desired delay, the 8254 will interrupt the CPU. This makes microprocessor to be free the tasks related to the counting process and can execute the programs in memory, while the timer device may perform the counting tasks. This minimizes the Software overhead on the microprocessor.

Application of 8254:

- Real time clock
- Event-counter
- Digital one-shot
- Programmable rate generator
- Square wave generator
- Binary rate multiplier
- Complex waveform generator
- Complex motor controller

It consists of

- Three independent 16-bit programmable counters (timers)
- a data bus buffer
- Read/Write control Logic
- Control register



8254 Functional Block diagram of 8253

DATA BUS BUFFER:

This 3- state, bi-directional, 8-bit buffer is used to interface the 8254 to the system bus.

READ/WRITE LOGIC:

- The Read/Write logic accepts inputs from the system bus and generates control signals for the other functional blocks of the 8254.
- A1 and A0 select one of the three contents counters or the control word register to be read from/written into.
- A "low" on the RD input tells the 8254 that the CPU is reading one of the counters.
- A "low" on the WR input tells the 8254 that the CPU is writing either a control word or an initial count.
- Both RD and WR are qualified by CS; RD and WR are ignored unless than 8254 has been selected by holding CS low.

CONTROL WORD REGISTER:

- The control word register is selected by the read/write logic when A1, A0=11.
- If the CPU then does a write operation to the 8254, the data is stored in the control word register and is interpreted as a control word used to define the operation of the counters.

- The control word register can only be written to; status information is available with the Read-Back command.

A_1	A_0	Function
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Control Word

COUNTER 0, COUNTER 1, COUNTER 2:

- Each is a 16 bit down counter
- The counters are fully independent. Each counter may operate in a different mode.
- Each counter has a separate clock input, count enable (gate) input lines and output lines.
- The control word register is not a part of the counter itself, but its contents determine how the counter operates.

8254 Pin Description

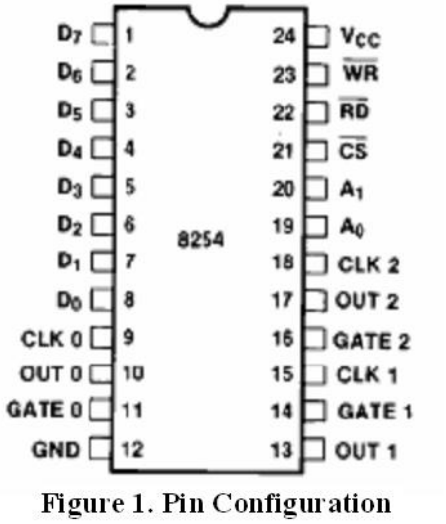
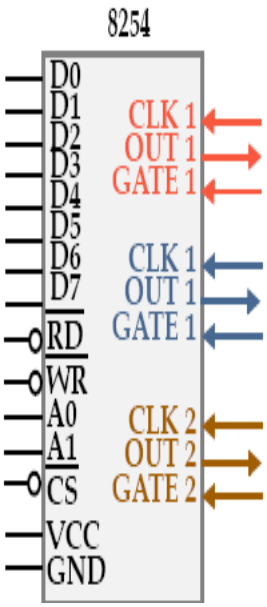


Figure 1. Pin Configuration



D_0 to D_7 : read, write, Chip select & Address pins A1 and A0 are connected to Microprocessor

A_1, A_0 **The *address inputs* select one of the four internal registers** for programming, reading, or writing to a counter.

CLK: The *clock* input is the timing source for each of the internal counters. It is often connected to the PCLK signal from the bus controller

CS: Chip Select enables the 8254 for programming, and reading and writing

Gate: The gate input controls the operation of the counter in some modes

OUT: A counter output is where the wave-form generated by the timer is available

$\overline{RD}/\overline{WR}$ *Read/Write* causes data to be read/written from the 8254 and often connects to the $\overline{IORC}/\overline{IOWC}$. Each counter is individually programmed by writing a control word, followed by the initial count.The control word allows the programmer to select the counter, model of operation, binary or BCD count and type of operation (read/write).

Command word of 8254

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC—Select Counter

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Read-Back Command (see Read Operations)

M—Mode

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

RW—Read/Write

RW1	RW0	
0	0	Counter Latch Command (see Read Operations)
0	1	Read/Write least significant byte only
1	0	Read/Write most significant byte only
1	1	Read/Write least significant byte first, then most significant byte

BCD

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

NOTE: Don't care bits (X) should be 0 to insure compatibility with future Intel products.

Figure 7. Control Word Format

Each counter may be programmed with a count of 1 to FFFFH. Minimum count is 1 all modes except 2 and 3 with minimum count of 2.

Each counter has a program control word used to select the way the counter operates.

If two bytes are programmed, then the first byte (LSB) stops the count, and the second byte (MSB) starts the counter with the new count.

****

Explain the various modes of operation of timer interface 8253/8254.[Dec 2013,Dec 2015,Dec 2016]

****

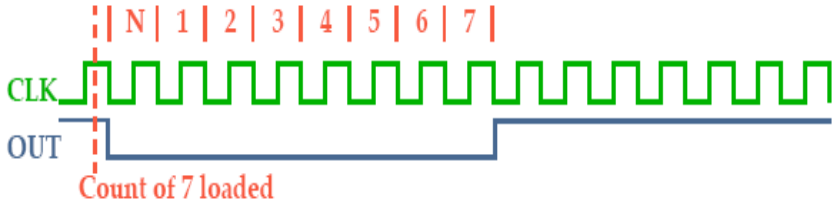
There are 6 modes of operation for each counter

1. MODE 0: INTERRUPT ON TERMINAL COUNT :
2. MODE 1: Programmable One-Shot:
3. MODE 2: RATE GENERATOR:
4. MODE 3: SQUARE WAVE GENERATOR
5. MODE 4: SOFTWARE TRIGGERED STROBE :
6. MODE 5: HARDWARE TRIGGERED STROBE (RETRIGGERABLE):

Modes of operation

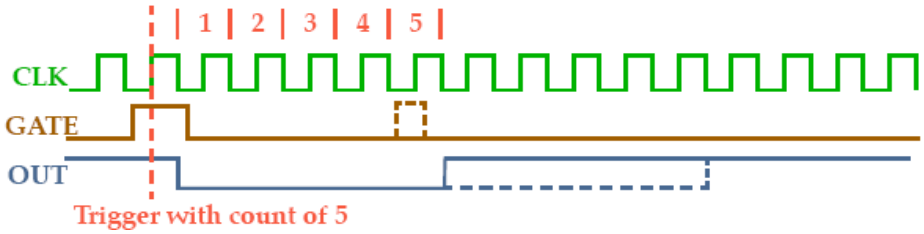
MODE 0: **INTERRUPT ON TERMINAL COUNT:**

- Mode 0 is typically used for event counting. After the Control Word is written, OUT is initially low, and will remain low until the Counter reaches zero.
- OUT then goes high and remains high until a new count or a new Mode 0 Control Word is written into the Counter.
- GATE = 1 enables counting; GATE = 0 disables counting. GATE has no effect on OUT.
- The output becomes a logic 0 when the control word is written and remains there until N plus the number of programmed counts.



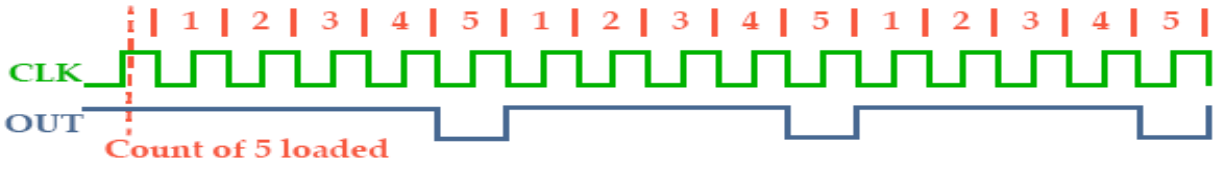
MODE 1: **PROGRAMMABLE ONE-SHOT**:

- OUT will be initially high.
- OUT will go low on the CLK pulse following a trigger to begin the one-shot pulse, and will remain low until the Counter reaches zero.
- OUT will then go high and remain high until the CLK pulse after the next trigger.
- The Gate input triggers the counter to output a 0 pulse for count clocks. Counter reloaded if Gate is pulsed again.



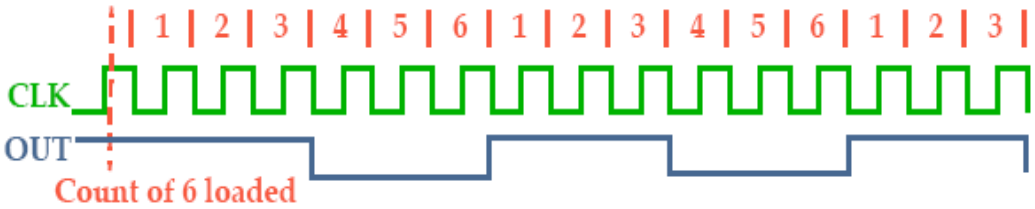
MODE 2: **RATE GENERATOR**:

- This Mode functions like a divide-by-N counter. It is typically used to generate a Real Time Clock interrupt.
- OUT will initially be high. When the initial count has decremented to 1, OUT goes low for one CLK pulse. OUT then goes high again, the Counter reloads the initial count and the process is repeated.
- Mode 2 is periodic, the same sequence is repeated indefinitely. For an initial count of N, the sequence repeats every N CLK cycles.
- GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low during an output pulse, OUT is set high immediately.
- Counter generates a series of pulses 1 clock pulse wide. The separation between pulses is determined by the count. The cycle is repeated until reprogrammed or G pin set to 0.



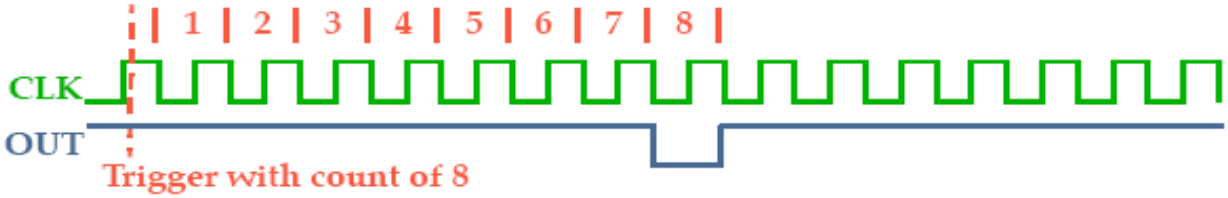
MODE 3: **SQUARE WAVE GENERATOR**

- Mode 3 is typically used for Baud rate generation. Mode 3 is similar to Mode 2 except for the duty cycle of OUT. OUT will initially be high.
- When half the initial count has expired, OUT goes low for the remainder of the count. Mode 3 is periodic; the sequence above is repeated indefinitely.
- An initial count of N results in a square wave with a period of N CLK cycles.GATE = 1 enables counting; GATE = 0 disables counting. If GATE goes low.while OUT is low, OUT is set high immediately; no CLK pulse is required.
- **Even counts**: OUT is initially high. The initial count is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses.
- When the count expires OUT changes value and the Counter is reloaded with the initial count. The above process is repeated indefinitely.so for **odd counts**, OUT will be high for (N + 1)/2 counts and low for (N - 1)/2 counts



MODE 4: SOFTWARE TRIGGERED STROBE :

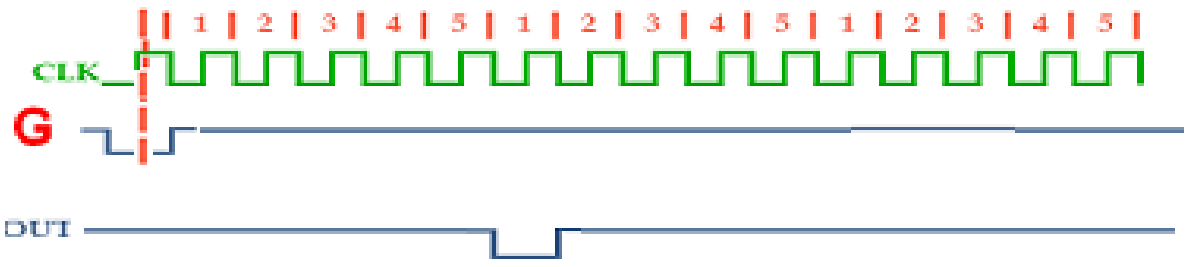
• OUT will be initially high. When the initial count expires, OUT will go low for one CLK pulse and then go high again. The counting sequence is ``triggered'' by writing the initial count. **(G must be 1).**



MODE 5: HARDWARE TRIGGERED STROBE (RETRIGGERABLE):

OUT will initially be high. Counting is triggered by a rising edge of GATE. When the initial count has expired, OUT will go low for one CLK pulse and then go high again. **G controls similar to Mode 1.**

Trigger with count of 5



****

Explain the working principle of 8257 DMA controller interface. (June 2016)

What is DMA? Explain the DMA based data transfer sing DMA controller (April 2015)

****

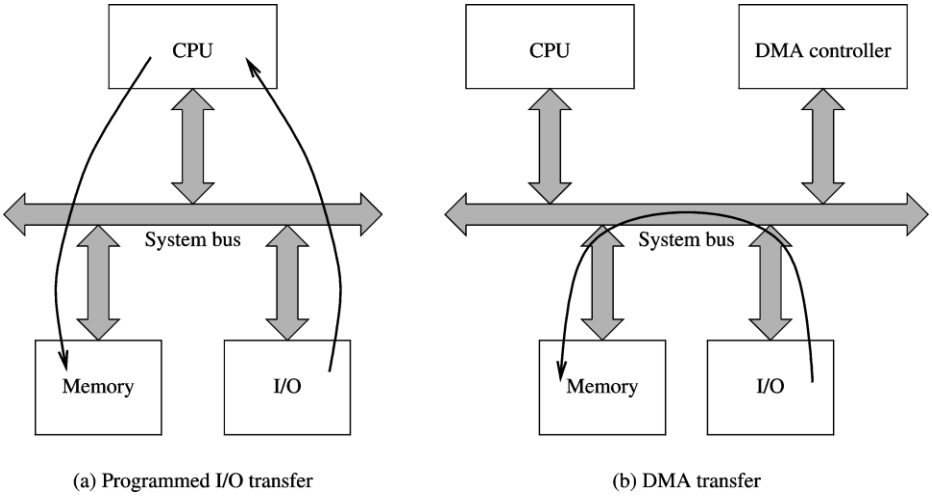
6. Direct Memory Access

Direct memory access (**DMA**) or DMA mode of data transfer is the fastest amongst all the modes of data transfer. In this mode, the device may transfer data directly to/from memory without any interference from the CPU.

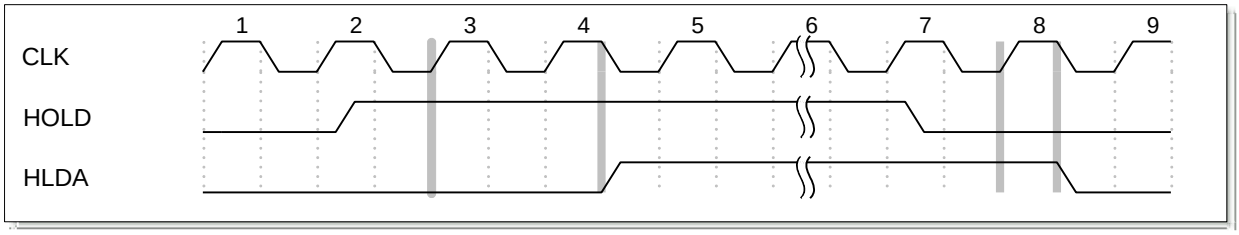
DMA Controller

The **DMA controller (8257)** allows certain hardware subsystems to read/write data to/from memory without microprocessor intervention, allowing the processor to do other work.

.



(a) Programmed I/O transfer (b) DMA transfer



It is used in disk controllers, video/sound cards etc, or between memory locations. Typically, the CPU initiates DMA transfer, does other operations while the transfer is in progress, and receives an interrupt from the DMA controller once the operation is complete.

It contains of five main blocks.

1. Data bus buffer
2. Read/Control logic
3. Control logic block
4. Priority resolver
5. DMA channels.

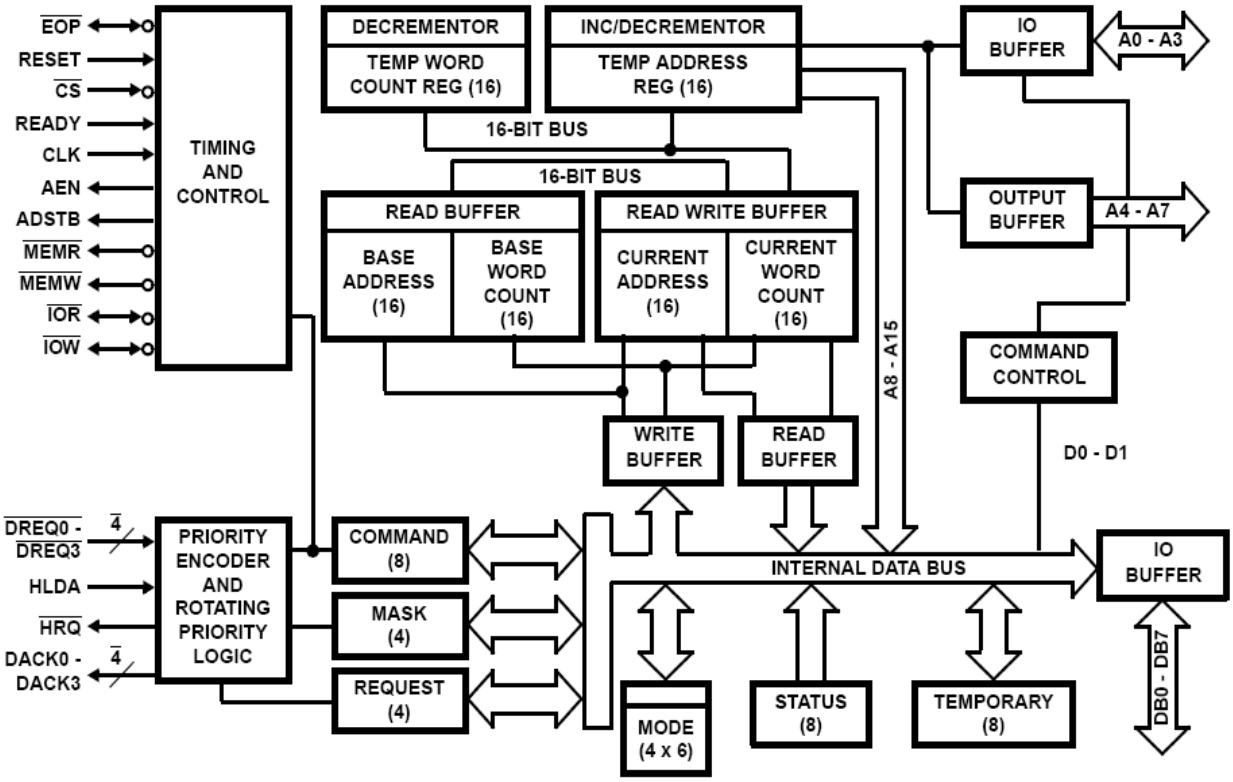
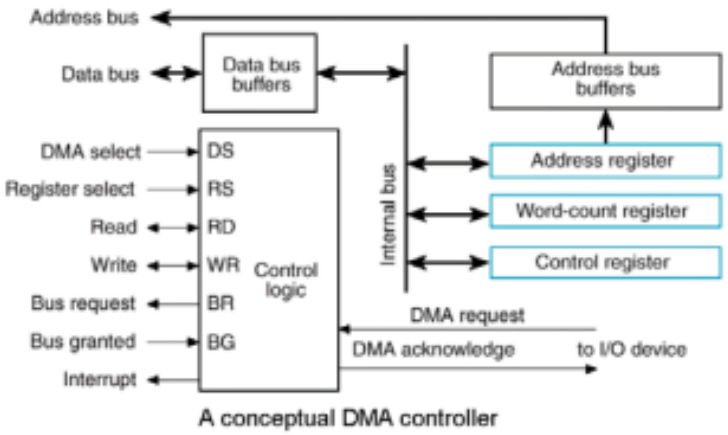


Figure:8237 DMA controller



A conceptual DMA controller

- **Programming the 8237**

There are 4 steps required to program the address and count registers first:

1. Clear the F/L flip-flop with a clear F/L command
2. Disable the channel
3. Program the LSB and then MSB of the address
4. Program the LSB and then MSB of the count

Additional programming is required to select the mode of operation before the channel is enabled and started.

Internal registers

- **The current address register (CAR)** is used to hold the 16-bit memory address used for the DMA transfer.
- The **current word count register (CWCR)** programs a channel for the number of bytes (up to 64K) transferred during a DMA action.
- The **base address (BA)** and **base word count (BWC) registers** are used when auto-initialization
- is selected for a channel. In this mode, their contents will be reloaded to the CAR and CWCR after the DMA action is completed.
- Each channel has its own CAR, CWCR, BA and BWC.
- The **command register (CR)** programs the operation of the 8237 DMA controller
- The **mode register (MR)** programs the mode of operation for a channel.
- The **request register (RR)** is used to request a DMA transfer via software, which is very useful in memory-to-memory transfers.
- The **mask register set/reset (MRSR)** sets or clears the channel mask to disable or enable particular DMA channels.
- The **status register** shows the status of each DMA channel.

Data bus buffer:

- It is a tri-state, bidirectional, 8 bit buffer which interfaces the 8257 to the system data in the slave mode; it is used to transfer data between microprocessor and internal registers.
- In master mode, it is used to send higher byte address (A8-A15) on the data bus.

Read/write logic:

- When the microprocessor is programming or reading one of the internal registers of the read/write logic accepts the I/O read (IOR) or low signal.
- Decodes least significant four address bits (A0-A7) and either writes the contents of the data bus addressed register or places the contents of the addressed register onto data bus.
- During DMA cycles the Read/write logic generates the I/O read and memory write or I/O write and memory read signals IOR control the data transfer between peripheral and memory device.

DMA channels:

The 8257 provides four identical channels labeled CH_0, CH_1, CH_2 and CH_3. Each channel has two-16 bit registers. They are

1. DMA address register
2. Terminal count register

1. DMA address register:

- It specifies the address of the first memory location to be accessed.

- It is necessary to load valid memory address in the DMA address register before channel is enabled.

2. Terminal count register:

- The value loaded into the low order 14 bits of TCR specifies the number of DMA cycles minus one (N-1) before TC output is activated.
- Therefore, for N number of desired DMA cycles it is necessary to load the value N-1 into the low order 14 bits of TCR.
- MSB 2 bits specify the type of operation to be performed.

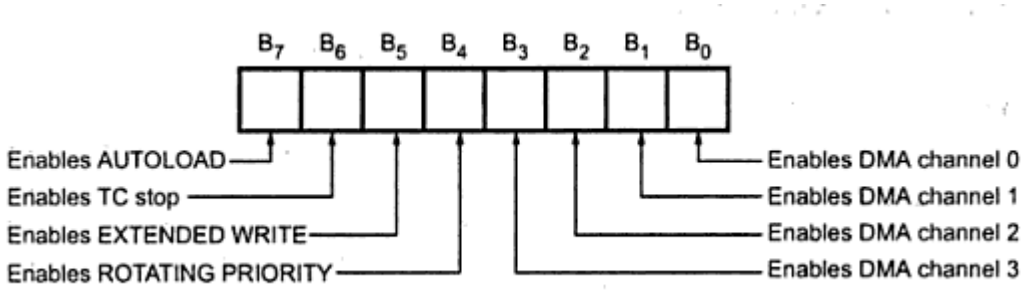
Control Logic:

- It controls the sequence of operations during all DMA cycles by generating the appropriate control signals and the 16 bit address that specified the memory location to be accessed.
- It consists of mode set register and status register.
- Mode set register is programmed by the CPU to configure 8257 whereas the status register is read by CPU to check which channels have reached a terminal count condition and status of update flag.

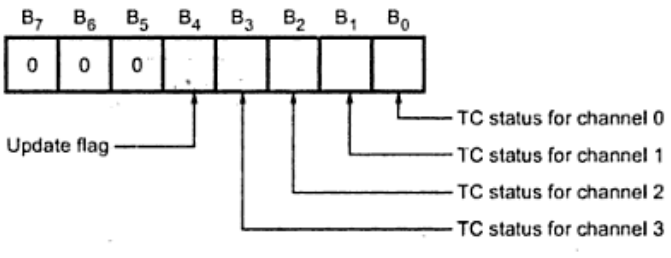
- **Mode set register:**
  - LSB 4 bits are the enable 4 DMA channels.
  - MSB 4 bits are the enable auto load, TC stop, extended write, rotating priority modes and terminal count registers.
  - It is cleared by RESET input, this disabling all options, inhibiting all channels and preventing bus conflicts on power-up.

  Master mode,
  - It controls the sequence of DMA operation during all DMA cycles.
  - It generates address and control signals.
  - It increments 16 bit address and decrement 14 bit counter registers.
  - It activates a HRQ signal on DMA channel Request.

  Slave mode it is disabled.



- **STATUS REGISTER:**



  - It indicates which channels have reached a terminal count condition and includes the update flag.

- The Tc status bit=1, terminal count has been reached for that channel.
- Tc bit remains set until the status register is read or the 8257 is reset.
- Update flag =1, 8257 is executing update cycle
- In update cycle 8257 lad parameters in channel 3 to channel 2.

PRIORITY RESOLVER:

It resolves the peripherals request. It can be programmed to work into two modes, either fixed mode or rotating priority mode.

Initializing of DMA controller

- A DMA controller is capable of becoming the bus master and supervising a transfer between an I/O or mass storage interface and memory. While making a transfer, it must be able to place memory address on the bus and send and receive handshaking signals in a manner similar to that of the bus control logic. The purpose of a DMA controller is to perform a sequence of transfers (ie a block transfer) by stealing bus cycles.

- A DMA controller is designed to service one or more I/O mass storage interfaces, and each interface is connected to the controller by a set of conductors. A portion of a DMA controller for servicing a single interface is called a channel..

- The general organization of a one channel DMA controller and its principal connection is shown in figure. In addition to the usual control and status registers, each channel must contain an address register and a byte (or word) count register.

- Initializing the controller consists of filling these registers with the beginning (or ending) address of the memory array that is to be used as a buffer and the number of bytes (words) to be transferred .For an input to memory, each time the interface has data to transfer it makes a DMA request

- The controller then makes a bus request and when it receives a bus grant, it puts the contents of the address register on the address bus, sends an acknowledgement back to the interface, and issues I/O read and memory write signals. The interface then puts the data on the data bus and drops its request.

- When the memory accepts the data it returns a ready signal to the controller, which then increments (or decrements) the address register, decrements the byte (word) count, and drops its bus request.

- Upon the count reaching zero, the process stops and a signal is sent to the processor as an interrupt request or to the interface to notify it that the transfers have terminated. An output is similarly executed except that the controller issues I/O write and memory read signals and the data are transferred in the other direction.

DRQ0-DRQ3 (DMA Request):

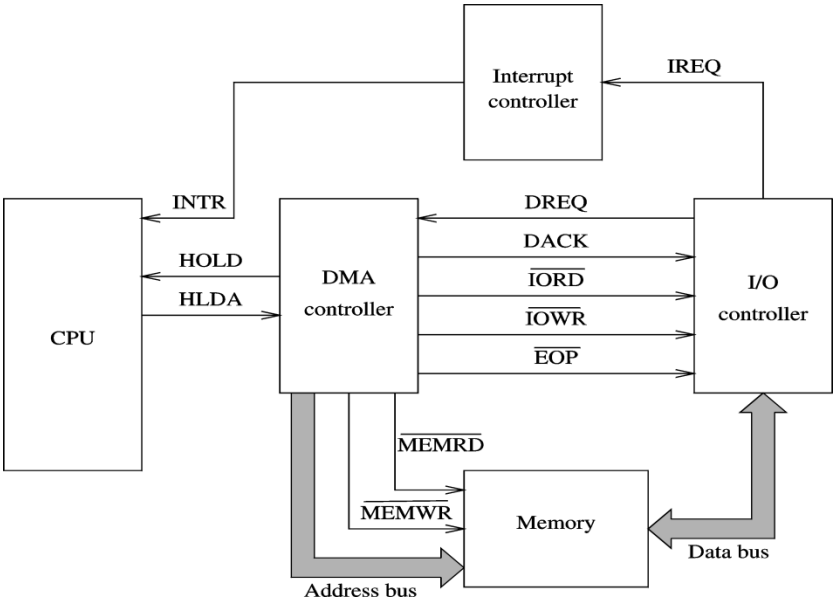
These are the asynchronous peripheral request input signal. The request signals is generated by external peripheral device.

DACK0-DACK3:

These are the active low DMA acknowledge output lines. Low level indicate that, peripheral is selected for giving the information (DMA cycle).In master mode it is used for chip select

HLDA becomes active to indicate the processor has placed its buses at high-impedance state as can be seen in the timing diagram,there are a few clock cycles between the time that HOLD changes and until HLDA changes

HLDA output is a signal to the requesting device that the processor has relinquished control of its memory and I/O space**.** one could call HOLD input a DMA request input and HLDA output a DMA grant signal



Steps in a DMA operation

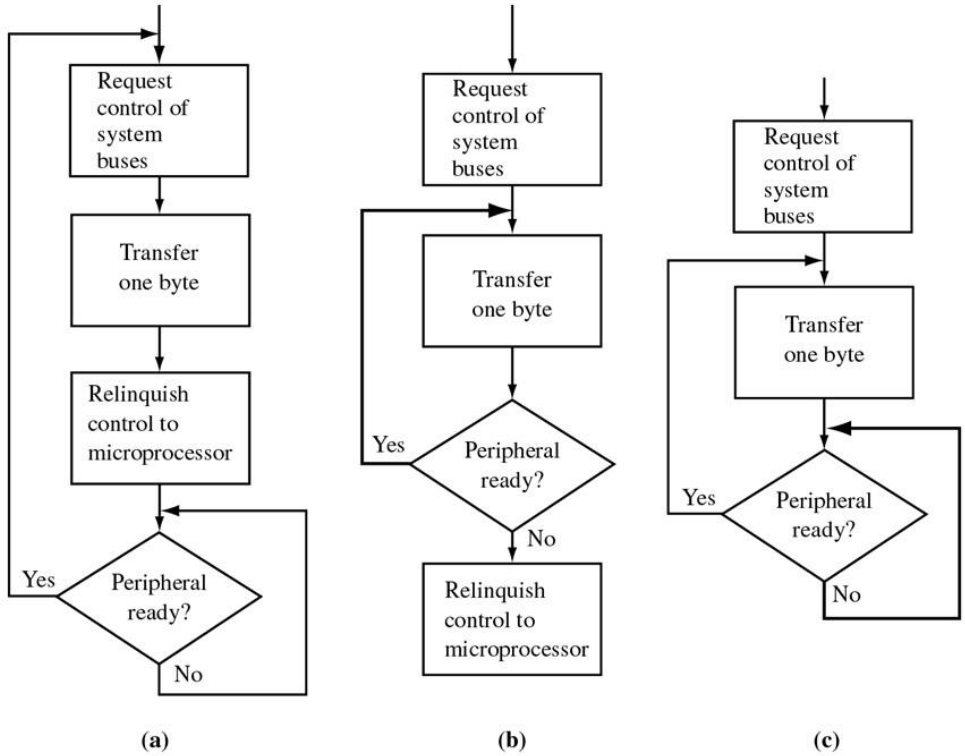
- Processor initiates the DMA controller gives device number, memory buffer pointer, called ***channel initialization.***
- Once initialized, it is ready for data transfer.
- When ready, I/O device informs the DMA controller .DMA controller starts the data transfer process
    - Obtains bus by going through bus arbitration
    - Places memory address and appropriate control signals
    - Completes transfer and releases the bus
    - Updates memory address and count value
    - If more to read, loops back to repeat the process

Notify the processor when done typically uses an interrupt

Modes of DMA operation

Each channel may be put in one of four modes,with its current mode being determined by bits 7 and6 of the channel's mode register .The four possible modes are

a) Byte **b) Burst** **c) Block**



- **Single transfer mode (01)**
  After each transfer the controller will release the bus to the processor for at least one bs cycle, but will immediately begin testing for DREQ inputs and proceed to steal another cycle as soon as a DREQ line becomes active.
- **Block transfer mode (10)**
  DREQ need only be active until DACK becomes active,after which the bus is not released until the entire block of data has been transferred.
- **Demand Transfer mode(00)**
  This is similar to the block mode except that DREQ is tested after each transfer. If DREQ is inactive , transfers are suspended until DREQ once again becomes active ,at which time the block transfer continues from the point at which it was suspended. This allows the interface to stop the transfer in the event that its device cannot keep up.
- **Cascade Mode(11)**
  In this mode 8237s may be cascaded so that more than four channels can be included in the DMA subsystem. In cascading the controllers ,those in the second level are connected to those in the first level by joining HRQ to DREQ and HLDA to DACK ,To conserve space, this mode will not be considered further.

In this mode

Single-cycle mode: DMA data transfer is done one byte at a time

Burst-mode: DMA transfer is finished when all data has been moved.

****

Draw the block diagram of 8259A and explain how to program 8259A (April 2010).(Dec 2018)
Explain the working of 8259 with a neat block diagram . (Nov/Dec 2016/April 2015)

****

7. Programmable Interrupt controller (8259)

Definition:

- The Intel 8259 Programmable Interrupt Controller handles up to eight vectored priority interrupts or the CPU.It is cascadable for up to 64 vectored priority interrupts without additional circuitry. It is packaged in a 28-pin DIP, uses NMOS technology and requires a single a5V supply.

- It accepts requests from the peripheral equipment, determines which of the incoming requests is of the highest importance (priority), ascertains whether the incoming request has a higher priority value than the level currently being serviced, and issues an interrupt to the CPU based on this determination.

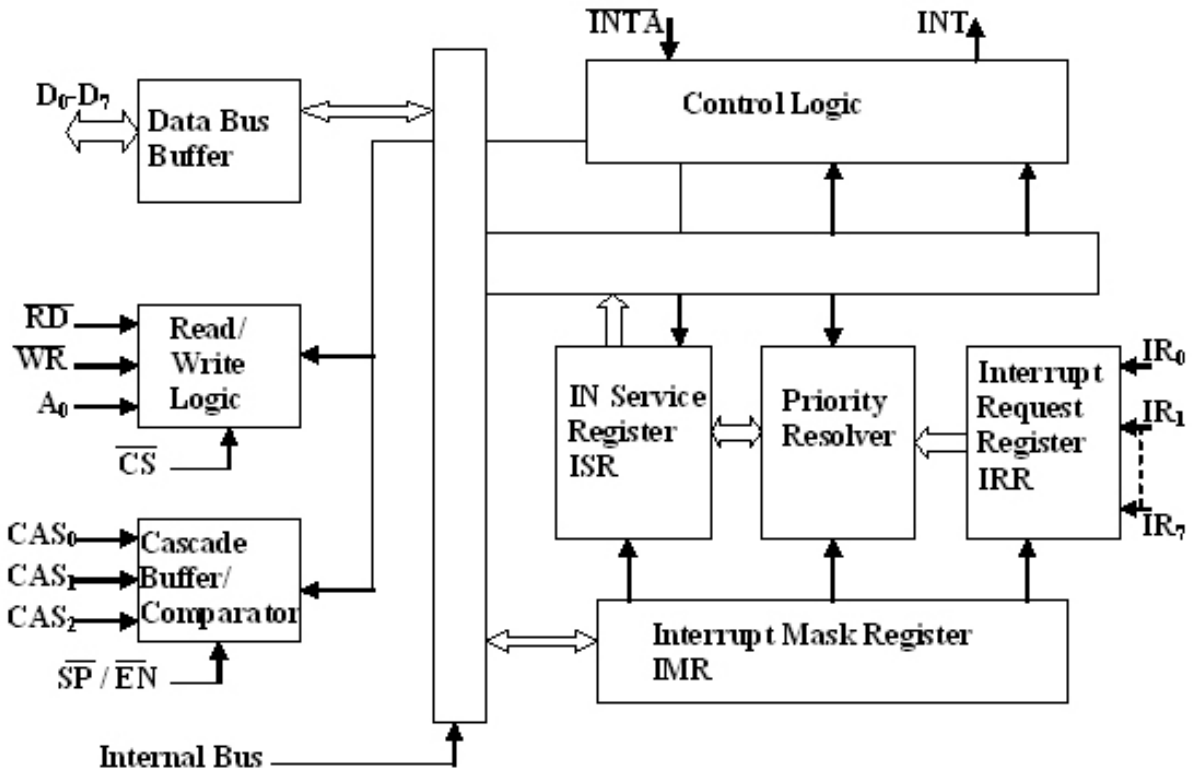


Fig:1 8259A Block Diagram

- **Interrupt Request Register (RR)**: IRR stores all the interrupt request in it in order to serve them one by one on the priority basis.

- **In-Service Register (ISR)**: This stores all the interrupt requests those are being served, i.e. ISR keeps a track of the requests being served.

- **Priority Resolver:** This unit determines the priorities of the interrupt requests appearing simultaneously. The highest priority is selected and stored into the corresponding bit of ISR during INTA pulse. The IR0 has the highest priority while the IR7 has the lowest one, normally in fixed priority mode. The priorities however may be altered by programming the 8259A in rotating priority mode.

- **Interrupt Mask Register (IMR)** : This register stores the bits required to mask the interrupt inputs. IMR operates on IRR at the direction of the Priority Resolver.

  Interrupt Control Logic: This block manages the interrupt and interrupt acknowledge signals to

  be sent to the CPU for serving one of the eight interrupt requests. This also accepts the interrupt acknowledge (INTA) signal from CPU that causes the 8259A to release vector address on to the data bus.

- **Data Bus Buffer**: This tristate bidirectional buffer interfaces internal 8259A bus to the microprocessor system data bus. Control words, status and vector information pass through data buffer during read or write operations.

- **Read/Write Control Logic**: This circuit accepts and decodes commands from the CPU. This block also allows the status of the 8259A to be transferred on to the data bus.

- **Cascade Buffer/Comparator**: This block stores and compares the ID's all the 8259A used in system. The three I/O pins CASO-2 are outputs when the 8259A is used as a master. The same pins act as inputs when the 8259A is in slave mode. The 8259A in master mode sends the ID of the interrupting slave device on these lines. The slave thus selected, will send its preprogrammed vector address on the data bus during the next INTA pulse.

- **CS**: This is an active-low chip select signal for enabling RD and WR operations of 8259A. INTA function is independent of CS.

- **WR:** This pin is an active-low write enable input to 8259A. This enables it to accept command words from CPU.

- **RD:** This is an active-low read enable input to 8259A. A low on this line enables 8259A to release status onto the data bus of CPU.

- **D0-D7** : These pins from a bidirectional data bus that carries 8-bit data either to control word or from status word registers. This also carries interrupt vector information.

- **CAS0 – CAS2 Cascade Lines:** A signal 8259A provides eight vectored interrupts. If more interrupts are required, the 8259A is used in cascade mode. In cascade mode, a master 8259A along with eight slaves 8259A can provide up to 64 vectored interrupt lines. These three lines act as select lines for addressing the slave 8259A.

- **PS/EN**: This pin is a dual purpose pin. When the chip is used in buffered mode, it can be used as buffered enable to control buffer transreceivers. If this is not used in buffered mode then the pin is used as input to designate whether the chip is used as a master (SP =1) or slave (SP = 0).

- **INT**: This pin goes high whenever a valid interrupt request is asserted. This is used to interrupt the CPU and is connected to the interrupt input of CPU.

- **IR0 – IR7 (Interrupt requests)** :These pins act as inputs to accept interrupt request to the CPU. In edge triggered mode, an interrupt service is requested by raising an IR pin from a low to a high

state and holding it high until it is acknowledged, and just by latching it to high level, if used in level triggered mode.

- **A0 :** This input signal is used in conjunction with WR and RD signals to write commands into the various command registers, as well as reading the various status registers of the chip. This line can be tied directly to one of the address lines.

Command Words of 8259A

The 8259A accepts two types of command words generated by the CPU:

1. **Initialization Command Words (ICWs):**
Before normal operation can begin, each 8259A in the system must be brought to a starting pointed by a sequence of 2 to 4 bytes timed by WR pulses.

2. Operational Command Words (OCWs):
These are the command words which command the 8259A to operate in various interrupt modes. These modes are:

a. Fully nested mode

b. Rotating priority mode

c. Special mask mode

d. Polled mode

The OCWs can be written into the 8259A anytime after initialization.

❖ **Interrupt Sequence of 8259 Programmable Interrupt Controller**

Interrupt Sequence with an 8085 system

1. One or more IR lines are raised high that set corresponding IRR bits.

2. 8259A resolves priority and sends an INT signal to CPU.

3. The CPU acknowledge with INTA pulse.

4. Upon receiving an INTA signal from the CPU, the highest priority ISR bit is set and the corresponding IRR bit is reset. The 8259 will also release a CALL instruction code (11001101) on to the 8 bit data through its D7 - D0 pins.

5. The CALL instruction will initiate a second INTA pulse. During this period 8259A releases an 8-bit pointer on to a data bus from two more INTA pulses to be sent to the 8259 from the CPU group.

6. These two INTA pulses allow the 8259 to release its programmed subroutine address onto the data bits. The lower 8 bit address is released at the first INTA pulse and the higher 8 bit address is released at the second INTA pulse.

7. This completes the 3 byte CALL instruction released by the 8259. Interrupt cycle. The ISR bit is reset at the end of the second INTA pulse if automatic end of interrupt (AEOI) mode is programmed. Otherwise ISR bit remains set until an appropriate EOI command is issued at the end of interrupt subroutine.

❖ **Priority Modes**

1.Fully Nested Mode

- IR0 is the highest and IR7 is the lowest one
- In addition any IR can be assigned the highest priority; the priority sequence will begin at that IR.

Example:

IR_0	IR_1	IR_2	IR_3	IR_4	IR_5	IR_6	IR_7
4	**5**	**6**	**7**	**0**	**1**	**2**	**3**

2.Automatic Rotation(equal Priority):

In this mode, a device which one is being serviced will be considered as a lowest priority in the next time

First Time	IR_0	IR_1	IR_2	IR_3	IR_4	IR_5	IR_6	IR_7
	6	**7**	**0**	**1**	**2**	**3**	**4**	**5**
Second Time	IR_0	IR_1	IR_2	IR_3	IR_4	IR_5	IR_6	IR_7
	1	**0**	**7**	**6**	**5**	**4**	**3**	**2**

3.Specific rotation mode(Specific Priority)

The programmer can change the priorities by programming the bottom priority and ths fixing all other priorities.ie if IR4 is programmed as the lowest priority ,then IR% will have the highest one**.**

End of Interrupt (EOI)

- After the completion of an interrupt service, the corresponding **ISR bit needs to be reset.**
- This is called the **End of Interrupt (EOI).**
- It can be issued in 3 formats. They are,
-

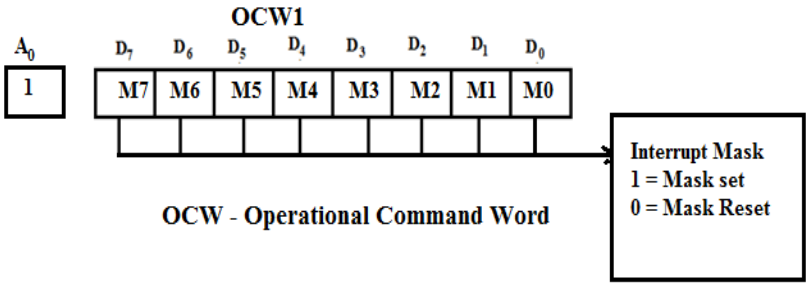
(i) **Non-specific EOI command**→When the 8259 receives this command, **it resets the highest priority ISR bit.**

(ii) **Specific EOI command**→ It specifies which **ISR bit to be reset**.

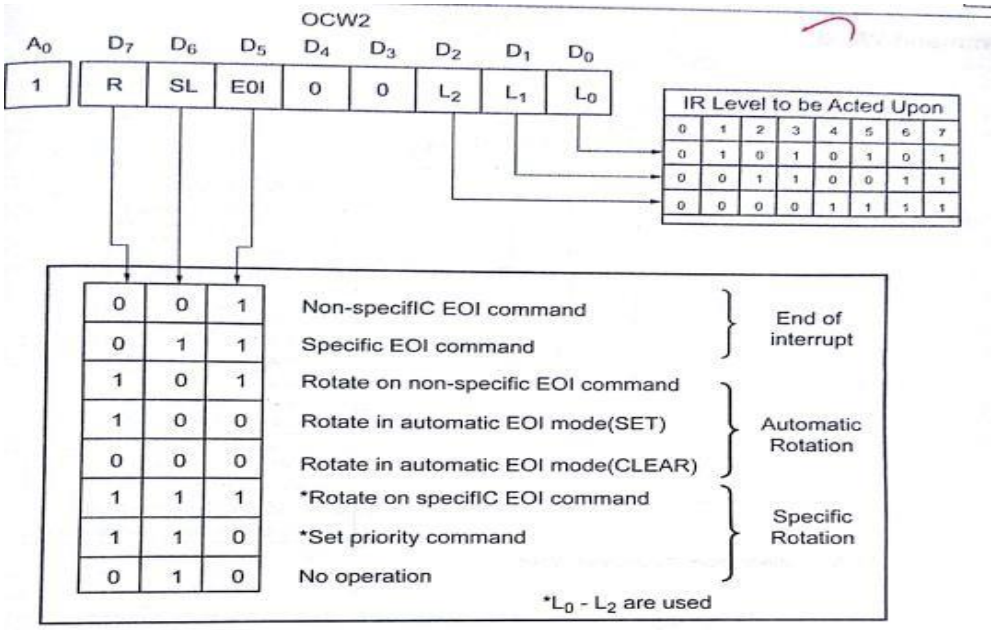
(iii) **Automatic EOI command** When the **8259 receives the third signal, the ISR bit is reset**

Command and Status Words of 8259

✓ **Command Word for Operational Command Words (OCWs)**



OCW - Operational Command Word



****

Explain A/D interface with 8085 with neat sketch.(Dec 2014)
Explain how D/A and A/D interfacing done with 8085with an application(April 2015)(Dec 2017)

****

8.Analog to digital conversion

- The process of analog to digital conversion is a slow process, and the microprocessor has to wait for the digital data till the conversion is over.
- After the conversion is over, the ADC sends end of conversion EOC signal to inform the microprocessor that the conversion is over and the result is ready at the output buffer of the ADC. These tasks of issuing an SOC pulse to ADC, reading EOC signal from the ADC and reading the digital output of the ADC are carried out by the CPU using 8255 I/O ports.
- The time taken by the ADC from the active edge of SOC pulse till the active edge of EOC signal is called as the conversion delay of the ADC.
- It may range anywhere from a few microseconds in case of fast ADC to even a few hundred milliseconds in case of slow ADCs.

- The available ADC in the market use different conversion techniques for conversion of analog signal to digitals. Successive approximation techniques and dual slope integration techniques are the most popular techniques used in the integrated ADC chip.

General algorithm for ADC interfacing contains the following steps:

1. Ensure the stability of analog input, applied to the ADC.
2. Issue start of conversion (SOC) pulse to ADC
3. Read end of conversion signal to mark the end of conversion processes.
4. Read digital data output of the ADC as equivalent digital output.

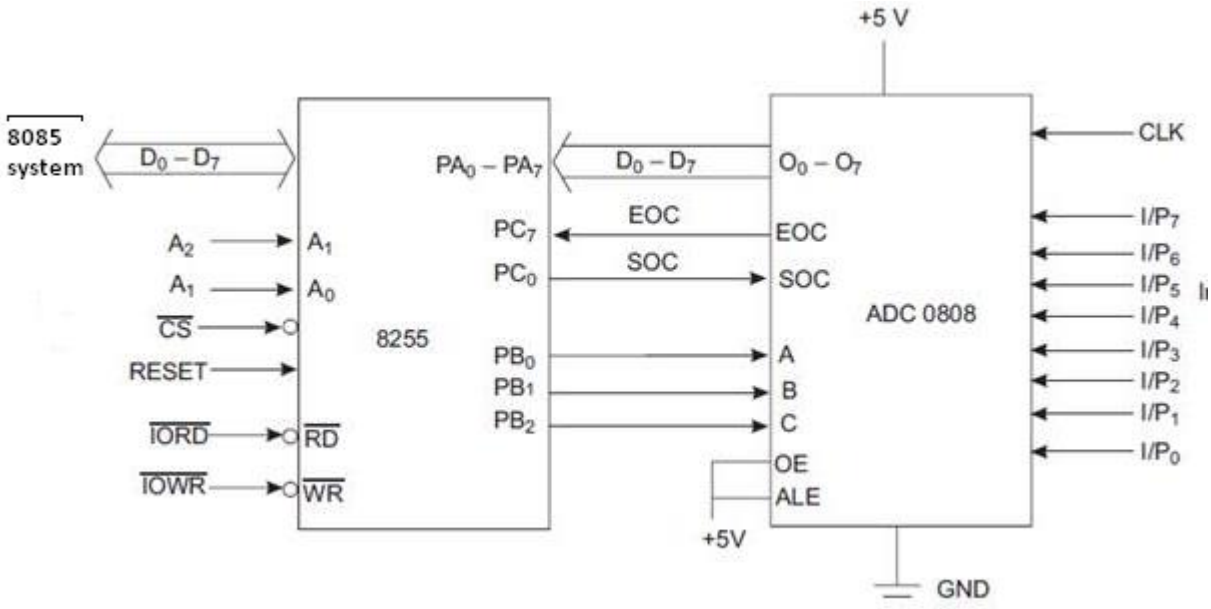


Figure:ADC0808 interfacing with 8085 using 8255

- Analog input voltage must be constant at the input of the ADC right from the start of conversion till the end of the conversion to get correct results.
- This may be ensured by a sample and hold circuit which samples the analog signal and holds it constant for a specific time duration.
- The microprocessor may issue a hold signal to the sample and hold circuit. If the applied input changes before the complete conversion process is over, the digital equivalent of the analog input calculated by the ADC may not be correct.

ADC 0808/0809 :

- The analog to digital converter chips 0808 and 0809 are 8-bit CMOS, successive approximation converters.

- This technique is one of the fast techniques for analog to digital conversion. The conversion delay is 100μs at a clock frequency of 640 KHz, which is quite low as compared to other converters.

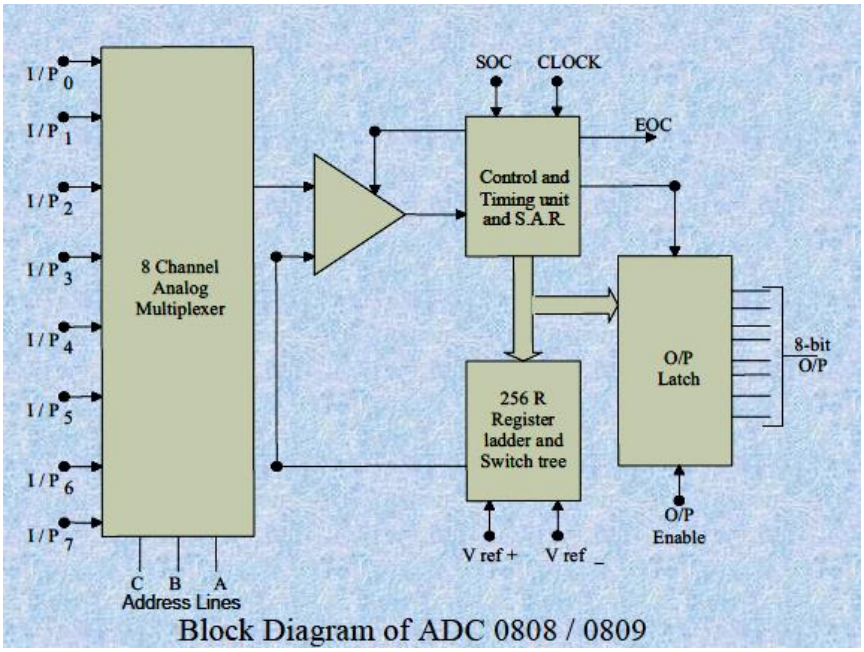
- These converters do not need any external zero or full scale adjustments as they are already taken care of by internal circuits. These converters internally have a 3:8 analog multiplexer so that at a time eight different analog conversion by using address lines ADD A, ADD B, ADD C.

- Using these address inputs, multichannel data acquisition system can be designed using a single ADC.

- The CPU may drive these lines using output port lines in case of multichannel applications. In case of single input applications, these may be hardwired to select the proper input.

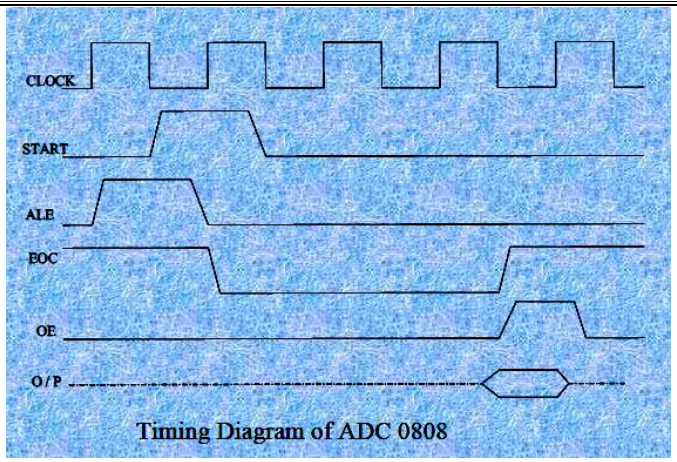
- There are unipolar analog to digital converters, i.e. they are able to convert only positive analog input voltage to their digital equivalent.

- These chips do not contain any internal sample and hold circuit.If one needs a sample and hold circuit for the conversion of fast signal into equivalent digital quantities, it has to be externally connected at each of the analog inputs.



Block Diagram of ADC 0808 / 0809

- Vcc Supply pins +5V
- GND GND
- Vref + Reference voltage positive +5 Volts maximum.
- Vref_ Reference voltage negative 0Volts minimum
- I/P0–I/P7 Analog inputs
- ADD A,B,C Address lines for selecting analog inputs.
- O7 – O0 Digital 8-bit output with O7 MSB and O0 LSB
- SOC Start of conversion signal pin
- EOC End of conversion signal pin
- OE Output latch enable pin, if high enables output
- CLK Clock input for ADC



Timing Diagram of ADC 0808

Example: Interfacing ADC 0808 with 8085using 8255 ports. Use port A of 8255 for transferring digital data output of ADC to the CPU and port C for control signals.Assume that an analog input is present at I/P2 of the ADC and a clock input of suitable frequency is available for ADC.

• **Solution**: The analog input I/P2 is used and therefore address pins A,B,C should be 0,1,0 respectively to select I/P2. The OE and ALE pins are already kept at +5V to select the ADC and enable the outputs. Port C upper acts as the input port to receive the EOC signal while port C lower acts as the output port to send SOC to the ADC.

Port A acts as a 8-bit input data port to receive the digital data output from the ADC. The 8255 control word is written as follows:

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0

1 0 0 1 1 0 0 0

The required ALP is as follows:

```
        MOV A, 98h   ; initialise 8255 as
        OUT CWR,     ;discussed above.
        MOV A, 02h   ; Select I/P2 as analog
        OUT Port B   , input.
        MOV AL, 00h; Give start of conversion
        OUT Port C   ; pulse to the ADC
        MOV AL, 01h
        OUT Port C
        MOV AL, 00h
        OUT Port C
WAIT:   IN Port C    ;Check for EOC by
        RCR          ; reading port C upper and
        JNC WAIT     ; rotating through carry.
        IN Port A    ; If EOC, read digital equivalent ;in AL
        HLT          ; Stop
```

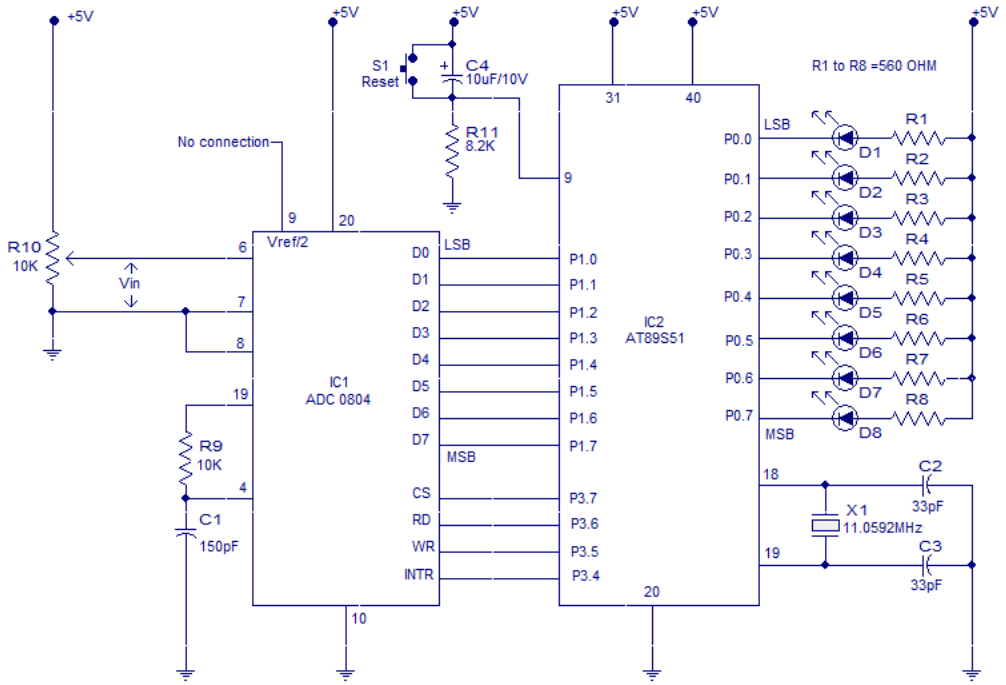


Figure:Interfacing ADC to 8051

The circuit initiates the ADC to convert a given analogue input , then accepts the corresponding digital data and displays it on the LED array connected at P0

ORG 00H

MOV P1,#11111111B // initiates P1 as the input port

MAIN: CLR P3.7 // makes CS=0

SETB P3.6 // makes RD high

CLR P3.5 // makes WR low

SETB P3.5 // low to high pulse to WR for starting conversion

WAIT:JB P3.4,WAIT // polls until INTR=0

CLR P3.7 // ensures CS=0

CLR P3.6 // high to low pulse to RD for reading the data from ADC

MOV A,P1 // moves the digital data to accumulator

CPL A // complements the digital data

MOV P0,A // outputs the data to P0 for the LEDs

SJMP MAIN // jumps back to the MAIN program

END

****

Explain the interfacing of D/A converter with 8085 and 8051.and Write a program for generating any typical waveform.(June 2016)(Dec 2018)

****

9.Interfacing Digital To Analog Converters

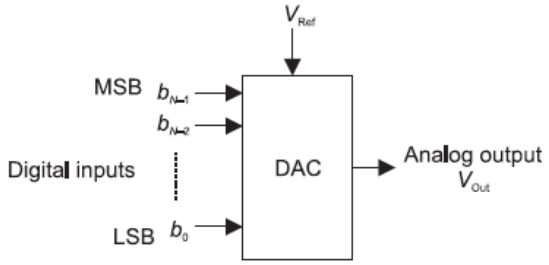
- The digital to analog converters convert binary number into their equivalent voltages. The DAC find applications in areas like digitally controlled gains, motors speed controls, programmable gain amplifiers etc.

- **AD 7523 8-bit Multiplying DAC** : This is a 16 pin DIP, multiplying digital to analog converter, containing R-2R ladder for D-A conversion along with single pole double thrown NMOS switches to connect the digital inputs to the ladder.

- The pin diagram of AD7523 is shown in fig the supply range is from +5V to +15V, while V_{ref} may be anywhere between -10V to +10V. The maximum analog output voltage will be anywhere between -10V to +10V, when all the digital inputs are at logic high state.

- Usually a zener is connected between OUT1 and OUT2 to save the DAC from negative transients. An operational amplifier is used as a current to voltage converter at the output of AD to convert the current output of AD to a proportional output voltage.

Digital to anlog converter



Block diagram of a digital to anlog converter

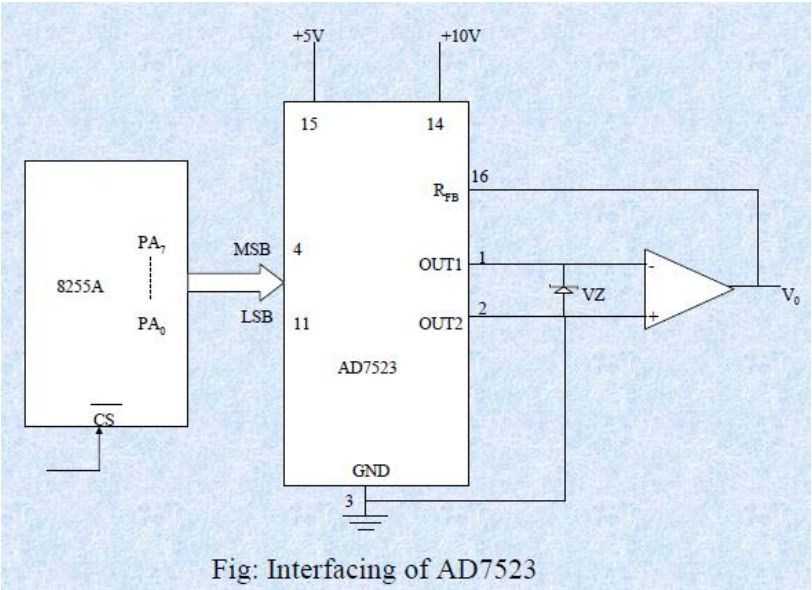


Fig: Interfacing of AD7523

Figure:DAC connected with 8085 via 8255

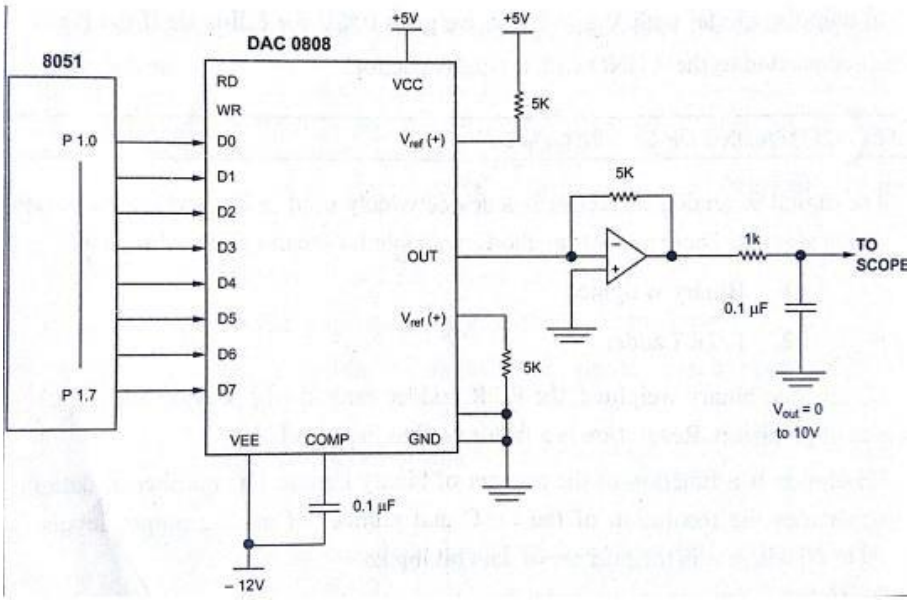


Figure:DAC connected with 8051

It also offers additional drive capability to the DAC output. An external feedback resistor acts to control the gain. One may not connect any external feedback resistor, if no gain control is required.

EXAMPLE: Interfacing DAC AD7523 with an 8085 CPU running at 8MHZ and write an assembly language program to generate a sawtooth waveform of period 1ms with V_{max} 5V.

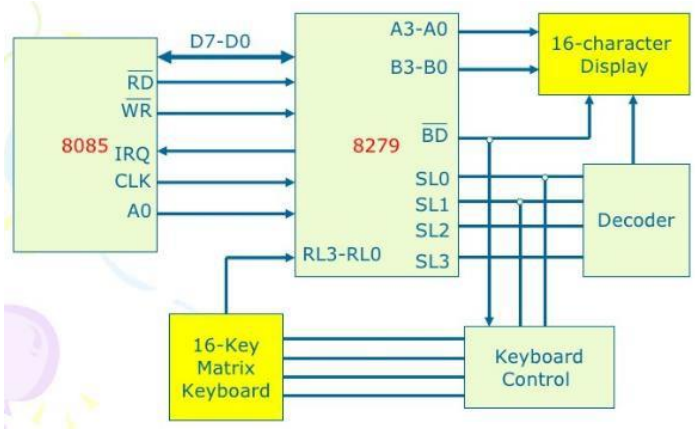
Solution: Fig shows the interfacing circuit of AD 74523 with 8086 using 8255 program gives an ALP to generate a sawtooth waveform using circuit.

```
        MOV A, 80h ;make all ports output
        OUT C0, AL
AGAIN: MOV AL, 00H ;start voltage for ramp
BACK: OUT PA
        INR A
        CPI 0FFh
        JB BACK
        JMP AGAIN
```

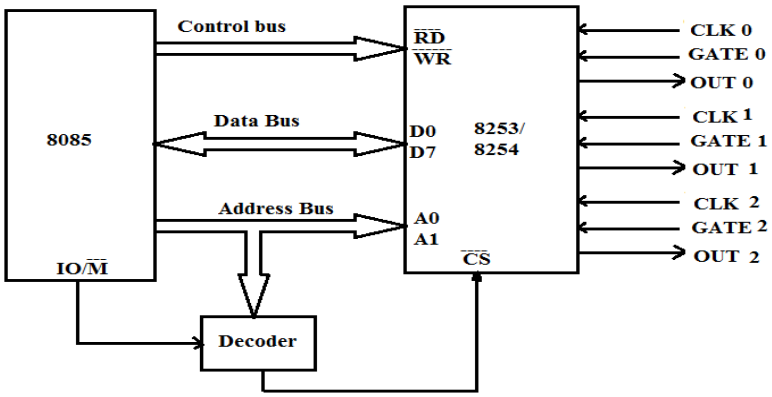
In the above program, port A is initialized as the output port for sending the digital data as input to DAC.

- The ramp starts from the 0V (analog), hence AL starts with 00H. To increment the ramp, the content of AL is increased during each execution of loop till it reaches F2H.
- After that the saw tooth wave again starts from 00H, i.e. 0V (analog) and the procedure is repeated. The ramp period given by this program is precisely 1.000625 ms.
- Here the count F2H has been calculated by dividing the required delay of 1ms by the time required for the execution of the loop once.
- The ramp slope can be controlled by calling a controllable delay after the OUT instruction.

INTERFACING 8279 WITH 8085



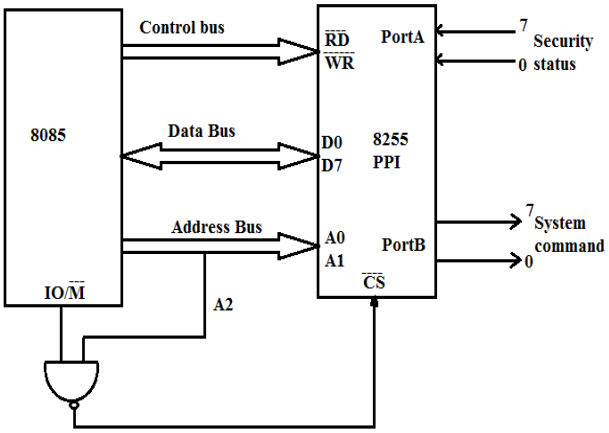
Interfacing 8254 with 8085



INTERFACING 8255 WITH 8085

8085 Microprocessor interfaced to the 8255.

- Port A has been used as the input port for the security status, whereas, port B has been used as the output port for the command.
- The port numbers assigned are 04 (port A), 05 (port B), 06 (port C) and 07 (Control Word) as evident from the circuit.



Explain the LED interfacing with 8086 microprocessor(April 2018)

****

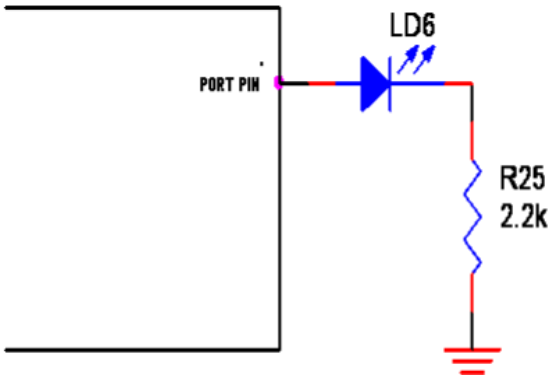
Interfacing LED with 8086

LED (LIGHT EMITTING DIODES)

Light Emitting Diodes (LED) is the most commonly used components, usually for displaying pins digital states. Typical uses of LEDs include alarm devices, timers and confirmation of user input such as a mouse click or keystroke.

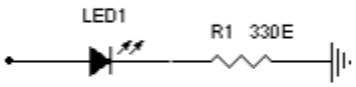
INTERFACING LED

Fig. 1 shows how to interface the LED to microprocessor. As you can see the Anode is connected through a resistor to GND & the Cathode is connected to the Microprocessor pin. So when the Port Pin is HIGH the LED is OFF & when the Port Pin is LOW the LED is turned ON.

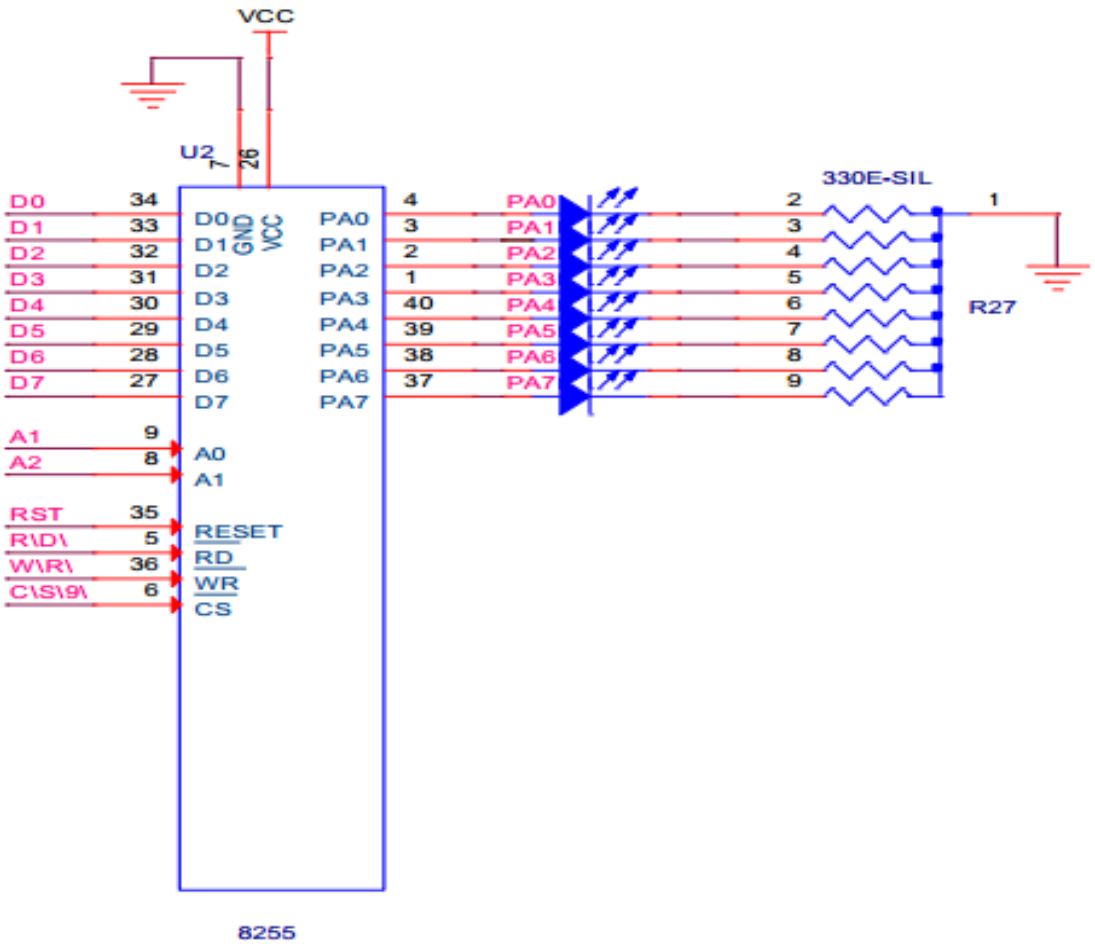


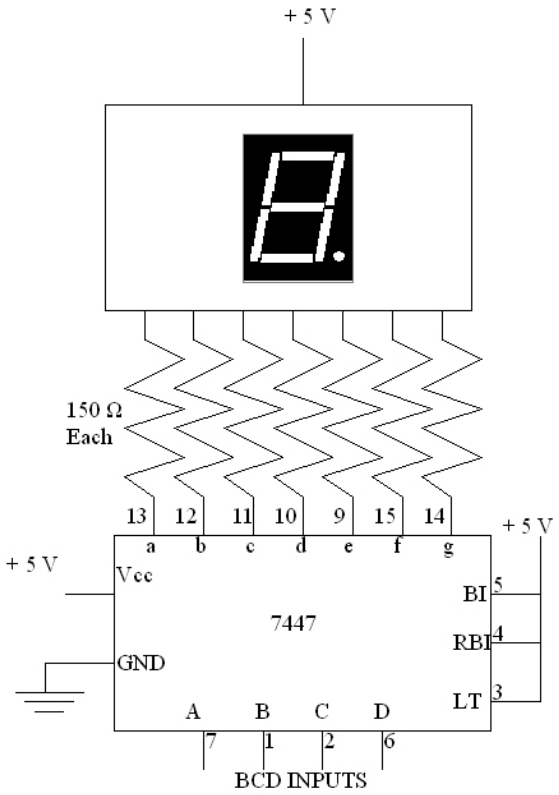
INTERFACING LED WITH 8086 We now want to flash a LED in 8086 Trainer Board. It works by turning ON a LED & then turning it OFF & then looping back to START. However the operating speed of microprocessor is very high

PIN ASSIGNMENT WITH 8086

	Point LEDs	8255 Lines	LED Selection
DIGITAL OUTPUTS	LD1	PA.0	
	LD2	PA.1	
	LD3	PA.2	
	LD4	PA.3	
	LD5	PA.4	
	LD6	PA.5	
	LD7	PA.6	
	LD8	PA.7	

CIRCUIT DIAGRAM TO INTERFACE LED WITH 8255





Circuit for driving single 7-segment LED display with 7447

ASSEMBLY PROGRAM TO ON AND OFF LED USING 8086

Title : Program to Blink LEDs

Title : Program to Blink LEDs

****

MEMORY ADDRESS	OPCODE	MNEMONICS
1100	B0 80	MOV AL, 80
1102	BA36 FF	MOV DX, FF36
1105	EE	OUT DX, AL
1106	B0 00	BEGIN:MOV AL, 00
1108	BA 30 FF	MOV DX, FF30
110B	EE	OUT DX, AL
110C	E8 08 00	CALL DELAY
110F	B0 FF	MOV AL, FF
1111	EE	OUT DX, AL
1112	E8 02 00	CALL DELAY
1115	EB EF	JMP BEGIN
1117	B9 FF FF	DELAY: MOV CX, FFFF
111A	49	P0: DEC CX
111B	75 FD	JNE PO
111D	C3	RET

UNIT – V
MICROCONTROLLER PROGRAMMING & APPLICATIONS

****

Simple programming exercises-key board and display interface – Closed loop control of servo motor- stepper motor control – Washing Machine Control

****

****

Explain the interfacing of Keyboard with 8051. [June 2016.December 2016.April 2018]

****

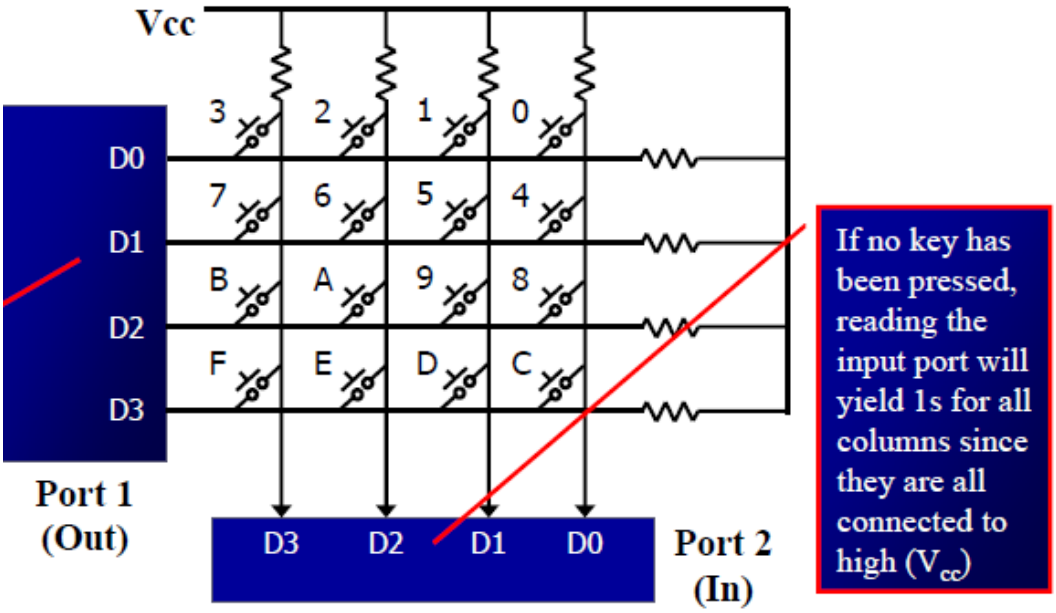
2.KEY BOARD AND DISPLAY INTERFACE

Keyboards are organized in a matrix of rows and columns

- The CPU accesses both rows and columns through ports. Therefore, with two 8-bit ports, an 8 x 8 matrix of keys can be connected to a microprocessor.
- When a key is pressed, a row and a column make a contact, Otherwise, there is no connection between rows and columns
- In IBM PC keyboards, a single microcontroller takes care of hardware and software interfacing.

A 4x4 matrix connected to two ports .The rows are connected to an output port and the columns are connected to an input port.

- It is the function of the microcontroller to scan the keyboard continuously to detect and identify the key pressed
- To detect a pressed key, the microcontroller grounds all rows by providing 0 to the output latch, then it reads the columns. If the data read from columns is D3 – D0 = 1111, no key has been pressed and the process continues till key press is detected.



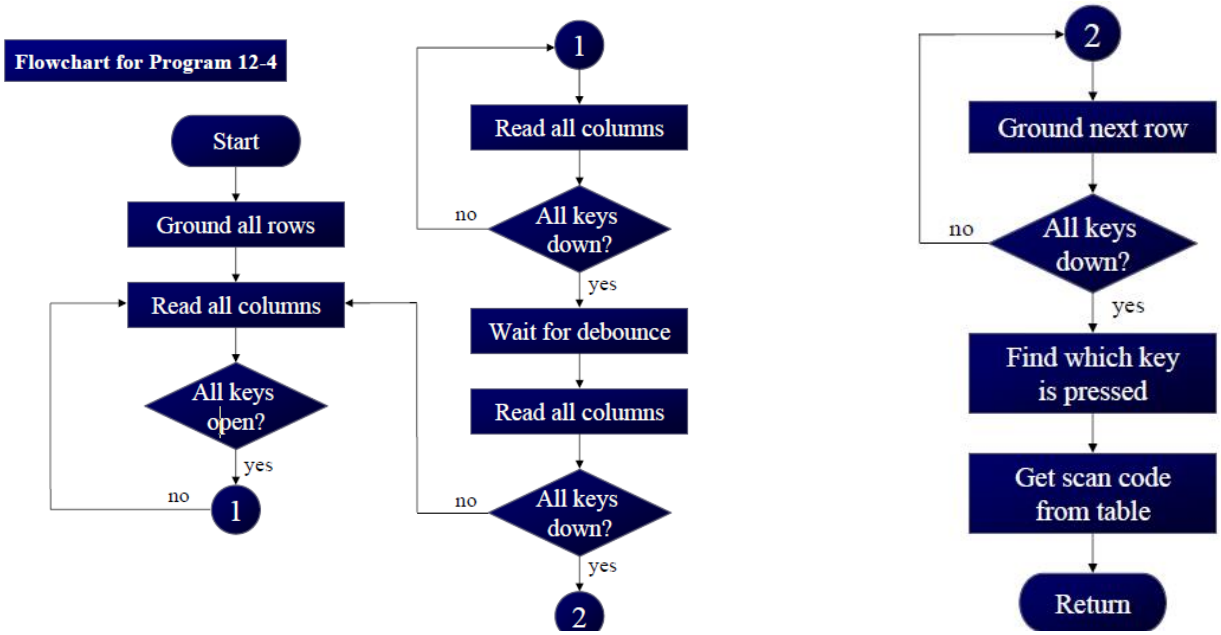
- If one of the column bits has a zero, this means that a key press has occurred, For example, if D3 – D0 = 1101, this means that a key in the D1 column has been pressed. After detecting a key press, microcontroller will go through the process of identifying the key.
- Starting with the top row, the microcontroller grounds it by providing a low to row D0 only.
- It reads the columns, if the data read is all 1s, no key in that row is activated and the process is moved to the next row.
- It grounds the next row, reads the columns, and checks for any zero
- This process continues until the row is identified
- After identification of the row in which the key has been pressed
- Find out which column the pressed key belongs to.

Program for detection and identification of key activation goes through the following stages:

1. To make sure that the preceding key has been released, 0s are output to all rows at once, and the columns are read and checked repeatedly until all the columns are high.
    - When all columns are found to be high, the program waits for a short amount of time before it goes to the next stage of waiting for a key to be pressed. To see if any key is pressed, the columns are scanned over and over in an infinite loop until one of them has a 0 on it.
    - Remember that the output latches connected to rows still have their initial zeros making them grounded.

2. After the key press detection, it waits 20 ms for the bounce and then scans the columns again.
    - It ensures that the first key press detection was not an erroneous one due to a spike noise.
    - If after the 20-ms delay the key is still pressed, it goes back into the loop to detect a real key press
3. To detect which row key press belongs to,it grounds one row at a time, reading the columns each time
    - If it finds that all columns are high, this means that the key press cannot belong to that row.Therefore, it grounds the next row and continues until it finds the row the key press belongs to.
    - Upon finding the row that the key press belongs to, it sets up the starting address for the look-up table holding the scan codes (or ASCII) for that row.

4. To identify the key press, it rotates the column bits, one bit at a time, into the carry flag and checks to see if it is low
    - Upon finding the zero, it pulls out the ASCII code for that key from the look-up table
    - otherwise, it increments the pointer to point to the next element of the look-up table



KEYBOARD INTERFACING WITH 8051:

<u>The steps in algorithm are as follows:</u>

1. Initialize P1.0, P1.1, P1.2 and P1.3 as inputs.
2. Check if all the keys are released by writing „0" to P1.4-P1.7 and check if all return lines are in state "1". If not then wait.
3. Call debounce.

4. Wait for key closure. Ground all scan lines by writing „0" and then check if at least one of return lines shows „0" level.

5. Call debounce.

6. Is key really pressed? (Check at least one of the return lines shows „0" level). No Step 4 , Yes step 7.

7. Find key code and display the key pressed on 7-segment display.

8. Go to step 1.

PROGRAM:

From the above figure identify the row and column of the pressed key for each of the following.

(a) D3 – D0 = 1110 for the row, D3 – D0 = 1011 for the column

(b) D3 – D0 = 1101 for the row, D3 – D0 = 0111 for the column Solution:

From the above figure, the row and column can be used to identify the key.

(a) The row belongs to D0 and the column belongs to D2; therefore, key number 2 was pressed.

(b) The row belongs to D1 and the column belongs to D3; therefore, key number 7 was pressed.

; Keyboard subroutine.
; This program sends the ASCII code for pressed key to P0.1.
; P1.0 – P1.3 connected to rows P2.0 – P2.3 connected to columns.

LOOK-UP TABLE FOR EACH ASCII
ROW ORG 300H

KCODE 0: DE '0', '1', '2', '3'	**; Row 0**
KCODE 1: DE '4', '5', '6', '7'	**; Row 1**
KCODE 2: DE '8', '9', 'A', 'B'	**; Row 2**
KCODE 3: DE 'C', 'D', 'E', 'F'	**; Row 3**

```
        MOV P2, #0FFH              ; make P2 an input port
K1:     MOV P1, #0                 ; ground all rows at once
        MCV A, P2                  ; read all column ensure all keys open.
        ANL A, #00001111B          ; masked unused bits
        CJNE A, #00001111B, K1     ; check till all keys released
K2:     ACALL DELAY                ; call 20 ms delay
        MCV A, P2                  ; see if any key is pressed
        ANL A, #00001111B          ; mask unused bits
        CJNE A, #00001111B, OVER   ; key pressed, await closure
        SJMP K2                    ; check if key pressed
```

```
OVER:    ACALL DELAY                ; wait 20 ms debounce time
         MCV A, P2                  ; check key closure
         ANL A, #00001111B          ;
         CJNE A, #00001111B, OVER1  ;
         SJMP K2                    ;
OVER1:   MOV P1, #11111110B         ;
         MOV A, P2                  ;
         ANL A, #00001111B          ;
```

```
         CJNE A, #00001111B, ROW_0
         MOV P1, #11111101B
         MOV A, P2
         ANL A, #00001111B
         CJNE A, #00001111B, ROW_1
         MOV P1, #11111011B
         MOV A, P2
         ANL A, #00001111B
         CJNE A, #00001111B, ROW_2
         MOV P1, #11110111B
         MOV A, P2
         ANL A, #00001111B
         CJNE A, #00001111B, ROW_3
         LJMP F2
         MOV DPTR, #KCODE 0
         SJMP FIND
          MOV DPTR,
          SJMP FIND
          MOV DPTR,
          SJMP FIND
          MOV DPTR,
          RRC A
          JNC MATCH
          INC DPTR
MATCH:    SJMP FIND
          CLR A
          MOVC A, @A +
          MOV P0, A
          LJMP K1
```

The steps in algorithm are as follows:

1. Initialize P1.0, P1.1, P1.2 and P1.3 as inputs.

2. Check if all the keys are released by writing „0" to P1.4-P1.7 and check if all return lines are in state '1'. If not then wait.

3. Call debounce.
4. Wait for key closure. Ground all scan lines by writing „0" and then check if at least one of return lines shows „0" level.
5. Call debounce.

****

Explain the interfacing of stepper motor with 8051. [June 2016/April 2018][DEC 2018]

****

3. Interfacing stepper motor with 8051

STEPPER MOTOR

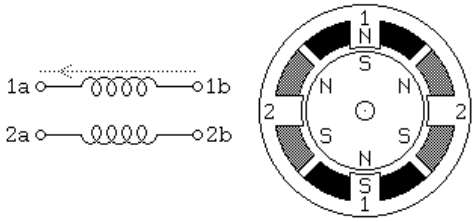
A stepper motor is a brushless, synchronous electric motor that converts digital pulses into mechanical shaft rotation. Every revolution of the stepper motor is divided into a discrete number of steps, and the motor must be sent a separate pulse for each step.

Stepper motors can be used in various areas of your microcontroller projects such as making robots, robotic arm, and automatic door lock system.

Fig. shows how to interface the Stepper Motor to microcontroller. As you can see the stepper motor is connected with Microcontroller output port pins through a ULN2803A array. So when the microcontroller is giving pulses with particular frequency to ls293A, the motor is rotated in clockwise or anticlockwise.

Step Angle

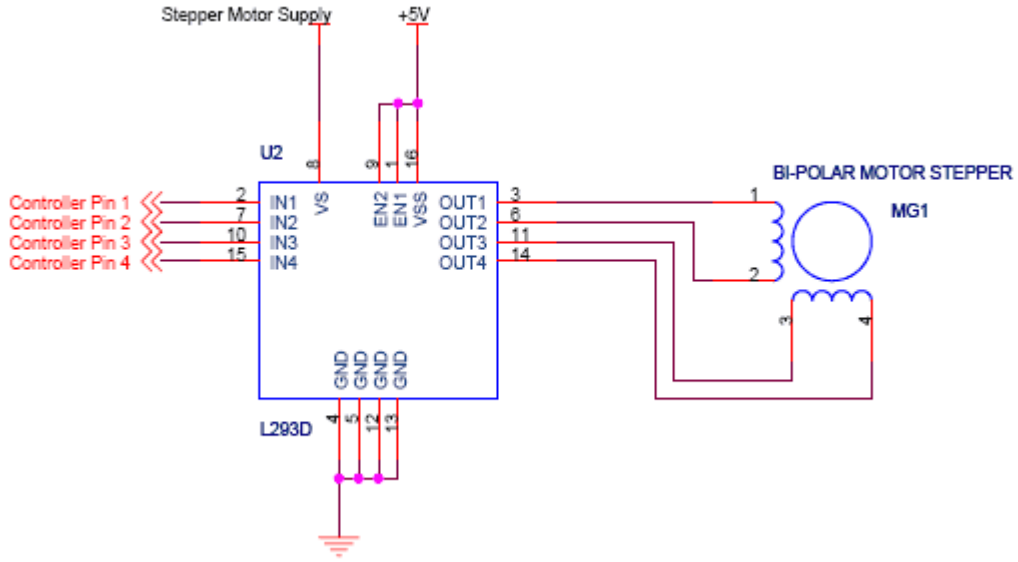
- Step angle of the stepper motor is defined as the angle traversed by the motor in one step.
- To calculate step angle, simply divide 360 by number of steps a motor takes to complete one revolution.
- Motor rotating in full mode takes 4 steps to complete a revolution ,so step angle can be calculated as step angle **θ = 360° / 4 =90.**



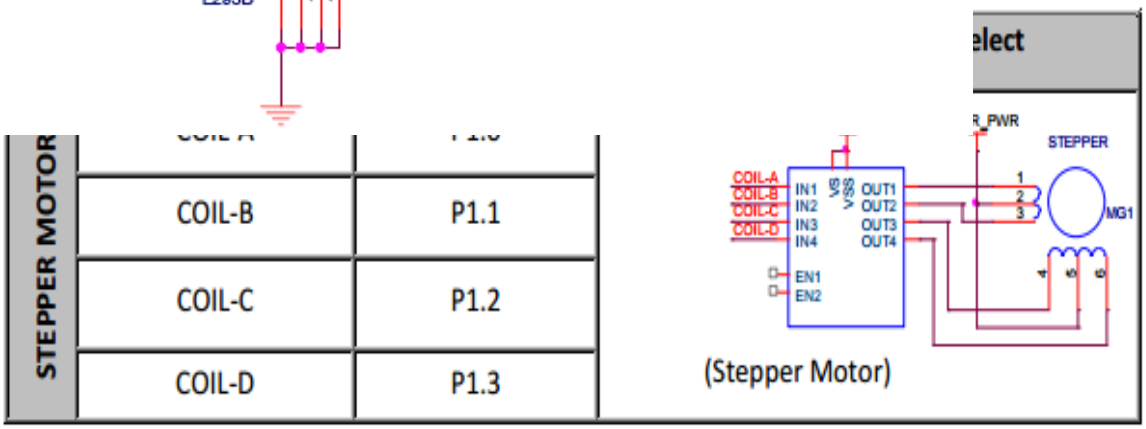
- By knowing the stepper motor step angle helps to move the motor in correct angular position**.**
- As you can see the stepper motor is connected with Microcontroller output port pins through a ULN2803A array.
- So when the microcontroller is giving pulses with particular frequency to ls293A, the motor is rotated in clockwise or anticlockwise

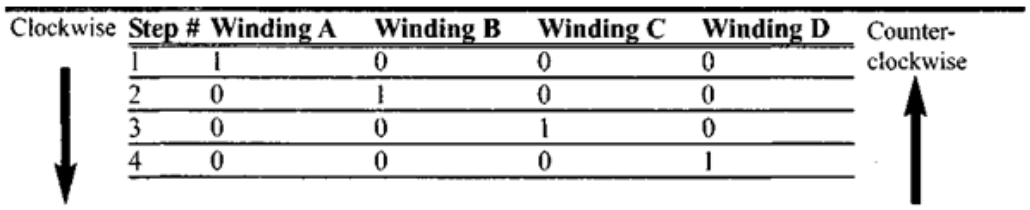
Program to interface Stepper motor with 8051

- To control a stepper motor in 8051 trainer by turning ON & OFF a four I/O port lines generating at a particular frequency.
- The 8051 trainer kit has three numbers of I/O port connectors, connected with I/O Port lines (P1.0 – P1.7), (P3.0 – P3.7) to rotate the stepper motor.
- LS293D is used as a driver for port I/O lines, drivers output connected to stepper motor, connector provided for external power supply if needed.



PIN A





Clockwise	Step #	Winding A	Winding B	Winding C	Winding D	Counter-clockwise
↓	1	1	0	0	0	↑
	2	0	1	0	0	
	3	0	0	1	0	
	4	0	0	0	1	

By giving the excitation as indicated above through port 1 we can rotate stepper motor in clockwise or anticlockwise direction.

NOTE: To turn the motor in the reverse direction enter as (RL A instead of RR A). The schematic sections given is, stepper motor connected to port 1 and the sample program is given based on 8255.

ADDRESS	OP CODE	MNEMONICS	COMMENTS
8500	74 80	MOV A, #80H	Initialize 80HEX value to Accumulator
8502	90 40 03	MOV DPTR, #4003	Move 4003 to R1 register
8505	F0	MOVX @DPTR, A	Store ACC value into R1 register
8506	90 40 00	MOV DPTR, #4000	Move 4000 to R1 register
8509	74 66	MOV A, #66H	Move 66HEX value to Accumulator
850B	F0	AGAIN:MOVX@DPTR, A	In AGAIN, Store ACC value into R1
850C	03	RR A	Rotate sequence for clockwise
850D	B1 11	ACALL DELAY	wait
850F	80 FA	SJMP AGAIN	Short Jump into AGAIN
8511	7D 05	DELAY: MOV R5, #5	In DELAY, Load 5 value into R5
8513	7C 0F	H3: MOV R4, #0F	In H3, Load 0F value into R4 register
8515	7B 43	H2: MOV R3, #43	In H2, Load 43 value into R3 register
8517	DB FE	H1 : DJNZ R3, H1	H1, Decrement R0& check if it's not 0
8519	DC FA	DJNZ R4, H2	Decrement R4& Check if it's not 0
851B	DD F6	DJNZ R5, H3	Decrement R5& Check if it's not 0
851D	22	RET	Return

Example 4: Describe the 8051 connection to the stepper motor of figure shows and code a program to rotate it continuously.

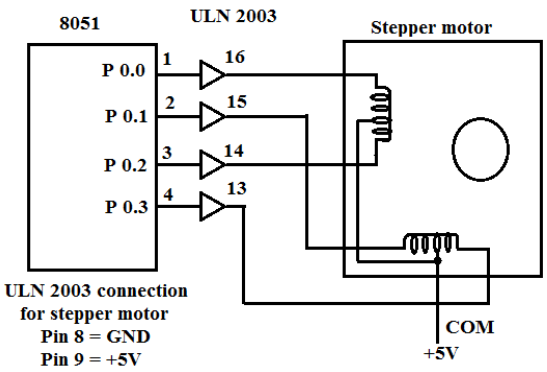


Figure : Interfacing Stepper Motor with 8051

Solution:

	MOV	A, #66H	Load step sequence
BACK:	MOV	P1, A	Issue sequence to motor
	RR	A	Rotate right clockwise
	ACALL	DELAY	Wait
	SJMP	BACK	Keep going
	…….		
DELAY			
	MOV	R2, #100H	
H1:	MOV	R3, #255H	H1:
H2:	DJNZ	R3, H2	H2:
	DJNZ	R2, H1	
	RET		

****

Describe with a neat diagram, the washing machine control using 8051 microcontroller. [April/May 2015]

Explain the working of a washing machine and how it is controlled by the 8051 controller. [Nov/Dec 2015, May/June 2014, Nov/Dec 2014, Dec 2016]

****

4. WASHING MACHINE CONTROL

INTRODUCTION:

- Washing machine consists of a washing basket that can rotate.
- In the centre of the basket is a cylindrical vertical column called Agitator.
- The Agitator can also move independently.
- The water, detergent and cloths are put in the washing basket.
- During washing, the agitator and the washing basket rotate in opposite directions in small steps.

- Due to this action, the clothes get washed.

1.Input Settings

There are four knobs for programming the washing machine.

1) Load select

- ✓ Load means the number of clothes intended to be washed together.
- ✓ There are three settings (high, medium and low).
- ✓ Based on the load selected, the machine decides the amount of water required.

2) Water Inlet Select

- ✓ Machine can take either hot, tap or mix water.
- ✓ There are two inlet pipes on the machine for hot and tap water.
- ✓ The knob setting "mix" allows 50% tap and 50% hot water as input.

3) Modes

Through this knob, the machine can be operated normal or save mode.

(i) Normal mode

1. The clothes are washed.
2. The detergent is drained.
3. The fresh water is put.
4. The clothes are rinsed.
5. The water is drained.
6. Using spin, the moisture from clothes is taken out.

(ii) Save mode

The save mode has been designed to save detergent, and is used when clothes need to be washed in a number of lots.

4) Program Select

- ✓ Using this knob, the machine is programmed to wash the clothes of different kinds.
- ✓ The various settings are Extra Heavy, Heavy, Normal, Light and Delicate.

2. Indications

1. Machine ON: There is an LED indication which glows when the machine is ON.
2. Washing Complete: A sound is generated to announce that the washing is complete.

3. Washing Cycle

Different operations performed by the machine in a typical wash cycles are

- ✓ **Fill**

- ✓ **Agitate**
- ✓ **Soak**
- ✓ **Drain**
- ✓ **Spin**

➢ **Fill:**
- Water is filled through the inlet.
- The quantity of water depends on the load setting (high, medium or low).
- In the first fill, water temperature is decided by the setting tap, hot or mix.
- In the second fill, after drain and spin, only tap water is filled for rising the clothes.

➢ **Agitate:**
- In this operation, the wash basket rotates in small steps.
- After every step, it waits for some second.
- Simultaneously, the agitator rotates in the opposite direction in small steps and after every step, there is wait state.

➢ **Soak:**
- The operation is used to allow the clothes to soak the detergent.
- The machine operation basically stops for a specified time period.

➢ **Drain:**
- All the water and detergent are taken out through the drain pipe.

➢ **Spin:**
- In this operation, the agitator does not move.
- The wash basket is rotated at high speed and most of the moisture from clothes is taken out through holes in the inner metallic basket.

4. <u>Control System Design</u>
- With the above knowledge about the operation of the washing machine, consider 8051 microcontroller based washing machine.

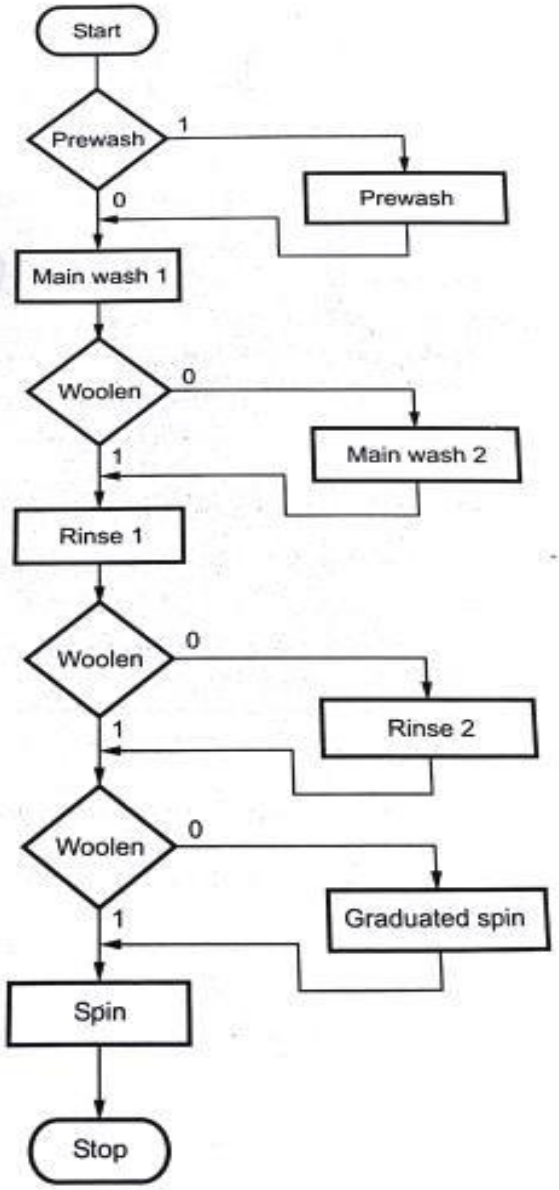
The various controls are:
- ❖ Inlet control of water
- ❖ Water quantity control
- ❖ Agitator control
- ❖ Spin control
- ❖ Drain control
- ❖ Program control
- ❖ Water inlet select
- ❖ Load select

<u>Inputs & Output port assignments:</u>

Pin Name	Pin
Start	P0.0
Prewash Input	P0.1
Cloth Types : 0 : Cotton 1 : Woolen	P0.2

Pin Name	Pin
Prewash Output	P1.0
Main Wash 1	P1.1
Main Wash 2	P1.2
Rinse 1	P1.3
Rinse 2	P1.4
Graduated Spin	P1.5
Spin	P1.6

FLOW CHART FOR WASHING MACHINE:



WASHING MACHINE INTERFACING USING 8051
The various indications are:

- ❖ Machine on indication (LED)
- ❖ Washing complete (LED + BUZZER)
- All the ports of 8051 can be used for input – output operations.
- Agitator control requires controlling of both stepper motors 1 and 2.
- Hence eight lines will be required for this purpose.

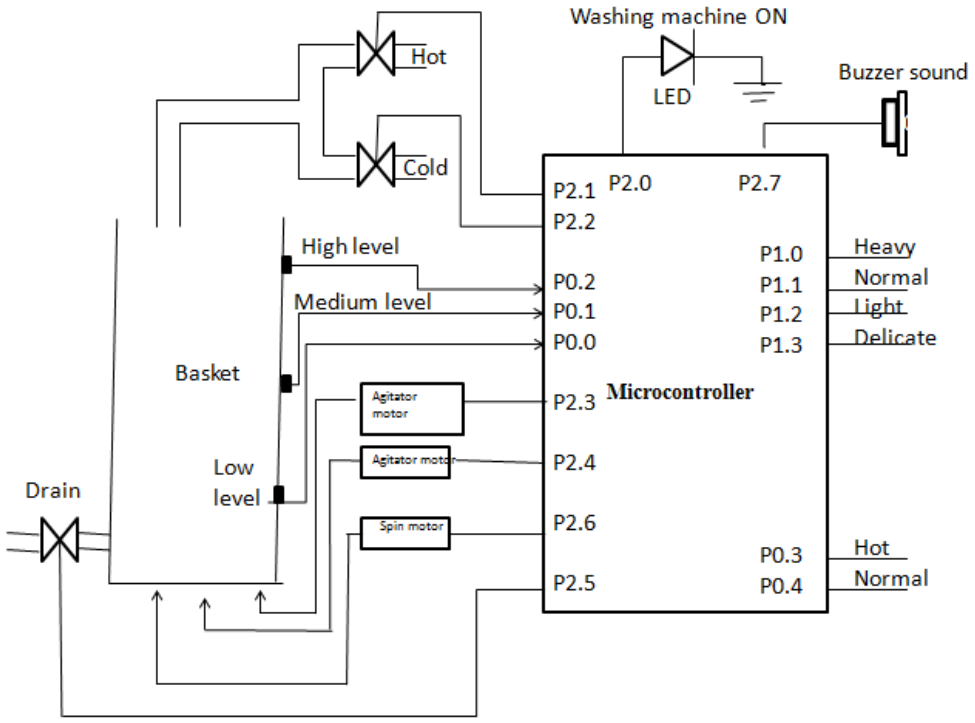


Figure: Washing Machine control Circuit using 8051

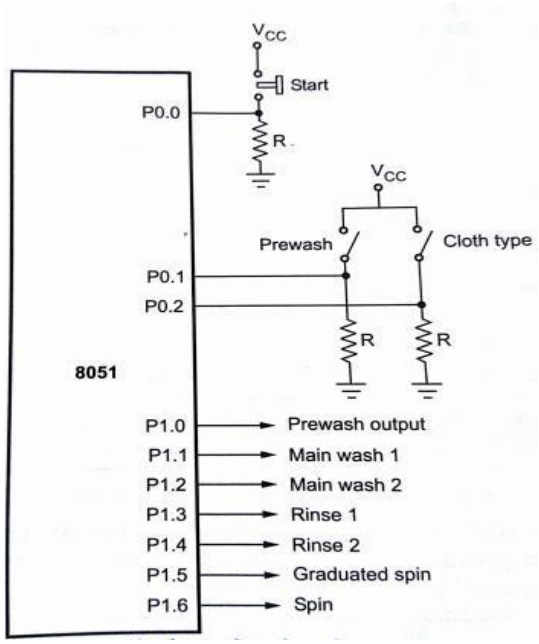


Figure: Hardware interfacing using 8051

PROGRAM:

```
SMRT:   JNB P0.0, START          ; check for star
        JNB P0.1, SKIPW          ; check if prewash is activated
        SETB P1.0                ; if yes do prewash CALL D_PREWASH
                                 ; wait for prewash
```

```
          CLRB P1.0          ; stop prewash
SKIPW:    SETB P1.1          ; Do Main wash 1
          CALL D_MAINWASH1   ; Wait for main wash 1
          CLRB P1.1          ; stop main wash 1
          JNB P0.2, SKIPMW2  ; check if cloth type is cotton
          SETB P1.2          ; if yes do main wash
          CALL D_MAINWASH2   ; Wait for main wash2
          CLRB P1.2          ; stop main wash 2
SKIPMW2:  SETB P1.3          ; Do rinse 1
          CALL D_RINSE1      ; wait for rinse 1
          CLRB P1.3          ; stop rinse 1
          JNB P0.2, SKIPRINSE2 ; Check for cloth type is cotton
          SETB P1.4          ; If yes do rinse 2
          CALL D_RINSE2      ; Wait for rinse 2
          CLRB P1.4          ; stop rinse 2
          JNB P0.2, SKIPGS   ; check for cloth type is cotton
          SETB P1.5          ; If yes do gradual spin
          CALL D_GS          ; wait for gradual spin
          CLRB P1.5          ; stop gradual spin
SKIPGS:   SETB P1.6          ; Do spin
          CALL D_SPIN        ; wait for spin
          CLRB P1.6          ; Stop Spin
          LJMP START         ; Go to start
```

****

Explain the closed loop control of a servo motor using 8051 with a neat diagram.[April/May 2017, May/June 2016, Nov/Dec 2014, May/June 2013,Nov/Dec 2015][December2017]
Explain the servo motor using 8051 microcontroller. [April/May 2011]

****

5.*CLOSED LOOP CONTROL OF SERVO MOTOR*

INTRODUCTION:

- ✓ Servo motor is based on servo mechanism and it is mainly used for position control.
- ✓ A servo system mainly consists of three basic components –
    - A controlled device
    - Output sensor
    - Feedback system.
- ✓ This is an automatic closed loop control system.
- ✓ Here instead of controlling a device by applying the variable input signal, the device is controlled by a

feedback signal generated by comparing output signal and reference input signal

- ✓ The servo motor is most commonly used in the industrial application like automation technology.
- ✓ It is a self contained electrical device that rotates parts of a machine with high efficiency and great precision.
- ✓ The output shaft of this motor can be moved to a particular angle.
- ✓ Servo motors are mainly used in home electronics, toys, cars, airplanes, etc

TYPES OF SERVO MOTOR:

- ✓ Servo motors are classified into different types based on their application, such as
    - AC servo motor
    - DC servo motor
    - Brushless DC servo motor
    - Positional rotation
    - Continuous rotation
    - Linear servo motor etc.
- ✓ Typical servo motors comprise of three wires namely, power control and ground.
- ✓ The shape and size of these motors depend on their applications. DC SERVO MOTOR.
- ✓ The motor which is used as a DC servo motor generally have a separate DC source in the field of winding & armature winding.
- ✓ DC servo motor provides very accurate and also fast respond to start or stop command signals due to the low armature inductive reactance.
- ✓ DC servo motors are used in similar equipments and computerized numerically controlled machines.

AC SERVO MOTOR:

- ✓ AC servo motor is an AC motor that includes encoder is used with controllers for giving closed loop control and feedback.
- ✓ This motor can be placed to high accuracy and also controlled precisely as compulsory for the applications.
- ✓ Applications of an AC motor mainly involve in automation, robotics, CNC machinery, and other applications a high level of precision and needful versatility.

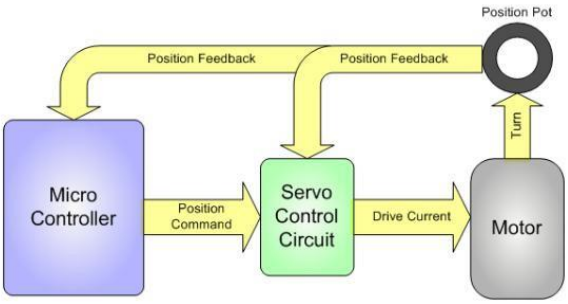


Figure : Feedback signal to the microcontroller

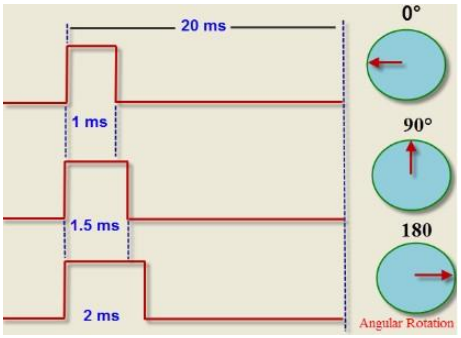


Figure : Controlling a servo motor with angle rotations.

POSITIONAL ROTATION SERVO MOTOR

- ✓ Positional rotation servo motor is a most common type of servo motor.
- ✓ These common servos involve in radio controlled water, radio controlled cars, aircraft, robots, toys and many other applications.

CONTINUOUS ROTATION SERVO MOTOR

- ✓ Continuous rotation servo motor is quite related to the common positional rotation servo motor, but it can go in any direction indefinitely.
- ✓ This type of motor is used in a radar dish if you are riding one on a robot or you can use one as a drive motor on a mobile robot.

ADVANTAGES OF SERVO MOTOR

- ✓ The servo motor is small and efficient.
- ✓ High-speed operation is possible by the servo motors.

APPLICATIONS OF SERVO MOTOR

- ✓ The applications of servo motors mainly involve in computers, robotics, toys, CD/DVD players, etc.

DC SERVO MOTOR INTERFACING WITH THE MICRO CONTROLLER

- ✓ The actual position of the motor is sensed with the sensor and it is compared with the desired position.
- ✓ The difference between the actual and desired position, the motor rotates either in clockwise direction or anticlockwise direction.
- ✓ Thus the position of the rotor is controlled by the controller output.

FLOWCHART:

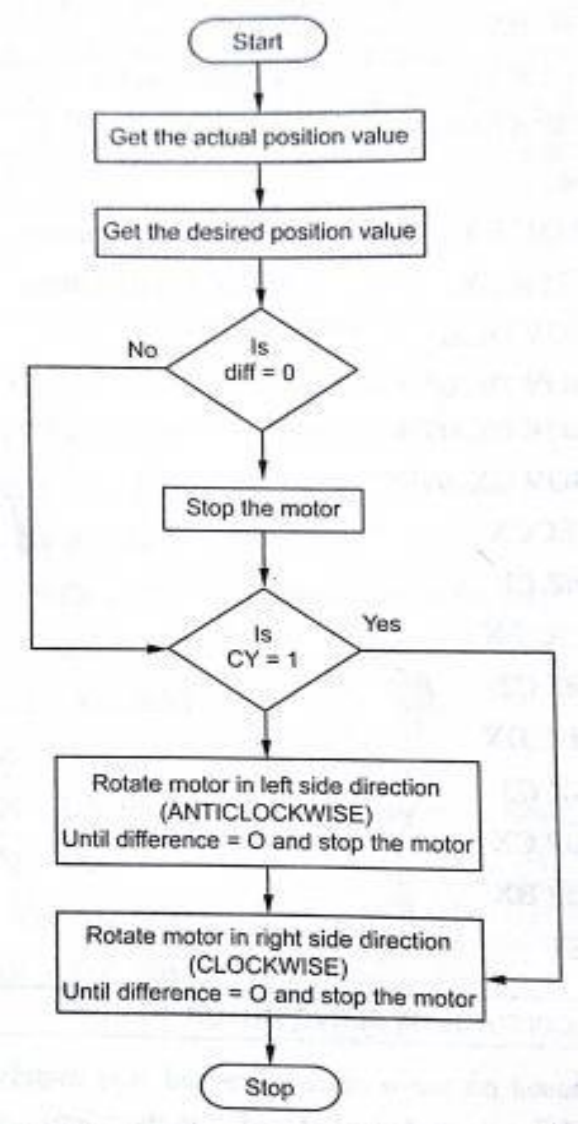


Figure : Flow chart for closed loop control of DC servo motor

DC SERVO MOTOR INTERFACING WITH 8051.

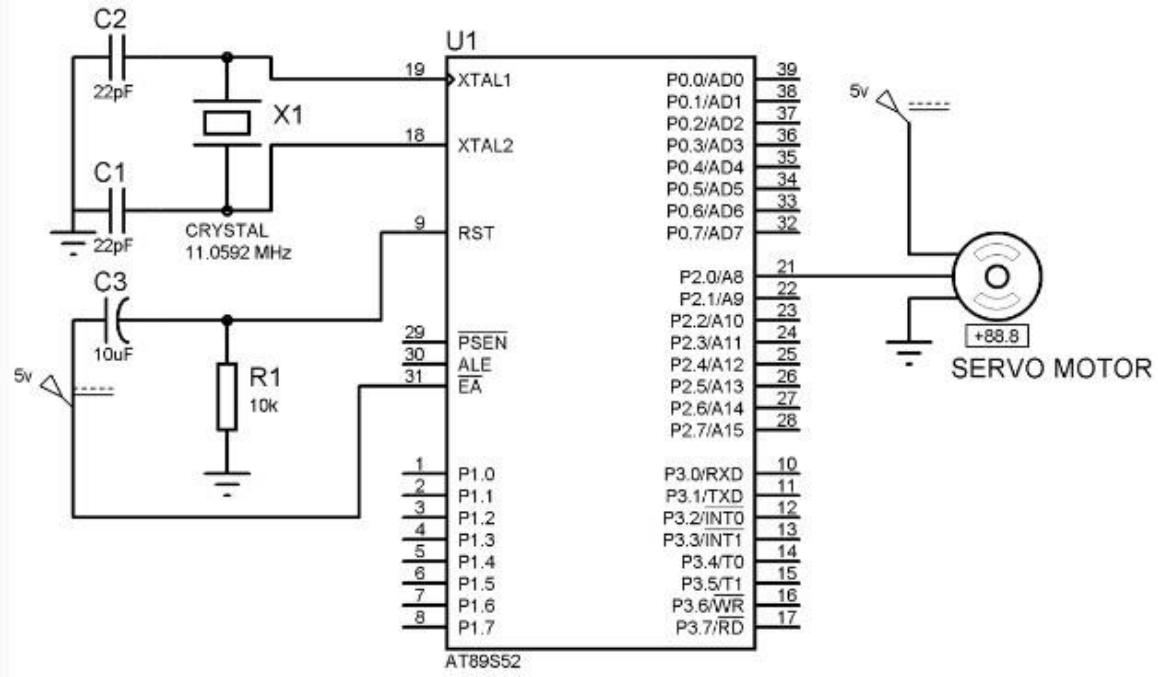


Figure: DC Servo Motor Interfacing With 8051

PROGRAM:

Label	Mnemonics	Operand	Comments
	MOV	A, P0	Get the actual position value from the input port P0.
	MOV	R0 , #data	Get the desired position value in R0 of selected register bank.
	SUBB	A, R0	Find the difference between actual and desired position.
	MOV	R1, A	Store the result in R1 of selected register bank.
	JZ	L1	If the result is zero, make the motor to stop.
	JC	RIGHT	If the magnitude of desired position is high, go to RIGHT.
LEFT:	RLC	A	
	DNZ	R1, LEFT	Or else make the motor to rotate left until the desired position is reached.
L1:	MOV	P1, 00H	Make the rotor to stop rotating.
RIGHT:	RRC		A make the motor to rotate right until the desired position is reached.
	DNZ	R1, RIGHT	
	SJMP	L1	Make the rotor to stop rotating
	RET		

SAMPLE PROGRAMS: (8051 Microcontroller)

1. Add two 8-bit numbers

```
MOV A, #30H        ; (A) 30
ADD A, #50H        ; (A)  →     (A) + 50H
```

2. Add two 16- bit numbers

```
MOV DPTR, #2040H; (DPTR)←  2040H (16 bit number)
MOV A, #2BH        ;(A)  ←2BH (lower byte of second 16 bit number)
MOV B, #20H        ; (B)  ←  20H (Higher byte of second 16 bit number)
ADD A, DPL         ; Add lower bytes
MOV DPL, A         ; Save result of lower byte addition
MOV A, B           ; Get higher byte of second number in A
ADD A, DPH         ; Add higher bytes with any carry from lower byte addition
MOV DPH, A         ; Save result of higher byte addition
```

3. Division two 8-bit numbers

```
MOV A, #90         ; Get the first number in A MOV B, #20    ; Get the second number in B
DIV A, B           ; A+B, Remainder in B and Quotient in A
```

4. To find the sum of 10 numbers stored in the array. (June 2016)

Statement: Calculate the sum of series of numbers. The length of the series is in memory location 2200H and the series itself begins from memory location 2201H.

a. Assumes the sum to be 8-bit number so you can ignore carries. Store the sum at memory location 2300H.
b. Assume the sum to be 16-bit number. Store the sum at memory locations 2300H and 2301H.

Sample program

2200H = 04H
2201H = 20H
2202H = 15H
2203H = 13H
2204H = 22H
Result = 20+15+13+22=6AH 2300H=6AH

Program

```
MOV DPTR, #2200H        ; Initialize memory
pointer MOVX A, @DPTR   ; Get the count
MOV R0, A               ; Initialize the iteration counter
INC DPTR                ; Initialize pointer to array of numbers
MOV R1, #00             ; Result = 0
BACK: MOVX A, @DPTR     ; get the umber
ADD A, R1               ; A← Result + A
MOV R1, A               ; Result ← A
INC DPTR                ; Increment the array pointer
DJNZ R0, BACK           ; Decrement iteration count if not zero repeat
MOV DPRT, #2300H        ; Initialize memory pointer
MOV A, R1               ; Get the result
MOVX @DPTR, A           ; Store the result
```

Sample program

2200H = 04H
2201H = 9AH
2202H = 52H
2203H = 89H
2204H = 3EH
Result = 9AH + 52H + 89H + 3EH = 6AH
2300H=B3H Lower byte
2301H = 01H Higher byte

Program

```
MOV DPTR, # 2200H       ; Initialize memory pointer
MOVX A, @DPTR           ; Get the count
MOV R0, A               ; Initialize the iteration counter
INC DPTR                ; Initialize pointer to array of numbers
MOV R2, #00             ; [Make result R2= 00H]
MOV R1, #00             ; [Make result R1= 00H]
BACK: MOVX A, @DPTR     ; get the number
ADD A, R1               ; A← Result + A
MOV R1, A               ; Result ← A
ADDC R2, #00            ; if carry exists, add it to MSD
INC DPTR                ; Increment the array pointer
DJNZ R0, BACK           ; Decrement iteration count if not zero repeat
MOV DPTR, #2300H        ; Initialize memory pointer
MOV A, R1               ; Get the lower byte of result
MOVX @DPTR, A           ; Store the lower byte of result
```

```
INC DPTR          ; Increment memory pointer
MOV A, R2         ; Get the higher byte of result
MOVX @DPTR, A     ; Store the higher byte of result
```

****

Design a microcontroller based water level control system in detail. (April 2018)

****

6. Water Level Controller using 8051 Circuit Principle

- This system mainly works on a principle that "water conducts electricity". The four wires which are dipped into the tank will indicate the different water levels. Based on the outputs of these wires, microcontroller displays water level on LCD as well as controls the motor.

- Initially when the tank is empty, LCD will display the message LOW and motor runs automatically. When water level reaches to half level, now LCD displays HALF and still motor runs.

- When the tank is full, LCD displays FULL and motor automatically stops. Again, the motor runs when water level in the tank becomes LOW.

- The water level probes are connected to the P0.0, P0.1 and P0.2 through the transistors (they are connected to the base of the transistors through corresponding current limiting resistors). P0.0 for LOW level, P0.1 for HALF Level and P0.2 for HIGH Level.

- The Collector terminals of the Transistors are connected to VCC and the Emitter terminals are connected to PORT0 terminals (P0.0, P0.1 and P0.2).

- PORT1 of the microcontroller is connected to the data pins of LCD and the control pins RS, RW and EN of the LCD Display are connected to the P3.6,
- GND and P3.7 respectively.

Algorithm for Water Level Controller Circuit

- First configure the controller pins P0.0, P0.1 and P0.2 as inputs and P0.7 as output.
- Now, initialize the LCD.
- Continuously check the water level input pins P0.0, P0.1 and P0.2.
- If all the pins are low, then display tank as "EMPTY" on the LCD and make P0.7 pin HIGH to run the motor automatically.

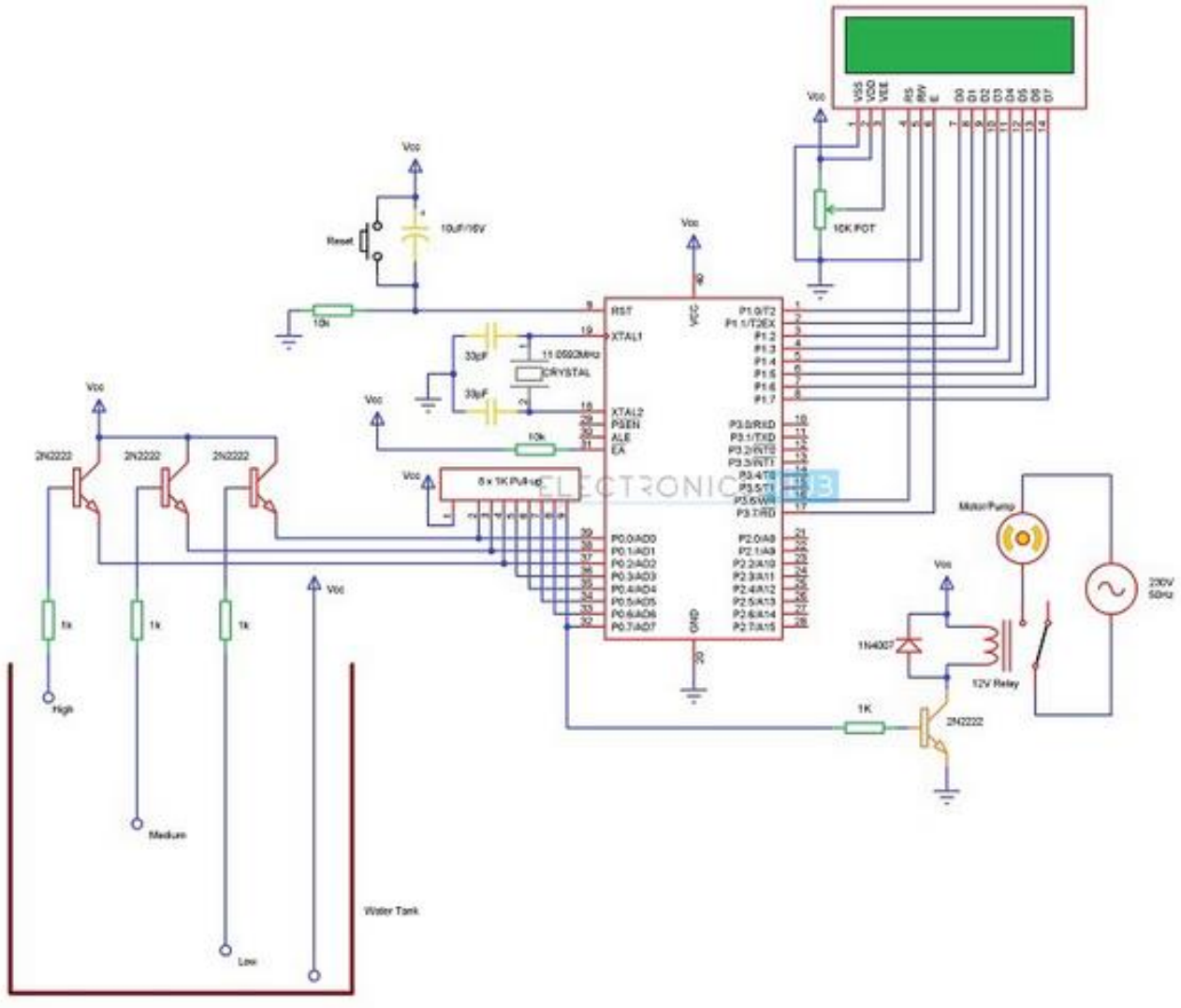


Figure: Water Level Controller using 8051

- If the level is low i.e. if P0.0 is HIGH, display the water level as "LOW" and continue to run the motor.
- A HIGH pulse on the pin P0.1 indicates that water has reached half level. So, display the same thing on LCD and run the motor normally.

- If P0.2 is HIGH, then the water level in the tank is FULL.
- Now, make the P0.7 pin as LOW to turn off the motor automatically

****

Develop a 8051 ALP to evaluate an arithmetic expression (A–B) XC Where A,B,C are8 bit data in internal memory.Assme A>B and store the result in external memory.Explain the program developed.(Dec 2018)

```
MOV A,#DATA1
MOV B,DATA2
MOV C,#DATA3
```

```
        SUBB A,B
        JC  L1
        MOV B,C
        MUL AB
LI      MOV DPTR,#4500
        MOV @DPTR,A
 L2     SJMP    L2
```