SREENIVASA INSTITUTE OF TECHNOLOGY AND MANAGEMENT STUDIES
(Autonomous)
DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
(Accredited by NBA)

III B.TECH.-V SEMESTER

MICROPROCESSORS AND MICROCONTROLLERS (23ECE354T)

COURSE OBJECTIVES:
1. To learn the fundamental architectural concepts of microprocessors.
2. To gain knowledge about assembly language programming concepts.
3. To get familiar about 8086 interfacing.
4. To understand the fundamentals of the 8051 Microcontroller.
5. To learn interfacing with the 8051 Microcontroller.

UNIT I - 8086 ARCHITECTURE : 8086 Architecture: Main features, pin diagram/description, 8086 microprocessor family, internal architecture, bus interfacing unit, execution unit, interrupts and interrupt response, 8086 system timing, minimum mode and maximum mode configuration.

UNIT II - 8086 PROGRAMMING
8086 Programming: Program development steps, instructions, addressing modes, assembler directives, writing simple programs with an assembler, assembly language program development tool

UNIT III - 8086 INTERFACING
8086 Interfacing: Semiconductor memories interfacing (RAM, ROM), Intel 8255 programmable peripheral interface, Interfacing switches and LEDS, Interfacing seven segment displays, software and hardware interrupt applications, Intel 8251 USART architecture and interfacing, Intel 8237a DMA controller, stepper motor, A/D and D/A converters, Need for 8259 programmable interrupt controller

UNIT IV – MICROCONTROLLER
Microcontroller: Architecture of 8051 – Special Function Registers (SFRs) - I/O Pins Ports and Circuits - Instruction set - Addressing modes - Assembly language programming.

UNIT V - INTERFACING MICROCONTROLLER
Interfacing Microcontroller: Programming 8051 Timers - Serial Port Programming - Interrupts Programming – LCD & Keyboard Interfacing - ADC, DAC & Sensor Interfacing - External Memory Interface- Stepper Motor and Waveform generation - Comparison of Microprocessor, Microcontroller, PIC and ARM processors

TEXT BOOKS:

1. Microprocessors and Interfacing – Programming and Hardware by Douglas V Hall, SSSP Rao, Tata McGraw Hill Education Private Limited, 3rdEdition,1994.
2. K M Bhurchandi, A K Ray, Advanced Microprocessors and Peripherals, 3rd edition, McGraw Hill Education, 2017.
3. Raj Kamal, Microcontrollers: Architecture, Programming, Interfacing and System Design, 2nd edition, Pearson, 2012.

REFERENCE BOOKS:

1. Ramesh S Gaonkar, Microprocessor Architecture Programming and Applications with the 8085, 6th edition, Penram International Publishing, 2013.
2. Kenneth J. Ayala, The 8051 Microcontroller, 3rd edition, Cengage Learning, 2004.

UNIT-1 : OVERVIEW OF 8086 µP

1. Introduction to 8086
2. Architecture of 8086
3. Memory segmentation
4. Registers of 8086
5. Pin configuration of 8086
6. Min. mode operation with Timing diagram
7. Max. mode operation with Timing diagramPhysical memory organization and Memory banks accessing
8. Interrupts of 8086 and Interrupt Vector Table (IVT)

1.1.INTRODUCTION TO 8086

The Intel released 8086 microprocessor in 1978, which is fabricated using HMOS technology. The features of Intel 8086 are given below

- The 8086 is a 16-bit microprocessor
- It has 16-bit ALU
- It has 16-bit data bus and 20-bit address bus
- It can address 2^{20} locations i.e., 1 M Bytes of memory
- It uses memory segmentation
- It uses 2-stage pipeline concept – Fetch and Execute
- It has 6-bytes of Instruction Queue
- It is packaged in 40-pin DIP
- It is available in three clock rates : 5 MHz, 10 MHz and 8 MHz
- It can be operated in two modes: *Min. mode and Max. Mode*
- The 8086 is operated in Minimum mode when only one processor is to be used in a microcomputer system. The Maximum mode is used when more than one processor are used in a system.

Note: The architecture of 8088 is similar to 8086 except for two changes:

- The 8088 has 4-byte instruction queue where as 8086 has 6-byte instruction queue.
- The 8088 has 8-bit data bus where as 8086 has 16-bit data bus
- There is no $\overline{BHE}$ signal in 8088

1.2. ARCHITECTURE OF 8086

The complete architecture of 8086 is divided into two parts

(A) Bus Interface Unit (BIU) and

(B) Execution Unit (EU).

The Bus Interface Unit is responsible for all read and write operations of Memory and I/O. It sends out 20-bit Physical address, fetches instruction code bytes from memory and stores them in Instruction Queue.

The Execution unit is responsible for decoding and executing the instructions. It receives instruction code bytes from Instruction Queue in BIU, decodes and executes them.

These two parts are operated in parallel for implementing pipelining concept and to increase the execution speed.

1.3.1. Bus interface unit (BIU) :

The BIU handles all transfer of address, data and code on Buses. The BIU consists of

(i) Instruction Queue

(ii) Segment Registers

(iii) Instruction Pointer

(iv) 20-bit Physical address calculation circuit

(i) Instruction Queue:

The pipelining concept is implemented in 8086 using an Instruction Queue. While Execution Unit (EU) is executing an instruction, the BIU will fetch upto 6- instruction code bytes from memory and stores them in a FIFO group of registers called as Instruction Queue. When EU is ready for its next instruction, it simply takes instruction bytes from Instruction Queue in BIU. This is much faster than sending out an address to memory, waiting for memory to send-back the next instruction bytes.

Note that the Instruction Queue fails in the case of branch instructions execution (Ex: JMP, CALL, RET, etc). After the execution of branch instructions, the program execution jumps to different location. Hence, the instructions available in the Instruction Queue will be erased during the execution of branch instructions.

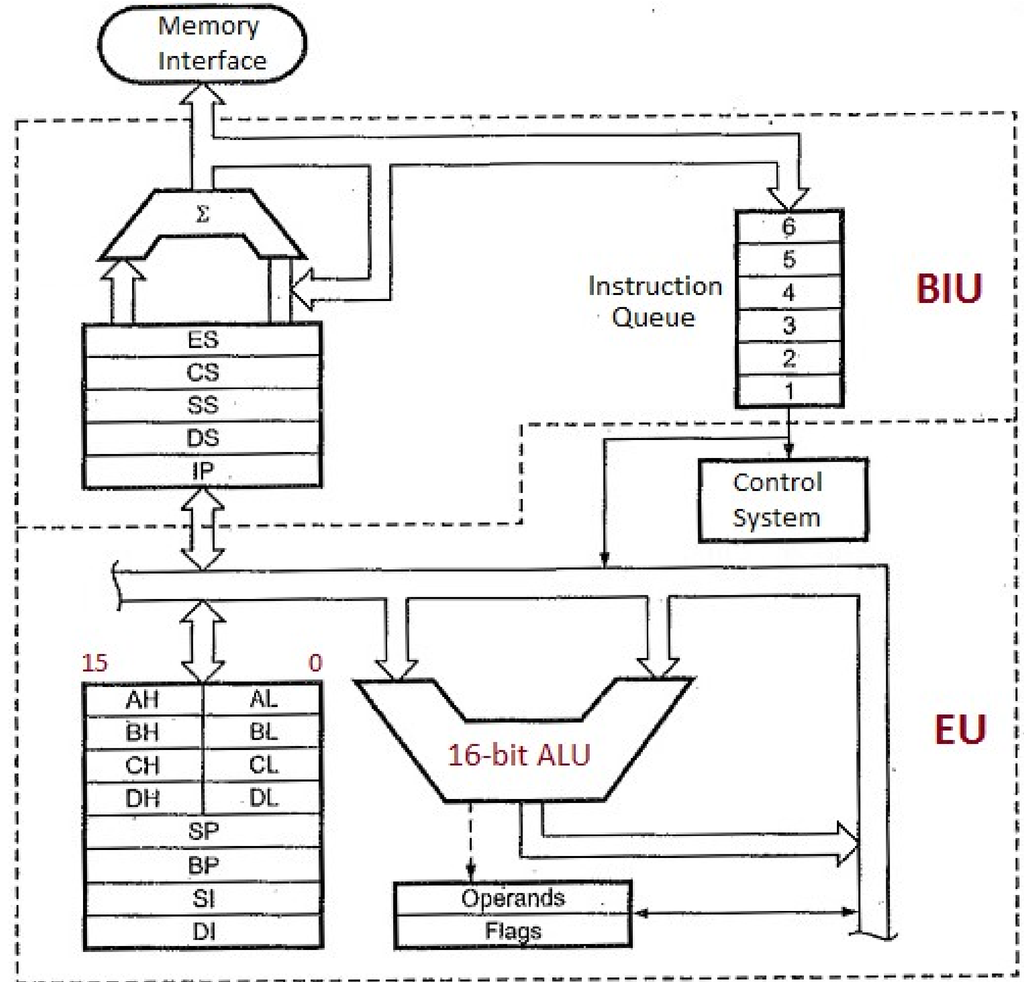


Figure 1.1. Internal architecture and registers of 8086

(ii) Segment Registers

The 8086 uses memory segmentation. In this scheme, the complete 1 MB physical memory is divided into number of logical segments. The size of each segment is 64 KB in size and is addressed by one of the segment registers. The segment registers in BIU are used to define the starting address of logical segments.

The four segment registers in 8086 are

Code segment register (CS) ☾ defines the starting address of Code segment Data segment register (DS) ☾ defines the starting address of Data segment Extra segment register (ES) ☾ defines the starting address of Extra segment Stack segment register (SS) ☾ defines the starting address of Stack segment

The segment registers in BIU hold the upper 16-bits of starting address of logical segments. The BIU inserts ZERO's for lower 4-bits of 20-bit starting address.

For example, if CS=2000 H, then the Code segment starts at 20000H.

(iii) Instruction Pointer (IP) :

The IP register holds the ***offset address*** of the next instruction to be fetched from Code segment. The Offset address is defined as the distance of operand from the Starting address of the Segment.

(iv) 20-bit Physical address calculation circuit

The BIU has a Physical Address Generation Circuit.
It generates the 20-bit physical address by adding Segment base to the Offset address.

The segment register defines the segment base address and a location within a segment can be addressed by *16-bit offset address*.

For example, if CS= 2000H and IP= 5678H then

Segment base = CS*10 = 20000 H

Offset address = IP = 5678 H

20-bit Physical address = Segment base + Offset = 25678 H

Note that the 20-bit Physical address is generally represented in the form of

Segment base : Offset = CS:IP = 2000:5678 H

1.2.2. Execution Unit (EU) :

The Execution unit is responsible for decoding and executing the instructions. It receives instruction code bytes from Instruction Queue in BIU, decodes and executes them.

The EU consists of

(i) 16-bit ALU
(ii) General purpose registers: AX,BX,CX,DX
(iii) Index Registers : SI and DI
(iv) Pointers : SP and BP
(v) Flag Register/Program Status Word (PSW)

(i) 16-bit ALU :

The EU consists of 16-bit ALU which can perform Arithmetic operations (such as Addition, Subtraction, Multiplication, Division, Increment, Decrement, etc) and Logic operations (such as Logic AND, OR, NOT, Ex-OR, Shift, Rotate, etc).

(ii) General purpose registers :

	15	0
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

The 8086 has 4- general purpose registers - AX, BX, CX, DX.
All these registers are 16-bit registers used to store 16-bit data
Each register is divided in two 8-bit registers to store 8-bit data
These registers have multiple functions, shown in following Table.

Register	*Name*	*Purpose*
AX	Accumulator	Used to hold the results of some operations like MUL, DIV, Shift, Rotate…etc
BX	Base Index	Used to hold the offset address (Based addressing)
CX	Counter	Used as a counter in LOOP and String instructions
DX	Data Index (or) Extended Accumulator	Used to hold a part of result from MUL & DIV. It is also used to hold the 16-bit I/O port address during I/O operation.

(iii) Index Registers :

- The Index Registers are used to hold the 16-bit offset address of data stored in Data and Extra segments.
- These registers are used in string operations to hold the offset address of source string and destination strings.
  The SI register holds the offset address of source string in Data Segment.
  The DI register holds the offset address of destination string in Extra Segment.
    - Address of source string ☾ DS:[SI]
    - Address of destination string ☾ ES:[DI]

- The directional flag (DF) selects the increment (or) decrement mode for SI and DI registers during String instructions manipulation.
  DF = 0 selects increment mode and
  DF = 1 selects decrement mode.

(iv) Pointers :

- The Pointers are used to hold the offset address relative to data and stack segments.
- The base pointer (BP) is used to access data in stack segment.
- The stack pointer (SP) is used to hold the address of stack top.

Register	Name	Purpose
SI	Source Index	It is used in string operations to hold the offset address of source string in Data segment.
DI	Destination Index	It is used in string operations to hold the offset address of destination string in Extra segment.
BP	Base Pointer	It is used to access data the stack segment. It is used to hold the offset address (Based addressing)
SP	Stack Pointer	It is used to hold the address of Stack top

(v) Flag Register / Program Status Word (PSW)

The flag register is used to indicate the status information (or) condition produced by an instruction execution.

The 8086 has 6- status flags and 3-control flags.

Conditional Flags / Status Flags :

These flags are Set or Reset according to the condition produced by an instruction execution. The 6- conditional flags of 8086 are CF, PF, AF, ZF, SF, OF

Control flags:

These flags control the operation of the processor.

The 3- control flags of 8086 are DF, IF, TF.

15 - 12	11	10	9	8	7	6	5	4	3	2	1	0
	OF	**DF**	**IF**	**TF**	**SF**	**ZF**		**AF**		**PF**		**CF**

Figure: Flag register of 8086

Carry Flag (CF) : It is set to 1, if a carry is generated in Addition (or) Subtraction

Parity Flag(PF) : It is set to 1, if the result of an operation has even number of 1's

Auxiliary carry Flag (AF) : It is used in BCD operations.
It is set to 1, if addition of lower nibble generates a carry.

Zero Flag (ZF) : It is set to 1, if the result of an operation is ZERO

Sign Flag (SF) : It is used with signed numbers only.
SF=1 for –ve results and SF =0 for +ve results

Overflow Flag (OF) : It is set to 1, if the result of a signed operation exceeds the register capacity.

Directional Flag (DF) :

It is used in string operations to select either increment (or) decrement mode for SI and DI registers. DF = 0 selects increment mode and
DF = 1 selects decrement mode.

Interrupt Flag (IF) :

- If IF = 1, all the maskable interrupts are recognized and processed by the CPU.
- Otherwise, the maskable interrupts are ignored and not processed by CPU.

Trap Flag (TF) :

- It is used for debugging purpose
- If TF = 1, the processor enters the 'Single step execution mode' i.e., the 8086 gets interrupted automatically at the end of every instruction execution.

Problem :

The content of registers AL and BL are 79H and 88H respectively. Determine the status of control flags of 8086, after the execution of the instruction ADD AL,BL?

Ans:

AL = 79 H	=	0 1 1 1 1 0 0 1	⇝	AL
BL = 88 H	=	1 0 0 0 1 0 0 0	⇝	BL
ADD AL, BL	:	0 0 0 0 0 0 0 1	⇝	AL

The status of conditional flags are :

CF = 1 ;
PF = 0;
AF = 1 ;
ZF = 0 ;
SF = 0 ;
OF = 0

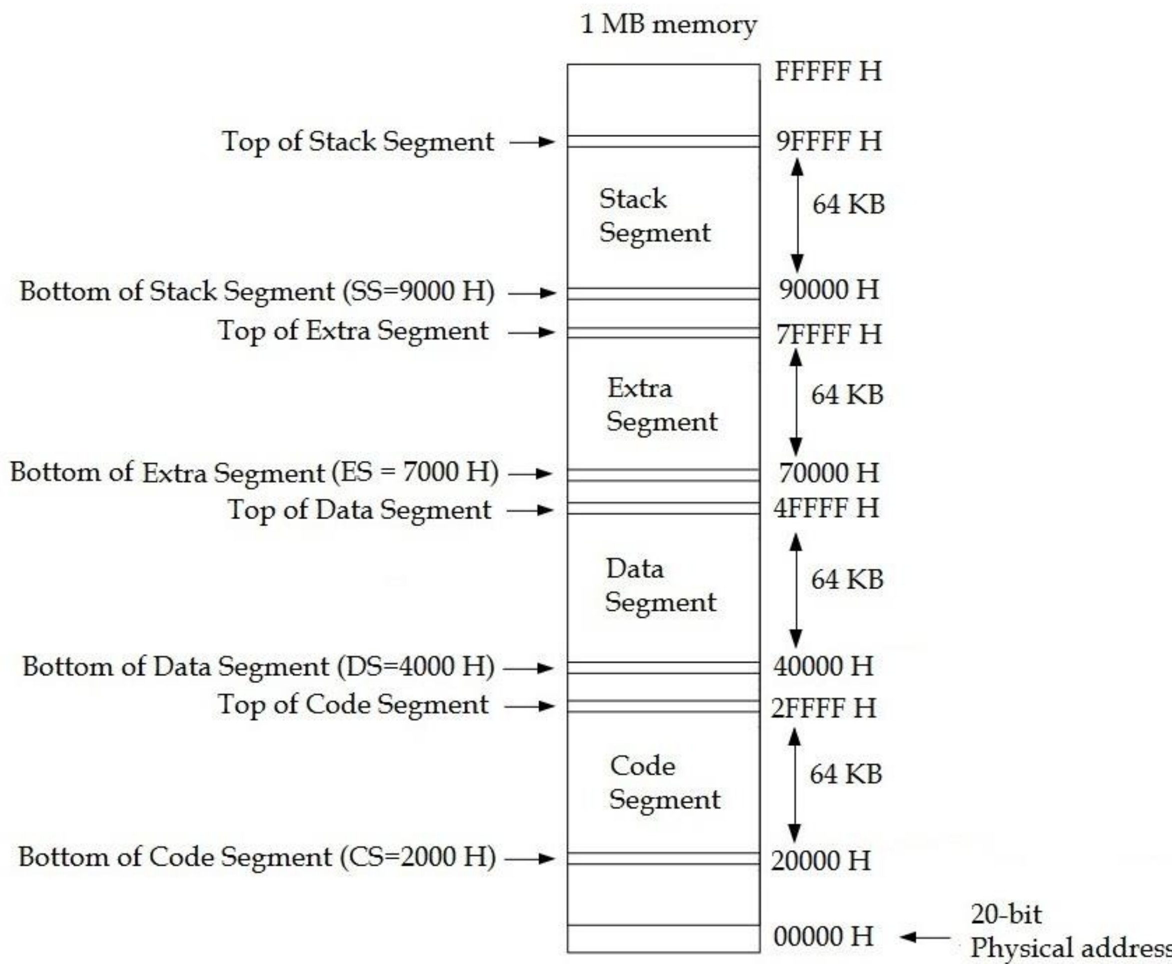
1.3. MEMORY SEGMENTATION

The 8086 uses memory segmentation. In this scheme, the complete 1 MB physical memory is divided into number of logical segments. The size of each segment is 64 KB in size and is addressed by one of the segment registers.

The 64 KB logical segment can be located anywhere in 1 MB memory, but the segment will always start at an address with ZERO's in lowest 4-bits.

Note that the 8086 does not work the whole 1 MB memory at any given time. However it works only with four 64 KB segments within the whole 1MB memory.

The four segment registers in BIU define the starting addresses of the four memory segments with which the 8086 is working at that instant of time. The segment registers in BIU are used to hold the upper 16-bits of starting address of logical segments. The BIU inserts ZERO's for lower 4-bits of 20-bit starting address.



Code Segment :

- The Code segment is used to store the program instruction codes.
- The Code Segment register (CS) is used to hold the upper 16-bits of starting address of the Code segment. The BIU inserts ZERO's for lower 4-bits of 20-bit starting address.
- For example, if CS=2000H then the Code segment starts at 20000H.

Data Segment :

- The Data segment is used to store data variables and constants of the program.
- The Data Segment register (DS) is used to hold the upper 16-bits of starting address of the Data segment.
- Data are accessed from Data segment by an Offset address.
- The SI, DI, BX, BP registers are used to store the offset address for data segment

Extra Segment :

- The Extra segment is an additional data segment.
- It is used in String operations to store the destination string.
- The Extra Segment register (ES) is used to hold the upper 16-bits of starting address of the Extra segment.
- The SI, DI, BX, BP registers are used to store the offset address for data segment

Stack Segment :

- The Stack segment defines the area of memory used for stack.
- Stack is a section of memory where the data are accessed in LIFO manner
- The stack memory is used to store data, address and status information. It is used by CPU to store return address during the execution of procedures and ISRs.
- The Stack Segment register (SS) is used to hold the upper 16-bits of starting address of the Stack segment.
- The Stack Pointer (SP) register holds the address of Stack top.

Addressing a location within a segment (20-bit Physical address calculation) :

The BIU has a Physical Address Generation Circuit.
It generates the 20-bit physical address by adding Segment base to the Offset address.

The segment register defines the segment base address and
a location within a segment can be addressed by *16-bit offset address* as shown in Figure.

For example, if CS= 2000H and IP= 5678H then

Segment base = CS*10 = 20000 H
Offset address = IP = 5678 H

20-bit Physical address = Segment base + Offset = 25678 H

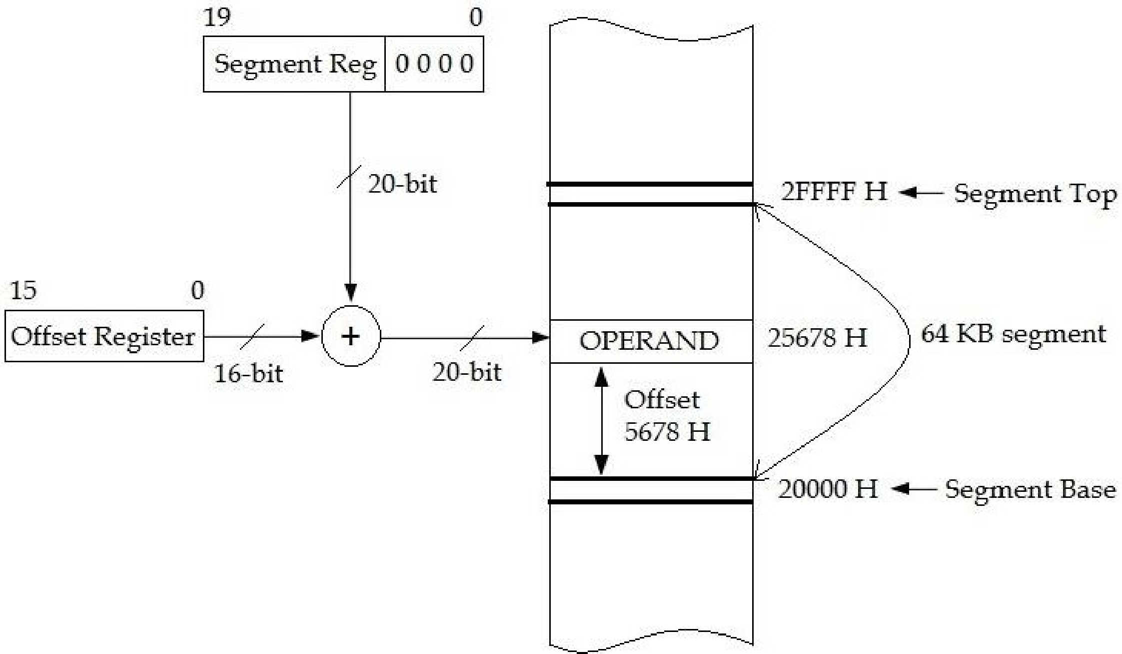


Figure : 20-bit Physical address calculation

The segment register holds the upper 16-bits of starting address of logical segments. The BIU inserts ZERO's for lower 4-bits of 20-bit starting address.

The Offset address is defined as the distance of operand from the Starting address of the Segment. The registers used to store the 16-bit offset addresses for various segments are given in Table.

Segment	***Segment Register***	***Offset Register***
Code Segment	CS	IP
Data Segment	DS	SI (DI/BX/BP)
Extra Segment	ES	DI (SI/BX/BP)
Stack Segment	SS	BP /SP

Advantages of Memory Segmentation :

- Allows to access 1 MB memory with 16-bit address.
- Allows Separate memory areas for program, data and stack
- Provides data and code protection
- Provision for relocation of programs and data
- Provides a powerful memory management mechanism
- Allows the processor to access the data from memory easily and fastly, which increases the speed of operation

1.4. REGISTER ORGANIZATION OF 8086

The Intel 8086 has a powerful set of registers as shown in figure.
The description of all registers is given in section 1.2

	15	0
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

General purpose registers

15 0
CS
DS
ES
SS

Segment Registers

15 0
IP
SI
DI
BP
SP

Pointers and
Index Registers

Flag Register

Figure: Register organisation of 8086

1.5. PIN CONFIGURATION OF 8086 :

The pin diagram of 8086 is shown in Figure.
The 8086 consists of 40-pins.
Among these, 21 are multiplexed pins to reduce the number of pins AD_0 – AD_{15}
A_{16}/S_3 - A_{19}/S_6 and
$\overline{BHE}$ / S_7

The 8086 issues two different sets of signals (Pins 24 to 31) in Min. mode and Max. mode.

Min. Mode ☾ $\overline{INTA}$, ALE, $\overline{DEN}$, DT/$\overline{R}$, M/$\overline{IO}$, $\overline{WR}$, HLDA, HOLD

Max. Mode ☾ QS1, QS0, $\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$, $\overline{WR}$, RQ/$\overline{GT_1}$, RQ/$\overline{GT_0}$

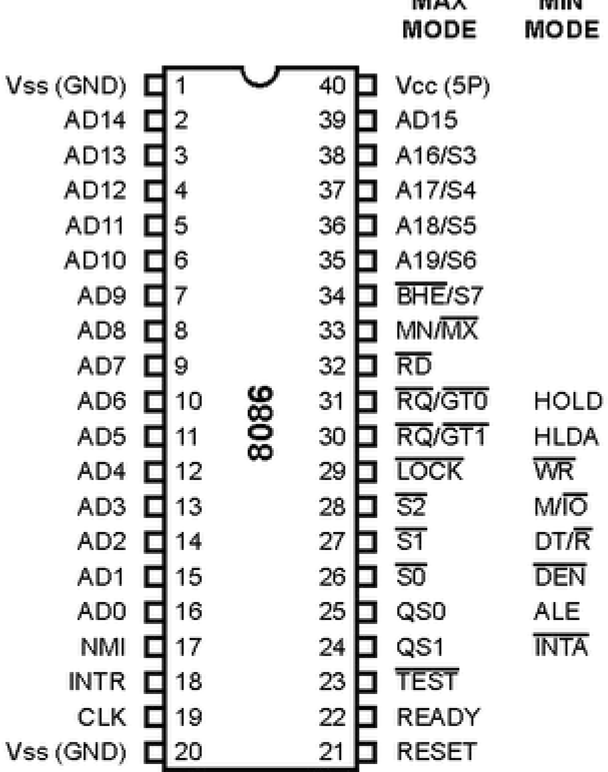


Figure: Pin diagram of 8086

AD_0 - AD_{15} (Address / Data lines) :

- The lower order 16-address lines (A_0 - A_{15}) of 8086 are multiplexed with 16- data lines (D_0 - D_{15}).
- During T_1 state of Bus-cycle, the 8086 sends out address on these lines and during later part of Bus-cycle this multiplexed bus is used as Data bus.
- All the above multiplexed pins are at high-impedance state during DMA operation.

A_{16}/ S_3 - A_{19}/ S_6 (Address/Status lines) :

- The higher order 4-address lines (A_{16} - A_{19}) are multiplexed with Status lines (S_3 - S_6).
- The **S_3 and S_4** shows which segment is accessed during current bus cycle.
- The S_5 indicates the status of IF flag.
- The S_6 is always logic 0 and it is not used.
- All the above multiplexed pins are at high-impedance state during DMA operation.

S4 S3	*Segment*
0 0	Code Segment
0 1	Data Segment
1 0	No Segment
1 1	Extra Segment

$\overline{BHE}$ / S_7 :

- The 1 MB Physical memory of 8086 is divided into two banks **for accessing 16-bit numbers**. Each bank size is 512 K bytes.
- The Bus High Enable $\overline{BHE}$ signal is used to enable the higher order data bus (D_8 - D_{15}) connected to HIGH BANK.
- The status signal S_7 is used by arithmetic co-processor 8087 to determine whether the CPU is 8086 (or) 8088.

$\overline{BHE}$	A_0	*Bank Selected*
0	0	Both banks are enabled for 16-bit operation
0	1	HIGH bank is enabled
1	0	LOW bank is enabled
1	1	No banks are enabled

NMI : Non-Maskable Interrupt:

- It is a positive going **edge triggered** Non-Maskable Interrupt
- It cannot be disabled by using software.
- This interrupt has highest priority than INTR.
- It is a vectored interrupt and the vector address of NMI is 0000:0008 H.

.

INTR: Interrupt Request :

- It is a **level triggered** maskable Interrupt Request.
- It can be disabled by using software i.e., the processor will get interrupted only if IF=1. Otherwise the processor will not get interrupted even INTR is active.
- It is a non-vectored interrupt.

CLK:

- The clock input provides the basic timing for μp and bus control activity.
- The clock frequencies of different versions of 8086 are 5 MHz, 10 MHz, 8 MHz.

RESET :

- It forces all the registers to a predefined values and microprocessor gets reset.
- When Reset is active DS, ES, SS, IP and FLAG registers are initialized to 0000H and CS is initialized to FFFF H.
- After Reset, the processor starts execution from FFFF0 H.

READY:

- A slow peripheral (or) memory device can be connected to microprocessor through READY line. It is used to insert wait states in bus cycles as needed to interface with slow memory & I/O
- It is used by the MPU to sense whether the peripheral is ready to transfer data or not

  If READY =1, peripheral is ready to transfer data.

  If READY=0, the processor WAITS until it goes to HIGH

$\overline{TEST}$:

- This input is examined by WAIT instruction.
- The 8086 enters into WAIT state after the execution of WAIT instruction. If

  $\overline{TEST}$ = 0, the WAIT instruction functions as NOP

  If $\overline{TEST}$ =1, the processor waits until $\overline{TEST}$ = 0.
- If the co-processor has finished its work then it makes $\overline{TEST}$= 0.

$\overline{RD}$:

The **Read Control Signal** is active whenever the processor is ready to read data from Memory (or) I/O.

MN/ $\overline{MX}$:

OPERATING MODES OF 8086/8088:

- The 8086 can be operated in two modes – Minimum mode and Maximum mode.
- The pin MN/$\overline{MX}$ is used to select the Min. (or) Max. mode of operation.

  MN/$\overline{MX}$ = 1 for Minimum mode operation

  MN/$\overline{MX}$ = 0 for Maximum mode operation

  - The pins (24-31) issues two different set of signals – for minimum & maximum mode operations. *These pins are described below:*

Note : The Min.mode pins and Max.mode pins are described in sections 1.6.and 1.7.

Minimum mode:

- The min. mode operation is selected by connecting the pin MN/MX to + 5 V
- The 8086 is operated in minimum mode in simple systems with a single CPU
- In min. mode operation, all the control signals are generated by CPU
- It is least expensive way to operate and the operation is similar to 8085A.
- Min. Mode signals ☾ $\overline{INTA}$, ALE, $\overline{DEN}$, DT/$\overline{R}$, M/$\overline{IO}$, $\overline{WR}$, HLDA, HOLD

Maximum mode:

- The max. mode operation is selected by connecting the pin MN/MX to GND
- The 8086 is operated in maximum mode in multi-processor system with more than one processor.
- In max. mode, the control signals are generated by external Bus-controller 8288.
- The maximum mode operation is used only when the system contains arithmetic co-processor such as 8087 numeric co-processor.
- Max. Mode signals ☾ QS_1, QS_0, $\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$, $\overline{LOCK}$, RQ/$\overline{GT_1}$, RQ/$\overline{GT_0}$

1.6. MINIMUM MODE CONFIGURATION OF 8086 :

- The min. mode operation is selected by connecting the pin MN/MX to + 5 V
- The 8086 is operated in minimum mode in simple systems with a single CPU
- In min. mode operation, all the control signals are generated by CPU
- It is least expensive way to operate and the operation is similar to 8085A.
- Min. Mode signals : $\overline{INTA}$, ALE, $\overline{DEN}$, DT/$\overline{R}$, M/$\overline{IO}$, $\overline{WR}$, HLDA, HOLD

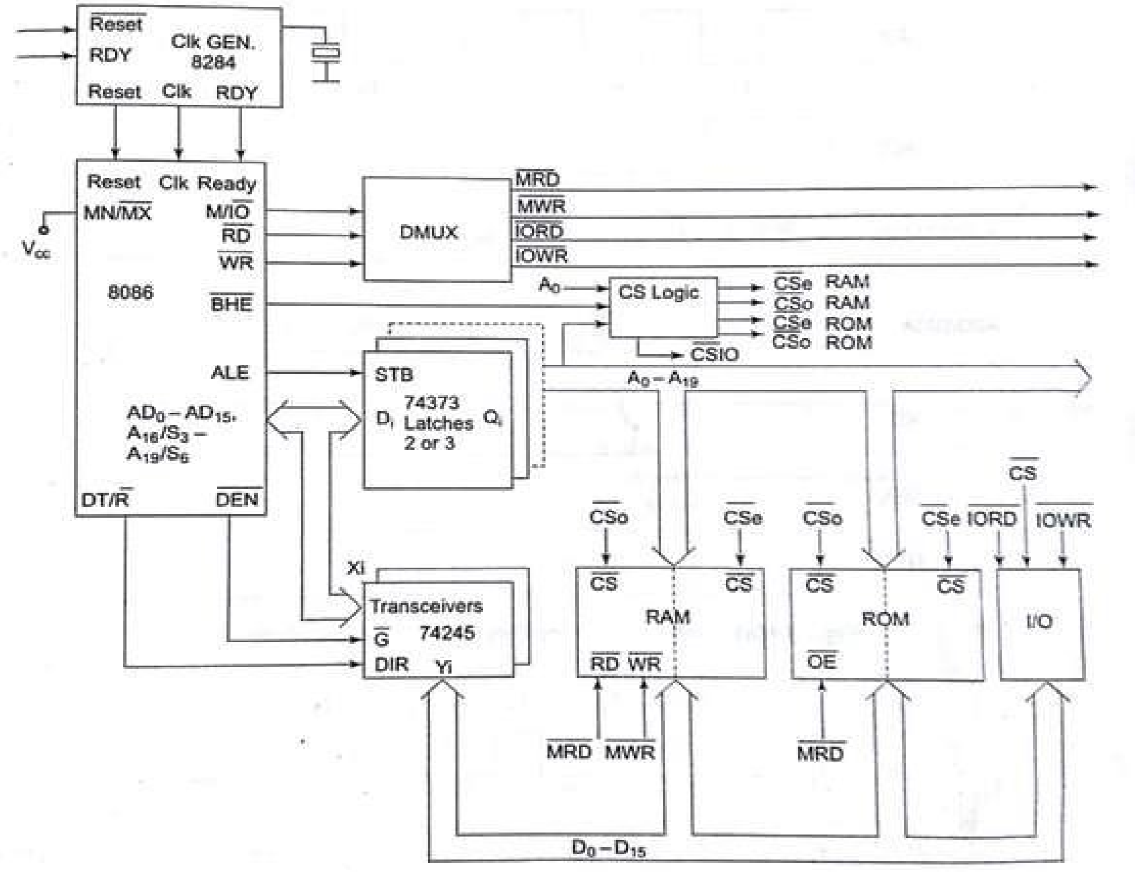


Fig.1.13 Minimum Mode 8086 System

Clock generator :

- It generates a CLK of frequency 5 MHz
- It provides the basic timing for µp and bus control activity
- It also synchronizes some external signals with the system clock

Address Latches :

- Latches are used for de-multiplexing of $AD_0 - AD_{15}$, A_{16}/S_3 - A_{19}/S_6 and $\overline{BHE}$ / S_7
- Latches are used for the separation of address and data lines
- Address latch enable (ALE) signal is used to enable the Latches

Transceivers/ Bi-directional data buffers

- The Bi-directional data buffers (transmitters/receivers) are used to maintain proper signal quality. i.e., to increase the fan-out of the system.
- The $\overline{DEN}$ signal is used to enable the bi-directional data buffers
- The DT/$\overline{R}$ signal controls the flow of data through data buffers

MINIMUM MODE SIGNALS:

$\overline{INTA}$ (pin-24) :

- The interrupt acknowledge signal is a response to the INTR.
- It indicates the recognition of an Interrupt Request.

ALE (pin-25) :

- The address latch enable signal is used to indicate the presence of valid address information on multiplexed bus (AD_0 - AD_{15})
- This signal is used to enable the Address Latches.
- The address latches are used to separate the address and data lines.(demultiplexing)

$\overline{DEN}$ (pin-26) :

- The data buffers enable signal is used to enable the bi-directional data bus buffers.
- The data bus buffers or transceivers (transmitters/receivers) are used to maintain proper signal quality.

DT/ $\overline{R}$ (pin-27) :

- The data transmit / receive signal is used to indicates the direction of data flow through the data bus buffers.

  DT/$\overline{R}$ = 1 for transmitting data

  DT/$\overline{R}$ = 0 for receiving data

M/$\overline{IO}$ (pin-28) :

It selects either Memory or I/O operation

M/$\overline{IO}$= 1 selects memory operation

M/$\overline{IO}$= 0 selects I/O operation

M/$\overline{IO}$	$\overline{RD}$	$\overline{WR}$	Operation
1	0	1	Memory Read
1	1	0	Memory Write
0	0	1	I/O Read
1	1	0	I/O Write

$\overline{WR}$ (pin-29) :

This control signal is active whenever the processor is writing data to Memory or I/O

HOLD & HLDA (pins 31 & 30):

- These two signals are used in DMA operation
- The HOLD input indicates the processor that other bus master (DMA controller) is requesting for the use of system bus.
- When HOLD =1, the µp stops the normal program execution and places address, data & control buses at high-impedance state and sends HLDA signal to the DMA controller.
- The HLDA signal indicates that the processor has accepted the HOLD request and the gain control of the system bus is transferred to the DMA controller.
- During HLDA =1 , the DMA controller is the master of the system bus.
- After removal of HOLD request, the HLDA becomes Low.

Timing diagrams for Min. mode

- The working of microprocessor based system can be explained with the help of Timing diagrams.
- The timing diagrams provide the information about the various conditions of the signal while a machine cycle is executed.

T- State : It is one cycle of the clock (clock period)

Machine cycle / Bus cycle :

- The group of T-States required for a basic bus operation is called as Machine cycle
- The microprocessor uses bus cycles for memory and I/O operations
  - Opcode fetch cycle
  - Memory Read cycle
  - Memory Write cycle
  - I/O Read cycle
  - I/O Write cycle

Instruction cycle:

- The time required for the microprocessor to fetch, decode & execute an instruction is called an Instruction cycle.
- An instruction cycle consists of one (or) more bus cycles.

The timing diagram for Memory Read cycle in Minimum mode is shown in figure.

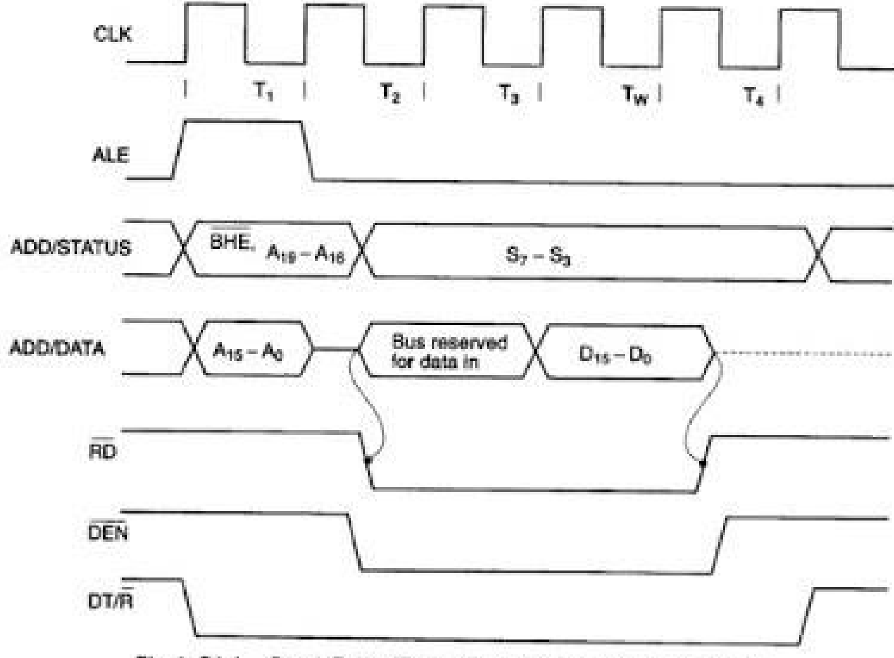


Fig. 1.9(a) *Read Cycle Timing Diagram for Minimum Mode*

During the time period T_1

- The micro processor sends out 20 bit address.
- Enables ALE Signal
- Issues $M/\overline{IO} = 1$ for memory operation
- Issues $DT/\overline{R} = 0$ for data reception

During the time period T_2

- The address information is removed from multiplexed bus and status signals are placed. However A0-A19 remain available as they were latched during T_1
- Enables $\overline{RD}$ signal
- Enables $\overline{DEN}$ signal

During the time period T_3

- The microprocessor checks the READY line. If READY =1, the µP reads the data from data bus. Otherwise, a WAIT state T_W is inserted

During the time period T_4

- All the control signals are deactivated to start the next cycle.
- $\overline{DEN}$, $DT/\overline{R}$, $M/\overline{IO}$ and $\overline{RD}$ signals are deactivated

1.7. MAXIMUM MODE CONFIGURATION OF 8086 :

- The max. mode operation is selected by connecting the pin $MN/\overline{MX}$ to GND
- The 8086 is operated in max. mode in multi-processor system with more than one processor.
- **In max. mode, the control signals are generated by external Bus-controller 8288.**
- The maximum mode operation is used only when the system contains arithmetic co-processor such as 8087 numeric co-processor.
- Max. Mode signals : QS_1, QS_0, $\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$, $\overline{LOCK}$, $RQ/\overline{GT_1}$, $RQ/\overline{GT_0}$

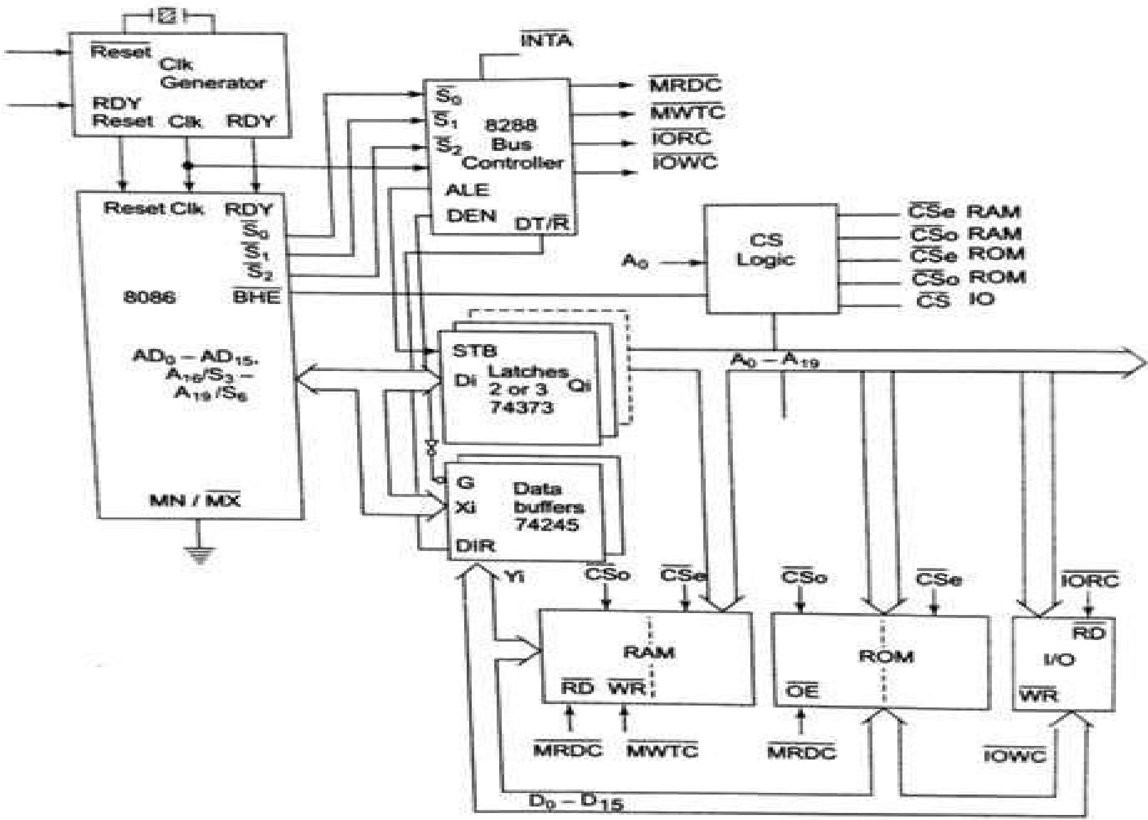


Fig. 1.15 Maximum Mode 8086 System

MAXIMUM MODE SIGNALS:

QS_1 & QS_0 (pins-24&25):

The Queue status pins indicate the status of Instruction Queue of 8086 processor.

QS_1 QS_0	Queue Status
0 0	No operation
0 1	First byte of Opcode
1 0	Queue is empty Next
1 1	byte of Opcode

$\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$ (pins- 26,27&28):

These status signals indicate the function of the current bus-cycle. i.e., the type of operation being carried out by the processor.

$\overline{S_2}$ $\overline{S_1}$ $\overline{S_0}$	Bus cycle
0 0 0	INTA
0 0 1	I/O Read
0 1 0	I/O Write
0 1 1	HALT
1 0 0	Op-code fetch
1 0 1	Memory read
1 1 0	Memory write
1 1 1	INACTIVE

LOCK : (pin-29)

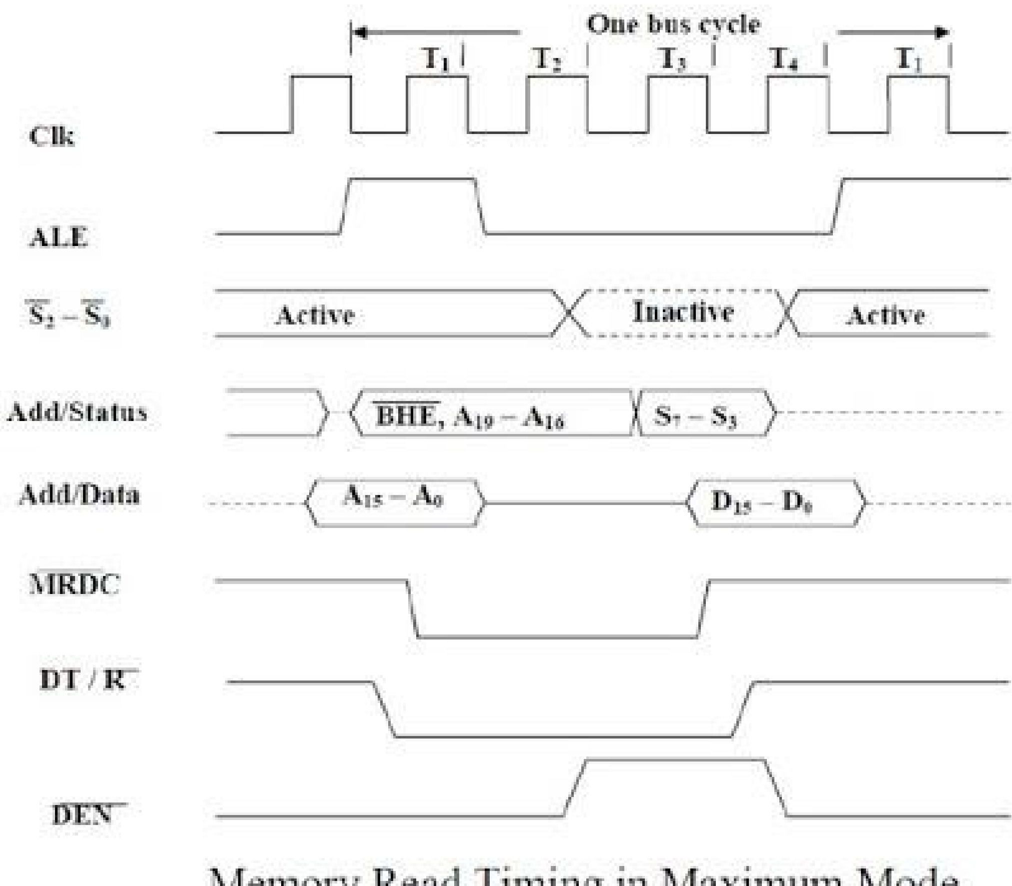
- This pin indicates that the processor is executing a LOCK prefixed instruction and the System bus is not to be used by another bus-master.
- This pin is used in multi-processor system to prevent other Bus masters from taking the control of System bus during the execution of a critical instruction.

RQ/GT_1 , RQ/GT_0 (pins -30 & 31) :

- The Request/Grant pins are used to request a DMA action during Maximum mode operation.
- These are bidirectional and used to request and grant the DMA operation.

Timing diagrams for Max. mode

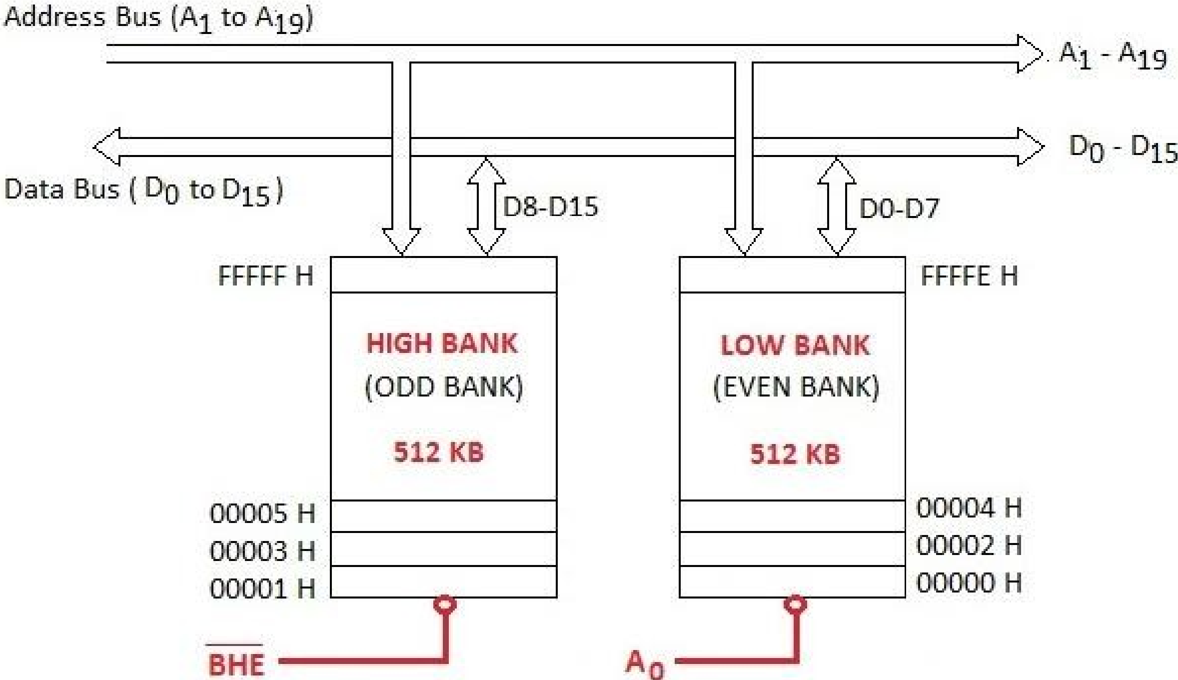
Op-code fetch cycle
Memory Read cycle
Memory Write cycle
I/O Read cycle
I/O Write cycle



Memory Read Timing in Maximum Mode

1.8. PHYSICAL MEMORY ORGANIZATION AND MEMORY BANKS ACCESSING

- The Physical memory of 8086 is divided into TWO banks **for accessing 16-bit numbers**. Each bank size is 512 K bytes.
- The data bus (D_0 - D_7) is connected to LOW bank /EVEN bank.
- The data bus (D_8 - D_{15}) is connected to HIGH bank/ ODD bank.
- Both Banks are enabled at a time for 16-bit operations



- Address lines (**A_1-A_{19}**) are used to select a location within the bank
- The LOW bank is selected by A_0
- The HIGH bank is selected by $\overline{BHE}$ signal.
- Both Banks are enabled at a time for 16-bit operations.
- Note that the address must be EVEN for 16-bit access.

$\overline{BHE}$ A_0	***Bank Selected***
0 0	Both banks are enabled for 16-bit operation
0 1	HIGH bank is enabled
1 0	LOW bank is enabled
1 1	No banks are enabled

1.9. INTERRUPTS OF 8086 & INTERRUPT VECTOR TABLE

- Interrupt is an event that causes the µp to stop the normal program execution.
- The µp services the interrupt by executing a subroutine called Interrupt Service Routine
- After executing ISR, the control is transferred back again to the main program.

There are 3 sources of interrupts for 8086

Hardware Interrupts ☾ signal applied to NMI, INTR pins
Software Interrupts ☾ by executing INT instructions
Error in program execution ☾ Divide by Zero, Overflow, etc

Hardware Interrupts :

These interrupts occur as signals on the external pins of the microprocessor. 8086 has two pins to accept hardware interrupts, NMI and INTR.

NMI (Non Maskable Interrupt)

- It is a positive going edge triggered Non-Maskable Interrupt
- It cannot be disabled by using software.
- This interrupt has highest priority than INTR.
- On receiving an interrupt on NMI line, the microprocessor executes INT 4
- The µp obtains the ISR address from location 2 x 4 = 00008H from the IVT

.

INTR (Interrupt Request)

- It is a **level triggered** maskable Interrupt Request.
- It can be disabled by using software i.e., the processor will get interrupted only if IF=1. Otherwise the processor will not get interrupted even INTR is active.
- It is masked by making IF = 0 by software through CLI instruction.
  It is unmasked by making IF = 1 by software through STI instruction.
- It is a non-vectored interrupt. On receiving an interrupt on INTR line, the µp issues INTA pulse to the interrupting device. Then the interrupting device sends the vector number 'N' to the processor. Now the µp multiplies N x 4 and goes to the corresponding location in the IVT to obtain the ISR address.

Software Interrupts :

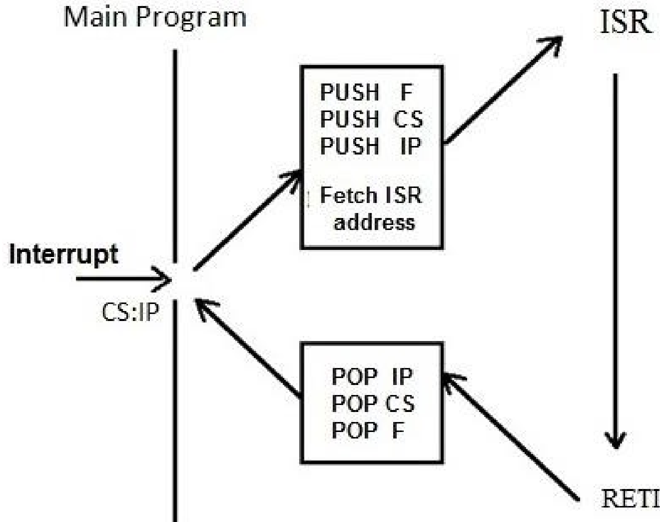
These interrupts are caused by writing the software interrupt instruction **INT N** where 'N' can be any value from 0 to 255 (00H to FFH). Hence all 256 interrupts can be invoked by software.

Error conditions :

The 8086 is interrupted when some special conditions occur while executing certain instructions in the program.

Example: Divide error, Overflow etc

Processing of Interrupts



When an interrupt is enabled, the 8086 µp performs the following actions

- ✓ It completes the execution of current instruction
- ✓ It pushes the Flag register (PSW) on to the stack
- ✓ The contents of CS and IP are pushed to stack (**return address**)
- ✓ It issues an Interrupt acknowledge (INTA)
- ✓ It computes the vector address from the type of interrupt
- ✓ The IP & CS are loaded with ISR address, which is available in Interrupt Vector Table
- ✓ Then the control is transferred to ISR and starts to execute the interrupt service routine at that address.

The code to handle an interrupt is called an *interrupt handler* or *Interrupt Service Routine* (ISR). An interrupt service routine must always finish with the special instruction IRET *(return from interrupt)*, which performs the following actions.

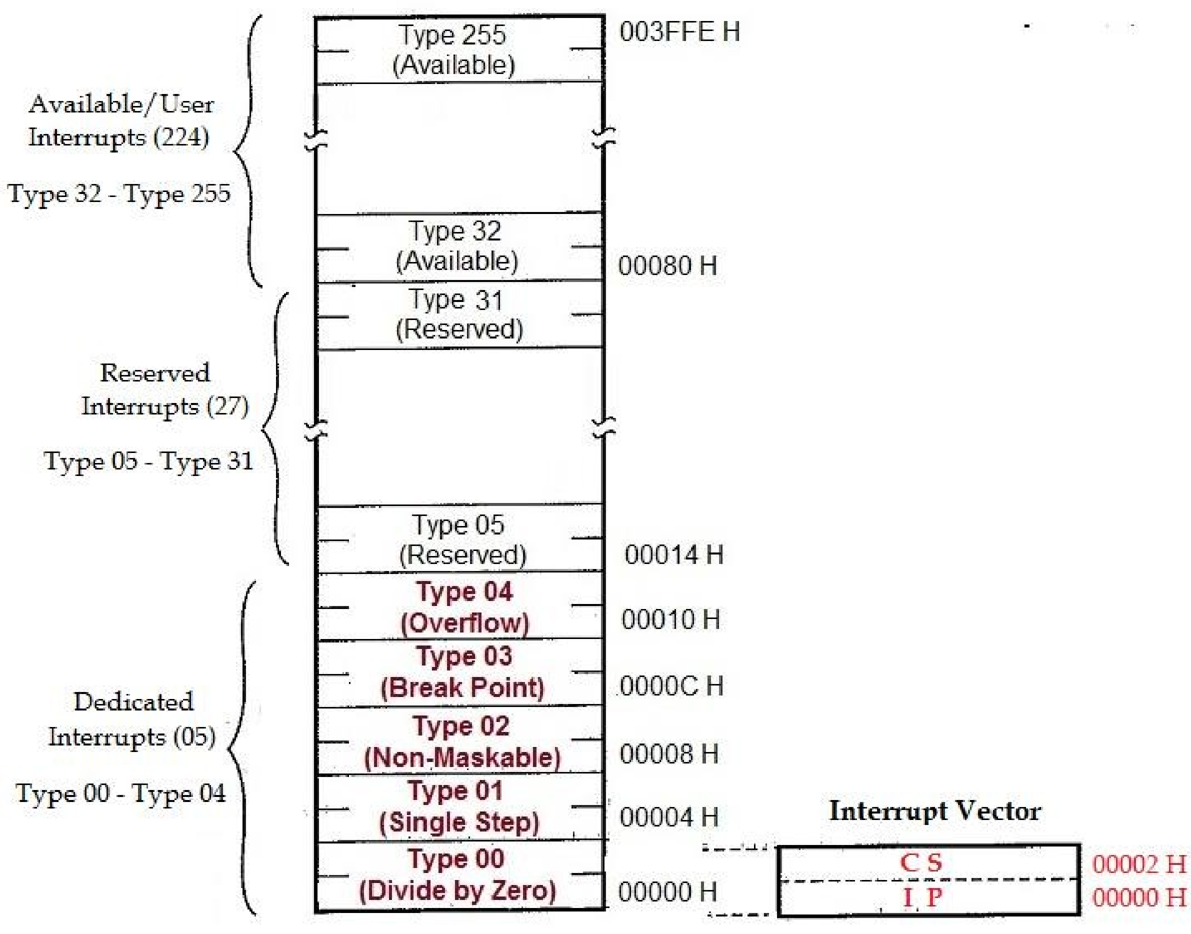
- ✓ The IP and CS are loaded with **return address** from stack
- ✓ The Flag register is popped from the stack
- ✓ The program execution returns to main line, where it was interrupted.

Interrupt Vector Table:

When the interrupt is enabled, the present IP and CS are pushed on to the stack and loaded with the address of ISR, which is available in Interrupt vector table.

Interrupt Vector ☾ It is a memory block which contains address of ISR
The size of Interrupt vector is 4-bytes (to store CS and IP of ISR)

Interrupt Vector Table ☾ It is a memory block which contains interrupt vectors
Since there are 256 types of interrupts, there are 256 interrupt vectors
Hence, the size of Interrupt Vector Table is 4*256 = 1024 bytes



8086 Interrupt Vector Table

In 8086 interrupt system, the first 1 KB memory from 00000 H to 003FF H is reserved for storing the starting addresses of ISRs. This block of memory is called as IVT.

- The Interrupt vector table contains 256 interrupt vectors
- **The interrupt vector contains IP and CS values of ISR**
- The address of interrupt vector can be calculated by multiplying the TYPE with 4
- *For example,* The interrupt vector address for Type-02 interrupt = 02 H*4= **00008 H**

- **The 8086 uses 256 types of interrupts – Type 00 to Type 255**

  These interrupts are classified as

  (A) Dedicated Interrupts (Type 00 to Type 04) : 05

  (B) Reserved Interrupts (Type 05 to Type 31) : 27

  (C) Available/User Interrupts (Type 32 to Type 255) : 224

(A) The dedicated interrupts (Type 0 – Type 4) are

(i)	INT 0	:	Divide Error
(ii)	INT 1	:	Single step execution (TRAP)
(iii)	INT 2	:	Non-Maskable Interrupt (NMI)
(iv)	INT 3	:	Break Point
(v)	INT 4	:	Overflow

(i) INT 0 (Divide Error)-

- This interrupt occurs whenever there is division error i.e. when the result of a division is too large to be stored. This condition normally occurs when the divisor is very small as compared to the dividend (or) the divisor is zero.
- Its ISR address is stored at location: Type 0 x 4 = **00000**H in the IVT.

(ii) INT 1 (Single Step)-

- The µp executes this interrupt after every instruction if the TF =1
- It puts µp in single stepping mode i.e. the microprocessor will get interrupted after executing every instruction. This is very useful during debugging.
- This ISR generally displays contents of all registers.
- Its ISR address is stored at location: Type 1 x 4 = **00004**H in the IVT.

(iii) INT 2 (Non Maskable Interrupt)-

- The microprocessor executes this ISR in response to an interrupt on the NMI (Non mask-able Interrupt) line.
- Its ISR address is stored at location: Type 2 x 4 = **00008**H in the IVT.

(iv) INT 3 (Breakpoint Interrupt)-

- This interrupt is used to cause breakpoints in the program.
- It is useful in debugging large programs
- This ISR generally displays contents of all registers
- Its ISR address is stored at location: Type 3 x 4 = **0000C**H in the IVT

(v) INT 4 (Overflow Interrupt)

- This interrupt occurs if the overflow flag is set
- It is used to detect overflow error in signed arithmetic operations.
- Its ISR address is stored at location: Type 4 x 4 = **0001**0H in the IVT

(B) Reserved interrupts (Type 5 to Type 31)

These levels are reserved by Intel to be used in higher processors like 80386, Pentium etc. They are not available to the user

(C) Available interrupts (Type 32 to Type 225)

- These are user defined, software interrupts.
- ISRs for these interrupts are written by the users to service various user defined conditions.
- These interrupts are invoked by writing the instruction INT N.
- Its ISR address is obtained by the microprocessor from location N x 4 in the IVT

Problem: ***The contents of memory location 0000:008C are given below 0000:008C ☾ 12, 34, 56, 78, 90, 92***

(a) What is the interrupt vector address for Type-23H interrupt?

(b) Find the address of ISR corresponding to INT 23H

(c) ***For which type of interrupt, the interrupt vector address is 0000:00C8H***

Sol: (a) Interrupt vector address = Type * 4

For Type-23H interrupt, Interrupt vector address = 23*4 = 8C H

Interrupt vector address for Type-23H is 0000:008C H

(b) The address of ISR for Type-23H is available at 0000:008C H ISR

IP is available at memory 0000:008C H = 3412 H

ISR CS is available at memory 0000:008E H = 7856 H

Address of ISR = CS: IP = 7856:1234

(c) Interrupt vector address = Type * 4

00C8 = Type * 4

Type = 00C8 / 4 = 32H

Hence the Type of Interrupt is 32H

UNIT-2
ASSEMBLY LANGUAGE PROGRAMMING WITH 8086

1. Addressing modes of 8086
2. Instruction formats of 8086
3. Instruction set of 8086
4. Assembler Directives
5. Procedures
6. Macros
7. Comparison between Procedures and Macros
8. Simple ALPs – Programming examples

2.1. ADDRESSING MODES OF 8086 :

Instruction ☾ An instruction is a command given to the microprocessor to perform a specific operation on specified data.

Addressing Mode ☾ The method of specifying data to be operated by an instruction is called as addressing mode

The 8086 supports the following addressing modes:

1. Immediate addressing
2. Register addressing
3. Direct addressing
4. Register Indirect addressing
   - (a) Based addressing
   - (b) Indexed addressing
   - (c) Based Indexed addressing
5. Register Relative [or] Register Indirect with Displacement
   - (a) Relative Based.
   - (b) Relative Indexed
   - (c) Relative Based Indexed addressing
6. Implicit addressing
7. I/O port addressing
8. Addressing modes for Control transfer / Branch Instructions
   - (a) Intra segment mode – Direct & Indirect
   - (b) Inter segment mode – Direct & Indirect

1. **Immediate addressing:** The operand (or) data is available in the instruction itself.

Ex: MOV AX, 1234H
ADD AX, 4567H

2. **Register addressing:** The data is available in any one of the general purpose registers.

Ex: MOV AX, BX
ADD AX, BX

3. **Direct addressing:** The offset /effective address of the data is available in the instruction.

Ex: MOV BX, DS:[2000H]
ADD AX, DS:[3000H]

4. **Register Indirect addressing:**

The effective address of data is available in any one of the Base (or) Index registers

(a) Based addressing:

The effective address of data is available in Base registers - BX or BP Ex:

MOV AX, DS:[BX]

(b) Indexed addressing:

The effective address of data is available in Index registers - SI or DI Ex:

MOV AX, DS:[SI]

(c) **Based Indexed addressing**

The effective address of data is the sum of Base and Index registers Ex:

MOV AX, DS:[BX+SI]
MOV AX, DS:[BX][SI]

5. **Register Relative addressing:**

The effective address is the sum of contents of Base/Index registers and 8-bit /16-bit signed Displacement.

(a) Relative Based addressing:

The effective address is the sum of Base register and 8-bit /16-bit displacement Ex:

MOV AX, DS:[BX+25H]
MOV AX, 25H DS:[BX]

(b) Relative Indexed addressing:

The effective address is the sum of Index register and 8-bit /16-bit displacement Ex:

MOV AX, DS:[SI+25H]
MOV AX, 25H DS:[SI]

(c) Relative Based Indexed addressing:

The effective address is the sum of Base register, Index register and Displacement Ex:

MOV AX, DS:[BX+SI+25H]
MOV AX, 25H DS:[BX][SI]

6. Implicit addressing:

There are some instructions which operate on the content of Accumulator. Such instructions do not require the address of operand. This type of addressing is called as Implicit (or) Implied addressing.

Ex: DAA - Decimal Adjust Accumulator after addition AAA -
ASCII Adjust Accumulator after addition

7. I/O port addressing:

The I/O port addressing is used to access the I/O ports.
The I/O read and I/O write operations are performed through Accumulator only.

(a) Fixed port addressing: The 8-bit I/O port address is available in the instruction

Ex: IN AL, 80H ; Reads data from port address 80H to AL
OUT 82H, AL ; Sends data from AL to port address 82H

(b) Variable port addressing: The 16-bit I/O port address is available in DX register. Ex: IN AL, DX
OUT DX, AL

8. Addressing modes for Control transfer / Branch Instructions

For the Control transfer instructions (or) Branch instructions such as JMP, CALL, RET,...etc, the addressing modes depend on whether the destination address lies in the same code segment (or) different code segment.

These are 2- types : (a) Intra segment mode
(b) Inter segment mode

(a) Intra-segment mode :

- In this mode, the destination address lies in the same Code segment.
- Here only IP is modified. CS remains the same.

(i) Intra-segment Direct : In this mode the instruction specifies the DISP value

destination IP = Present IP + 8-bit (or) 16-bit DISP given in instruction

Ex: JMP displacement
JMP SHORT Lable

Note:

(i) If the displacement is 8-bit, the destination lies within -128 bytes to +127 bytes. This type of JUMP is called as SHORT JUMP.

(ii) If the displacement is 16-bit, the destination lies within -32 K bytes to +32 K bytes This type of JUMP is called as LONG JUMP.

(ii) Intra-segment Indirect :

In this mode, the destination address is found as content of Memory location.

destination IP = 16-bit Content of memory location

Ex: JMP WORD PTR[BX]
CALL WORD PTR[BX]

destination IP ← [BX]

(b) Inter segment mode :

- In this mode, the destination address lies in different Code segment.
- It provides branching from one code segment to another code segment.
- Here both CS and IP registers will be modified.

(i) Inter segment Direct :

In this mode, the CS and IP values of destination are specified directly in the Instruction

Ex: JMP 4000:6000
CALL 4000:6000

destination CS = 4000H,
destination IP = 6000H

(ii) Inter segment Indirect:

In this mode, the CS and IP values of destination are found as content of memory locations.

Ex: JMP DWORD PTR[BX]
CALL DWORD PTR[BX]

destination IP ← [BX]
destination CS ← [BX+2]

Problem :

Calculate the offset address and 20-bit physical address for the following addressing modes. The content of different registers are given below:

DS = 4000H, ES =6000H, SS= 8000H, SP=1998H

BX = 6688H, SI= 3333H, DI= 4444H

(a) MOV AX, DS:[5060H] (or) MOV AX, [5060H] -- Direct addressing

Offset address	=	5060 H
Segment base = DS*10	=	40000 H
20-bit Physical address	=	45060 H

(b) ADD AX, DS:[BX][SI] -- Based Indexed addressing

Offset address = BX+SI	=	99BB H
Segment base = DS*10	=	40000 H
20-bit Physical address	=	499BB H

(c) XOR AX, 25H [DI] - Relative Indexed addressing

Offset address = DI+25H	=	4469 H
Segment base = ES*10	=	60000 H
20-bit Physical address	=	64469 H

(d) MOV AX, 5000 [BX] [SI] - Relative Based Indexed addressing

Offset address = BX+SI	=	99CC H
Segment base = DS*10	=	40000 H
20-bit Physical address	=	499CC H

(e) PUSH AX : It decrements SP by 2 and AX is stored at stack top pointed by SP

Here, Source is AX
Destination is Stack

SP ← SP - 2
[SP] ← AX

Destination address = SP - 2 = 1996 H
20-bit Physical address = SS*10 + 1996 =81996 H

(f) POP CX : It copies the content of stack top to CX and SP is incremented by 2

Here, Source is Stack
Destination is CX

CX ← [SP]
SP ← SP + 2

Source address = SP = 1998 H
20-bit Physical address = SS*10 + 1998 = 81998 H

2.2. INSTRUCTION FORMATS OF 8086 :

- **Instruction** is a command given to the microprocessor to perform a specific task on specified data.

- Each instruction has 2- parts

Instruction

OPCODE	OPERAND

Opcode (Operation code) ☾ The task to be performed
Operand ☾ The data to be operated on

- The method of specifying data to be operated by an instruction is called as addressing mode. It may immediate data/content of register /content of memory location.

- There are 6- instruction formats in 8086 instruction set. The length of an instruction may vary from 1 to 6 bytes.

(i) One byte Instruction :

D7 DO

OPCODE

- This format is only one byte long.
- It may have implied data (or) register operands.
- The least significant 3-bits of Op-code represent the register operand, if any. Otherwise, all 8-bits are Opcode bits and Operands are implied.

Ex: DAA ; Decial Adjsut accumulator after Addition
STC ; Set carry flag

(ii) Register to Register :

D7 D0

OPCODE	W

D7 D6	D5 D4 D3	D2 D1 D0
1 1	REG	R/M

- This format is two bytes long.
- The first byte represents Op-code and width of the operand specified by ' W' bit.
- W=1 for 16-bit operand and W=0 for 8-bit operand
- The second byte represents the register operands and R/M fields Ex:

MOV AX, BX

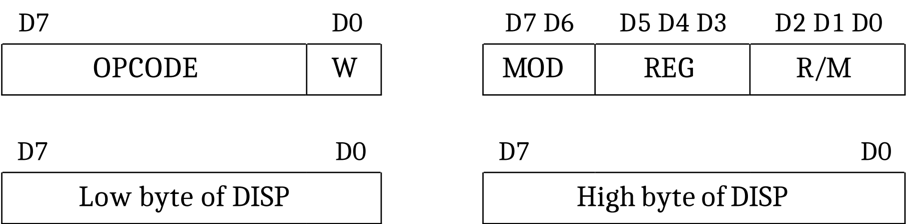
(iii) Register to/from Memory without Displacement

D7	D0	D7 D6	D5 D4 D3	D2 D1 D0
OPCODE	W	MOD	REG	R/M

- This format is also two bytes long, and similar to Register to Register format, except the MOD filed.
- The first byte represents Op-code and width of the operand
- The second byte represents the register operands and R/M fields
- The MOD filed represents mode of addressing.

Ex: MOV AX, [SI]

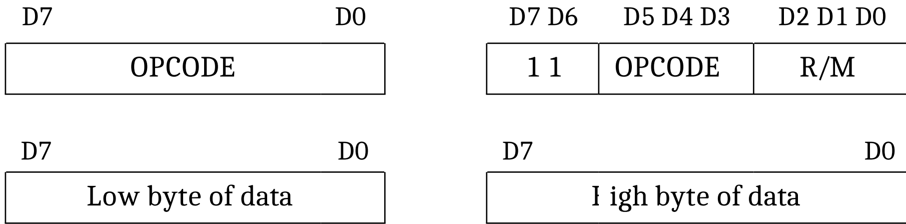
(iv) Register to/from Memory with Displacement



- This format contains one (or) two additional bytes for DISP along with 2-byte format of Register to Memory without Displacement.

Ex: MOV AX, [SI+2000H]

(v) Immediate operand to Register



- In this format, the first byte and 3-bits from the second byte are used to represent the Op-code. (Immediate addressing mode)
- It also contains one (or) two additional bytes of immediate data Ex:

MOV AX, 1234 H

(vi) Immediate operand to Mmeory with Displacement

D7 OPCODE D0	D7 D6 MOD	D5 D4 D3 OPCODE	D2 D1 D0 R/M

D7 Low byte of DISP D0	D7 High byte of DISP D0
D7 Low byte of data D0	D7 High byte of data D0

- This format is 5 (or) 6 bytes long
- The first 2-bytes represent OPCODE, MOD and R/M fields.
- The next 2-bytes represent Displacement
- The last 2-bytes represent Immediate data

Registers Codes

REG	W=1	W=0
000	AX	AL
001	CX	CL
010	DX	DL
011	BX	BL
100	SP	AH
101	BP	CH
110	SI	DH
111	DI	BH

SREG	Segment Register
00	ES
01	CS
10	SS
11	DS

MOD, REG and R/M filed Codes

MOD / R/M	Memory operands			Reg. operands	
	00 No Disp	01 8-bit Disp	10 16-bit Disp	11 W=1	11 W=0
000	[BX+SI]	[BX]+[SI]+D8	[BX]+[SI]+D16	AX	AL
001	[BX+DI]	[BX]+[DI]+D8	[BX]+[DI]+D16	CX	CL
010	[BP+SI]	[BP]+[SI]+D8	[BP]+[SI]+D16	DX	DL
011	[BP+DI]	[BP]+[DI]+D8	[BP]+[DI]+D16	BX	BL
100	[SI]	[SI]+D8	[SI]+D16	SP	AH
101	[DI]	[DI]+D8	[DI]+D16	BP	CH
110	D16	[BP]+D8	[BP]+D16	SI	DH
111	[BX]	[BX]+D8	[BX]+D16	DI	BH

2.3. INSTRUCTION SET OF 8086 :

The 8086 Instruction set is classified as

1. Data transfer instructions
2. Arithmetic instructions
3. Logic and bit manipulation instructions
4. Branch instructions / Control transfer instructions
5. String manipulation instructions
6. Processor control instructions

   (a) Flag manipulation instructions (b) Machine control instructions

(1) Data Transfer Instructions		
1	MOV dest, source	Copies data from source to destination
2	PUSH source	Pushes the content of source onto the stack.
3	POP dest	Pop a word from stack top to destination
4	XCHG dest, source	Exchange the contents of source and destination
5	IN AL, port_addr	Read data from specified input port to Accumulator
6	OUT port_addr, AL	Send data from Accumulator to specified output port
7	LEA Reg, Operand	Load specified register with the effective address of operand
8	LDS Reg, Addr.	Load specified register and DS registers with contents of two words from the effective address
9	LES Reg, Addr.	Load specified register and ES registers with contents of two words from the effective address
10	LAHF	Load AH from lower byte of Flag
11	SAHF	Store AH to lower byte of Flag
12	XLAT	Translate byte using look-up table

(2) Arithmetic Instructions		
1	ADD dest, source	The content of source is added to the destination
2	ADC dest, source	The content of source along with carry are added to the destination
3	SUB dest, source	The content of the source is subtracted from destination.
4	SBB dest, source	The content of the source along with borrow is subtracted from destination

5	INC dest	Increases the content of destination by 1
6	DEC dest	Decreases the content of destination by 1
7	CMP dest, source	Compares destination and source operands by performing Subtraction of source from destination. Only flags are effected
8	MUL source	Unsigned multiplication of Accumulator with source
9	IMUL source	Signed multiplication of Accumulator with source
10	DIV source	Unsigned division of Accumulator by the source
11	IDIV source	Signed division of Accumulator by the source
12	CBW	Converts a signed byte in AL to a signed word in AX
13	CWD	Converts a signed word in AX to a signed word in DS:AX
14	DAA	Decimal adjust Accumulator after Addition
15	DAS	Decimal adjust Accumulator after Subtraction
16	AAA	ASCII adjust Accumulator after Addition
17	AAS	ASCII adjust Accumulator after Subtraction
18	AAM	ASCII adjust Accumulator after Multiplication
19	AAD	ASCII adjust Accumulator before Division

(3) Logical Instructions		
1	AND dest, source	Performs bitwise AND operation of Source and Destination
2	OR dest, source	Performs bitwise OR operation of Source and Destination
3	XOR dest, source	Performs bitwise Ex-OR operation of Source and Destination
4	NOT dest	Performs 1's complement of destination
5	TEST dest, source	Performs bitwise logical AND operation on the two operands. Only flags are affected. Result is not stored anywhere
6	SHL / SAR	Logical shift Left / Arithmetic shift Left (by 1 or CL)
7	SHR	Logical shift Right
8	SAR	Arithmetic shift Right
9	ROL	Rotate Left
10	RCL	Rotate Left through carry
11	ROR	Rotate Right
12	RCR	Rotate Right through carry

(4) Control transfer instructions / Branching instructions		
1	JMP	Unconditional jump
2	CALL	Call a sub-routine
3	RET	Return to Main program
4	INT N	Software Interrupt Type N
5	IRET	Return from ISR
6	INTO	Interrupt on Overflow
7	LOOP LOOPE LOOPNE	Loop while CX $\neq$ 0 Loop while CX $\neq$ 0 and ZF = 1 Loop while CX $\neq$ 0 and ZF = 0
8	JCXZ	Jump if CX equals zero

(5) String manipulation instructions		
1	MOVSB/ MOVSW	Move string byte / word from DS:[SI] to ES:[DI]
2	CMPSB/ CMPSW	Compare two string bytes / words
3	LODS	Load accumulator with string byte / word from DS:[SI]
4	STOS	Store string byte / word in accumulator at ES:[DI]
5	SCAS	Compare string byte in Accumulator with ES:[DI]
6	REP REPE REPNE	Repeat while CX $\neq$ 0 Repeat while CX $\neq$ 0 and ZF = 1 Repeat while CX $\neq$ 0 and ZF = 0

(5) Processor control instructions		
		(a) Flag manipulation instructions
1	STC, CLC, CMC	Set , Clear , Complement Carry flag
2	STD, CLD	Set , Clear Directional flag
3	STI, CLI	Set , Clear Interrupt flag
4	LAHF, SAHF	Load AH from flags, store AH into flags
5	PUSHF, POPF	Push flags onto stack, pop flags off stack
		(a) Machine control instruction
1	ESC	Escape to external processor interface
2	WAIT	Wait for signal on $\overline{TEST}$ input pin active
3	LOCK	Lock bus during next instruction
4	NOP	No operation
5	HLT	Halt processor (Stops execution)

2.4. ASSEMBLER DIRECTIVES :

- Assembly language program (ALP) consists of two types of statements
  - Instructions and Directives
- **Instructions** are used to perform a specific operation on specified data. The instructions are translated into machine codes by the assembler.
- **Directives** are used to direct the assembler during assembly process, for which no machine code is generated. The Assembler directives are hints given to the assembler

S.No.	Type	Directives
1	Program Organization Directives	SEGMENT, ENDS, END ASSUME, GROUP
2	Data definition and Storage Directives	DB, DW, DD, DQ, DT
3	Alignment Directives	ORG, EVEN
4	Procedure definition Directives	PROC, ENDP
5	Macro definition Directives	MACRO, ENDM, EQU
6	Value returning attribute Directives	OFFSET, TYPE, LENGTH, SIZE, SEG
7	Data Control directives	PTR, PUBLIC, EXTRN

1. Program Organization Directives

(i) SEGMENT : This directive is used to indicate the starting of the logical segment

Syntax : Seg_name SEGMENT
Example : DATA SEGMENT

(ii) ENDS : This directive is used to indicate the ending of the logical segment

Syntax : Seg_name ENDS
Example : DATA ENDS

(iii) END : This directive is used after the last statement of the program to indicate the ending of the program

Syntax : END

(iv) ASSUME : This directive is used to inform the name of the logical segments to be used as Code segment, Data segment, Extra segment and Stack segment

Syntax : ASSUME Seg_Reg : Seg_name
Example : ASSUME CS: CODE, DS: DATA, ES: EXTRA

(v) GROUP : This directive is used to group the logical segments into one logical segment. i.e., the grouped segments will have same segment base.

Syntax : Group_name GROUP: Seg1_name, Seg2_name, ...
Example : SMALL_SYSTEM GROUP DATA, CODE, EXTRA

2. Data definition and Storage Directives

These directives are used to define the program variables and **allocate a specified amount of memory to them**. They are of type Byte, Word, Double word and Quad word.

(i) DB : This directive is used to define a variable of BYTE type.

Syntax : Var_name DB value
Examples : N1 DB 25H
N2 DB ?
ARRAY DB 10H, 20H, 30H, 40H, 50H
GRADE DB 'A' NAME
DB "MICRO"

(ii) DW : This directive is used to define a variable of WORD type (2-bytes)

Syntax : Var_name DW value
Examples : N1 DW 1234 H
ARRAY DW 1000H, 2000H, 3000H, 4000H

(iii) DD : This directive is used to define a variable of type Double word (4-bytes)

Syntax : Var_name DD value
Examples : N DD 11223344 H

(iv) DQ : This directive is used to define a variable of type Quad word (8-bytes)

Syntax : Var_name DQ value
Examples : N DD 1122334455667788 H

(v) DT : This directive is used to define a variable of type Ten bytes (10-bytes)

Syntax : Var_name DT value
Examples : N DT 11223344556677889900 H

3. Alignment Directives

These directives are used to modify the memory location counter

(i) ORG: This directive is used to set the location counter to desired value.

Syntax : ORG value
Examples : ORG 4000 H ; location counter = 4000 H
ORG $+1000 H ; location counter is incremented by 1000H

(ii) EVEN : This directive is used to increment the location counter to next Even address, if the present address is Odd. The 8086 can access a word in one bus-cycle, if the address is Even. Hence a series of words can be quickly accessed, if they are stored at Even address

Syntax : EVEN

4. Procedure definition Directives

(i) PROC : This directive is used to indicate the beginning of a Procedure.
After PROC directive, the term NEAR (or) FAR is used to specify the type of the procedure.

Syntax : Procedure_name PROC NEAR/FAR
Example : Fact PROC NEAR

(ii) ENDP : This directive is used to indicate the ending of a Procedure

Syntax : Procedure_name ENDP
Example : Fact ENDP

5. Macro definition Directives

(i) MACRO : This directive is used to indicate the beginning of a Macro

Syntax : Macro_name MACRO arg1, arg2, ...
Example : Fact MACRO n

(ii) ENDM : This directive is used to indicate the ending of a Macro

Syntax : ENDM

6. Value returning attribute Directives

(i) OFFSET : This directive is used to determine the **offset address** of the variable
Example : MOV SI, OFFSET ARRAY

(ii) TYPE : This directive is used to determine the **Type** of the variable
(1- Byte , 2- Word, 4- Double word, 8- Quad word)
Example : MOV AX, TYPE ARRAY

(iii) LENGTH : This directive is used to determine the **no. of elements** in a data item
Example : MOV AX, LENGTH ARRAY

(iv) SIZE : This directive is used to determine the **no. of bytes** allocated to data item
Example : MOV AX, SIZE ARRAY

(v) SEG : This directive is used to determine the **segment base**, in which the specified data item is defined
Example : MOV AX, SEG ARRAY

7. Data control Directives

(i) PTR : This directive is used to indicate the type of memory access

Example : INC BYTE PTR [BX]
JMP WORD PTR[BX]

(ii) PUBLIC :

- This directive informs the assembler that the data items /procedures declared as PUBLIC can be accessed from any other program module.
- It helps in managing multiple program modules by sharing the global variables

Syntax : PUBLIC Var1, Var2, Var3 ....
Example : PUBLIC N1, N2, ARRAY
PUBLIC fact

(iii) EXTRN :

- This directive informs the assembler that the data items declared after EXTRN have already been defined in some other program module.
- Note that, only PUBLIC variables are accessible when EXTRN is used d variables can be accessed from any other program module.
- It helps in managing multiple program modules by sharing the global variables

Syntax : EXTRN Var1: Type, Var2: Type , Var3: Type ....
Example : EXTRN N1: Word, ARRAY :Word
EXTRN fact: FAR

Example:

```
DATA1   SEGMENT
        N1  DB   25H
        N2  DW  1000H
        ARRAY  DB  10H, 20H, 30H, 40H
DATA1   ENDS

PUBLIC  N1, N2

CODE1   SEGMENT
        ASSUME  CS: CODE1,  DS: DATA1
        -----------
        -----------
        -----------
        -----------
CODE1   ENDS
        END
```

```
DATA2   SEGMENT
        N3  DB  12H
        N4  DW  2000H
        N5  DW  4000H
DATA2   ENDS

EXTRN   N1 : BYTE,   N2 : WORD

CODE2   SEGMENT
        ASSUME  CS: CODE2, DS: DATA2
        ----------
        ----------

        MOV  AL, N1
        MOV  BX, N2
        -----------
        -----------
CODE2   ENDS
        END
```

The data items N1 and N2 are defined in Program module-1 and declared as PUBLIC. These variables can be accessed in Program module-2, by declaring them as EXTRN

Example:

Let us consider a data segment, which is defined as follows

```
DATA        SEGMENT   ORG
            4000 H N1 DW
            1234 H
            ARRAY  DW   1000H, 2000H, 3000H, 4000H
            N2   DW   ?
DATA        ENDS
```

```
(i)   MOV    SI, OFFSET  ARRAY      ; SI  =  4002H
(ii)  MOV    AX,  TYPE    ARRAY     ; AX =  2      (Word Type)
(iii) MOV    AX,  LENGTH ARRAY      ; AX =  4      (No. of  elements)
(iv)  MOV    AX,  SIZE    ARRAY     ; AX =  8      (No. Of bytes)
(v)   MOV    AX,  SEG   ARRAY       ; AX =  Segment Base address of DATA
```

2.5. PROCEDURES

A procedure is a group of instructions that usually perform one task and stored in memory once, but used as often as necessary.

Advantages : (1) saves memory space
(2) makes easier to develop software
(3) we can break large program into several modules and each module can be called from main program

Disadvantages : (1) it takes some time to link from main program to procedure and Procedure to main program
(2) it uses stack memory (to store the Return address)

Two types of Procedures

(i) NEAR PROCEDURE
(ii) FAR PROCEDURE

(i) Near Procedure :

- A procedure which lies in the same code segment is called as NEAR procedure
- Since, the procedure lies in the same code segment, only IP will be modified and CS remains the same. (Intra-segment)
- The near CALL instruction is used to call a near procedure
- The Near CALL instruction performs the following **two actions**
  - It pushes the return address (only IP value) on to the stack
  - It loads IP with the offset address of procedure, so that the flow of execution is transferred to procedure

Ex:
```
CALL NEAR fact
CALL 2000H
CALL WORD PTR[BX]
```

- The RET instruction at the end of the NEAR procedure returns the flow of execution from Procedure to main program. This instruction pops the return address from stack memory.

(i) Far Procedure :

- A procedure which lies in different code segment is called as FAR procedure.
- Since, the procedure lies in different code segment, Both IP and CS will get modified. (Inter-segment)
- The FAR CALL instruction is used to call a FAR procedure

- The Far CALL instruction performs the following **two actions**
  - It pushes the return address (both CS and IP) on to the stack
  - It loads CS and IP with the address of procedure, so that the flow of execution is transferred to procedure

  Ex:
  ```
  CALL FAR fact  CALL
  2000H : 5000H
  CALL  DWORD PTR[BX]
  ```

- The RET instruction at the end of the NEAR procedure returns the flow of execution from Procedure to main program. This instruction pops the return address from stack memory.

Note: The procedures are defined using directives PROC and ENDP
PROC ☾ indicates the beginning of the procedure
ENDP ☾ indicates the ending of the procedure

Example: **Procedure to find the factorial of given number N**

```
fact  PROC  NEAR
       MOV  AL, N            // to find the factorial of N
       MOV CL, AL
       DEC CL
       UP:  MUL CL
            DEC CL
            JNZ  UP
      MOV BX, AX             // Result is copied to BX
      RET
fact  ENDP
```

PASSING PARAMETERS TO/FROM PROCEDURES :

The data values passed from main program to procedure and from procedure to main program are called as parameters. The different methods of passing parameters are

(a) Using registers
(b) Using pointes / general purpose memory
(c) Using stack memory

(a) Using registers ☾ The parameters to be passed are stored in register and then CALL instruction is executed.

```
Ex:     MOV AL, N
        CALL fact
        MOV  RES, BX
```

In above example, the register AL is used to pass the input number N to the procedure and register BX is used to pass the result from procedure to main program.
Using this method, we can't pass more number of parameters.

(b) Using Pointers ☾ In this method, the parameters can be directly accessed from memory using pointers from procedure.

```
Ex:     MOV  SI, OFFSET N
        MOV  DI, OFFSET RES
        CALL fact
```

In above example, before calling the procedure, the SI and DI registers are initialized to set as source and destination pointers. Hence, the parameters can be directly accessed from memory using these pointers.
Using this method, more number of parameters can be passed by incrementing the pointers.

(c) Using Pointers ☾ In this method, the parameters to be passed are pushed onto the stack memory, before calling the procedure.

```
Ex:     PUSH  AX
        CALL fact
        POP  BX
```

In above example, before calling the procedure, the parameters to be passed are pushed on to the stack. In procedure, we can read the parameter from stack by using POP instruction.

2.6. MACROS

- If a group of instructions are repeating again and again in the main program, the listing will be lengthy. **The process of assigning a label (or) macro name to the group of repeated instructions is called Macro.** The macro name is then used throughout the main program to refer that group of instructions.

- **During the assembly process, the assembler generates machine codes for the group of instructions and replaces the macro name with the group of instructions**.

  Advantages : (1) Execution time is less (no branching takes place)
  (2) It doesn't use stack memory

  Disadvantages : (1) Main program length increases, because the macro name is replaced with group of instructions

- The macros are defined using directives MACRO and ENDM
  MACRO ☾ indicates the beginning of the Macro
  ENDM ☾ indicates the ending of the Macro

- The parameters can be directly passed to macro along with macro name

Example(1): Macro to find the factorial of given number N

```
FACT   MACRO  N
         MOV   AL,   N
         MOV   CL,   AL
         DEC CL
           UP:  MUL CL
                DEC CL
          JNZ  UP MOV
                 BX, AX
   ENDM
```

Example(2): Macro to move a string length N of from SRC to DST

```
MOVSTR    MACRO   SRC, DST, N
            MOV   SI,  OFFSET  SRC
            MOV   DI,  OFFSET  DST
            MOV  CX, N
            CLD
            REP :  MOVSB
      ENDM
```

2.7. COMPARISION BETWEEN PROCEDURES & MACROS

S.No.	***PROCEDURES***	***MACROS***
1	Procedure is nothing but branching to a sub-routine	Macros are nothing but substitution of macro name with definition
2	Machine code will be put in memory only once, and called many times from main program	Machine code of macro are added to main program each time the macro is called
3	Program takes up less memory space	Program takes up more memory space
4	Control transfer of flow of execution is required	No control transfer of flow of execution
5	Overhead of using stack for transferring control	No overhead of using stack
6	Execution time is more	Execution time is less
7	Procedures are processed in execution time	Macros are processed in assembling time
8	Assembly time is less	Assembly time is more
9	Procedures are called by CALL instruction	Macros are called by their names
10	The directives PROC and ENP are used to define a procedure. The parameters can be passed to the procedure by using registers, pointers and stack.	The directives MACRO and ENDM are used to define a macro. The parameters can be directly passed to macro along with macro name

2.8. EXAMPLES OF ASSEMBLY LANGUAGE PROGRAMS

(1) Write an ALP to find the average of N- words which are located at ARRAY.

```
DATA        SEGMENT
            N  DW  0005H
            ARRAY DW 1000H, 2000H, 3000H, 4000H, 5000H RES
            DW ?
DATA        ENDS

CODE        SEGMENT
            ASSUME CS:CODE, DS:DATA MOV
            AX, DATA
            MOV DS, AX

            MOV  DX,  0000H
            MOV  AX,  0000H              ; to store the sum
            MOV  CX,  N                  ;  counter
            MOV SI, OFFSET ARRAY         ; address of ARRAY

     UP:    ADD  AX,  DS:[SI]            ; add each number to AX
            JNC down
            INC  DX                      ; increment DX, only if carry

     down :    INC  SI
               INC  SI
               LOOP UP

            MOV BX, N DIV
            BX
            MOV  RES, AX HLT

CODE        ENDS
            END
```

(2) Write a program to move a word from location 2000:5000H to 4000:6000H.

```
CODE    SEGMENT
        ASSUME CS:CODE

        MOV  AX, 2000H          ;  address of source
        MOV DS, AX
        MOV SI, 5000H

        MOV  AX, 4000H          ; address of destination
        MOV ES, AX
        MOV  DI, 6000H

        MOV   AX, DS:[SI]       ; Read data from source
        MOV  ES:[DI], AX        ; Store data at destination
        HLT
CODE    ENDS
        END
```

(3) An array of 16-bit numbers are located at ARRAY1. Find the 2's complement of each word and store them at ARRAY2.

```
CODE    SEGMENT
        ASSUME CS:CODE

        MOV   SI,   OFFSET   ARRAY1
        MOV   DI,   OFFSET   ARRAY2
        MOV CX, N

    UP: MOV   AX, [SI]       ;  Read number

        NOT   AX             ;  Find 1's complement
        INC   AX             ;  Find 2's complement

        MOV [DI], AX         ; Store result

        INC SI
        INC SI INC
        DI INC DI
        LOOP UP
        HLT
CODE    ENDS
        END
```

(4) An array of 8-bit numbers are located at ARRAY. Write an ALP to separate the ODD and EVEN numbers. Store the ODD numbers at ARRAY2 and EVEN numbers at at ARRAY3.

```
CODE    SEGMENT
        ASSUME CS:CODE

        MOV SI,  OFFSET  ARRAY          ; input array address
        MOV BX,  OFFSET ARRAY2          ; to store EVEN numbers
        MOV DI, OFFSET ARRAY3           ; to store ODD numbers
        MOV CX, N                       ; no. of elements

   UP:  MOV   AL,  DS:[SI]              ; Read the number

        TEST  AL, 01H
        JNZ  odd_num

        MOV  DS:[BX] AL                 ; Store Even number
        INC DI
        JMP down

   odd_num : MOV DS:[DI],  AL           ; Store the Odd number
             INC BX

   down:  INC SI
        LOOP UP HLT

CODE    ENDS
        END
```

Note: To check whether the given word is Positive (or) Negative

Method(1) : By using TEST instruction : TEST AX, 8000H

- The TEST instruction performs AND operation. Here only flags are affected and result is not stored anywhere.
- The given word is ANDed with data 8000H

  if the result is zero (ZF=1) then, the given number is Positive number

Method(2) : By using CMP instruction : CMP AX, 8000H

- The given word is compared with data 8000H

  If given number < 8000H (i.e., CF=1) it is a positive number

Method(3) : By using RCL instruction : RCL AX, 1

- For negetive numbers,MSB=1
- By performing Rotate Left with carry operation, we can move the MSB to CF
- After RCL operation, if CF=1 then, it is a negative number

(5) Write a program to count the number of positive numbers and negative numbers in a given series of signed numbers

```
CODE      SEGMENT
          ASSUME CS:CODE

          MOV SI, OFFSET ARRAY
          MOV CX, N

          MOV BX, 0000H         ; to count number of positive numbers
          MOV DX,0000H          ; to count number of negative numbers

     UP:  MOV  AL, DS:[SI]

          TEST AL, 80H
          JNZ  odd_num

          INC BX
          JMP down

     odd_num : INC DX

     down:  INC SI

          LOOP UP
          HLT

CODE      ENDS
          END
```

(6) Write an 8086 ALP to find the largest number in given array of N-numbers

```
CODE      SEGMENT
          ASSUME CS:CODE

          MOV  SI, OFFSET  ARRAY MOV
          CX, N
          MOV  AL,00H           ;   to store the largest number

     UP:  CMP  AL, [SI]
          JNC  down

          MOV  AL, [SI]

     down:  INC SI
          LOOP UP

          HLT
CODE      ENDS
          END
```

(7) Write an 8086 assembly language program to generate fibonacci series

```
CODE    SEGMENT
        ASSUME CS:CODE

        MOV DI, OFFSET RES ;  to store the fibonacci series MOV
        CX, N

        MOV AL, 00H MOV
        BL, 01H

UP:     ADD AL, BL
        MOV [DI], AL
        MOV AL, BL          ; copy previous value to AL
        MOV BL, [DI]        ; copy present value to BL
        INC DI
        LOOP UP

        HLT
CODE    ENDS
        END
```

(8) Write an 8086 ALP to search a number N in given array of 10-numbers. If the number is found, store 1111H in AX. Otherwise store 0000H in AX

```
CODE    SEGMENT
        ASSUME CS:CODE

        MOV DI, OFFSET   ARRAY
        MOV CX, 000AH              ; Counter = 10 numbers

        MOV AL, N                  ; Number to be searched

UP:     CMP AL, [DI]
        JZ down

        INC DI
        LOOP UP
        MOV AX,0000H
        JMP exit

down:  MOV AX,1111H

exit   : HLT

CODE    ENDS
        END
```

(9) Write an 8086 ALP to search a character 'R' in the given string. If the character is found, store 1111H in AX. Otherwise store 0000H in AX

```
CODE    SEGMENT
        ASSUME CS:CODE

        MOV  DI,  OFFSET  STRING
        MOV CX,  SIZE  STRING           ;  Counter

        MOV AL,  'R'                    ; Character to be searched

REPNE :  SCASB
         JE down

        MOV  AX,0000H
        JMP  exit   down:

MOV AX,1111H

exit    : HLT

CODE    ENDS
        END
```

(10) Write an 8086 ALP to count the number of characters 'R' in the given string.

```
CODE    SEGMENT
        ASSUME CS:CODE

        MOV  DI,  OFFSET  STRING
        MOV CX,  SIZE  STRING           ;  Counter

        MOV  AL,   'R'                  ;  Character  to be searched
        MOV  BX,  0000H                 ;  To store the count value

UP:     NOP
        SCASB
        JNE down

        INC  BX                  ; increment BX only if 'R' is available

down: LOOP UP

        HLT

CODE    ENDS
        END
```

(11) Write an 8086 assembly language program to move a string of length of 8-bytes from 'OLD_HOME' to 'NEW_HOME'

```
CODE        SEGMENT
            ASSUME CS:CODE

            MOV SI, OFFSET OLD_HOME MOV
            DI, OFFSET NEW_HOME MOV CX,
            N
            CLD REP:

        MOVSB

            HLT
CODE        ENDS
            END
```

(12) Write a Procedure to move a string from 'OLD_HOME' to 'NEW_HOME'

```
movstr      PROC  NEAR

            MOV SI, OFFSET OLD_HOME MOV
            DI, OFFSET NEW_HOME MOV CX,
            N
            CLD

        REP:  MOVSB
            RET

movstr      ENDP
```

(13) Write a Macro to move a string from 'OLD_HOME' to 'NEW_HOME'

```
MOVSTR      MACRO  OLD_HOME, NEW_HOME, N MOV

            SI, OFFSET OLD_HOME
            MOV DI, OFFSET NEW_HOME MOV
            CX, N
            CLD REP:

        MOVSB

ENDM
```

(14) Write an Assembly language program to reverse a string of length 5 bytes

```
CODE    SEGMENT
        ASSUME CS:CODE

        MOV  SI,  OFFSET  ARRAY1        ; address of source string
        MOV  DI, OFFSET  ARRAY2         ;  address of result
        MOV CX, N                       ; length of the string
        CLD
        ADD DI, CX

    UP: MOV  AL,   [SI]                 ; Read a byte from [SI]
        MOV [DI],  AL                   ; Store the byte at [DI]
        INC SI
        DEC  DI
        LOOP UP

        HLT
CODE    ENDS
        END
```

(15) Write a Program to compare two strings, which are located at ARRAY1 and ARRAY2 If two strings are equal than store 1111H in AX , otherwise store 0000H in AX

```
CODE    SEGMENT
        ASSUME CS:CODE

        MOV   SI,   OFFSET   ARRAY1
        MOV   DI,   OFFSET   ARRAY2
        MOV CX, N
        CLD

        MOV AX,1111H

  REPE: CMPSB
        JE down

        MOV AX,0000H                    ; if not equal only

 down : NOP
        HLT
CODE    ENDS
        END
```

(16) Write a Program to check whether the given string is Palindrome (or) not?

Combine – Reversing a string and Comparison of two strings

(17) Write an ALP to sort the given series of numbers in ascending order

```
CODE        SEGMENT
            ASSUME  CS:CODE

            MOV CX, N                    ; counter

     BACK : MOV BX, N
            DEC BX
            MOV  SI, OFFSET ARRAY        ; address of array

       UP  : MOV AL,[SI]
            INC SI
            CMP  AL,[SI]
            JB down

            XCHG AL,[SI] DEC
             SI
            MOV [SI],AL
            INC SI

     down:  DEC  BX
            JNZ UP LOOP
            BACK

            HLT
CODE        ENDS
            END
```

(18) Write an ALP to covert the given BCD number into equivalent Binary.
The BCD number is available at BCD_IN and store the result at BIN_OUT.

(19) Write an Aeembly language Program to find the value of ${}^{n}C_{r}$
using NEAR procedure

UNIT III

8086 Interfacing

MEMORY AND I/O INTERFACING

I/O Interface

Any application of a microprocessor system requires the transfer of data between microprocessor and external environment and also with in the microprocessor. This is known as Input/Output. There are three different ways that the data transfer can take place. They are

(1) Program controlled I/O
(2) Interrupt Program Controlled I/O
(3) Hardware controlled I/O

In program controlled I/O data transfer scheme the transfer of data is completely under the control of the microprocessor program. In this case an I/O operation takes place only when an I/O transfer instruction is executed.

In an interrupt program controlled I/O an external device indicates directly to the microprocessor its readiness to transfer data by a signal at an interrupt input of the microprocessor. When microprocessor receives this signal the control is transferred to ISS (Interrupt service subroutine) which performs the data transfer.

Hardware controlled I/O is also known as direct memory access DMA. In this case the data transfer takes place directly between an I/O device and memory but not through microprocessors. Microprocessor only initializes the process of data transfer by indicating the starting address and the number of words to be transferred.

The instruction .set of any microprocessor contains instructions that transfer information to an I/O device and to read information from an I/O device. In 8086 we have IN, OUT instructions for this purpose. OUT instruction transfers information to an I/O device where as IN instruction is used to read information from an I/O device. Both the instructions perform the data transfer using accumulator AL or AX. The I/O address is stored in register DX.

The port number is specified along with IN or OUT instruction. The external I/O interface decodes to find the address of the I/O device. The 8 bit fixed port number appears on address bus A_0 - A_7 with A_8 - A_{15} all zeros. The address connections above A_{15} are
undefined for an I/O instruction. The 16 bit variable port number appears on address connections A_0 - A_{15}. The above notation indicates that first 256 I/O port addresses 00 to FF are accessed by both the fixed and variable I/O instructions. The I/O addresses from 0000 to FFFF are accessed by the variable
I/O address.

I/O devices can be interfaced to the microprocessors using two methods. They are I/O mapped I/O and memory mapped I/O. The I/O mapped I/O is also known as isolated I/O or direct I/O. In I/O mapped I/O the IN and OUT instructions transfer data between the accumulator or memory and I/O device. In memory mapped I/O the instruction that refers memory can perform the data transfer.

I/O mapped I/O is the most commonly used I/O transfer technique. In this method I/O locations are placed separately from memory. The addresses for isolated I/O devices are separate from memory. Using this method user can use the entire memory. This method allows data transfer only by using instructions IN, OUT. The pins M/ IO and W/R are used to indicate I/O read or an I/O write operations. The signals on these lines indicate that the address on the address bus is for I/O devices.

Memory mapped I/O does not use the IN, OUT instruction it uses only the instruction that transfers data between microprocessor and memory. A memory mapped I/O device is treated as memory location. The disadvantage in this system is the overall memory is reduced. The advantage of this system is that any memory transfer instruction can be used for data transfer and control signals like I/O read and I/O write are not necessary which simplify the hardware.

Memory interfacing

Memory is an integral part of a microcomputer system. There are two main types of memory.

(i) **Read only memory (ROM):** As the name indicates this memory is available only for reading purpose. The various types available under this category are PROM, EPROM, EEPROM which contain system software and permanent system data.

(ii) **Random Access memory (RAM):** This is also known as Read Write Memory. It is a volatile memory. RAM contains temporary data and software programs generally for different applications.

While executing particular task it is necessary to access memory to get instruction codes and data stored in memory. Microprocessor initiates the necessary signals when read or write operation is to be performed. Memory device also requires some signals to perform read and write operations using various registers. To do the above job it is necessary to have a device and a circuit, which performs this task is known as interfacing device and as this is involved with memory it-is known as memory interfacing device. The basic concepts of memory interfacing involve three different tasks. The microprocessor should be able to read from or write into the specified register. To do this it must be able to select the required chip, identify the required register and it must enable the appropriate buffers.

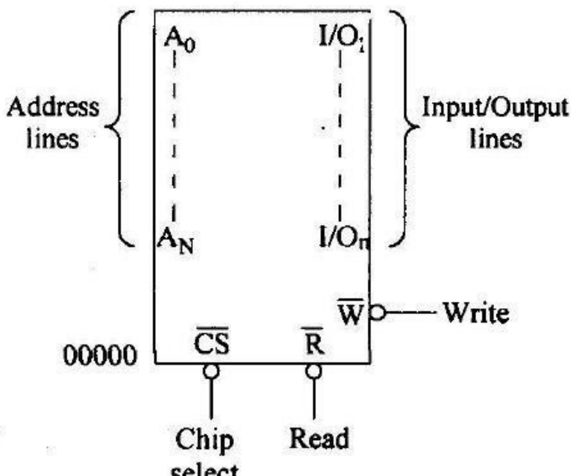


Fig. 3.13 Simple memory device

Any memory device must contain address lines and Input, output lines, selection input, control input to perform read or write operation. All memory devices have address inputs that select memory location with in the memory device. These lines are labeled as A_0 A_N. The number of address lines indicates the total memory capacity of the memory device. A 1K memory requires 10 address lines A_0-A_9. Similarly a 1MB requires 20 lines A_0-A_{19} (in the case of 8086). The memory devices may have separate I/O lines or a common set of bidirectional I/O lines. Using these lines data can be transferred

in either direction. Whenever output buffer is activated the operation is read whenever input buffers are activated the operation is write. These lines are labelled as I/O,......... I/O_n or D_O D_n. The size of a memory location is dependent upon the number of data bits. If the number of data lines are eight D_0 - D_7 then 8 bits or 1 byte of data can be stored in each location. Similarly if numbers of data bits are 16 (D_0 - D_{15}) then the memory size is 2 bytes. For example 2K x 8 indicates there are 2048 memory locations and each memory location can store 8 bits of data.

Memory devices may contain one or more inputs which areused to select the memory device or to enable the memory device. This pin is denoted by $\overline{CS}$ (Chip select) or $\overline{CE}$ (Chip enable). When this pin is at logic '0' then only the memory device performs a read or a write operation. If this pin is at logic '1' the memory chip is disabled. If there are more than one CS input then all these pins must be activated to perform read or write operation.

All memory devices will have one or more control inputs. When ROM is used we find $\overline{OE}$ output enable pin which allows data to flow out of the output data pins. To perform this task both $\overline{CS}$ and $\overline{OE}$ must be active. A RAM contains one or two control inputs. They are $\overline{R}$ / $\overline{W}$ or $\overline{RD}$ and $\overline{WR}$. If there is only one input R/ $\overline{W}$ then it performs read operation when R/ $\overline{W}$ pin is at logic 1. If it is at logic 0 it performs write operation. Note that this ispossible only when $\overline{CS}$ is also active.

Memory Interface using RAMS, EPROMS and EEPROMS

(Ref: Advanced Microprocessors and Peripherals by A.K. Ray & K.M. Bhurchandi, McGraw-Hill, 2nd Edition.P.158- 164)

Semiconductor Memory Interfacing:

Semiconductor memories are of two types, viz. RAM (Random Access Memory) and ROM (Read Only Memory).

Static RAM Interfacing:

The semiconductor RAMs are of broadly two types-static RAM and dynamic RAM. The semiconductor memories are organised as two dimensional arrays of memory locations. For example, 4K x 8 or 4K byte memory contains 4096 locations, where each location contains 8-bit data and only one of the 4096 locations can be selected at a time. Obviously, for addressing 4K bytes of memory, twelve address lines are required. In general, to address a memory location out of N memory locations, we will require at least *n* bits of address, i.e. *n* address lines where $n = \text{Log}_2 N$. Thus if the microprocessor has *n* address lines, then it is able to address at the most N locations of memory, where $2^n = N$. However, if out of *N* locations only *P* memory locations are to be interfaced, then the least significant *p* address lines out of the available *n* lines can be directly connected from the microprocessor to the memory chip while the remaining *(n-p)* higher order address lines may be used for address decoding (as inputs to the chip selection logic). The memory address depends upon the hardware circuit used for decoding the chip select ($\overline{CS}$). The output of the decoding circuit is connected with the $\overline{CS}$ pin of the memory chip. The general procedure of static memory interfacing with 8086 is briefly described as follows:

1. Arrange the available memory chips so as to obtain 16-bit data bus width. The upper 8-bit bank is called 'odd address memory bank' and the lower 8-bit bank is called 'even address memory bank'.
2. Connect available memory address lines of memory chips with those of the microprocessor and also connect the memory $\overline{RD}$ and $\overline{WR}$ inputs to the corresponding processor control signals.Connect the 16-bit data bus of the memory bank with that of the microprocessor 8086.
3. The remaining address lines of the microprocessor, $\overline{BHE}$ and A_0 are used for decoding the required chip select signals for the odd and even memory banks. $\overline{CS}$ of memory is derived from theO/P of the decoding circuit.

As a good and efficient interfacing practice, the address map of the system should be continuous as far as possible, i.e. there should be no windows in the map. A memory location should have a single address corresponding to it, i.e. absolute decoding should be preferred, and minimum hardware should be used for decoding. In a number of cases, linear decoding may be used to minimise the required hardware.Let us now consider a few example problems on memory interfacing with 8086.

Problem 5.1

Interface two 4K × 8 EPROMS and two 4K × 8 RAM chips with 8086. Select suitable maps.

Solution We know that, after reset, the IP and CS are initialised to form address FFFF0H. Hence, this address must lie in the EPROM. The address of RAM may be selected any where in the 1MB address space of 8086, but we will select the RAM address such that the address map of the system is continuous, as shown in Table 5.1.

Table 5.1 *Memory Map for Problem 5.1*

Address	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
							EPROM						8K × 8							
FE000H	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
FDFFFH	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
							RAM					8K × 8								
FC000H	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Total 8K bytes of EPROM need 13 address lines $A_0 - A_{12}$ (since $2^{13} = 8K$). Address lines $A_{13} - A_{19}$ are used for decoding to generate the chip select. The $\overline{BHE}$ signal goes low when a transfer is at odd address or higher byte of data is to be accessed. Let us assume that the latched address, $\overline{BHE}$ and demultiplexed data lines are readily available for interfacing. Figure 5.1 shows the interfacing diagram for the memory system.

The memory system in this example contains in total four 4K × 8 memory chips.

The two 4K × 8 chips of RAM and ROM are arranged in parallel to obtain 16-bit data bus width. If A_0 is 0, i.e. the address is even and is in RAM, then the lower RAM chip is selected indicating 8-bit transfer at an even address. If A_0 is 1, i.e. the address is odd and is in RAM, the $\overline{BHE}$ goes low, the upper RAM chip is selected, further indicating that the 8-bit transfer is at an odd address. If the selected addresses are in ROM, the respective ROM chips are selected. If at a time A_0 and $\overline{BHE}$ both are 0, both the RAM or ROM chips are selected, i.e. the data transfer is of 16 bits. The selection of chips here takes place as shown in Table 5.2.

Table 5.2 *Memory Chip Selection for Problem 5.1*

Decoder I/P → Address/$\overline{BHE}$ →	A_2 A_{13}	A_1 A_0	A_0 $\overline{BHE}$	Selection/ Comment
Word transfer on $D_0 - D_{15}$	0	0	0	Even and odd addresses in RAM
Byte transfer on $D_7 - D_0$	0	0	1	Only even address in RAM
Byte transfer on $D_8 - D_{15}$	0	1	0	Only odd address in RAM
Word transfer on $D_0 - D_{15}$	1	0	0	Even and odd addresses in ROM
Byte transfer on $D_0 - D_7$	1	0	1	Only even address in ROM
Byte transfer on $D_8 - D_{15}$	1	1	0	Only odd address in ROM

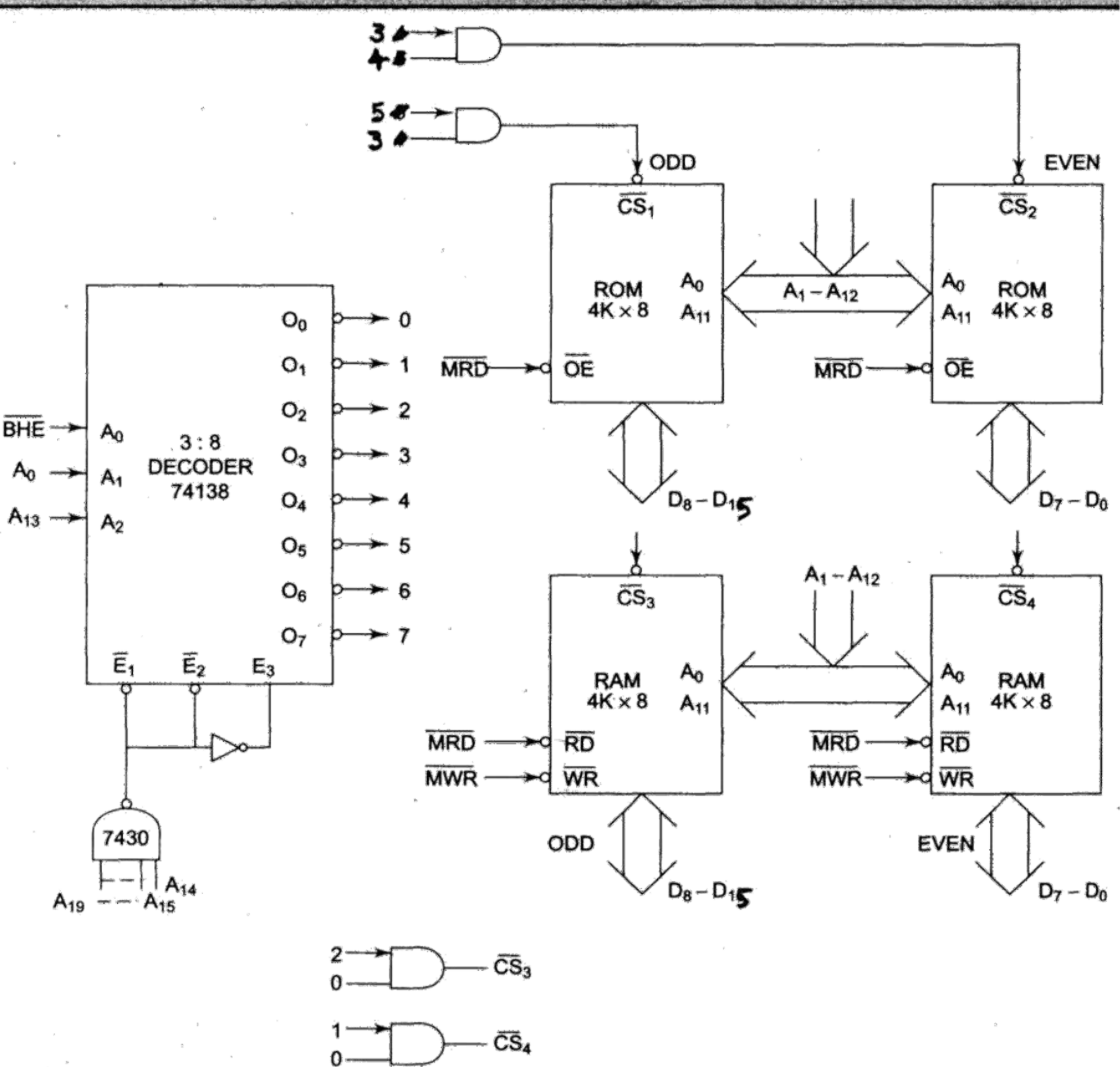


Fig. 5.1 *Interfacing Problem 5.1*

Problem 5.2

Design an interface between 8086 CPU and two chips of 16K × 8 EPROM and two chips of 32K × 8 RAM. Select the starting address of EPROM suitably. The RAM address must start at 00000H.

Solution: The last address in the map of 8086 is FFFFFH. After resetting, the processor starts from FFFF0H. Hence this address must lie in the address range of EPROM. Figure 5.2 shows the interfacing diagram, and Table 5.3 shows complete map of the system.

Table 5.3 *Address Map for Problem 5.2*

Addresses	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
							32KB			EPROM										
F8000H	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0FFFFH	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
						64KB RAM														
00000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

It is better not to use a decoder to implement the above map because it is not continuous, i.e. there is some unused address space between the last RAM address (0FFFFH) and the first EPROM address (F8000H). Hence the logic is implemented using logic gates, as shown in Fig. 5.2.

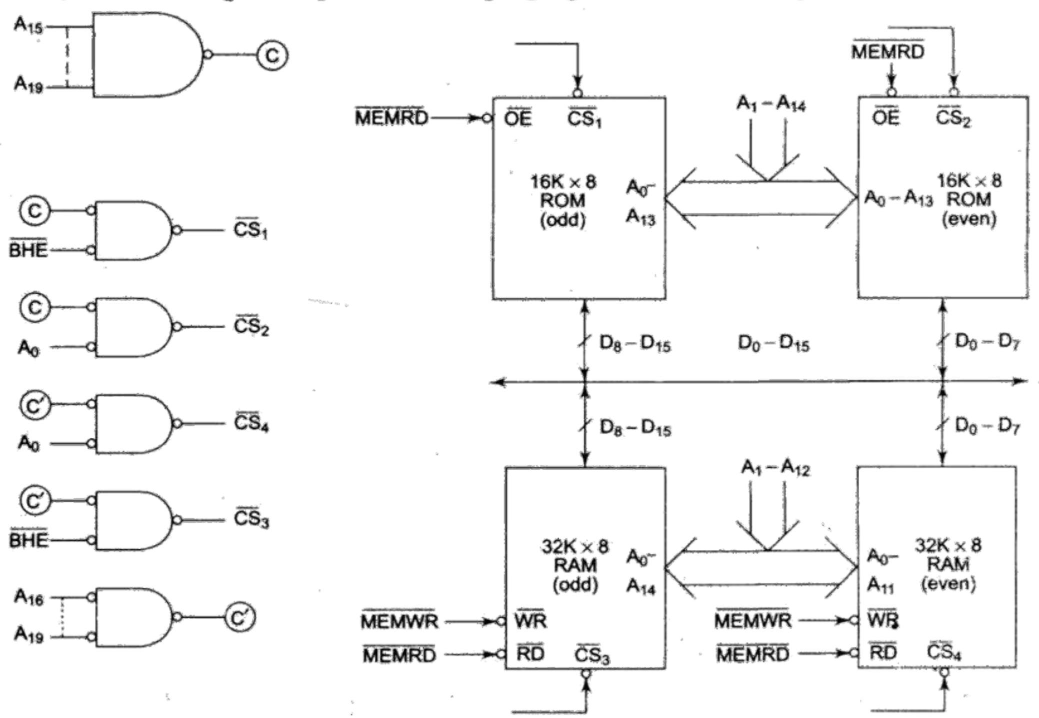


Fig. 5.2 *Interfacing Problem 5.2*

Problem 5.3

It is required to interface two chips of 32K × 8 ROM and four chips of 32K × 8 RAM with 8086, according to the following map.

```
ROM 1 and 2 F0000H - FFFFFH, RAM 1 and 2 D0000H - DFFFFH
RAM 3 and 4 E0000H - EFFFFH
```

Show the implementation of this memory system.

Solution Let us write the memory map of the system as shown in Table 5.6.

The implementation of the above map is shown in Fig. 5.3 using the same technique as in Problem 5.1 and Problem 5.2. All the address, data and control signals are assumed to be readily available.

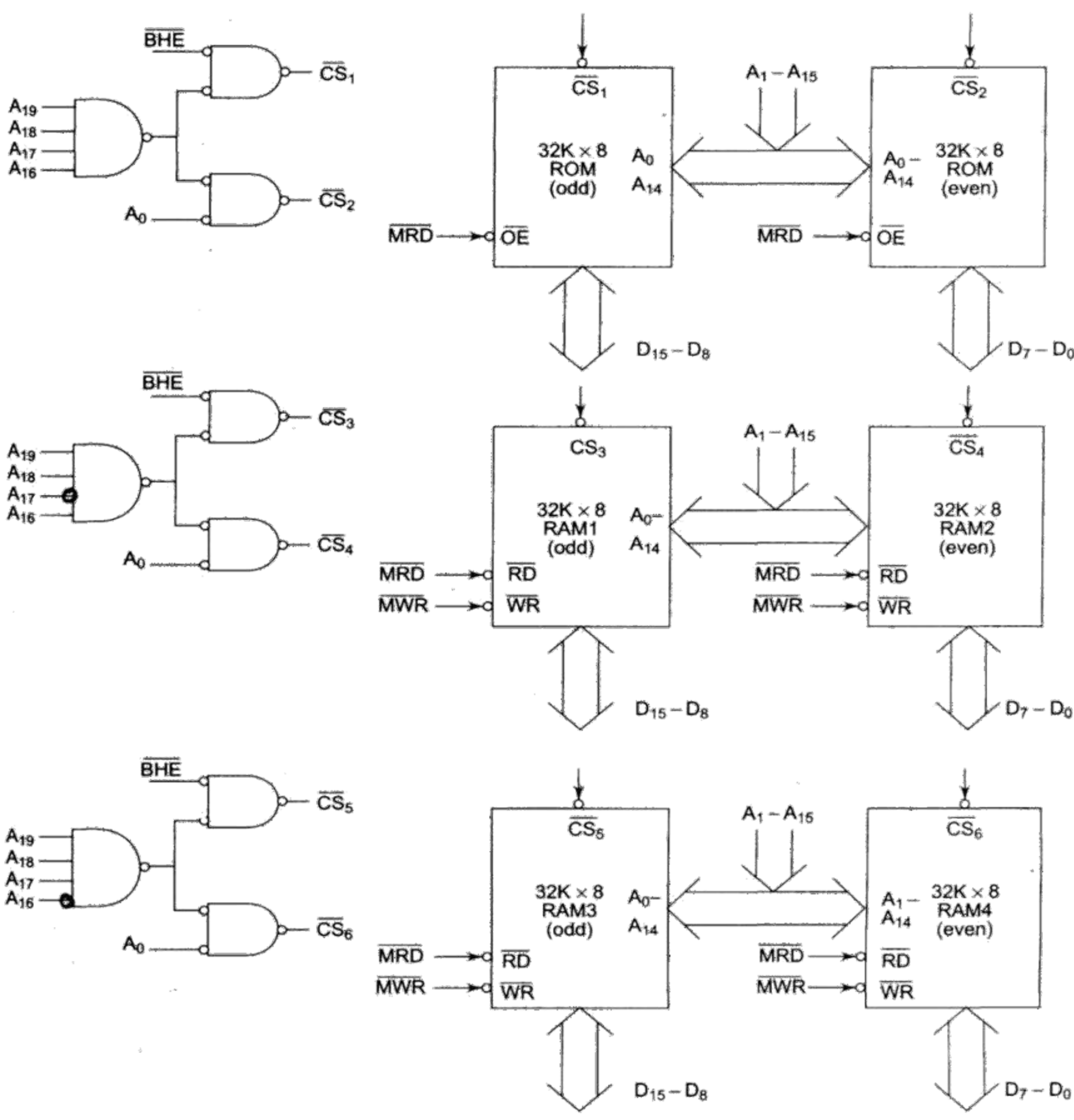


Fig. 5.3 *Interfacing Problem 5.3*

PIO 8255:

The parallel input-output port chip 8255 is also called as programmable **peripheral input-output port.** The Intel‟s 8255 are designed for use with Intel‟s 8-bit, 16-bit and higher capability microprocessors. It has 24 input/output lineswhich may be individually programmed in two groups of twelve lines each, orthree groups of eight lines.

The two groups of I/O pins are named as Group A and Group B. Each of thesetwo groups contains a subgroup of eight I/O lines called as 8-bit port and anothersubgroup of four lines or a 4-bit port. Thus Group A contains an 8-bit port Aalong with a 4-bit port C upper.

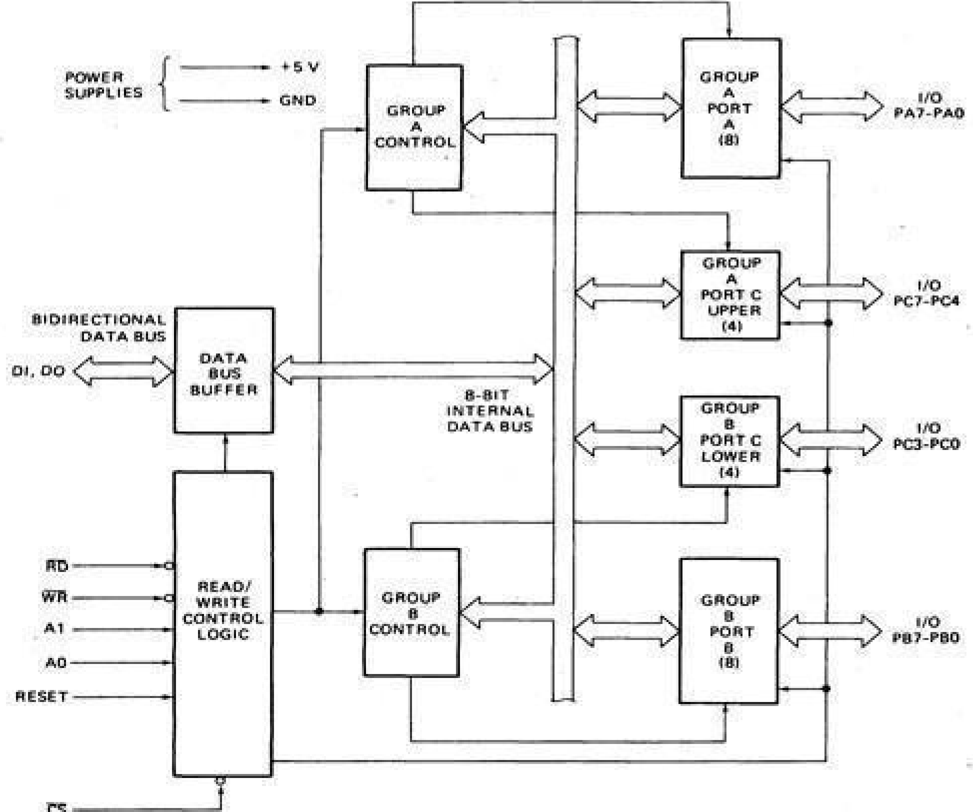


FIGURE Internal block diagram of 8255A programmable parallel port device. (*Intel Corporation*)

The port A lines are identified by symbols PA0-PA7 while the port C lines are identified as PC4-PC7 similarly. Group B contains an 8-bit port B, containing lines PB0-PB7 and a 4-bit port C with lower bits PC0-PC3. The port C upper and port C lower can be used in combination as an 8-bit port C. Both the port Cs is assigned the same address. Thus one may have either three 8-bit I/O ports or two 8-bit and two 4-bit I/O ports from 8255. All of these ports can function independently either as input or as output ports. This can be achieved by programming the bits of an internal register of 8255 called as control word register (CWR). The internal block diagram and the pin configuration of 8255 are shown in figs.

The 8-bit data bus buffer is controlled by the read/write control logic. The read/write control logic manages all of the internal and external transfer of both data and control words. RD, WR, A1, A0 and RESET are the inputs, provided by the microprocessor to READ/WRITE control logic of 8255. The 8-bit, 3-state bidirectional buffer is used to interface the 8255 internal data bus with the external system data bus. This buffer receives or transmits data upon the execution of input or output instructions by the microprocessor. The control words or status information is also transferred through the buffer.

Pin Diagram of 8255A

8255A Pin Configuration

The pin configuration of 8255 is shown in fig.

- The port A lines are identified by symbols PA0-PA7 while the port C lines are
- Identified as PC4-PC7. Similarly, Group B contains an 8-bit port B, containing lines PB0-PB7 and a 4-bit port C with lower bits PC0- PC3. The port C upper and port C lower can be used in combination as an 8-bit port C.
- Both the port C is assigned the same address. Thus one may have either three 8-bit I/O ports or two 8-bit and two 4-bit ports from 8255. All of these ports can function independently either as input or as output ports. This can be achieved by programming the bits of an internal register of 8255 called as control word register (CWR).

The 8-bit data bus buffer is controlled by the read/write control logic. The read/write control logic manages all of the internal and external transfers of both data and control words.

RD,WR, A1, A0 and RESET are the inputs provided by the microprocessor to the READ/ WRITE control logic of 8255. The 8-bit, 3-state bidirectional buffer is used to interface the 8255 internal data bus with the external system data bus.

This buffer receives or transmits data upon the execution of input or output instructions by the microprocessor. The control words or status information is also transferred through the buffer.

The signal description of 8255 is briefly presented as follows:

PA7-PA0: 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.

PC7-PC4: 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 mode1 or mode2.

PC3-PC0: These are the lower port C lines; other details are the same as PC7-PC4 lines.

PB0-PB7: 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.

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

WR: This is an input line driven by the microprocessor. A low on this line indicates write operation.

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.

D0-D7: These are the data bus lines those carry data or control word to/from the microprocessor.

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

A1-A0: These are the address input lines and are driven by the microprocessor.

These lines A1-A0 with RD, WR and CS from the following operations for 8255. These address lines are used for addressing any one of the four registers, i.e. three ports and a control word register as given in table below.

In case of 8086 systems, if the 8255 is to be interfaced with lower order data bus, the A0 and A1 pins of 8255 are connected with A1 and A2 respectively.

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	Input (Read) cycle
0	1	0	0	0	Port A to Data bus
0	1	0	0	1	Port B to Data bus
0	1	0	1	0	Port C to Data bus
0	1	0	1	1	CWR to Data bus

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	Output (Write) cycle
1	0	0	0	0	Data bus to Port A
1	0	0	0	1	Data bus to Port B
1	0	0	1	0	Data bus to Port C
1	0	0	1	1	Data bus to CWR

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	Function
X	X	1	X	X	Data bus tristated
1	1	0	X	X	Data bus tristated

Control Word Register

Modes of Operation of 8255

These are two basic modes of operation of 8255. I/O mode and Bit Set-Reset mode (BSR).

In I/O mode, the 8255 ports work as programmable I/O ports, while in BSR mode only port C (PC0-PC7) can be used to set or reset its individual port bits.

Under the I/O mode of operation, further there are three modes of operation of 8255, so as to support different types of applications, mode 0, mode 1 and mode 2.

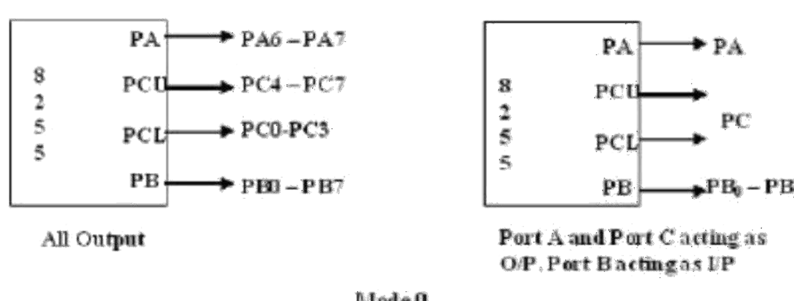
BSR Mode: In this mode any of the 8-bits of port C can be set or reset depending on D0 of the control word. The bit to be set or reset is selected by bit select flags D3, D2 and D1 of the CWR as given in table.

I/O Modes:

a) Mode 0 (Basic I/O mode): This mode is also called as basic input/output Mode. This mode provides simple input and output capabilities using each of the three ports. Data can be simply read from and written to the input and output ports respectively, after appropriate initialization.

D_3	D_2	D_1	Selected bits of port C
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

BSR Mode : CWR Format



Mode 0

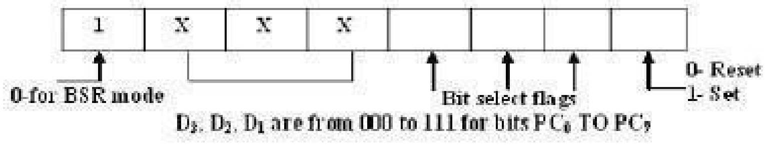
The salient features of this mode are as listed below:

1. Two 8-bit ports (port A and port B) and two 4-bit ports (port C upper and lower) are available. The two 4-bit ports can be combined used as a third 8-bit port.
2. Any port can be used as an input or output port.
3. Output ports are latched. Input ports are not latched.
4. A maximum of four ports are available so that overall 16 I/O configurations arepossible.

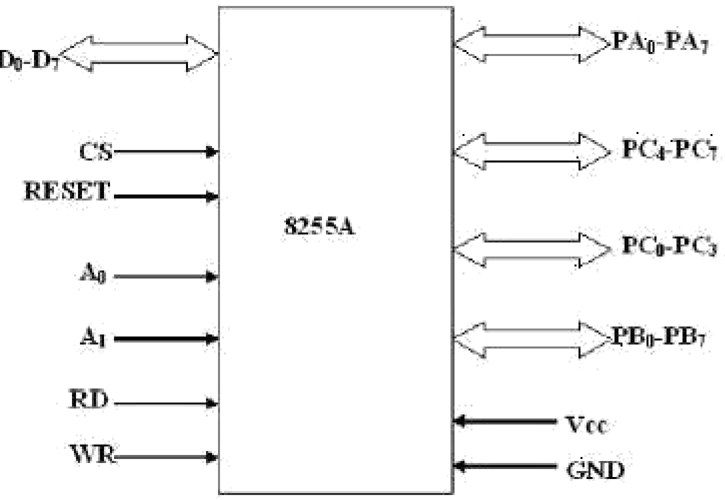
All these modes can be selected by programming a register internal to 8255known as CWR.

The control word register has two formats. The first format is valid for I/O modesof operation, i.e. modes 0, mode 1 and mode 2 while the second format is validfor bit set/reset (BSR) mode of operation.

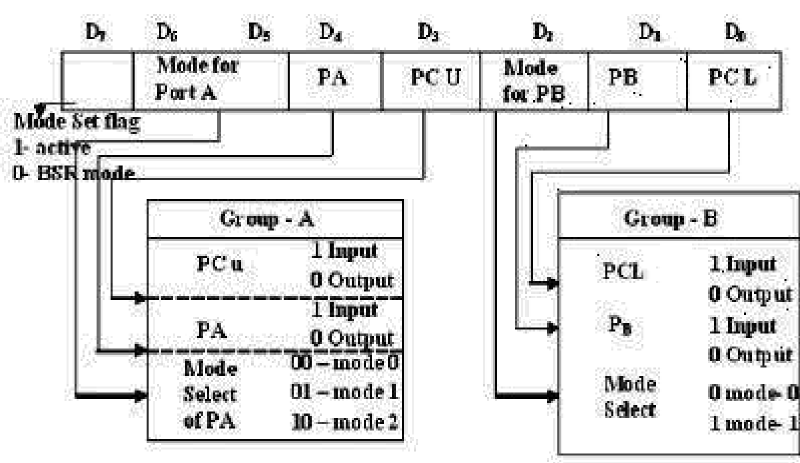
These formats are shown in followingfig.



I/O Mode Control Word Register Format and
BSR Mode Control Word Register Format



Signals of 8255



Control Word Format of 8255

b) Mode 1: (Strobed input/output mode) In this mode the handshaking control the input and output action of the specified port. Port C lines PC0-PC2, provide strobe orhandshake lines for port B. This group which includes port B and PC0-PC2 is called asgroup B for Strobed data input/output. Port C lines PC3-PC5 provides strobe lines for portA.This group including port A and PC3-PC5 from group A. Thus port C is utilized forgenerating handshake signals.

The salient features of mode 1 are listed as follows:

1. Two groups – group A and group B are available for strobed data transfer.
2. Each group contains one 8-bit data I/O port and one 4-bit control/data port.
3. The 8-bit data port can be either used as input and output port. The inputs andoutputs both are latched.
4. Out of 8-bit port C, PC0-PC2 are used to generate control signals for port B andPC3-PC5 are used to generate control signals for port A. the lines PC6, PC7 may be used as independent data lines.

The control signals for both the groups in input and output modes are explained as follows:

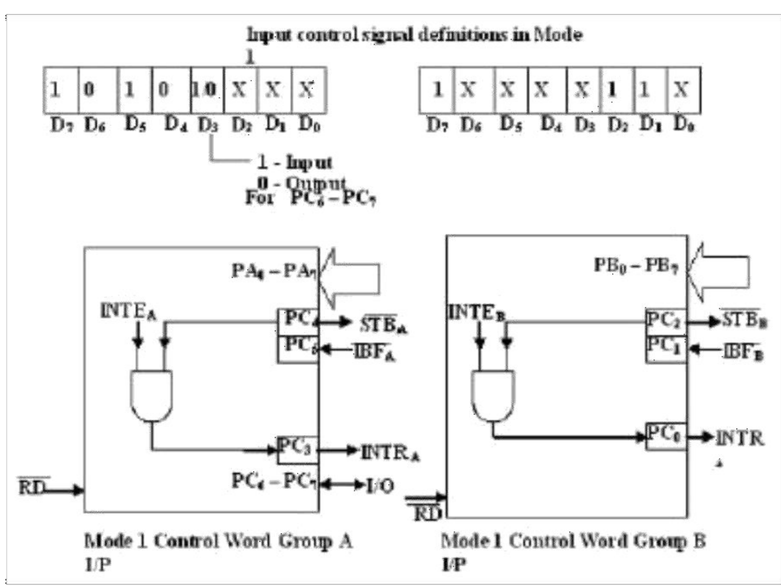
Input control signal definitions (mode 1):

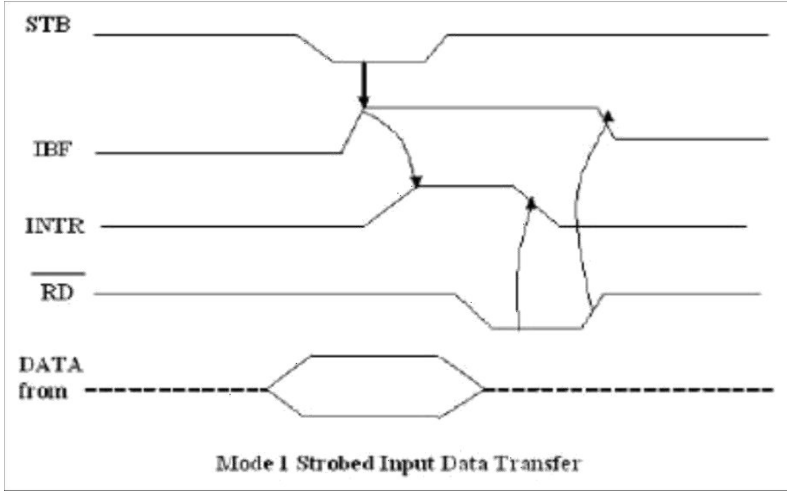
- **STB** (Strobeinput) – If this lines falls to logic low level, the data available at 8-bit input port is loaded into input latches.
- **IBF** (Input buffer full) – If this signal rises to logic 1, it indicates that data hasbeen loaded into latches, i.e. it works as an acknowledgement. IBF is set by a lowon STB and is reset by the rising edge of RD input.
- **INTR** (Interruptrequest) – This active high output signal can be used tointerrupt the CPU whenever an input device requests the service. INTR is set by ahigh STBpin and a high at IBF pin. INTE is an internal flag that can be controlledby the bit set/reset mode of either PC4 (INTEA) or PC2 (INTEB) as shown in fig.
- INTR is reset by a falling edge of RD input. Thus an external input device can berequest the service of the processor by putting the data on the bus and sending thestrobe signal.

Output control signal definitions (mode 1):

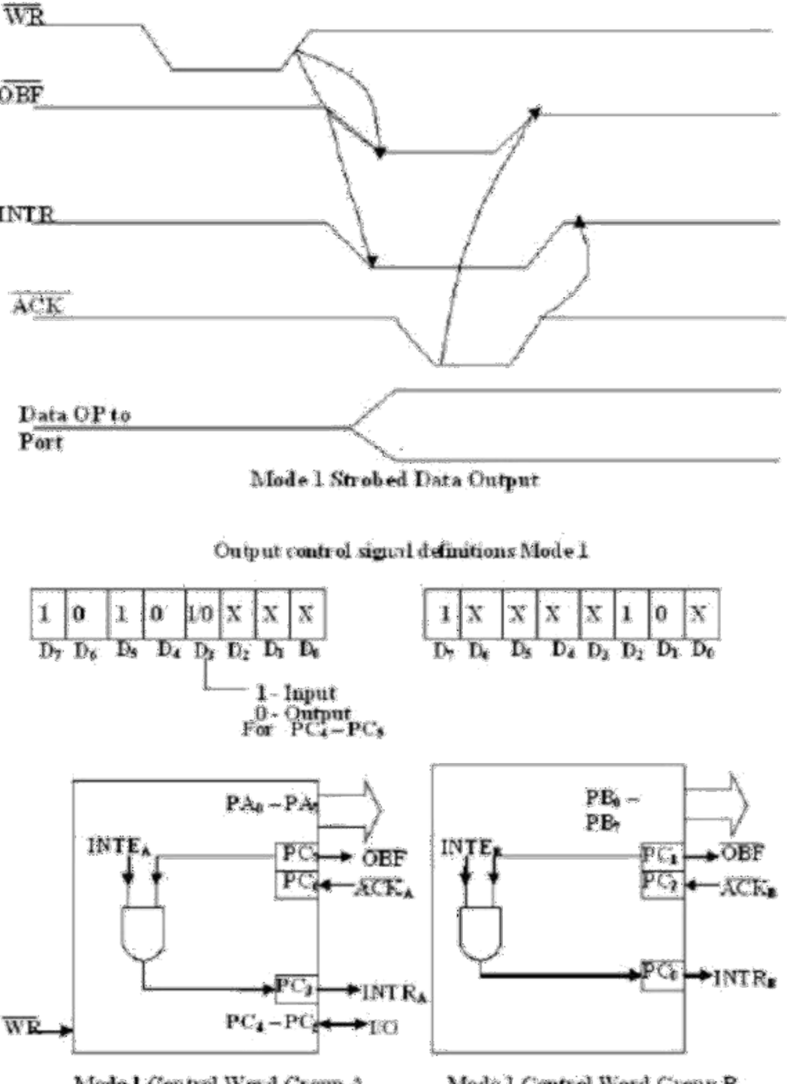
- **OBF** (Output buffer full) – This status signal, whenever falls to low, indicatesthat CPU has written data to the specified output port. The OBF flip-flop will beset by a rising edge of WR signal and reset by a low going edge at the ACKinput.
- ACK (Acknowledgeinput) – ACK signal acts as an acknowledgement to begiven by an output device. ACK signal, whenever low, informs the CPU that thedata transferred by the CPU to the output device through the port is received bythe output device.
- **INTR** (Interruptrequest) – Thus an output signal that can be used to interruptthe 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 bitset-reset mode ofPC6 and PC2 respectively.





Mode 1 Strobed Input Data Transfer



c) Mode 2 (Strobed bidirectional I/O): This mode of operation of 8255 is also called as strobed bidirectional I/O. This mode of operation provides 8255 with additional features for communicating with a peripheral device on an 8-bit data bus. Handshaking signals are provided to maintain proper data flow and synchronization between the data transmitter and receiver. The interrupt generation and other functions are similar to mode 1.
In this mode, 8255 is a bidirectional 8-bit port with handshake signals. The Rdand WR signals decide whether the 8255 is going to operate as an input port or output port.

The Salient features of Mode 2 of 8255 are listed as follows:

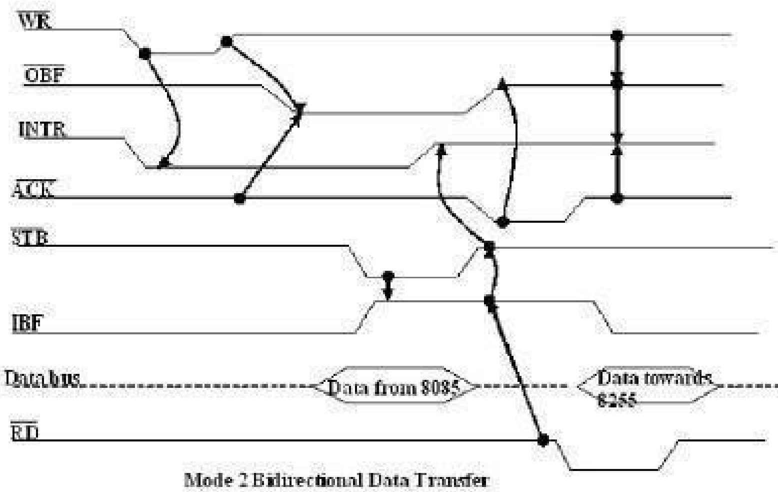
1. The single 8-bit port in group A is available.
2. The 8-bit port is bidirectional and additionally a 5-bit control port is available.
3. Three I/O lines are available at port C.(PC2 – PC0)
4. Inputs and outputs are both latched.
5. The 5-bit control port C (PC3-PC7) is used for generating / accepting handshakesignals for the 8-bit data transfer on port A.

Control signal definitions in mode 2:

- **INTR** – (Interrupt request) As in mode 1, this control signal is active high and isused to interrupt the microprocessor to ask for transfer of the next data byteto/from it. This signal is used for input (read) as well as output (write) operations.
- **Control Signals for Output operations**:
- OBF (Output buffer full) – This signal, when falls to low level, indicates that theCPU has written data to port A.
- ACK (Acknowledge) This control input, when falls to logic low level, Acknowledges that the previous data byte is received by the destination and nextbyte may be sent by the processor. This signal enables the internal tristate buffersto send the next data byte on port A.
- **INTE1** (A flag associated with OBF) This can be controlled by bit set/resetmode with PC6.

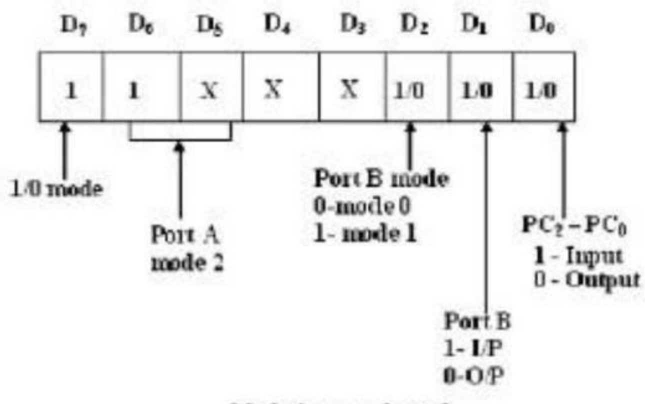
Control signals for input operations:

- STB (Strobe input)a low on this line is used to strobe in the data into the inputLatches of 8255.
- **IBF** (Input buffer full) when the data is loaded into input buffer, this signal risesto logic „1". This can be used as an acknowledge that the data has been receivedby the receiver.
- The waveforms in fig show the operation in Mode 2 for output as well as inputport.
- Note: WR must occur before ACK and STB must be activated before RD.



Mode 2 Bidirectional Data Transfer

- The following fig shows a schematic diagram containing an 8-bit bidirectionalport, 5-bit control port and the relation of INTR with the control pins. Port B caneither be set to Mode 0 or 1 with port A(Group A) is in Mode 2.
- Mode 2 is not available for port B. The following fig shows the control word.
- The INTR goes high only if IBF, INTE2, STB and RD go high or OBF,
- INTE1, ACK and WR go high. The port C can be read to know the status of theperipheral device, in terms of the control signals, using the normal I/Oinstructions.



Mode 2 control word

Mode 2 pins

Interfacing switches and LEDS

LED (light emitting diode):

Light emitting diodes is the most commonly used comp[onents, 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 resister to GND & the cathide is connected to the microprocessor pin. So when the port pin is HIGH the LED is OFF 7 when the port pin is LOW the LED is turned ON.

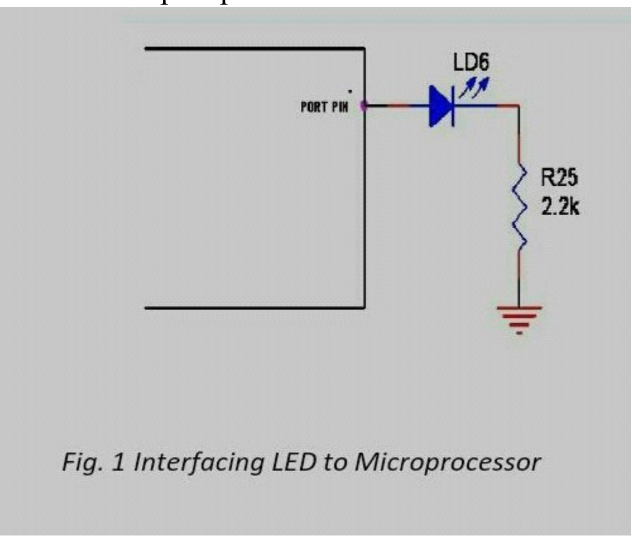


Fig. 1 Interfacing LED to Microprocessor

INTERFACING THE LED WITH 8086:

we now want to flash a LED in 8086 trainer boarde. 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:

PIN ASSIGNMENT WITH 8086

	Point LEDs	8255 Lines	LED Selection
DIGITAL OUTPUTS	LD1	PA.0	LED1 R1 330E
	LD2	PA.1	
	LD3	PA.2	
	LD4	PA.3	
	LD5	PA.4	
	LD6	PA.5	
	LD7	PA.6	
	LD8	PA.7	

ASSEMBLY PROGRAM TO ON AND OFF LED USING 8086:

ASSEMBLY PROGRAM TO ON AND OFF LED USING 8086

****

Title : Program to Blink LEDs

****

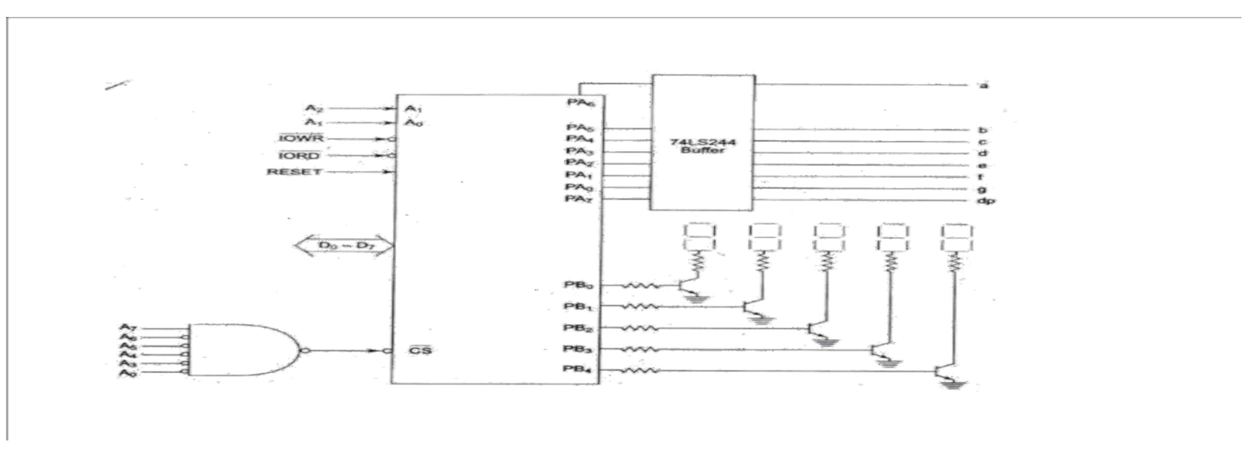
MEMORY ADDRESS	OPCODE	MNEMONICS
1100	B0 80	MOV AL, 80
1102	BA36 FF	MOV DX, FF36
1105	EE	OUT DX, AL
1106	B0 00	BEGIN:MOV AL, 00
1108	BA 30 FF	MOV DX, FF30
110B	EE	OUT DX, AL
110C	E8 08 00	CALL DELAY
110F	B0 FF	MOV AL, FF
1111	EE	OUT DX, AL
1112	E8 02 00	CALL DELAY
1115	EB EF	JMP BEGIN
1117	B9 FF FF	DELAY: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

Interfacing seven segment displays

Number to be displayed	PA7dp	PA6 a	PA5 b	PA4 c	PA3 d	PA2 e	PA1 f	PA0 g	Code
1	1	1	0	0	1	1	1	1	CF
2	1	0	0	1	0	0	1	0	92
3	1	0	0	0	0	1	1	0	86
4	1	1	0	0	1	1	0	0	CC
5	1	0	1	0	0	1	0	0	A4

Display Interface

Interfacing multiplexed 7-segment display



Software and Hardware interrupt applications

1. Hardware Interrupt :
Hardware Interrupt is caused by some hardware device such as request to start an I/O, a hardware failure or something similar. Hardware interrupts were introduced as a way to avoid wasting the processor's valuable time in polling loops, waiting for external events.

For example, when an I/O operation is completed such as reading some data into the computer from a tape drive.

2. Software Interrupt :
Software Interrupt is invoked by the use of INT instruction. This event immediately stops execution of the program and passes execution over to the INT handler. The INT handler is usually a part of the operating system and determines the action to be taken. It occurs when an application program terminates or requests certain services from the operating system.

For example, output to the screen, execute file etc.

Difference between Hardware Interrupt and Software Interrupt :

SR.NO.	Hardware Interrupt	Software Interrupt
1	Hardware interrupt is an interrupt generated from an external device or hardware.	Software interrupt is the interrupt that is generated by any internal system of the computer.
2	It do not increment the program counter.	It increment the program counter.
3	Hardware interrupt can be invoked with some external device such as request to start an I/O or occurrence of a hardware failure.	Software interrupt can be invoked with the help of INT instruction.
4	It has lowest priority than software interrupts	It has highest priority among all interrupts.
5	Hardware interrupt is triggered by external hardware and is considered one of the ways to communicate with the outside peripherals, hardware.	Software interrupt is triggered by software and considered one of the ways to communicate with kernel or to trigger system calls, especially during error or exception handling.
6	It is an asynchronous event.	It is synchronous event.

SR.NO.	Hardware Interrupt	Software Interrupt
7	Hardware interrupts can be classified into two types they are: 1. Maskable Interrupt. 2. Non Maskable Interrupt.	Software interrupts can be classified into two types they are: 1. Normal Interrupts. 2. Exception
8	Keystroke depressions and mouse movements are examples of hardware interrupt.	All system calls are examples of software interrupts

Applications of interrupts:

1. Applications of interrupts include the following: system timers, disk I/O, power-off signals, and traps.
2. Other interrupts exist to transfer data bytes using UARTs or Ethernet; sense key-presses; or anything else the equipment must do.
3. Another typical use is to generate periodic interrupts by dividing the output of a crystal oscillator and having an interrupt handler count the interrupts in order for a processor to keep time.
4. These periodic interrupts are often used by the OS's task scheduler to reschedule the priorities of running processes.
5. Some older computers generated periodic interrupts from the power line frequency because it was controlled by the utilities to eliminate long-term drift of electric clocks.
6. For example, a disk interrupt signals the completion of a data transfer from or to the disk peripheral; a process waiting to read or write a file starts up again.
7. As another example, a power-off interrupt predicts or requests a loss of power, allowing the computer equipment to perform an orderly shut-down.
8. Also, interrupts are used in type ahead features for buffering events like keystrokes.

Intel 8251 USART architecture and interfacing

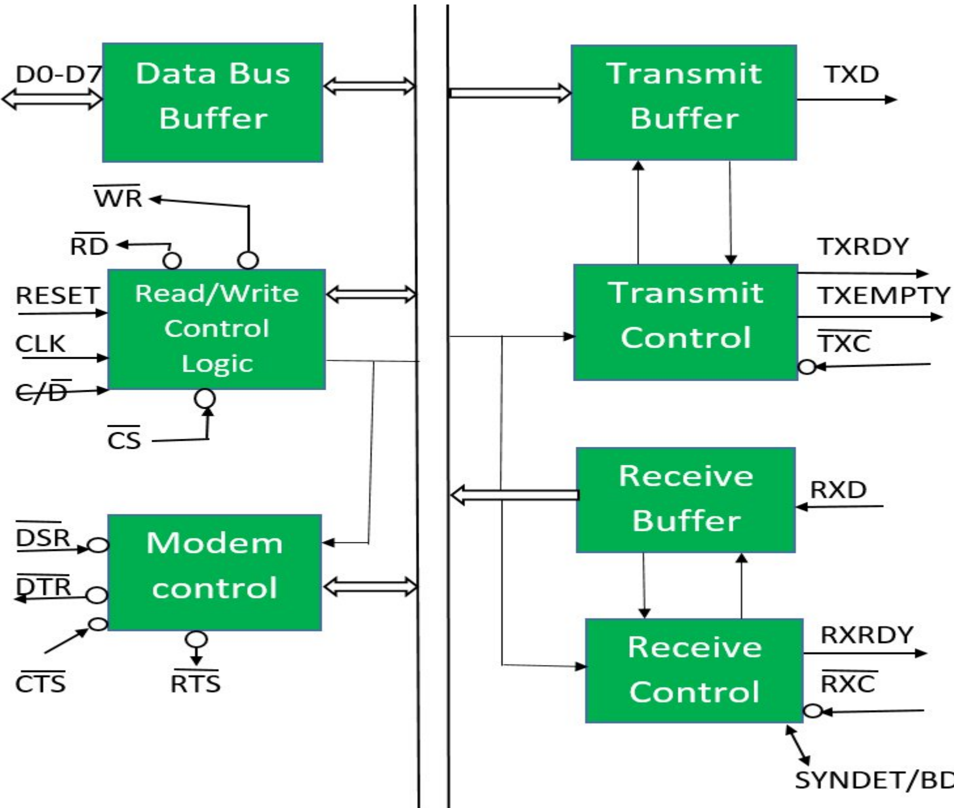
USART stands for **U**niversal **S**ynchronous and **A**synchronous **R**eceiver **T**ransmitter.

8251 universal synchronous asynchronous receiver transmitter (USART) acts as a mediator between microprocessor and peripheral to transmit serial data into parallel form and vice versa.

1. It takes data serially from peripheral (outside devices) and converts into parallel data.
2. After converting the data into parallel form, it transmits it to the CPU.
3. Similarly, it receives parallel data from microprocessor and converts it into serial form.
4. After converting data into serial form, it transmits it to outside device (peripheral)

.

Block Diagram/ Architecture of 8251 USART –



It contains the following blocks:

1. **Data bus buffer –**
   This block helps in interfacing the internal data bus of 8251 to the system data bus. The data transmission is possible between 8251 and CPU by the data bus buffer block.
2. **Read/Write control logic –**
   It is a control block for overall device. It controls the overall working by selecting the operation to be done. The operation selection depends upon input signals as:

$\overline{CS}$	$C/\overline{D}$	$\overline{RD}$	$\overline{WR}$	Operation
1	X	X	X	Invalid
0	0	0	1	data CPU< ----- 8251
0	0	1	0	data CPU ----- > 8251
0	1	0	1	Status word CPU < ------------8251
0	1	1	0	Control word CPU------------ > 8251

3.
In this way, this unit selects one of the three registers- data buffer register, control register, status register.

4. **Modem control (modulator/demodulator) –**
A device converts analog signals to digital signals and vice-versa and helps the computers to communicate over telephone lines or cable wires. The following are active-low pins of Modem.

- **DSR:** Data Set Ready signal is an input signal.
- **DTR:** Data terminal Ready is an output signal.
- **CTS:** It is an input signal which controls the data transmit circuit.

RTS: It is an output signal which is used to set the status RTS.

5. **Transmit buffer –**
This block is used for parallel to serial converter that receives a parallel byte for conversion into serial signal and further transmission onto the common channel.

- **TXD:** It is an output signal, if its value is one, means transmitter will transmit the data.

6. **Transmit control –**
This block is used to control the data transmission with the help of following pins:

- **TXRDY:** It means transmitter is ready to transmit data character.
- **TXEMPTY:** An output signal which indicates that TXEMPTY pin has transmitted all the data characters and transmitter is empty now.
- **TXC:** An active-low input pin which controls the data transmission rate of transmitted data.

7. **Receive buffer –**
This block acts as a buffer for the received data.

- **RXD:** An input signal which receives the data.

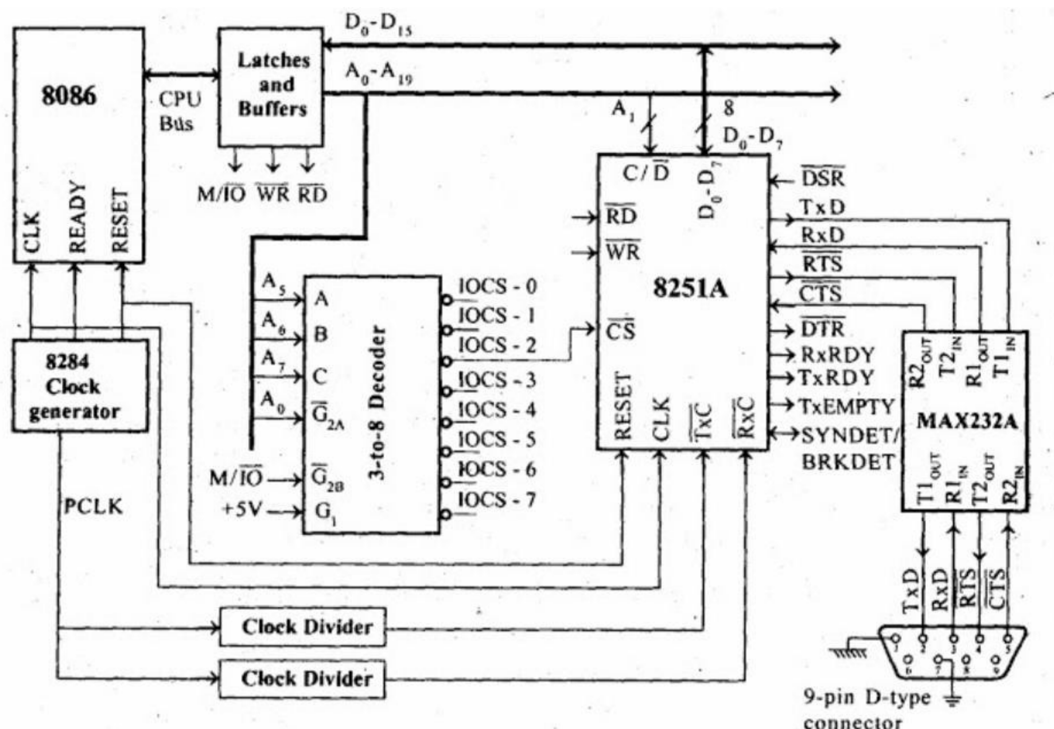
8. **Receive control –**
This block controls the receiving data.

- **RXRDY:** An input signal indicates that it is ready to receive the data.
- **RXC:** An active-low input signal which controls the data transmission rate of received data.
- **SYNDET/BD:** An input or output terminal. External synchronous mode-input terminal and asynchronous mode-output terminal.

INTERFACING 8251A TO 8086 PROCESSOR

• The chip select for I/O mapped devices are generated by using a 3-to-8 decoder. • The address lines A5, A6 and A7 are decoded to generate eight chip select signals (IOCS-0 to IOCS-7) and in this, the chip select signal IOCS-2 is used to select 825lA. • The address line A0 and the control signal M/IO(low) are used as enable for decoder. • The line A1 of 8086 is connected to C/D(low) of 8251A to provide the internal addresses. • The lines D0 – D7 connected to D0 – D7 of the processor to achieve parallel data transfer. • The

RESET and clock signals are supplied by 8284 clock generator. Here the processor clock is directly connected to 8251A. This clock controls the parallel data transf er between the processor and 825lA. • 8251A in I/O mapped in the system is shown in the figure.



- The peripheral clock (PCLK) supplied by 8284, is divided by suitable clock dividers like programmable timer 8254 and then used as clock for serial transmission and reception.
- In 8251A the transmission and reception baud rates can be different or same. • The TTL logic levels of the serial data lines and the control signals necessary for serial transmission and reception are converted to RS232 logic levels using MAX232 and then terminated on a standard 9-pin D-.type connector. • The device, which requires serial communication with processor, can be connected to this 9-pin D-type connector using 9-core cable. • The signals TxEMPTY, TxRDY and RxRDY can be used as interrupt signals to initiate interrupt driven data transfer scheme between processor and 8251 A. • The I/O addresses allotted to the internal devices of 8251A are listed in table.

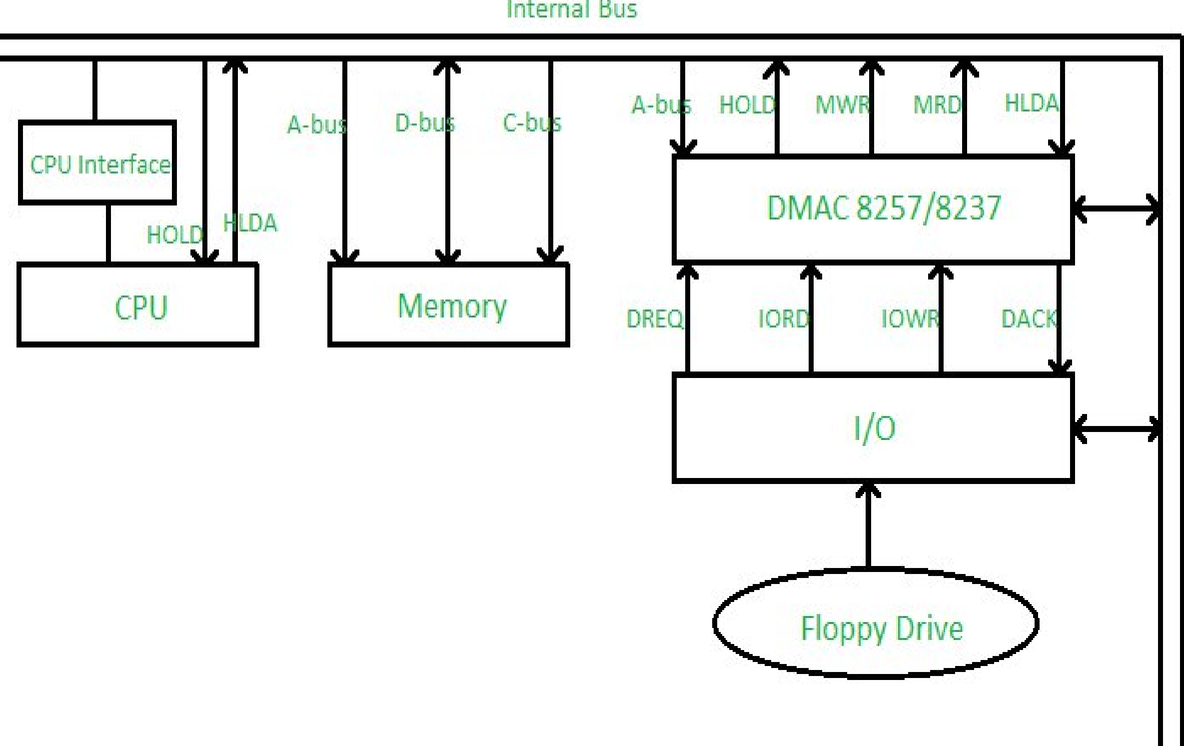
Internal Device of 8251A	Binary Address								Hexa Address
	Decoder input			Input to address pin of 8251				Decoder enable	
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
Data buffer	0	1	0	x	x	x	0	0	40
Control register	0	1	0	x	x	x	1	0	42

Intel 8237a DMA controller

Suppose any device which is connected to input-output port wants to transfer data to memory, first of all it will send input-output port address and control signal, input-output read to input-output port, then it will send memory address and memory write signal to memory where data has to be transferred. In normal input-output technique the processor becomes busy in checking whether any input-output operation is completed or not for next input-output operation, therefore this technique is slow.

This problem of slow data transfer between input-output port and memory or between two memory is avoided by implementing Direct Memory Access (DMA) technique. This is faster as the microprocessor/computer is bypassed and the control of address bus and data bus is given to the DMA controller.

- HOLD – hold signal
- HLDA – hold acknowledgment
- DREQ – DMA request
- DACK – DMA acknowledgment



Suppose a floppy drive that is connected at input-output port wants to transfer data to memory, the following steps are performed:

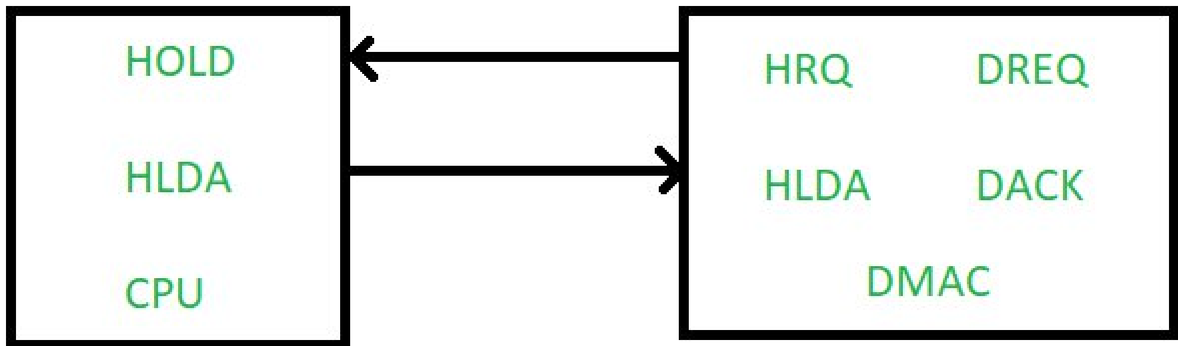
- **Step-1:** First of all the floppy drive will send a DMA request (DREQ) to the DMAC, it means the floppy drive wants its DMA service.
- **Step-2:** Now the DMAC will send a HOLD signal to the CPU.
- **Step-3:** After accepting the DMA service request from the DMAC, the CPU will send hold acknowledgment (HLDA) to the DMAC, it means the microprocessor has released control of the address bus the data bus to DMAC and the microprocessor/computer is bypassed during DMA service.

- **Step-4:** Now the DMAC will send one acknowledgement (DACL) to the floppy drive which is connected at the input-output port. It means the DMAC tells the floppy drive be ready for its DMA service.

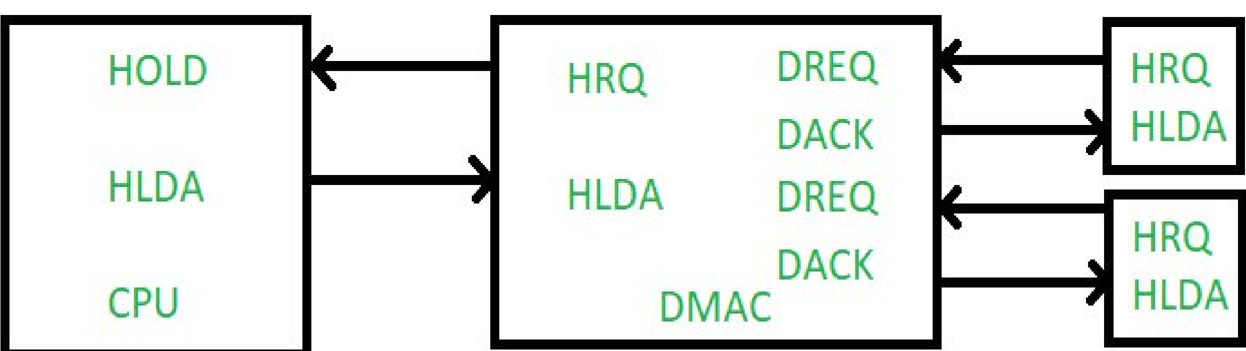
- **Step-5:** Now with the help of input-output read and memory write signal the data is transferred from the floppy drive to the memory.

Modes of DMAC:

1. Single Mode – In this only one channel is used, means only a single DMAC is connected to the bus system.



2. Cascade Mode – In this multiple channels are used, we can further cascade more number of DMACs.



The method that is used to transfer information between internal storage and external I/O devices is known as I/O interface. The CPU is interfaced using special communication links by the peripherals connected to any computer system. These communication links are used to resolve the differences between CPU and peripheral. There exists special hardware components between CPU and peripherals to supervise and synchronize all the input and output transfers that are called interface units.

Mode of Transfer:

The binary information that is received from an external device is usually stored in the memory unit. The information that is transferred from the CPU to the external device is originated from the memory unit. CPU merely processes the information but the source and target is always the memory unit. Data transfer between CPU and the I/O devices may be done in different modes.

Data transfer to and from the peripherals may be done in any of the three possible ways

1. Programmed I/O.
2. Interrupt- initiated I/O.
3. Direct memory access(DMA).

Now let's discuss each mode one by one.

1. **Programmed I/O:** It is due to the result of the I/O instructions that are written in the computer program. Each data item transfer is initiated by an instruction in the program. Usually the transfer is from a CPU register and memory. In this case it requires constant monitoring by the CPU of the peripheral devices.

Example of Programmed I/O: In this case, the I/O device does not have direct access to the memory unit. A transfer from I/O device to memory requires the execution of several instructions by the CPU, including an input

instruction to transfer the data from device to the CPU and store instruction to transfer the data from CPU to memory. In programmed I/O, the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process since it needlessly keeps the CPU busy. This situation can be avoided by using an interrupt facility. This is discussed below.

2. **Interrupt- initiated I/O:** Since in the above case we saw the CPU is kept busy unnecessarily. This situation can very well be avoided by using an interrupt driven method for data transfer. By using interrupt facility and special commands to inform the interface to issue an interrupt request signal whenever data is available from any device. In the meantime the CPU can proceed for any other program execution. The interface meanwhile keeps monitoring the device. Whenever it is determined that the device is ready for data transfer it initiates an interrupt request signal to the computer. Upon detection of an external interrupt signal the CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing.

Note: Both the methods programmed I/O and Interrupt-driven I/O require the active intervention of the processor to transfer data between memory and the I/O module, and any data transfer must transverse a path through the processor. Thus both these forms of I/O suffer from two inherent drawbacks.

- The I/O transfer rate is limited by the speed with which the processor can test and service a device.
- The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer.

3. **Direct Memory Access**: The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU. Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as DMA or direct memory access. During DMA the CPU is idle and it has no control over the memory buses. The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.

High Impedance (disable) when BG is enable

Bus Request → BR | ABUS ← Address Bus

DBUS → Data Bus

RD → Read

Bus Grant ← BG | WR → Write

Figure - CPU Bus Signals for DMA Transfer

Bus Request : It is used by the DMA controller to request the CPU to relinquish the control of the buses.

Bus Grant : It is activated by the CPU to Inform the external DMA controller that the buses are in high impedance state and the requesting DMA can take control of the buses. Once the DMA has taken the control of the buses it transfers the data. This transfer can take place in many ways.

Types of DMA transfer using DMA controller:

Burst Transfer :
DMA returns the bus after complete data transfer. A register is used as a byte count, being decremented for each byte transfer, and upon the byte count reaching zero, the DMAC will release the bus. When the DMAC operates in burst mode, the CPU is halted for the duration of the data transfer.
Steps involved are:

1. Bus grant request time.
2. Transfer the entire block of data at transfer rate of device because the device is usually slow than the speed at which the data can be transferred to CPU.

3. Release the control of the bus back to CPU

So, total time taken to transfer the N bytes

= Bus grant request time + (N) * (memory transfer rate) + Bus release control time.

Where,

X µsec =data transfer time or preparation time (words/block)

Y µsec =memory cycle time or cycle time or transfer time (words/block)

% CPU idle (Blocked)=(Y/X+Y)*100

% CPU Busy=(X/X+Y)*100

Cyclic Stealing :

An alternative method in which DMA controller transfers one word at a time after which it must return the control of the buses to the CPU. The CPU delays its operation only for one memory cycle to allow the direct memory I/O transfer to "steal" one memory cycle.

Steps Involved are:

1. Buffer the byte into the buffer
2. Inform the CPU that the device has 1 byte to transfer (i.e. bus grant request)
3. Transfer the byte (at system bus speed)
4. Release the control of the bus back to CPU.

Before moving on transfer next byte of data, device performs step 1 again so that bus isn't tied up and the transfer won't depend upon the transfer rate of device.

So, for 1 byte of transfer of data, time taken by using cycle stealing mode (T).

= time required for bus grant + 1 bus cycle to transfer data + time required to release the bus, it will be N x T

In cycle stealing mode we always follow pipelining concept that when one byte is getting transferred then Device is parallel preparing the next byte. "The fraction of CPU time to the data transfer time" if asked then cycle stealing mode is used.

Where,

X µsec =data transfer time or preparation time

(words/block)

Y µsec =memory cycle time or cycle time or transfer

time (words/block)

% CPU idle (Blocked) =(Y/X)*100

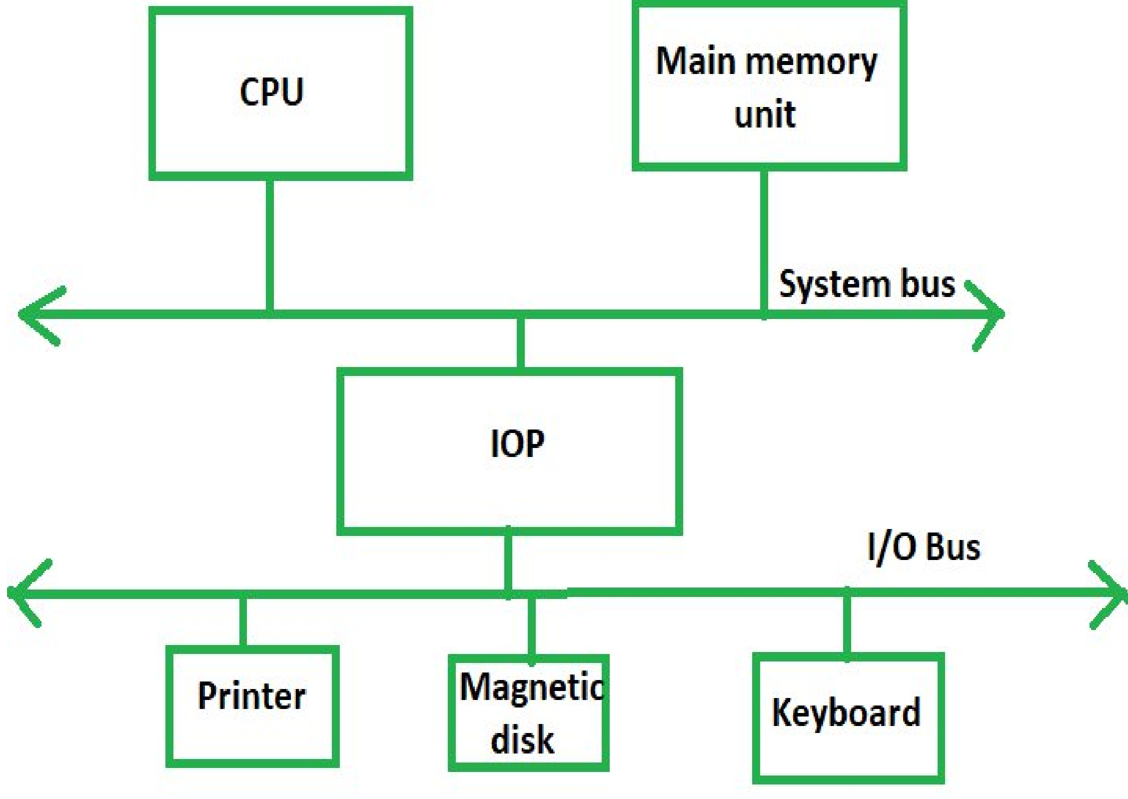
% CPU busy=(X/Y)*100

Interleaved mode: In this technique , the DMA controller takes over the system bus when the microprocessor is not using it.An alternate half cycle i.e. half cycle DMA + half cycle processor.

The **DMA mode** of data transfer reduces CPU's overhead in handling I/O operations. It also allows parallelism in CPU and I/O operations. Such parallelism is necessary to avoid wastage of valuable CPU time while handling I/O devices whose speeds are much slower as compared to CPU. The concept of DMA operation can be extended to relieve the CPU further from getting involved with the execution of I/O operations. This gives rises to the development of special purpose processor called **Input-Output Processor (IOP) or IO channel**.

The Input Output Processor (IOP) is just like a CPU that handles the details of I/O operations. It is more equipped with facilities than those are available in typical DMA controller. The IOP can fetch and execute its own instructions that are specifically designed to characterize I/O transfers. In addition to the I/O – related tasks, it can perform other processing tasks like arithmetic, logic, branching and code translation. The main memory unit takes the pivotal role. It communicates with processor by the means of DMA.

The block diagram –



The Input Output Processor is a specialized processor which loads and stores data into memory along with the execution of I/O instructions. It acts as an interface between system and devices. It involves a sequence of events to executing I/O operations and then store the results into the memory.

Advantages –

- The I/O devices can directly access the main memory without the intervention by the processor in I/O processor based systems.

It is used to address the problems that are arises in Direct memory access method.

Stepper Motor Interfacing:

- A stepper motor is a device used to obtain an accurate position control of rotating shafts. It employs rotation of its shaft in terms of steps, rather than continuous rotation as in case of AC or DC motors. To rotate the shaft of the stepper motor, a sequence of pulses is needed to be applied to the windings of the stepper motor, in a proper sequence.
- The number of pulses required for one complete rotation of the shaft of the stepper motor is equal to its number of internal teeth on its rotor. The stator teeth and the rotor teeth lock with each other to fix a position of the shaft.
- With a pulse applied to the winding input, the rotor rotates by one teeth position or an angle x. The angle x may be calculated as:

X=360^0/no. of rotor teeth

- After the rotation of the shaft through angel x, the rotor locks itself with the next tooth in the sequence on the internal surface of stator.
- The internal schematic of a typical stepper motor with four windings is shown in fig.1.
- The stepper motors have been designed to work with digital circuits. Binary level pulses of 0-5V are required at its winding inputs to obtain the rotation of shafts. The sequence of the pulses can be decided, depending upon the required motion of the shaft.
  - Fig.2 shows a typical winding arrangement of the stepper motor.
  - Fig.3 shows conceptual positioning of the rotor teeth on the surface of rotor, for a six teeth rotor.

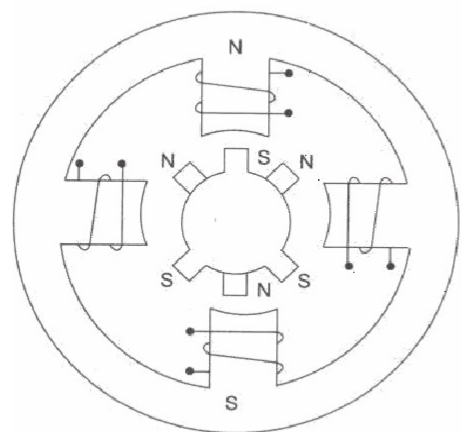
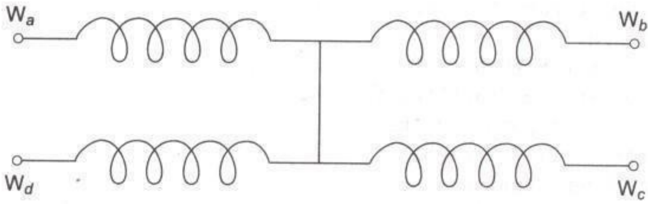


Fig.1 Internal schematic of a four winding stepper motor.



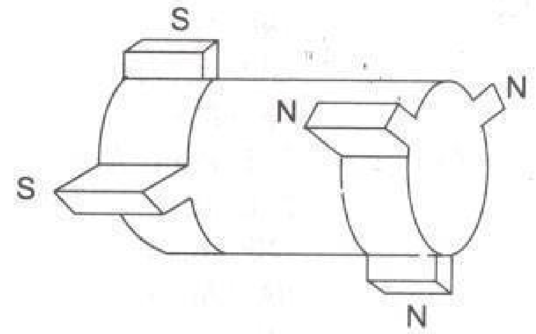


Fig.2 Winding arrangement of a stepper motor.

Fig.3 Stepper motor rotor

- The circuit for interfacing a winding Wn with an I/O port is given in fig.4. Each ofthe windings of a stepper motor needs this circuit for its interfacing with theoutput port. A typical stepper motor may have parameters like torque 3 Kg-cm, operating voltage 12V, current rating 0.2 A and a step angle 1.8^0 i.e. 200steps/revolution (number of rotor teeth).
- A simple schematic for rotating the shaft of a stepper motor is called a wave scheme. In this scheme, the windings Wa, Wb, Wc and Wd are applied with the required voltages pulses, in a cyclic fashion. By reversing the sequence of excitation, the direction of rotation of the stepper motor shaft may be reversed.
- Table.1 shows the excitation sequences for clockwise and anticlockwise rotations. Another popular scheme for rotation of a stepper motor shaft applies pulses to two successive windings at a time but these are shifted only by one position at a time. This scheme for rotation of stepper motor shaft is shown in table2.

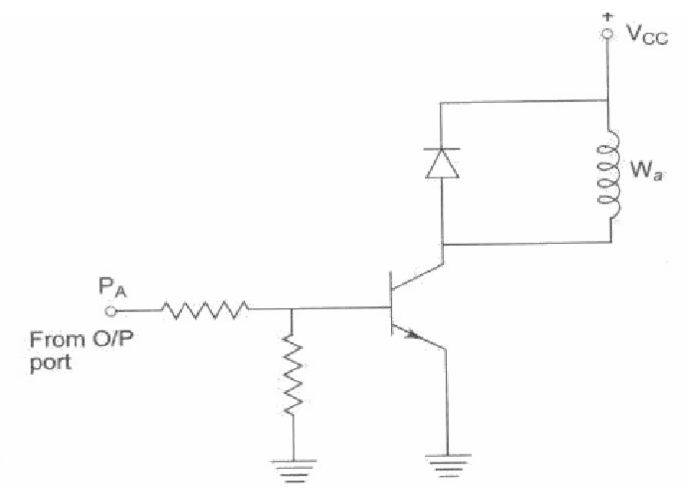


Fig.4 interfacing stepper motor winding.

Motion	step	A	B	C	D
Clock wise	1	1	0	0	0
	2	0	1	0	0
	3	0	0	1	0
	4	0	0	0	1
	5	1	0	0	0
Anticlock wise	1	1	0	0	0
	2	0	0	0	1
	3	0	0	1	0
	4	0	1	0	0
	5	1	0	0	0

Table.1 Excitation sequence of a stepper motor using wave switching scheme.

Table.2 An alternative scheme for rotating stepper motor shaft

Motion	step	A	B	C	D
Clock wise	1	0	0	1	1
	2	0	1	1	0
	3	1	1	0	0
	4	1	0	0	1
	5	0	0	1	1
Anticlock wise	1	0	0	1	1
	2	1	0	0	1
	3	1	1	0	0
	4	0	1	1	0
	5	0	0	0	0

Interfacing Analog to Digital Data Converters:

- In most of the cases, the PIO 8255 is used for interfacing the analog to digital converters with microprocessor.
- We have already studied 8255 interfacing with 8086 as an I/O port, in previous section. This section we will only emphasize the interfacing techniques of analog to digital converters with 8255.
- The analog to digital converters is treated as an input device by the microprocessor that sends an initializing signal to the ADC to start the analogy to digital data conversation process. The start of conversation signal is a pulse of a specific duration.

- The process of analog to digital conversion is a slow
- Process and the microprocessor have to wait for the digitaldata till the conversion is over. After the conversion isover, the ADC sends end of conversion EOC signal toinform themicroprocessor that the conversion is over andthe result is ready at the output buffer of the ADC. Thesetasks of issuing an SOC pulse to ADC, reading EOC signalfrom the ADC and reading the digital output of the ADCare carried out by the CPU using 8255 I/O ports.
- The time taken by the ADC from the active edge of SOCpulse till the active edge of EOCsignal is called as theconversion delay of the ADC.
- It may range anywhere from a few microseconds in caseof fast ADC to even a few hundred milliseconds in case ofslow ADCs.
- The available ADC in the market use different conversiontechniques for conversion of analog signal to digitals.Successive approximation techniques and dual slopeintegration techniques are the most popular techniquesused in the integrated ADC chip.
- General algorithm for ADC interfacing contains thefollowing steps:
- Ensure the stability of analog input, applied to the ADC.
- Issue start of conversion pulse to ADC
- Read end of conversion signal to mark the end ofconversion processes.
- Read digital data output of the ADC as equivalent digitaloutput.
- Analog input voltage must be constant at the input of theADC right from the start of conversion till the end of theconversion to get correct results. This may be ensured by asample and hold circuit which samples the analog signaland holds it constant for specific time duration. Themicroprocessor may issue a hold signal to the sample andhold circuit.
- If the applied input changes before the completeconversion process is over, the digital equivalent of theanalog input calculated by the ADC may not be correct.

ADC 0808/0809:

19.1 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.

19.2 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, as shown. 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.

19.3 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.

19.4 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.

Fig (1) and Fig (2) show the block diagrams and pin diagrams for ADC 0808/0809.

Table.1

Analog I/P selected	**Address lines**		
	C	**B**	**A**
I/P 0	0	0	0
I/P 1	0	0	1
I/P 2	0	1	0
I/P 3	0	1	1
I/P 4	1	0	0
I/P 5	1	0	1
I/P 6	1	1	0
I/P 7	1	1	1

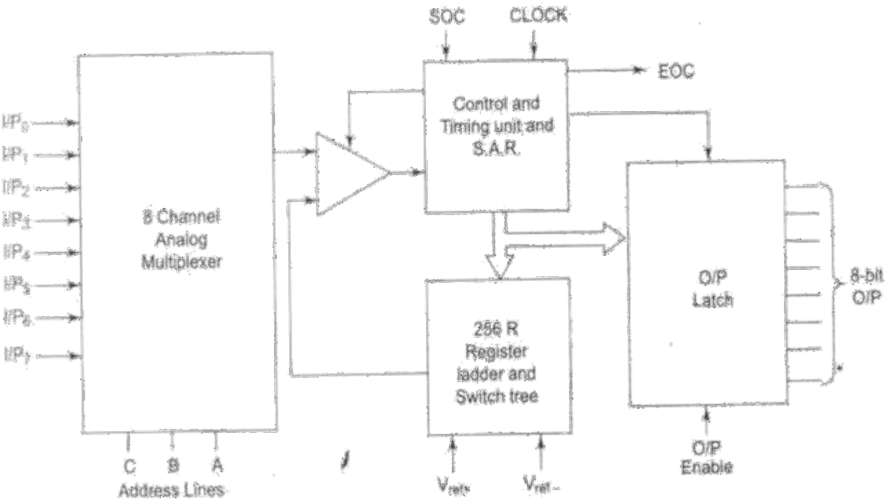
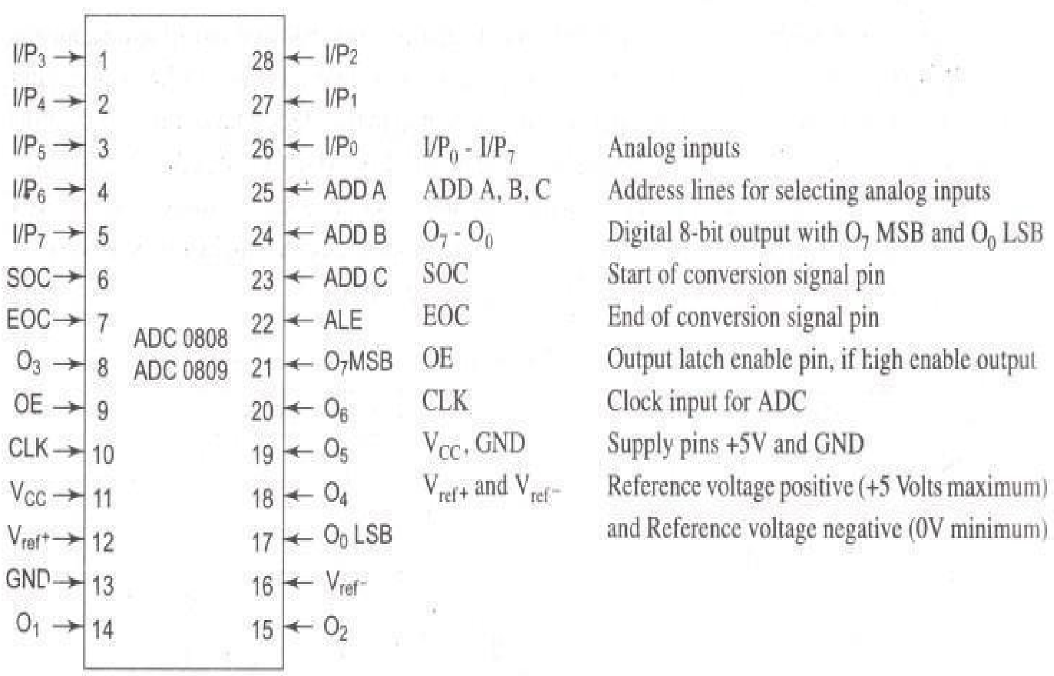


Fig.1 Block Diagram of ADC 0808/0809



I/P_0 - I/P_7	Analog inputs
ADD A, B, C	Address lines for selecting analog inputs
O_7 - O_0	Digital 8-bit output with O_7 MSB and O_0 LSB
SOC	Start of conversion signal pin
EOC	End of conversion signal pin
OE	Output latch enable pin, if high enable output
CLK	Clock input for ADC
V_{CC}, GND	Supply pins +5V and GND
V_{ref+} and V_{ref-}	Reference voltage positive (+5 Volts maximum) and Reference voltage negative (0V minimum)

Fig.2 Pin Diagram of ADC 0808/0809

Some Electrical Specifications Of The ADC 0808/0809 Are Given In Table.2.

Table.2

Minimum SOC pulse width	100 ns
Minimum ALE pulse width	100 ns
Clock frequency	10 to 1280 kHz
Conversion time	100 ms at 640 kHz
Resolution	8-bit
Error	+/–1 LSB
V_{ref+}	Not more than +5V
V_{ref-}	Not less than GND
+ V_{cc} supply	+ 5 V DC
Logical 1 i/p voltage	minimum V_{cc} –1.5 V
Logical 0 i/p voltage	maximum 1.5 V
Logical 1 o/p voltage	minimum V_{cc}–0.4 V
Logical 0 o/p voltage	maximum 0.45 V

The Timing Diagram Of Different Signals Of Adc0808 Is Shown In Fig.3

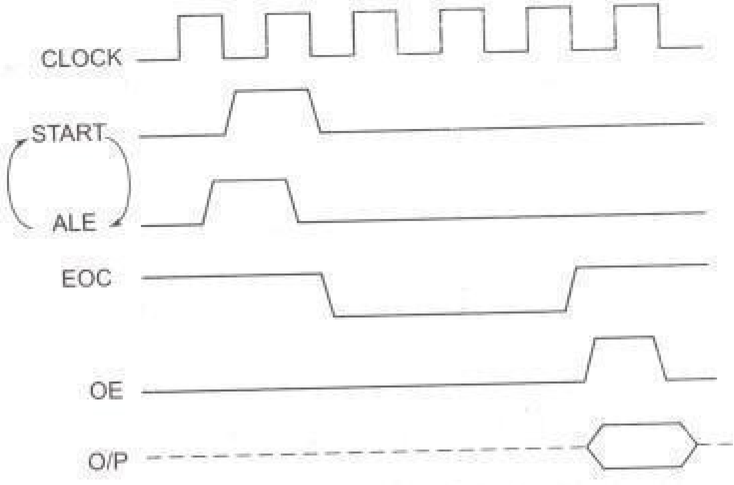
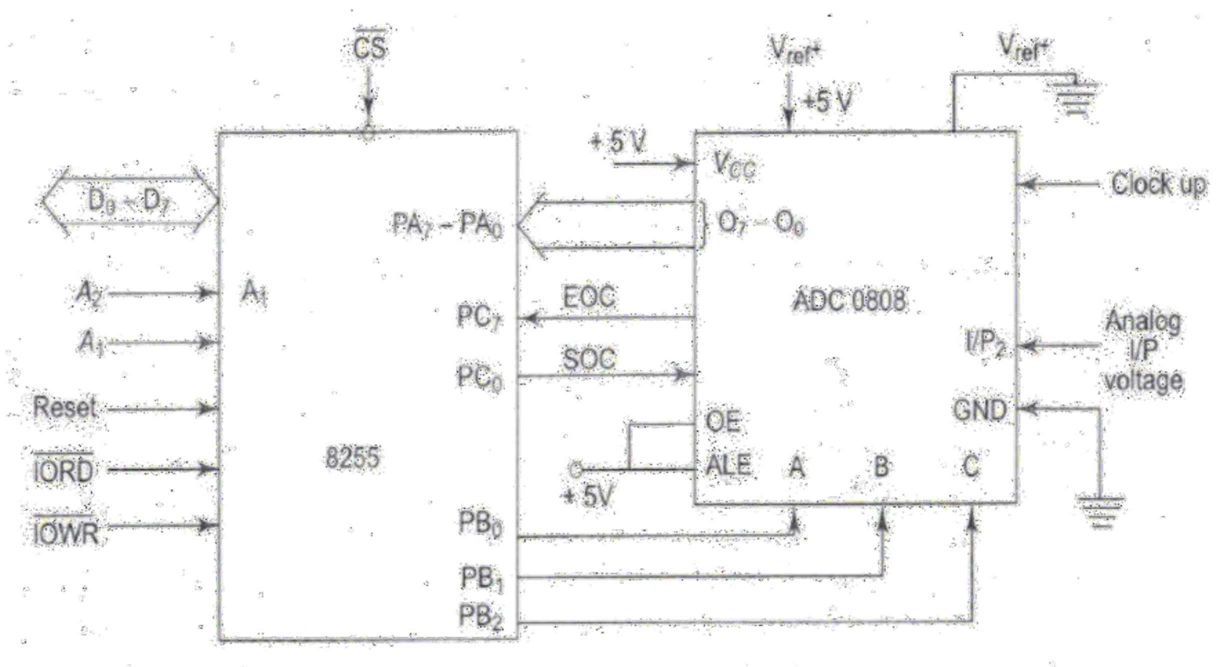


Fig.3 Timing Diagram Of ADC 0808.



Interfacing ADC0808 with 8086

Interfacing Digital To Analog Converters:

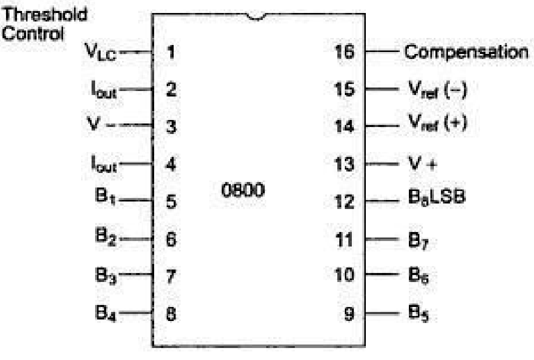
The digital to analog converters convert binary numbers into their analog equivalent voltages. The DAC find applications in areas like digitally controlled gains, motor speed controls, programmable gain amplifiers, etc.

DAC0800 8-bit Digital to Analog Converter

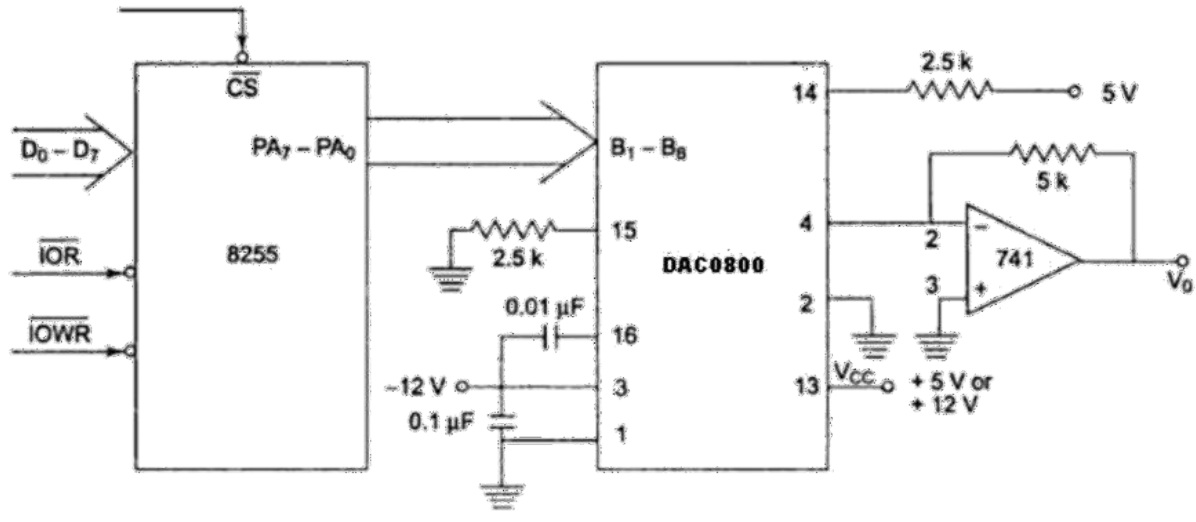
- The DAC 0800 is a monolithic 8-bit DAC manufactured by National Semiconductor.

It has settling time around 100ms and can operate on a range of power supply voltages i.e. from 4.5V to +18V.

Usually the supply V+ is 5V or +12V. The V-pin can be kept at a minimum of -12V.



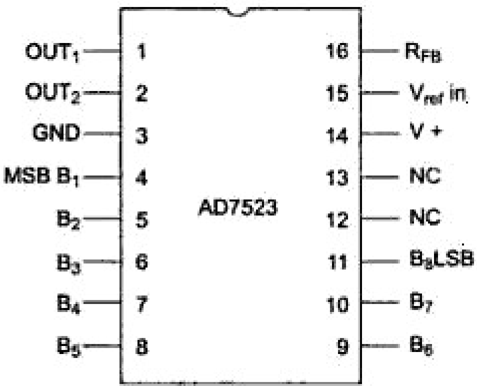
Pin Diagram of DAC 0800



Interfacing DAC0800 with 8086Ad

7523 8-Bit Multiplying DAC:

Intersil‟s AD 7523 is a 16 pin DIP, multiplying digital to analog converter, containing R-2R ladder(R=10KΩ) for digital to analog conversion along with single pole double through NMOS switches to connect the digital inputs to the ladder.

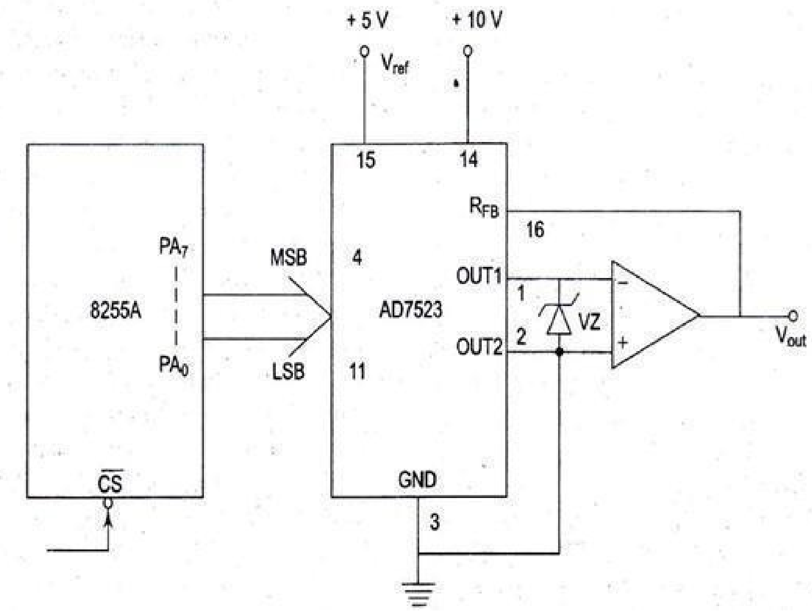


Pin Diagram of AD7523

The supply range extends from +5V to +15V , while Vref may be anywhere between -10V to +10V. The maximum analog output voltage will be +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 7523 to convert the current output of AD7523 to a proportional output voltage.

- 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.



Interfacing AD7523 with 8086

Need for 8259 programmable interrupt controllers

Programmable interrupt controllers are used to enhance the number of interrupts of a microprocessor. 8259 is a programmable interrupt controller which shows compatibility with 8085 microprocessor.

It is also known as a **priority interrupt controller** and was designed by **Intel** to increase the interrupt handling ability of the microprocessor. An 8259 PIC never services an interrupt; it simply forwards the interrupt to the processor for the execution of interrupt service routine.

Need of Programmable Interrupt Controller

We know whenever an interrupt occurs then the microprocessor suspends the current program and switches to the Interrupt Service Routine (ISR).

We know 8085 has 5 interrupts, which are: Trap, RST7.5, RST6.5, RST5.5 and INTR.

Among all these, only INTR is a non-vectored type of interrupt, rest are vectored interrupts.

We know vectored interrupts are those interrupts whose ISR address is known to the processor. Or we can say in case of vectored interrupts, the processor holds the address of the memory location where ISR is stored.

But in case of non-vectored interrupts, the processor has to reach the ISR but it does not hold the address of ISR. So, in this case, the interrupt generating device provides the ISR address to the microprocessor.

An 8085 has 5 major interrupts for which a fixed number of lines are present in the chip. But there are many devices connected to a processor. So, for such a case the processor must have more number of lines to handle several interrupts.

But it is not practically possible to increase the number of lines each time with the increase in the number of interrupts.

So, to overcome this problem 8259 PIC chip is used. 8259 allows the combining of multiple interrupts and providing them to the processor based on priority through a common line.

As we have already discussed that the processor holds the address of ISR in case of vectored interrupts. So, it is not possible to combine a non-vectored interrupt with a vectored one.

Therefore, 8259 is used to combine various interrupts which are non-vectored in nature.

Also, suppose in some way or the other, two devices generate interrupt simultaneously through a common line without the involvement of 8259.

So, the processor gets two INTR signals at the same time but how does the processor get to know that from where the interrupt is generating and where to send the INTA in order to have the ISR address.

This shows the necessity of 8259. The programmable interrupt controller tells the microprocessor about the interrupt. Basically the external devices initially interrupt the 8259 and further the 8259 interrupts the microprocessor.

INTRODUCTION TO MICROCONTROLLERS

8051 MICROCONTROLLER:

The Intel 8051 microcontroller is one of the most popular general purpose microcontrollers in use today. The success of the Intel 8051 spawned a number of clones which are collectively referred to as the MCS-51 family of microcontrollers, which includes chips from vendors such as Atmel, Philips, Infineon, and Texas Instruments

The Intel 8051 is an 8-bit microcontroller which means that most available operations are limited to 8 bits. There are 3 basic "sizes" of the 8051: Short, Standard, and Extended. The Short and Standard chips are often available in DIP (dual in-line package) form, but the Extended 8051 models often have a different form factor, and are not "drop-in compatible". All these things are called 8051 because they can all be programmed using 8051 assembly language, and they all share certain features (although the different models all have their own special features).

Some of the features that have made the 8051 popular are:

- 64 KB on chip program memory.
- 128 bytes on chip data memory (RAM).
- 4 register banks.
- 128 user defined software flags.
- 8-bit data bus
- 16-bit address bus
- 32 general purpose registers each of 8 bits
- 16 bit timers (usually 2, but may have more, or less).
- 3 internal and 2 external interrupts.
- Bit as well as byte addressable RAM area of 16 bytes.
- Four 8-bit ports, (short models have two 8-bit ports).
- 16-bit program counter and data pointer.
- 1 Microsecond instruction cycle with 12 MHz Crystal.

8051 models may also have a number of special, model-specific features, such as UARTs, ADC, Op-Amps, etc...

Typical applications:

8051 chips are used in a wide variety of control systems, telecom applications, robotics as well as in the automotive industry. By some estimation, 8051 family chips make up over 50% of the embedded chip market.

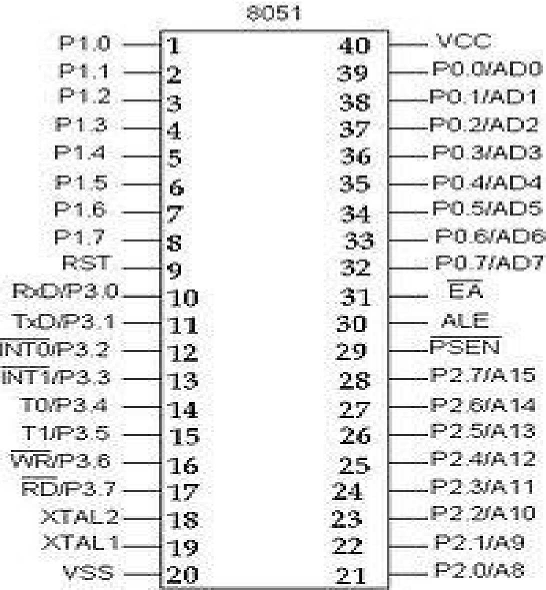


Fig: Pin diagram of the 8051 DIP

Basic Pins:

Pin 9: PIN 9 is the reset pin which is used reset the microcontroller‘s internal registers and ports upon starting up. (Pin should be held high for 2 machine cycles.)

Pins 18 & 19: The 8051 has a built-in oscillator amplifier hence we need to only connect a crystal at these pins to provide clock pulses to the circuit.

Pin 40 and 20: Pins 40 and 20 are VCC and ground respectively. The 8051 chip needs +5V 500mA to function properly, although there are lower powered versions like the Atmel 2051 which is a scaled down version of the 8051 which runs on +3V.

Pins 29, 30 & 31: As described in the features of the 8051, this chip contains a built-in flash memory. In order to program this we need to supply a voltage of +12V at pin 31. If external memory is connected then PIN 31, also called EA/VPP, should be connected to ground to indicatethe presence of external memory. PIN 30 is called ALE (address latch enable), which is used when multiple memory chips are connected to the controller and only one of them needs to be selected. We will deal with this in depth in the later chapters. PIN 29 is called PSEN. This is "program store enable". In order to use the external memory it is required to provide the low voltage (0) on both PSEN and EA pins.

Ports:

There are 4 8-bit ports: P0, P1, P2 and P3.

PORT P1 (Pins 1 to 8): The port P1 is a general purpose input/output port which can be used for a variety of interfacing tasks. The other ports P0, P2 and P3 have dual roles or additional functions associated with them based upon the context of their usage.

PORT P3 (Pins 10 to 17): PORT P3 acts as a normal IO port, but Port P3 has additional functions such as, serial transmit and receive pins, 2 external interrupt pins, 2 external counter inputs, read and write pins for memory access.

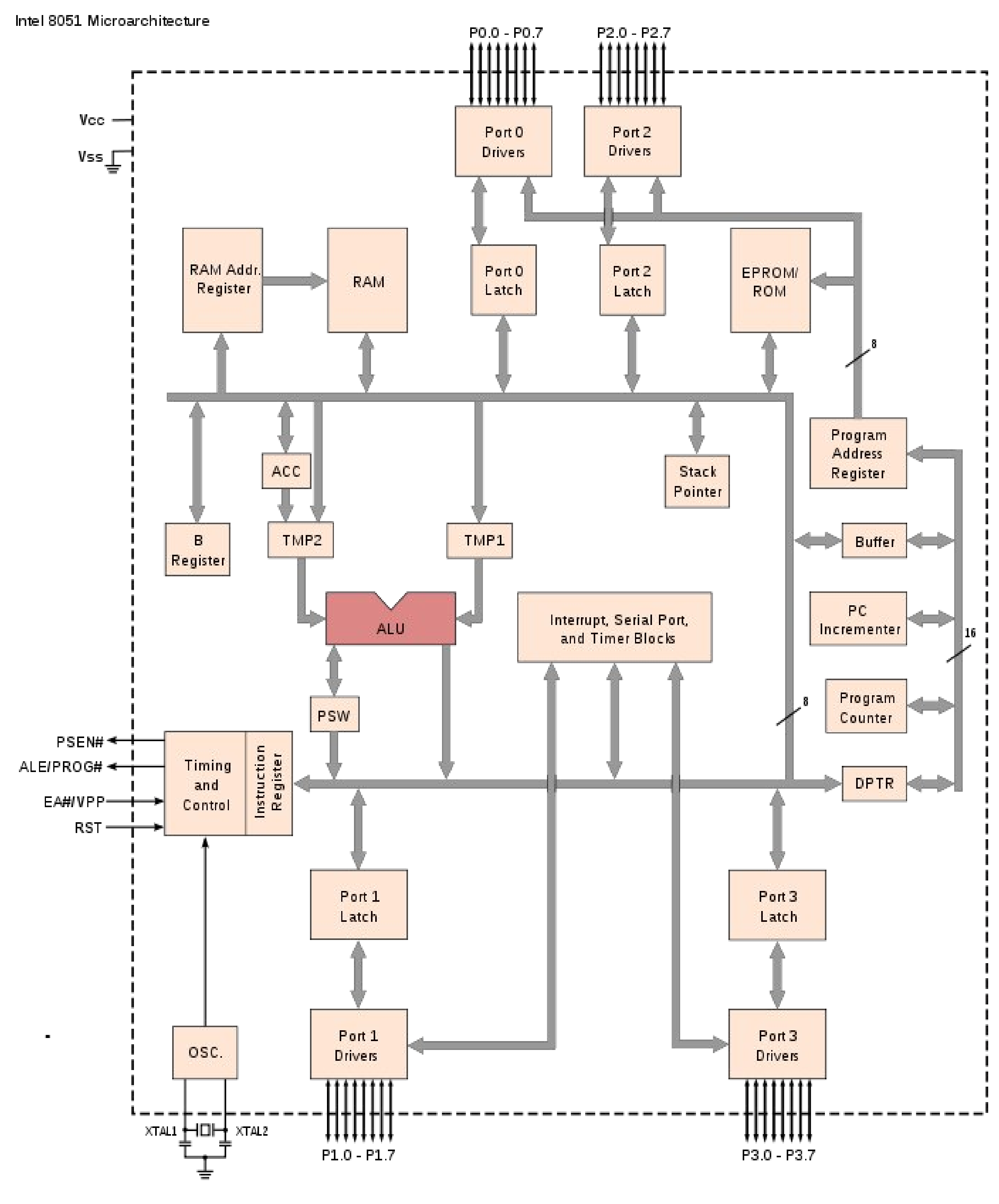
PORT P2 (pins 21 to 28): PORT P2 can also be used as a general purpose 8 bit port when no external memory is present, but if external memory access is required then PORT P2 will act as an address bus in conjunction with PORT P0 to access external memory. PORT P2 acts as A8-A15, as can be seen from fig 1.1

PORT P0 (pins 32 to 39) PORT P0 can be used as a general purpose 8 bit port when no external memory is present, but if external memory access is required then PORT P0 acts as a multiplexed address and data bus that can be used to access external memory in conjunction with PORT P2. P0 acts as AD0-AD7, as can be seen from fig 1.1

Oscillator Circuits

The 8051 requires the existence of an external oscillator circuit. The oscillator circuit usually runs around 12MHz, although the 8051 (depending on which specific model) is capable of running at a maximum of 40MHz. Each machine cycle in the 8051 is 12 clock cycles, giving an effective cycle rate at 1MHz (for a 12MHz clock) to 3.33MHz (for the maximum 40MHz clock).

Internal Architecture



Data and Program Memory

The 8051 Microprocessor can be programmed in PL/M, 8051 Assembly, C and a number of other high-level languages. Many compilers even have support for compiling C++ for an 8051.

Program memory in the 8051 is read-only, while the data memory is considered to be read/write accessible. When stored on EEPROM or Flash, the program memory can be rewritten when the microcontroller is in the special programmer circuit.

Program Start Address

The 8051 starts executing program instructions from address 0000 in the program memory.

Direct Memory

The 8051 has 256 bytes of internal addressable RAM, although only the first 128 bytes are available for general use by the programmer. The first 128 bytes of RAM (from 0x00 to 0x7F) are called the **Direct Memory**, and can be used to store data.

Special Function Register

The **Special Function Register** (SFR) is the upper area of addressable memory, from address 0x80 to 0xFF. A, B, PSW, DPTR are called SFR.This area of memory cannot be used for data or program storage, but is instead a series of memory-mapped ports and registers. All port input and output can therefore be performed by memory **mov** operations on specified addresses in the SFR. Also, different status registers are mapped into the SFR, for use in checking the status of the 8051, and changing some operational parameters of the 8051.

General Purpose Registers

The 8051 has 4 selectable banks of 8 addressable 8-bit registers, R0 to R7. This means that there are essentially 32 available general purpose registers, although only 8 (one bank) can be directly accessed at a time. To access the other banks, we need to change the current bank number in the flag status register.

A and B Registers

The A register is located in the SFR memory location 0xE0. The A register works in a similar fashion to the AX register of x86 processors. The A register is called the **accumulator**, and by default it receives the result of all arithmetic operations. The B register is used in a similar manner, except that it can receive the extended answers from the multiply and divide operations. When not being used for multiplication and Division, the B register is available as an extra general-purpose register.

Comparison between Microprocessor and Microcontroller

We have discussed what is a microprocessor and a microcontroller. Let us see the points of differences between them.

No.	Microprocessor	Microcontroller
1.	Microprocessor contains ALU, control unit (clock and timing circuit), different register and interrupt circuit.	Microcontroller contains microprocessor, memory (ROM and RAM), I/O interfacing circuit and peripheral devices such as A/D converter, serial I/O, timer etc.
2.	It has many instructions to move data between memory and CPU.	It has one or two instructions to move data between memory and CPU.
3.	It has one or two bit handling instructions.	It has many bit handling instructions.
4.	Access times for memory and I/O devices are more.	Less access times for built-in memory and I/O devices.
5.	Microprocessor based system requires more hardware.	Microcontroller based system requires less hardware reducing PCB size and increasing the reliability.
6.	Microprocessor based system is more flexible in design point of view.	Less flexible in design point of view.
7.	It has single memory map for data and code.	It has separate memory map for data and code.
8.	Less number of pins are multifunctioned.	More number pins are multifunctioned.

Features of 8051

The features of the 8051 family are as follows :

1) 4096 bytes on - chip program memory.
2) 128 bytes on - chip data memory.
3) Four register banks.
4) 128 User-defined software flags.
5) 64 Kilobytes each program and external RAM addressability.
6) One microsecond instruction cycle with 12 MHz crystal.
7) 32 bidirectional I/O lines organized as four 8-bit ports (16 lines on 8031).
8) Multiple mode, high-speed programmable serial port.
9) Two multiple mode, 16-bit Timers/Counters.
10) Two-level prioritized interrupt structure.
11) Full depth stack for subroutine return linkage and data storage.
12) Direct Byte and Bit addressability.
13) Binary or Decimal arithmetic.
14) Signed-overflow detection and parity computation.
15) Hardware Multiple and Divide in 4 μsec.
16) Integrated Boolean Processor for control applications.
17) Upwardly compatible with existing 8084 software.

8051 Microcontroller Hardware

The Fig. 11.1 shows the internal block diagram of 8051. It consists of a CPU, two kinds of memory sections (data memory - RAM and program memory - EPROM/ROM), input/output ports, special function registers and control logic needed for a variety of peripheral functions. These elements communicate through an eight bit data bus which runs throughout the chip referred as internal data bus. This bus is buffered to the outside world through an I/O port when memory or I/O expansion is desired.

Central Processing Unit (CPU)

The CPU of 8051 consists of eight-bit Arithmetic and Logic unit with associated registers like A, B, PSW, SP, the sixteen bit program counter and "Data pointer" (DPTR) registers. Alongwith these registers it has a set of special function registers. Along with these registers it has a set of special function registers.

The 8051's ALU can perform arithmetic and logic functions on eight bit variables. The arithmetic unit can perform addition, subtraction, multiplication and division. The logic unit can perform logical operations such as AND, OR, and Exclusive-OR, as well as rotate, clear, and complement. The ALU also looks after the branching decisions. An important and unique feature of the 8051 architecture is that the ALU can also manipulate one bit as well as eight-bit data types. Individual bits may be set, cleared, complemented, moved, tested, and used in logic computation.

Internal RAM

The 8051 has 128-byte internal RAM. It is accessed using RAM address register. The Fig. 11.3 shows the organisation of internal RAM. As shown in the Fig. 11.3, internal RAM of 8051 is organised into three distinct areas :

- Working registers
- Bit Addressable
- General Purpose

1. First thirty-two bytes from address 00H to 1FH of internal RAM constitute 32 working registers. They are organised into four banks of eight registers each. The four register banks are numbered 0 to 3 and are consists of eight registers named R_0 to R_7. Each register can be addressed by name or by its RAM address. Only one register bank is in use at a time. Bits RS_0 and RS_1 in the PSW determine which bank of registers is currently in use. Register banks when not selected can be used as general purpose RAM. On reset, the Bank 0 is selected.
2. The 8051 provides 16 bytes of a bit-addressable area. It occupies RAM byte addresses from 20H to 2FH, forming a total of 128 (16 × 8) addressable bits. An addressable bit may be specified by its bit address of 00H to 7FH, or 8 bits may form any byte address from 20H to 2FH. For example, bit address 4EH refers bit 6 of the byte address 29H.
3. The RAM area above bit addressable area from 30H to 7FH is called general purpose RAM. It is addressable as byte.

See Fig. 11.3 on next page.

11.3.4 Internal ROM

The 8051 has 4 Kbyte of internal ROM with address space from 0000H to 0FFFH. It is programmed by manufacturer when the chip is built. This part cannot be erased or altered after fabrication. This is used to store final version of the program.

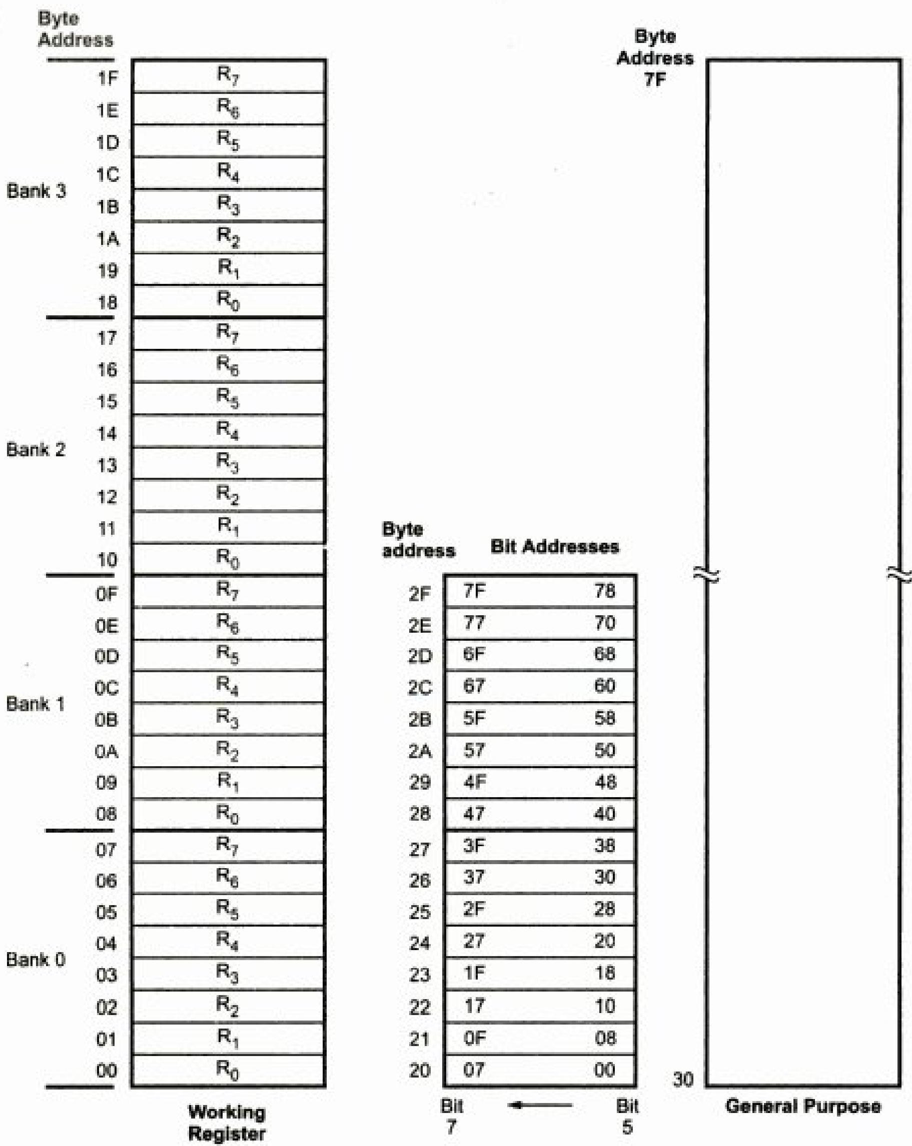


Fig. 11.3 Organisation of internal RAM of 8051

Input/Output Ports

The 8051 has 32 I/O pins configured as four eight-bit parallel ports (P0, P1, P2, and P3). All four ports are bidirectional, i.e. each pin will be configured as input or output (or both) under software control. Each port consists of a latch, an output driver, and an input buffer.

The output drives of Ports 0 and 2 and the input buffers of Port 0, are used to access external memory. As mentioned earlier, Port 0 outputs the low order byte of the external memory address, time multiplexed with the data being written or read, and Port 2 outputs the high order byte of the external memory address when the address is 16 bits wide. Otherwise Port 2 gives the contents of special function register P2.

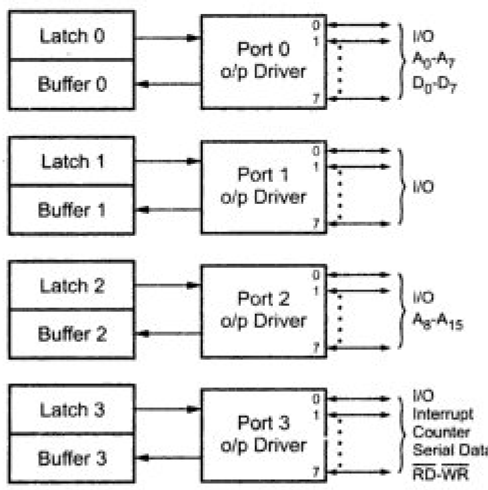


Fig. 11.4 I/O Ports

11.3.6 Register Set of 8051

11.3.6.1 Register A (Accumulator)

It is an 8-bit register. It holds a source operand and receives the result of the arithmetic instructions (addition, subtraction, multiplication, and division). The accumulator can be the source or destination for logical operations and a number of special data movement instructions, including look-up tables and external RAM expansion. Several functions apply exclusively to the accumulator : rotate, parity computation , testing for zero , and so on.

11.3.6.2 Register B

In addition to accumulator, an 8-bit B-register is available as a general purpose register when it is not being used for the hardware multiply/divide operation.

11.3.6.3 Program Status Word (Flag Register)

Many instructions implicitly or explicitly affect (or are affected by) several status flags, which are grouped together to form the Program Status Word. Fig. 11.5 shows the bit pattern of the program status word. It is an 8-bit word, containing the information as follows.

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
CY	AC	F0	RS1	RS0	OV	-	P

CY Bit 7 - Carry flag

AC Bit 6 - Auxiliary carry flag for BCD operations

F0 Bit 5 - User defined flag (Flag zero)

RS1, RS0 Bit 4-3 - Select the working register banks as follows :

RS1	RS0	Bank Selection	
0	0	00H - 07H	Bank 0
0	1	08H - 0FH	Bank 1
1	0	10H - 17H	Bank 2
1	1	18H - 1FH	Bank 3

OV Bit 2 - Overflow flag

– Bit 1 - Reserved

P Bit 0 - Parity flag (1 = Even parity)

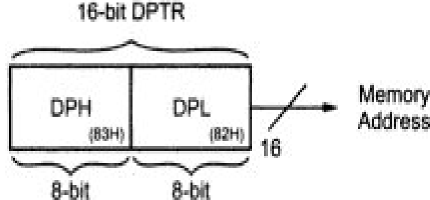
Fig. 11.5 Program status word

11.3.6.4 Stack and Stack Pointer

The stack refers to an area of internal RAM that is used in conjunction with certain opcodes data to store and retrieve data quickly. The stack pointer register is used by the 8051 to hold an internal RAM address that is called **top of stack.** The stack pointer register is 8-bit wide. It is increased **before** data is stored during PUSH and CALL instructions and decremented **after** data is restored during POP and RET instructions. Thus stack array can reside anywhere in on-chip RAM. The stack pointer is initialized to 07H after a reset. This causes the stack to begin at location 08H. The operation of stack and stack pointer is illustrated in Fig. 11.6.

11.3.6.5 Data Pointer (DPTR)

The data pointer (DPTR) consists of a high byte (DPH) and a low byte (DPL). Its function is to hold a 16 bit address. It may be manipulated as a 16 bit data register or as two independent 8 bit registers. It serves as a base register in indirect jumps, lookup table instructions and external data transfer. The DPTR does not have a single internal address; DPH (83H) and DPL (82H) have separate internal addresses.



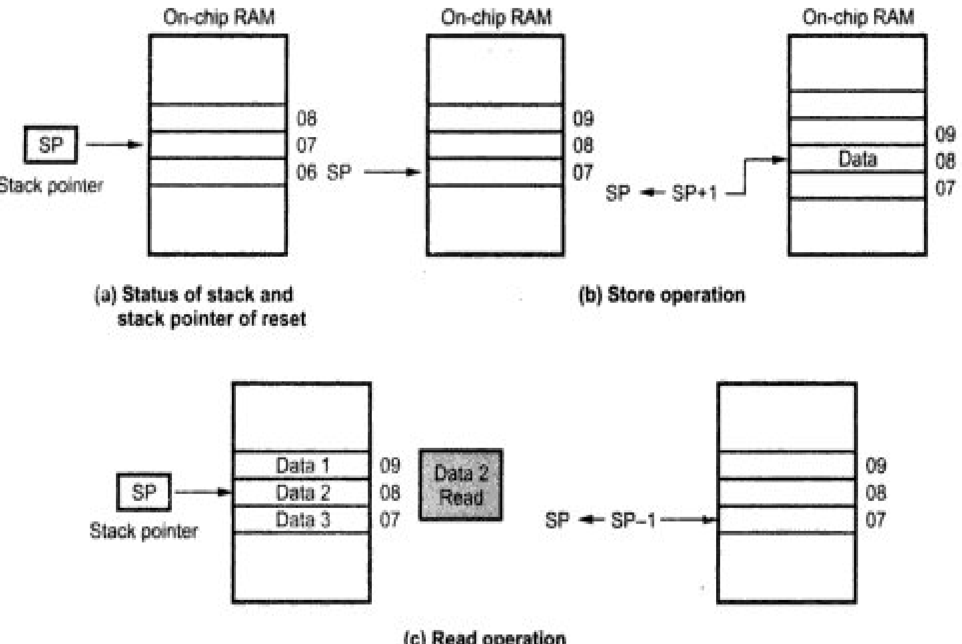


Fig. 11.6

11.3.6.6 Program Counter

The 8051 has a 16-bit program counter. It is used to hold the address of memory location from which the next instruction is to be fetched. Due to this the width of the program counter decides the maximum program length in bytes. For example, 8051 is 16-bit hence it can address upto 2^{16} bytes (64 K) of memory.

The PC is automatically incremented to point the next instruction in the program sequence after execution of the current instruction. It may also be altered by certain instructions. The PC is the only register that does not have an internal address.

11.3.6.7 Special Function Registers

Unlike other microprocessors in the Intel family, 8051 uses memory mapped I/O through a set of special function registers that are implemented in the address space immediately above the 128 bytes of RAM. Fig. 11.7 shows special function bit addresses. All access to the four I/O ports, the CPU registers, interrupt-control registers, the timer/counter, UART, and power control are performed through registers between 80H and FFH.

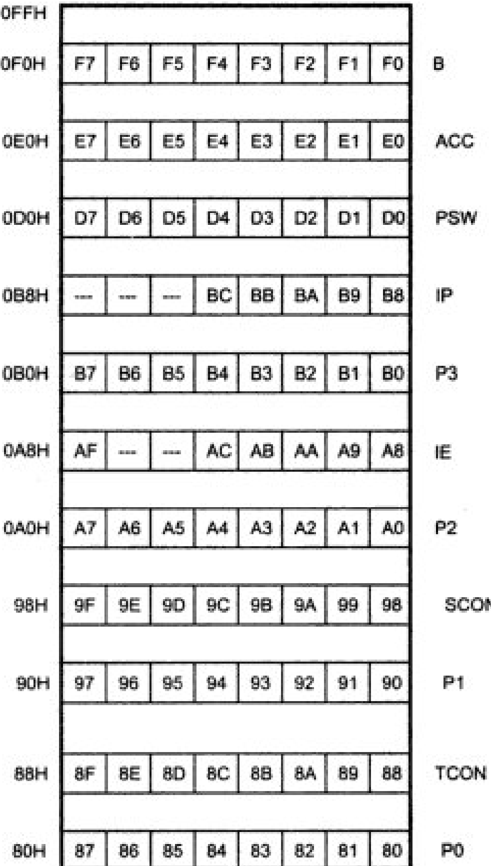


Fig. 11.7 SFR bit address

Symbol	Name	Address	Value in Binary
*ACC	Accumulator	0E0H	0 0 0 0 0 0 0 0
*B	B Register	0F0H	0 0 0 0 0 0 0 0
*PSW	Program Status Word	0D0H	0 0 0 0 0 0 0 0
SP	Stack Pointer	81H	0 0 0 0 0 1 1 1
DPTR	Data Pointer 2 Bytes		
DPL	Low Byte	82H	0 0 0 0 0 0 0 0
DPH	High Byte	83H	0 0 0 0 0 0 0 0
*P0	Port 0	80H	1 1 1 1 1 1 1 1
*P1	Port 1	90H	1 1 1 1 1 1 1 1
*P2	Port 2	0A0H	1 1 1 1 1 1 1 1
*P3	Port 3	0B0H	1 1 1 1 1 1 1 1
*IP	Interrupt Priority Control	0B8H	8051 X X X 0 0 0 0 0 8052 X X 0 0 0 0 0 0
*IE	Interrupt Enable Control	0A8H	8051 0 X X 0 0 0 0 0 8052 0 X 0 0 0 0 0 0
TMOD	Timer/Counter Mode Control	89H	0 0 0 0 0 0 0 0
*TCON	Timer/Counter Control	88H	0 0 0 0 0 0 0 0
* + T2CON	Timer/Counter 2 Control	0C8H	0 0 0 0 0 0 0 0
TH0	Timer/Counter 0 High Byte	8CH	0 0 0 0 0 0 0 0
TL0	Timer/Counter 0 Low Byte	8AH	0 0 0 0 0 0 0 0
TH1	Timer/Counter 1 High Byte	8DH	0 0 0 0 0 0 0 0
TL1	Timer/Counter 1 LowByte	8BH	0 0 0 0 0 0 0 0
+ TH2	Timer/Counter 2 High Byte	0CDH	0 0 0 0 0 0 0 0
+ TL2	Timer/Counter 2 Low Byte	0CCH	0 0 0 0 0 0 0 0
+ RCAP2H	T/C 2 Capture Reg. High Byte	0CBH	0 0 0 0 0 0 0 0
+ RCAP2L	T/C 2 Capture Reg. Low Byte	0CAH	0 0 0 0 0 0 0 0
* SCON	Serial Control	98H	0 0 0 0 0 0 0 0
SBUF	Serial Data Buffer	99H	Interminate
PCON	Power Control	87H	HMOS 0 X X X X X X X CHMOS 0 X X X 0 0 0 0

Table 11.3 List of all SFRs (* = Bit addressable, + = 8052 only)

Memory Organization in 8051

Fig. 11.8 shows the basic memory structure for 8051. It can access upto 64 K program memory and 64 K data memory. The 8051 has 4 Kbytes of internal program memory and 256 bytes of internal data memory.

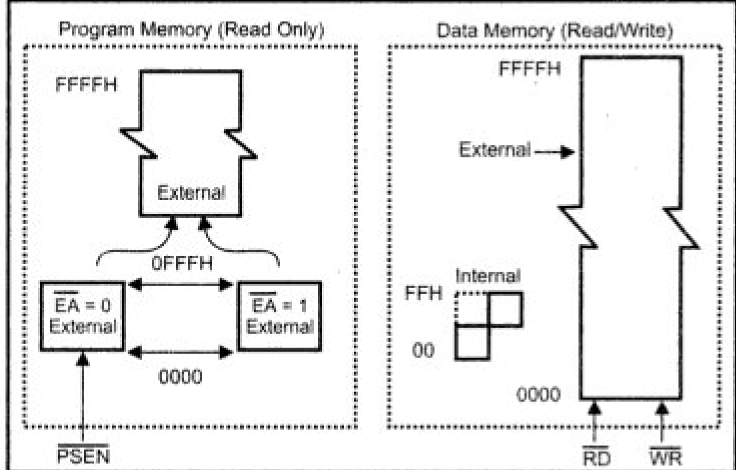


Fig. 11.8 Memory structure

External Data Memory and Program Memory

We have seen that 8051 has internal data and code memory with limited memory capacity. This memory capacity may not be sufficient for some applications. In such situations, we have to connect external ROM/EPROM and RAM to 8051 microcontroller to increase the memory capacity. We also know that ROM is used as a program memory and RAM is used as a data memory. Let us see how 8051 accesses these memories.

External Program Memory

Fig. 11.10 shows a map of the 8051 program memory.

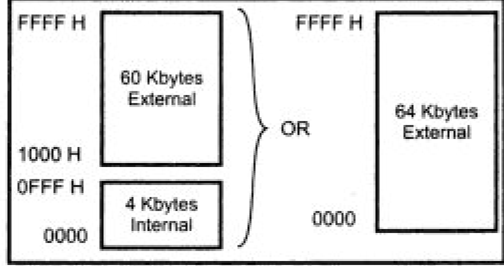


Fig. 11.10 The 8051 program memory

In 8051, when the $\overline{EA}$ pin is connected to V_{CC}, program fetches to addresses 0000H through 0FFFH are directed to the internal ROM and program fetches to addresses 1000H through FFFFH are directed to external ROM/EPROM. On the other hand when $\overline{EA}$ pin is grounded, all addresses (0000H to FFFFH) fetched by program are directed to the external ROM/EPROM. The $\overline{PSEN}$ signal is used to activate output enable signal of the external ROM/EPROM, as shown in the Fig. 11.11.

External Data Memory and Program Memory

We have seen that 8051 has internal data and code memory with limited memory capacity. This memory capacity may not be sufficient for some applications. In such situations, we have to connect external ROM/EPROM and RAM to 8051 microcontroller to increase the memory capacity. We also know that ROM is used as a program memory and RAM is used as a data memory. Let us see how 8051 accesses these memories.

External Program Memory

Fig. 11.10 shows a map of the 8051 program memory.

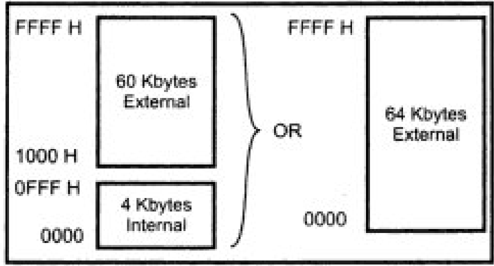


Fig. 11.10 The 8051 program memory

In 8051, when the $\overline{EA}$ pin is connected to V_{CC}, program fetches to addresses 0000H through 0FFFH are directed to the internal ROM and program fetches to addresses 1000H through FFFFH are directed to external ROM/EPROM. On the other hand when $\overline{EA}$ pin is grounded, all addresses (0000H to FFFFH) fetched by program are directed to the external ROM/EPROM. The $\overline{PSEN}$ signal is used to activate output enable signal of the external ROM/EPROM, as shown in the Fig. 11.11.

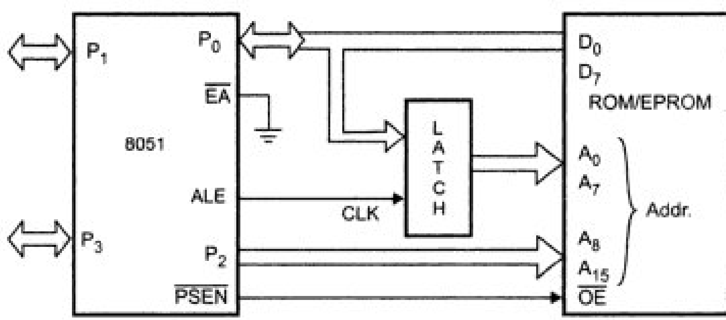
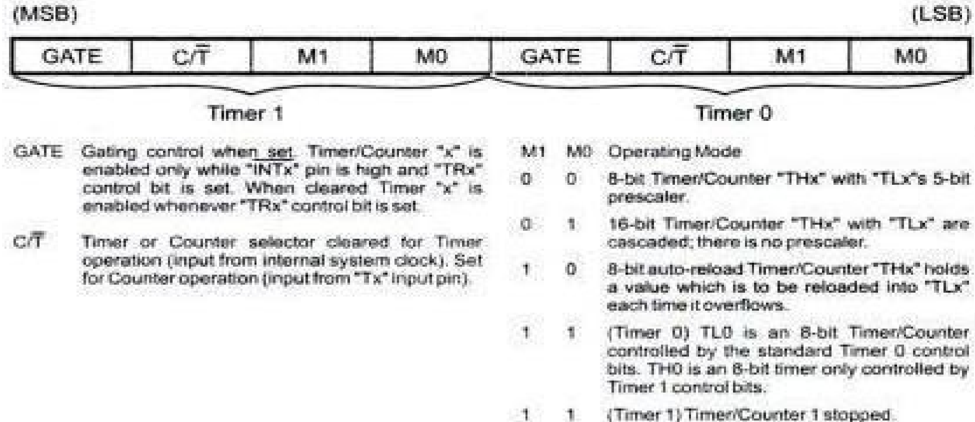


Fig. 11.11 Accessing external program memory

Copyrighted material

Timer 0 and Timer 1

In this timers, "Timer" or Counter" mode is selected by control bits $C/\overline{T}$ in the Special Function Register TMOD (Fig. 11.18). These two Timer/Counters have four operating modes, which are selected by bit-pairs (M1, M0) in TMOD. Modes 0, 1 and 2 are same for both Timer/Counters. Mode 3 is different. The four operating modes are described as follows :



TMOD : Timer/counter mode control register

MODE 0

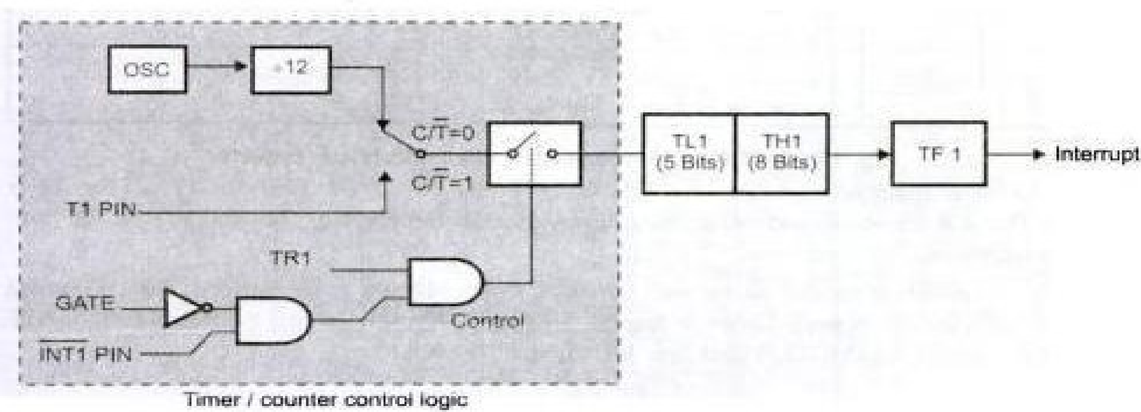


Fig. 11.19 Timer/counter 1 mode 0 : 13-bit counter

Both Timers in Mode 0 is an 8-bit Counter with a divide-by-32 prescaler. This 13-bit timer is MCS-48 compatible. Fig. 11.19 shows the Mode 0 operation as it applies to Timer 1. In this mode, the Timer register is configured as a 13-bit register. As the count rolls over from all 1s to all 0s, it sets the Timer interrupt flag TF1. The counted input is enabled to the Timer when TR1 = 1 and either GATE = 0 or $\overline{INT1}$ = 1. (Setting GATE = 1 allows the Timer to be controlled by external input $\overline{INT1}$, to facilitate pulse width measurements.) TR1 is a control bit in the Special Function Register TCON (Fig. 11.20) GATE is in TMOD.

(MSB)							(LSB)
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Symbol	Position	Name and Significance
TF1	TCON.7	Timer 1 Overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.
TR1	TCON.6	Timer 1 Run control bit. Set/cleared by software to turn timer/counter on/off.
TF0	TCON.5	Timer 0 Overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.
TR0	TCON.4	Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.
IE1	TCON.3	Interrupt 1 Edge Flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
IT1	TCON.2	Interrupt 1 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.
IE0	TCON.1	Interrupt 0 Edge Flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
IT0	TCON.0	Interrupt 0 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.

Fig 11.20 TCON-timer/counter control/status register

The 13-bit register consists of all 8 bits of TH1 and the lower 5 bits of TL1. The upper 3 bits of TL1 are indeterminate and should be ignored. Setting the run flag (TR1) does not clear the registers.

RI	SCON.0	Receive Interrupt flag. Set by hardware when byte received. Cleared by software after servicing.
	Note : Mode	The state of (SM0, SM1) selects ; **SM0 SM1**
	0	0 0 - Shift register ; baud = f/12
	1	0 1 - 8-bit UART, variable data rate.
	2	1 0 - 9-bit UART, fixed data rate ; baud = f/32 or f/64
	3	1 1 - 9-bit UART, variable data rate.

Fig. 11.24 (a) SCON-serial port control/status register

(MSB) 7	6	5	4	3	2	1	(LSB) 0
SMOD	–	–	–	GF1	GF0	PD	IDL

Symbol	Position	Name and Significance
SMOD	PCON.7	Serial baud rate modify bit. It is 0 at reset. It is set to 1 by program to double the baud rate.
–	PCON.6-4	Not defined
GF1	PCON.3	General purpose user flag bit 1. Set/cleared by program.
GF0	PCON.2	General purpose user flag bit 0. Set/cleared by program.
PD	PCON.1	Power down bit. It is set to 1 by program to enter power down configuration for CHMOS microcontrollers.
IDL	PCON.0	Idle mode bit. It is set to 1 by program to enter idle mode configuration for CHMOS microcontrollers.
Note : PCON is not bit addressable		

Fig. 11.24 (b) PCON register

Operating Modes for Serial Port

MODE 0

In this mode, serial data enters and exits through RXD. TXD outputs the shift clock. 8 bits are transmitted/received : 8 data bits (LSB first). The baud rate is fixed at 1/12 the oscillator frequency.

MODE 1

In this mode, 10 bits are transmitted (through TXD) or received (through RXD) : a start bit (0), 8 data bits (LSB first), and a stop bit (1). On receive, the stop bit goes into RB8 in Special Function Register SCON. The baud rate is variable.

MODE 2

In this mode, 11 bits are transmitted (thorugh TXD) or received (through RXD) : a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On Transmit, the 9th data bit (TB8 in SCON) can be assigned the value of 0 or 1. Or, for example, the parity bit (P, in the PSW) could be moved into TB8. On receive, the 9th data bit goes into RB8 in Special Function Regsiter SCON, while the stop bit is ignored. The baud rate is programmable to either $\frac{1}{32}$ or $\frac{1}{64}$ the oscillator frequency.

MODE 3

In this mode, 11 bits are transmitted (through TXD) or received (through RXD) : a start bit (0), 8 data bits (LSB first), a programmable 9th data bit and a stop bit (1). In fact, Mode 3 is the same as Mode 2 in all respects except the baud rate. The baud rate in Mode 3 is variable.

In all four modes, transmission is initiated by any instruction that uses SBUF as a destination register. Reception is initiated in Mode 0 by the condition RI = 0 and REN = 1. Reception is initiated in the other modes by the incoming start bit if REN = 1.

The Table 11.9 summaries the four serial port modes provided by 8051.

Mode	Transmission Format	Baud Rate
0	8-data bits	$\frac{1}{12}$ oscillator frequency
1	10-bit (start bit + 8-data bits + stop bit)	Variable
2	11-bit (start bit + 8-data bits + programmable 9th data bit + stop bit)	Programmable to either $\frac{1}{32}$ or $\frac{1}{64}$ oscillator frequency
3	11-bit (start bit + 8 data bit + programmable 9th data bit + stop bit)	Variable

Table 11.9 Summary of serial port modes

ADDRESSING MODES OF 8051

Various methods of accessing the data are called addressing

modes.8051 addressing modes are classified as follows.

1. Immediate addressing.
2. Register addressing.
3. Direct addressing.
4. Indirect addressing.
5. Relative addressing.
6. Absolute addressing.
7. Long addressing.
8. Indexed addressing.
9. Bit inherent addressing.
10. Bit direct addressing.

1. Immediate addressing.

In this addressing mode the data is provided as a part of instruction itself. In other words data immediately follows the instruction.

Eg. MOV A,#30H

ADD A, #83   # Symbol indicates the data is immediate.

2. *Register addressing.*

In this addressing mode the register will hold the data. One of the eight general registers(R0 to R7) can be used and specified as the operand.

Eg. MOV
A,R0
ADD
A,R6

R0 – R7 will be selected from the current selection of register bank. The default register bank will be bank 0.

3. *Direct addressing*

There are two ways to access the internal memory. Using direct address and indirect address. Using direct addressing mode we can not only address the internal memory but SFRs also. In direct addressing, an 8 bit internal data memory address is specified as part of the instruction and hence, it can specify the address only in the range of 00H to FFH. In this addressing mode, data is obtained directly from the memory.

Eg. MOV
A,60h
ADD
A,30h

4. *Indirect addressing*

The indirect addressing mode uses a register to hold the actual address that will be used in data movement. Registers R0 and R1 and DPTR are the only registers that can be used as data pointers. Indirect addressing cannot be used to refer to SFR registers. Both R0 and R1 can hold 8 bit address and DPTR can hold16 bit address.

Eg. MOV A,@R0
ADD A,@R1
MOVX
A,@DPTR

5. *Indexed addressing.*

In indexed addressing, either the program counter (PC), or the data pointer (DTPR)—is used to hold the base address, and the A is used to hold the offset address. Adding the value of the base address to the value of the offset address forms the effective address. Indexed addressing is used with JMP or MOVC instructions. Look up tables are easily implemented with the help of index addressing.

Eg. MOVC A, @A+DPTR // *copies the contents of memory location pointed by the sum of the accumulator A and the DPTR into accumulator A.*

MOVC A, @A+PC // *copies the contents of memory location pointed by the sum of the accumulator A and the program counter into accumulator A.*

6. *Relative Addressing.*

Relative addressing is used only with conditional jump instructions. The relative address, (offset), is an 8 bit signed number, which is automatically added to the PC to make the address of the next instruction. The 8 bit signed offset value gives an address range of +127 to —128 locations.The jump destination is usually specified using a label and the assembler calculates the jump offset accordingly. The advantage of relative addressing is that the program code is easy to relocate and the address is relative to position in the memory.

Eg. SJMP
LOOP1JC
BACK

7. *Absolute addressing*

Absolute addressing is used only by the AJMP (Absolute Jump) and ACALL (Absolute Call) instructions. These are 2 bytes instructions. The absolute addressing mode specifies the lowest 11bit of the memory address as part of the instruction. The upper 5 bit of the destination address are the upper 5 bit of the current program counter. Hence, absolute addressing allows branching onlywithin the current 2 Kbyte page of the program memory.

Eg. AJMP LOOP1
ACALL
LOOP2

8. *Long Addressing*

The long addressing mode is used with the instructions LJMP and LCALL. These are 3 byte instructions. The address specifies a full 16 bit destination address so that a jump or a call can be made to a location within a 64 Kbyte code memory space.

Eg. LJMP FINISH
LCALL
DELAY

9. *Bit Inherent Addressing*

In this addressing, the address of the flag which contains the operand, is implied in the opcodeof the instruction.

Eg. CLR C ; *Clears the carry flag to 0*

10. *Bit Direct Addressing*

In this addressing mode the direct address of the bit is specified in the instruction. The RAM space 20H to 2FH and most of the special function registers are bit addressable. Bit address values are between 00H to 7FH.

Eg. CLR 07h ; *Clears the bit 7 of 20h RAM space*
SETB 07H ; *Sets the bit 7 of 20H RAM space.*

INSTRUCTION SET OF 8051

1. *Instruction Timings*

The 8051 internal operations and external read/write operations are controlled by the oscillatorclock.

T-state, Machine cycle and Instruction cycle are terms used in instruction timings.

T-state is defined as one subdivision of the operation performed in one clock period. The terms 'T-state' and 'clock period' are often used synonymously.

Machine cycle is defined as 12 oscillator periods. A machine cycle consists of six states and each state lasts for two oscillator periods. An instruction takes one to four machine cycles to execute aninstruction. ***Instruction cycle*** is defined as the time required for completing the execution of an instruction. The 8051 instruction cycle consists of one to four machine cycles.

Eg. If 8051 microcontroller is operated with 12 MHz oscillator, find the execution time for the following four instructions.

1. *ADD A, 45H*
2. *SUBB A, #55H*
3. *MOV DPTR, #2000H*
4. *MUL AB*

Since the oscillator frequency is 12 MHz, the clock period is, Clock period = 1/12 MHz = ***0.08333 µS****.*
*Time for 1 machine cycle = 0.08333 µS x 12 =**1 µS**.*

	Instruction	**No. of machine cycles**	**Execution time**
1.	*ADD A, 45H*	*1*	*1 µs*
2.	*SUBB A, #55H*	*2*	*2 µs*
3.	*MOV DPTR, #2000H*	*2*	*2 µs*
4.	*MUL AB*	*4*	*4 µs*

2. *8051 Instructions*

The instructions of 8051 can be broadly classified under the following headings.

1. Data transfer instructions
2. Arithmetic instructions
3. Logical instructions
4. Branch instructions
5. Subroutine instructions
6. Bit manipulation instructions

Data transfer instructions.

In this group, the instructions perform data transfer operations of the following types.

a. Move the contents of a register Rn to A
   i. MOV A,R2
   ii. MOV A,R7
b. Move the contents of a register A to Rn
   i. MOV R4,A
   ii. MOV R1,A
c. Move an immediate 8 bit data to register A or to Rn or to a memory location(direct or indirect)
   i. MOV A, #45H
   ii. MOV R6, #51H
   iii. MOV 30H, #44H
   iv. MOV @R0, #0E8H
   v. MOV DPTR, #0F5A2H
   vi. MOV DPTR, #5467H
d. Move the contents of a memory location to A or A to a memory location using direct and indirect addressing
   i. MOV A, 65H
   ii. MOV A, @R0
   iii. MOV 45H, A
   iv. MOV @R1, A
e. Move the contents of a memory location to Rn or Rn to a memory location using direct addressing
   i. MOV R3, 65H
   ii. MOV 45H, R2
f. Move the contents of memory location to another memory location using direct and indirect addressing
   i. MOV 47H, 65H
   ii. MOV 45H, @R0
g. Move the contents of an external memory to A or A to an external memory
   i. MOVX A,@R1
   ii. MOVX @R0,A
   iii. MOVX A,@DPTR
   iv. MOVX@DPTR,A
h. Move the contents of program memory to A
   i. MOVC A, @A+PC
   ii. MOVC A, @A+DPTR

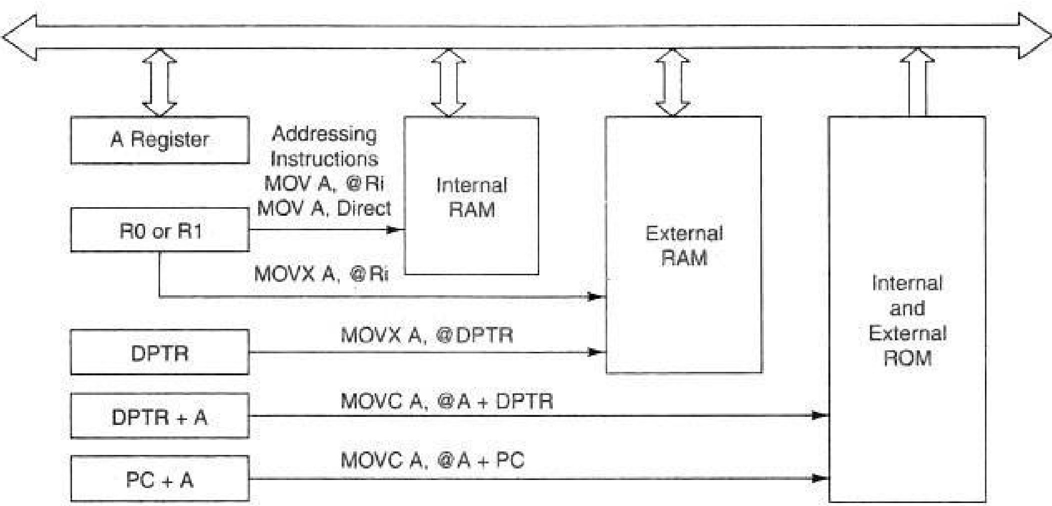


FIG. Addressing Using MOV, MOVX and MOVC

i. Push and Pop instructions

```
                [SP]=07          //CONTENT OF SP IS 07 (DEFAULT
VALUE)MOV R6, #25H               [R6]=25H     //CONTENT OF R6 IS 25H
MOV R1, #12H    [R1]=12H         //CONTENT OF R1 IS
12HMOV R4, #0F3H                 [R4]=F3H     //CONTENT
OF R4 IS F3H

PUSH 6          [SP]=08          [08]=[06]=25H //CONTENT OF 08 IS 25H
PUSH 1          [SP]=09          [09]=[01]=12H //CONTENT OF 09 IS 12H
PUSH 4          [SP]=0A          [0A]=[04]=F3H //CONTENT OF 0A IS
F3H

POP 6           [06]=[0A]=F3H [SP]=09      //CONTENT OF 06 IS
F3HPOP 1        [01]=[09]=12H [SP]=08      //CONTENT OF 01 IS 12H
POP 4           [04]=[08]=25H [SP]=07      //CONTENT OF 04 IS 25H
```

j. Exchange instructions

The content of source ie., register, direct memory or indirect memory will be exchangedwith the contents of destination ie., accumulator.

i. XCH A,R3
ii. XCH A,@R1
iii. XCH A,54h

k. Exchange digit. Exchange the lower order nibble of Accumulator (A0-A3) with lower order nibble of the internal RAM location which is indirectly addressed by the register.

i. XCHD A,@R1
ii. XCHD A,@R0

Arithmetic instructions.

The 8051 can perform addition, subtraction. Multiplication and division operations on 8 bit numbers.

Addition

In this group, we have instructions to

i. Add the contents of A with immediate data with or without carry.
   i. ADD A, #45H
   ii. ADDC A, #OB4H

ii. Add the contents of A with register Rn with or without carry.
   i. ADD A, R5
   ii. ADDC A, R2

iii. Add the contents of A with contents of memory with or without carry using direct and indirect addressing
   i. ADD A, 51H
   ii. ADDC A, 75H
   iii. ADD A, @R1
   iv. ADDC A, @R0

CY AC and OV flags will be affected by this operation.

Subtraction

In this group, we have instructions to

i. Subtract the contents of A with immediate data with or without carry.
   i. SUBB A, #45H
   ii. SUBB A, #OB4H

ii. Subtract the contents of A with register Rn with or without carry.
   i. SUBB A, R5
   ii. SUBB A, R2

iii. Subtract the contents of A with contents of memory with or without carry using direct and indirect addressing
   i. SUBB A, 51H
   ii. SUBB A, 75H
   iii. SUBB A, @R1
   iv. SUBB A, @R0

CY AC and OV flags will be affected by this operation.

Multiplication

MUL AB. This instruction multiplies two 8 bit unsigned numbers which are stored in A and B register. After multiplication the lower byte of the result will be stored in accumulator and higher byte of result will be stored in B register.

```
Eg.   MOV A,#45H      ;[A]=45H
      MOV B,#0F5H     ;[B]=F5H
      MUL AB          ;[A] x [B] = 45 x F5 = 4209
                      ;[A]=09H, [B]=42H
```

Division

DIV AB. This instruction divides the 8 bit unsigned number which is stored in A by the 8 bit unsigned number which is stored in B register. After division the result will be stored in accumulator and remainder will be stored in B register.

```
Eg.   MOV A,#45H        ;[A]=0E8H
      MOV B,#0F5H       ;[B]=1BH
      DIV AB            ;[A] / [B] = E8 /1B = 08 H with remainder 10H
                        ;[A] = 08H, [B]=10H
```

DA A (Decimal Adjust After Addition).

When two BCD numbers are added, the answer is a non-BCD number. To get the result in BCD, weuse DA A instruction after the addition. DA A works as follows.

- If lower nibble is greater than 9 or auxiliary carry is 1, 6 is added to lower nibble.
- If upper nibble is greater than 9 or carry is 1, 6 is added to upper nibble.

```
Eg 1:   MOV A,#23H
        MOV R1,#55H
        ADD A,R1        // [A]=78
        DA A            // [A]=78          no changes in the accumulator after da a

Eg 2:   MOV A,#53H
        MOV R1,#58H
        ADD A,R1        // [A]=ABh
        DA A            // [A]=11, C=1 . ANSWER IS 111. Accumulator data is changed after DA A
```

Increment: *increments the operand by one.*

INC A **INC Rn** **INC DIRECT** **INC @RiINC DPTR**

INC increments the value of source by 1. If the initial value of register is FFh, incrementing the value will cause it to reset to 0. The Carry Flag is not set when the value "rolls over" from 255 to 0.

In the case of "INC DPTR", the value two-byte unsigned integer value of DPTR is incremented. If the initial value of DPTR is FFFFh, incrementing the value will cause it to reset to 0.

Decrement: *decrements the operand by one.*

DEC A **DEC Rn DEC DIRECT** **DEC @Ri**

DEC decrements the value of *source* by 1. If the initial value of is 0, decrementing the value will cause it to reset to FFh. The Carry Flag is not set when the value "rolls over" from 0 to FFh.

Logical Instructions

Logical AND

ANL destination, source: ANL does a bitwise "AND" operation between *source* and *destination*, leaving the resulting value in *destination*. The value in source is not affected. "AND" instruction logically AND the bits of source and destination.

ANL A,#DATA ANL A,
Rn ANL A,DIRECT ANL
A,@Ri
ANL DIRECT,A ANL DIRECT, #DATA

Logical OR

ORL destination, source: ORL does a bitwise "OR" operation between *source* and *destination*,

leaving the resulting value in *destination*. The value in source is not affected. " OR " instruction logically OR the bits of source and destination.
ORL A,#DATA ORL A,
Rn ORL A,DIRECT ORL
A,@Ri
ORL DIRECT,A ORL DIRECT, #DATA

Logical Ex-OR

XRL destination, source: XRL does a bitwise "EX-OR" operation between *source* and *destination*, leaving the resulting value in *destination*. The value in source is not affected. " XRL " instruction logically EX-OR the bits of source and destination.
XRL A,#DATA XRL A,Rn
XRL A,DIRECT XRL
A,@Ri
XRL DIRECT,A XRL DIRECT, #DATA

Logical NOT

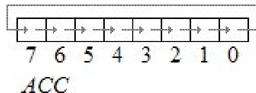
CPL complements *operand*, leaving the result in *operand*. If *operand* is a single bit then the state ofthe bit will be reversed. If *operand* is the Accumulator then all the bits in the Accumulator will be reversed.

CPL A, CPL C, CPL bit address

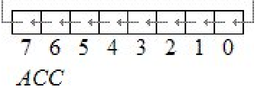
SWAP A – Swap the upper nibble and lower nibble of A.

Rotate Instructions

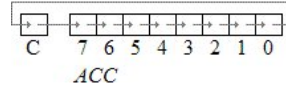
RR A
This instruction is rotate right the accumulator. Its operation is illustrated below. Each bit is shifted one location to the right, with bit 0 going to bit 7.



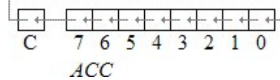
RL A
Rotate left the accumulator. Each bit is shifted one location to the left, with bit 7 going to bit 0



RRC A
Rotate right through the carry. Each bit is shifted one location to the right, with bit 0 going into the carry bit in the PSW, while the carry was at goes into bit 7



RLC A
Rotate left through the carry. Each bit is shifted one location to the left, with bit 7 going into the carry bit in the PSW, while the carry goes into bit 0.



Branch (JUMP) Instructions

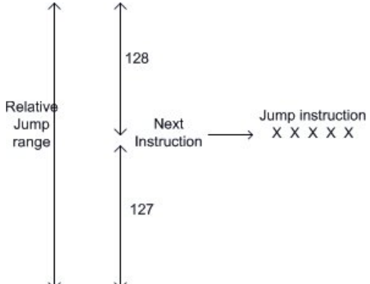
Jump and Call Program Range

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

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

Relative Jump

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



The advantages of the relative jump are as follows:-

1. Only 1 byte of jump address needs to be specified in the 2's complement form, ie. For jumping ahead, the range is 0 to 127 and for jumping back, the range is -1 to -128.
2. Specifying only one byte reduces the size of the instruction and speeds up programexecution.
3. The program with relative jumps can be relocated without reassembling to generateabsolute jump addresses.

Disadvantages of the absolute jump: -

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

Instructions that use Relative Jump

SJMP <relative address>; *this is unconditional jump*

The remaining relative jumps are conditional jumps

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

Short Absolute Jump

In this case only 11bits of the absolute jump address are needed. The absolute jump address is calculated in the following manner.

In 8051, 64 kbyte of program memory space is divided into 32 pages of 2 kbyte each. The hexadecimal addresses of the pages are given as follows:-

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

It can be seen that the upper 5bits of the program counter (PC) hold the page number and the lower 11bits of the PC hold the address within that page. Thus, an absolute address is formed by taking page numbers of the instruction (from the program counter) following the jump and attaching the specified 11bits to it to form the 16-bit address.

Advantage: The instruction length becomes 2

bytes.Example of short absolute jump: -

ACALL <address 11>
AJMP <address 11>

Long Absolute Jump/Call

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

Example: -

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

Another classification of jump instructions is

1. Unconditional Jump
2. Conditional Jump

1. **The unconditional jump** is a jump in which control is transferred unconditionally to the target location.
   a. **LJMP** (long jump). This is a 3-byte instruction. First byte is the op-code and second and third bytes represent the 16-bit target address which is any memory location from 0000 to FFFFH
   eg: LJMP 3000H
   b. **AJMP:** this causes unconditional branch to the indicated address, by loading the 11 bit address to 0 -10 bits of the program counter. The destination must be therefore within the same 2K blocks.
   c. **SJMP** (short jump). This is a 2-byte instruction. First byte is the op-code and second byte is the relative target address, 00 to FFH (forward +127 and backward -128 bytes from the current PC value). To calculate the target address of a short jump, the second byte is added to the PC value which is address of the instruction immediately below the jump.

2. Conditional Jump instructions.

JBC	Jump if bit = 1 and clear bit
JNB	Jump if bit = 0
JB	Jump if bit = 1
JNC	Jump if CY = 0
JC	Jump if CY = 1
CJNE reg,#data	Jump if byte ≠ #data
CJNE A,byte	Jump if A ≠ byte
DJNZ	Decrement and Jump if A ≠ 0
JNZ	Jump if A ≠ 0
JZ	Jump if A = 0

All conditional jumps are short jumps.

Bit level jump instructions:

Bit level JUMP instructions will check the conditions of the bit and if condition is true, it jumps to theaddress specified in the instruction. All the bit jumps are relative jumps.

JB bit, rel ; jump if the direct bit is set to the relative address specified. JNB
bit, rel ; jump if the direct bit is clear to the relative address specified.
JBC bit, rel ; jump if the direct bit is set to the relative address specified and then clear the bit.

Subroutine CALL And RETURN Instructions

Subroutines are handled by CALL and RET instructionsThere

are two types of CALL instructions

1. **LCALL address(16 bit)**
   This is long call instruction which unconditionally calls the subroutine located at the indicated 16 bit address. This is a 3 byte instruction. The LCALL instruction works as follows.
   a. During execution of LCALL, [PC] = [PC]+3; (if address where LCALL resides is say, 0x3254; during execution of this instruction [PC] = 3254h + 3h = 3257h
   b. [SP]=[SP]+1; (if SP contains default value 07, then SP increments and [SP]=08
   c. [[SP]] = [PC_{7-0}]; (lower byte of PC content ie., 57 will be stored in memory location 08.
   d. [SP]=[SP]+1; (SP increments again and [SP]=09)
   e. [[SP]] = [PC_{15-8}]; (higher byte of PC content ie., 32 will be stored in memory location 09.

   With these the address (0x3254) which was in PC is stored in stack.
   f. [PC]= address (16 bit); the new address of subroutine is loaded to PC. No flags are affected.

2. **ACALL address(11 bit)**
   This is absolute call instruction which unconditionally calls the subroutine located at the indicated 11 bit address. This is a 2 byte instruction. The SCALL instruction works as follows.
   a. During execution of SCALL, [PC] = [PC]+2; (if address where LCALL resides is say, 0x8549; during execution of this instruction [PC] = 8549h + 2h = 854Bh
   b. [SP]=[SP]+1; (if SP contains default value 07, then SP increments and [SP]=08
   c. [[SP]] = [PC_{7-0}]; (lower byte of PC content ie., 4B will be stored in memory location 08.
   d. [SP]=[SP]+1; (SP increments again and [SP]=09)
   e. [[SP]] = [PC_{15-8}]; (higher byte of PC content ie., 85 will be stored in memory location 09.

   With these the address (0x854B) which was in PC is stored in stack.

f. $[PC_{10-0}]$= address (11 bit); the new address of subroutine is loaded to PC. No flags are affected.

RET instruction

RET instruction pops top two contents from the stack and load it to PC.

g. $[PC_{15-8}]$ = [[SP]] ;content of current top of the stack will be moved to higher byte of PC.
h. [SP]=[SP]-1; (SP decrements)
i. $[PC_{7-0}]$ = [[SP]] ;content of bottom of the stack will be moved to lower byte of PC.
j. [SP]=[SP]-1; (SP decrements again)

Bit manipulation instructions.

8051 has 128 bit addressable memory. Bit addressable SFRs and bit addressable PORT pins. It is possible to perform following bit wise operations for these bit addressable locations.

1. LOGICAL AND
   a. ANL C,BIT(BIT ADDRESS) ; 'LOGICALLY AND' CARRY AND CONTENT OF BIT ADDRESS, STORE RESULT IN CARRY
   b. ANL C, /BIT; ; 'LOGICALLY AND' CARRY AND COMPLEMENT OF CONTENT OF BIT ADDRESS, STORE RESULT IN CARRY
2. LOGICAL OR
   a. ORL C,BIT(BIT ADDRESS) ; 'LOGICALLY OR' CARRY AND CONTENT OF BIT ADDRESS, STORE RESULT IN CARRY
   b. ORL C, /BIT; ; 'LOGICALLY OR' CARRY AND COMPLEMENT OF CONTENT OF BIT ADDRESS, STORE RESULT IN CARRY
3. CLR bit
   a. CLR bit ; CONTENT OF BIT ADDRESS SPECIFIED WILL BE CLEARED.
   b. CLR C ; CONTENT OF CARRY WILL BE CLEARED.
4. CPL bit
   a. CPL bit ; CONTENT OF BIT ADDRESS SPECIFIED WILL BE COMPLEMENTED.
   b. CPL C ; CONTENT OF CARRY WILL BE COMPLEMENTED.

ASSEMBLY LANGUAGE PROGRAMS IN 8051

1. Write a program to add the values of locations 50H and 51H and store the result in locationsin 52h and 53H.

```
ORG 0000H          ; Set program counter 0000H
MOV A,50H          ; Load the contents of Memory location 50H into A ADD ADD A,51H
        ; Add the contents of memory 51H with CONTENTS A
MOV 52H,A          ; Save the LS byte of the result in 52H
MOV A, #00         ; Load 00H into A
ADDC A, #00        ; Add the immediate data and carry to A
MOV 53H,A          ; Save the MS byte of the result in location 53h
END
```

2. Write a program to store data FFH into RAM memory locations 50H to 58H using direct addressing mode

```
ORG 0000H          ; Set program counter 0000H
MOV A, #0FFH  ; Load FFH into A
MOV 50H, A         ; Store contents of A in location 50H
```

```
MOV 51H, A      ; Store contents of A in location 5IH
MOV 52H, A      ; Store contents of A in location 52H
MOV 53H, A      ; Store contents of A in location 53H
MOV 54H, A      ; Store contents of A in location 54H
MOV 55H, A      ; Store contents of A in location 55H
MOV 56H, A      ; Store contents of A in location 56H
MOV 57H, A      ; Store contents of A in location 57H
MOV 58H, A      ; Store contents of A in location 58H
END
```

3. **Write a program to subtract a 16 bit number stored at locations 51H-52H from 55H-56H and store the result in locations 40H and 41H. Assume that the least significant byte of data or theresult is stored in low address. If the result is positive, then store 00H, else store 01H in 42H.** ORG 0000H

```
; Set program counter 0000H
MOV A, 55H    ; Load the contents of memory location 55 into A
CLR C         ; Clear the borrow flag
SUBB A,51H    ; Sub the contents of memory 51H from contents of A
MOV 40H, A    ; Save the LSByte of the result in location 40H
MOV A, 56H    ; Load the contents of memory location 56H into A
SUBB A, 52H   ; Subtract the content of memory 52H from the content A
MOV 41H,      ; Save the MSbyte of the result in location 415.
MOV A, #00    ; Load 005 into A
ADDC A, #00   ; Add the immediate data and the carry flag to A MOV
42H, A        ; If result is positive, store00H, else store 0lH in 42H
END
```

4. **Write a program to add two 16 bit numbers stored at locations 51H-52H and 55H-56H and store the result in locations 40H, 41H and 42H. Assume that the least significant byte of data and the result is stored in low address and the most significant byte of data or the resultis stored in high address.**

```
ORG 0000H       ; Set program counter 0000H
MOV A,51H       ; Load the contents of memory location 51H into A
ADD A,55H       ; Add the contents of 55H with contents of A MOV
40H,A           ; Save the LS byte of the result in location 40H
MOV A,52H       ; Load the contents of 52H into A
ADDC A,56H      ; Add the contents of 56H and CY flag with A
MOV 41H,A       ; Save the second byte of the result in 41H
MOV A,#00       ; Load 00H into A
ADDC A,#00 ; Add the immediate data 00H and CY to A
MOV 42H,A       ; Save the MS byte of the result in location 42H
END
```

5. **Write a program to store data FFH into RAM memory locations 50H to 58H using indirect addressing mode.**

```
        ORG  0000H            ; Set program counter 0000H
        MOV  A, #0FFH         ; Load FFH into A
        MOV  RO, #50H         ; Load pointer, R0-50H
        MOV  R5, #08H         ; Load counter, R5-08H
Start:MOV  @RO, A             ; Copy contents of A to RAM pointed by R0
        INC  RO               ; Increment pointer
         DJNZ R5, start ; Repeat until R5 is zero
         END
```

6. **Write a program to add two Binary Coded Decimal (BCD) numbers stored at locations 60H and 61H and store the result in BCD at memory locations 52H and 53H. Assume that the least significant byte of the result is stored in low address.**

```
ORG 0000H     ; Set program counter 00004
MOV A,60H     ; Load the contents of memory location 6.0.H into A
ADD A,61H     ; Add the contents of memory location 61H with contents of A
DA A          ; Decimal adjustment of the sum in A
MOV 52H, A    ; Save the least significant byte of the result in location 52H
MOV A,#00     ; Load 00H into .A
ADDC A,#00H ; Add the immediate data and the contents of carry flag to A
MOV 53H,A  ; Save the most significant byte of the result in location 53:,
END
```

7. **Write a program to clear 10 RAM locations starting at RAM address 1000H.**

```
         ORG 0000H              ;Set program counter 0000H
         MOV DPTR, #1000H   ;Copy address 1000H to DPTR
         CLR A          ;Clear A
         MOV R6, #0AH           ;Load 0AH to R6
again: MOVX @DPTR,A             ;Clear RAM location pointed by DPTR

         INC DPTR               ;Increment DPTR
         DJNZ R6, again         ;Loop until counter R6=0
         END
```

8. **Write a program to compute 1 + 2 + 3 + N (say N=15) and save the sum at70H**

```
          ORG  0000 H                  ;  Set program counter 0000H
         N EQU 15
         MOV  R0,#00                 ; Clear R0
         CLR  A                              ; Clear A
again:  INC  R0                      ; Increment R0
         ADD  A, R0                  ; Add the contents of R0 with A
         CJNE  R0,#N,again  ; Loop  until  counter,  R0,  N
         MOV  70H,A                  ;  Save  the  result  in  location  70H  END
```

9. **Write a program to multiply two 8 bit numbers stored at locations 70H and 71H and store theresult at memory locations 52H and 53H. Assume that the least significant byte of the result is stored in low address.**

ORG 0000H ; Set program counter 00 OH
MOV A, 70H ; Load the contents of memory location 70h into A
MOV B, 71H ; Load the contents of memory location 71H into B
MUL AB ; Perform multiplication
MOV 52H,A ; Save the least significant byte of the result in location 52H MOV 53H,B ; Save the most significant byte of the result in location 53
END

10. **Ten 8 bit numbers are stored in internal data memory from location 5oH. Write a program to increment the data.**

Assume that ten 8 bit numbers are stored in internal data memory from location 50H, henceR0 or R1 must be used as a pointer.

The program is as follows.

```
OPT 0000H
MOV
R0,#50HMOV
R3,#0AH
Loopl: INC
@R0INC RO
DJNZ R3, loopl
ENDEND
```

11. **Write a program to find the average of five 8 bit numbers. Store the result in H. (Assume that after adding five 8 bit numbers, the result is 8 bit only).**

```
ORG 0000H
MOV 40H,#05H
MOV 41H,#55H
MOV 42H,#06H
MOV
43H,#1AHMOV
44H,#09HMOV
R0,#40H MOV
R5,#05H
MOV
B,R5CLR
A
Loop: ADD
A,@ROINC RO
DJNZ R5,Loop
DIV AB
MOV 55H,A                END
```

12. Write a program to find the cube of an 8 bit number program is as follows

```
ORG 0000H
MOV
R1,#NMOV
A,R1 MOV
B,R1
MUL AB                  //SQUARE IS
COMPUTEDMOV R2, B
MOV B, R1
MUL AB
MOV 50,A
MOV 51,B
MOV A,R2
MOV B, R1
MUL AB
ADD A, 51H
MOV 51H, A
MOV 52H, B
MOV A, #
00HADDC
A, 52H
MOV 52H, A              //CUBE IS STORED IN
52H,51H,50HEND
```

13. Write a program to exchange the lower nibble of data present in external memory 6000H and6001H

```
ORG 0000H                ; Set program counter 00h
MOV DPTR, #6000 H ; Copy address 6000 H to DPTR
MOVX A, @DPTR           ; Copy contents of 60008 to A
MOV R0, #45H            ; Load pointer, R0=45H
MOV @RO, A              ;Copy cont of A to RAM pointed by 80
INC DPL                 ;Increment pointer
MOVX A, @DPTR           ;Copy contents of 60018 to A
XCHD A, @R0             ; Exchange lower nibble of A with RAM pointed by RO
MOVX @DPTR, A           ;Copy contents of A to 60018
DEC DPL                 ; Decrement pointer
MOV A, @R0              ; Copy cont of  RAM pointed by R0 to A
MOVX @DPTR, A           ; Copy cont of A to RAM pointed by DPTR
END
```

14. Write a program to count the number of and o's of 8 bit data stored in location 6000H.

```
ORG 00008               ; Set program counter 00008
MOV DPTR, #6000h     ; Copy address 6000H to DPTR
MOVX A, @DPTR           ; Copy num be r to A
MOV R0,#08              ; Copy 08 in RO
MOV R2,#00              ; Copy 00 in R2
MOV R3,#00              ; Copy 00 in R3
CLR C                   ; Clear carry flag
BACK:  RLC A         ; Rotate A through carry flag
```

```
JC NEXT                    ; If CF = 1, branch to next
INC R2                     ; If CF = 0, increment R2 AJMP NEXT2
NEXT: INC R3               ; If CF = 1, increment R3
NEXT2: DJNZ RO,BACK        ; Repeat until RO is zero
END
```

15. Write a program to shift a 24 bit number stored at 57H-55H to the left logically four places. Assume that the least significant byte of data is stored in lower address.

```
        ORG 0000H       ; Set program counter 0000h
        MOV R1,#04      ; Set up loop count to 4
again:  MOV A,55H       ; Place the least significant byte of data in A
        CLR C           ; Clear tne carry flag
        RLC A           ; Rotate contents of A (55h) left through carry
        MOV 55H,A
        MOV A,56H
        RLC A           ; Rotate contents of A (56H) left through carry
        MOV 56H,A
        MOV A,57H
        RLC A           ; Rotate contents of A (57H) left through carry
        MOV 57H,A
        DJNZ R1,again ; Repeat until R1 is zero
        END
```

16. Two 8 bit numbers are stored in location 1000h and 1001h of external data memory. Write a program to find the GCD of the numbers and store the result in 2000h.

ALGORITHM

- *Step 1 :Initialize external data memory with data and DPTR with address*
- *Step 2 :Load A and TEMP with the operands*
- *Step 3 :Are the two operands equal? If yes, go to step 9*
- *Step 4 :Is (A) greater than (TEMP) ? If yes, go to step 6*
- *Step 5 :Exchange (A) with (TEMP) such that A contains the bigger number*
- *Step 6 :Perform division operation (contents of A with contents of TEMP)*
- *Step 7 :If the remainder is zero, go to step 9*
- *Step 8 :Move the remainder into A and go to step 4*
- *Step 9 :Save the contents 'of TEMP in memory and terminate the program*

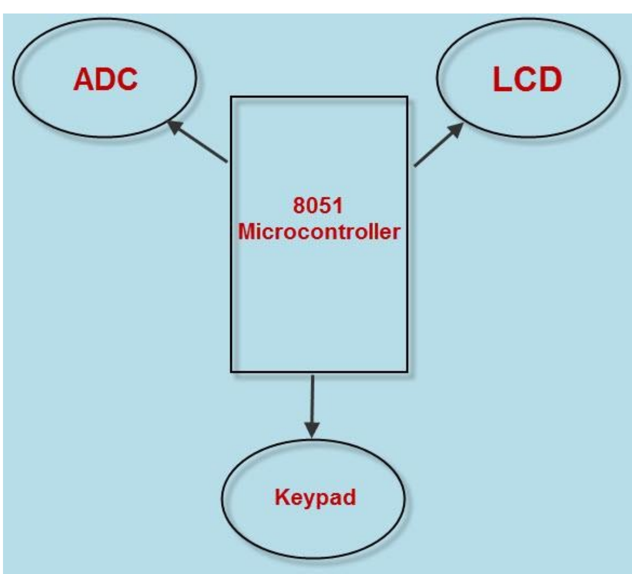
```
        ORG 0000H             ; Set program counter 0000H
        TEMP EQU 70H
        TEMPI EQU 71H
        MOV DPTR, #1000H      ; Copy address 100011 to DPTR
        MOVX A, @DPTR         ; Copy First number to A
        MOV TEMP, A           ; Copy First number to temp INC DPTR
        MOVX A, @DPTR         ; Copy Second number to A
LOOPS: CJNE A, TEMP, LOOP1   ; (A) /= (TEMP) branch to LOOP1
        AJMP LOOP2            ; (A) = (TEMP) branch to L00P2
LOOP1:  JNC LOOP3             ; (A) > (TEMP) branch to LOOP3
        NOV TEMPI, A          ; (A) < (TEMP) exchange (A) with (TEMP)
        MOV A, TEMP
        MOV TEMP, TEMPI
LOOP3:  MOV B, TEMP
        DIV AB                ; Divide (A) by (TEMP)
        MOV A, B              ; Move remainder to A
        CJNE A,#00, LOOPS     ; (A)/=00 branch to LOOPS
LOOP2:  MOV A, TEMP
        MOV DPTR, #2000H
        MOVX @DPTR, A         ; Store the result in 2000H
        END
```

UNIT V- Interfacing Microcontroller

Interfacing Microcontroller - Programming 8051 Timers - Serial Port Programming - Interrupts Programming – LCD & Keyboard Interfacing - ADC, DAC & Sensor Interfacing - External Memory Interface- Stepper Motor and Waveform generation - Comparison of Microprocessor, Microcontroller, PIC and ARM processors

Interfacing Microcontroller:

Every electrical and electronics project designed to develop electronic gadgets that are frequently used in our day-to-day life utilizes microcontrollers with appropriate interfacing devices. There are different types of applications that are designed using microcontroller-based projects. In maximum number of applications, the microcontroller is connected with some external devices called as interfacing devices for performing some specific tasks. For example, consider security system with a user changeable password project, in which an interfacing device, keypad is interfaced with microcontroller to enter the password.



In Intel 8051, there are two 16-bit timer registers. These registers are known as Timer0 andTimer1. The timer registers can be used in two modes. These modes areTimer mode and the Counter mode. The only difference between these two modes is the source for incrementing the timer registers.

Timer Mode

In the timer mode, the internal machine cycles are counted. So this register is incremented in each machine cycle. So when the clock frequency is 12MHz, then the timer register is incremented in each millisecond. In this mode it ignores the external timer input pin.

Counter Mode

In the counter mode, the external events are counted. In this mode, the timer register is incremented for each 1 to 0 transition of the external input pin. This type of transitions is treated as events. The external input pins are sampled once in each machine cycle, and to determine the 1or 0 transitions, another machine cycle will be needed. So in this mode, at least two machine cycles are needed. When

the frequency is12MHz, then the maximum count frequency will be 12MHz/24 = 500KHz. So for event counting the time duration is 2 μs.

There are four different modes of the Timer or Counter. The Mode 0 to Mode 2 are for both of the Timer/Counter. Mode 3 has a different meaning for each timer register. There is a register called TMOD. This register can be programmed to configure these timers or counters.

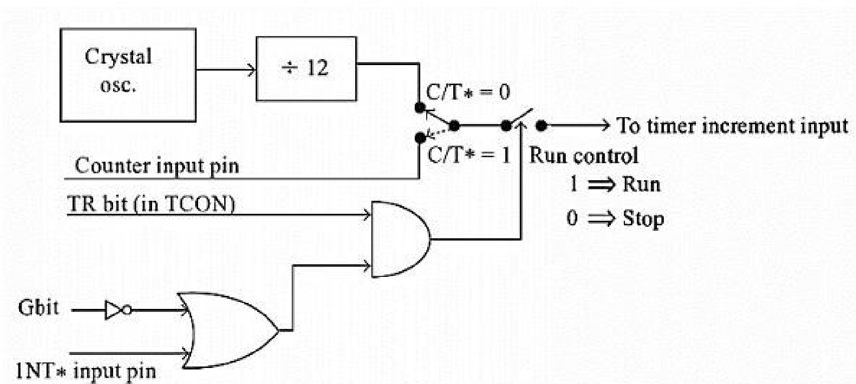
The Serial port is used for serial communication in mode 1 and 3. Timer1 is used for generating the baud rate. So only Timer0 is available for timer or counter operations.

TMOD Register

TMOD(Timer Mode) is an SFR. The address of this register is 89H. This is not bit-addressable.

Timer	Timer1 Mode				Timer0 Mode			
Bit Details	Gate (G)	C/T	M1	M0	Gate (G)	C/T	M1	M0

Now, let us see the circuit that controls the running of the timers.



In the following table, we will see the bit details and their different operations for high or low value.

Bit Details	High Value(1)	Low Value(0)
C/T	Configure for the Counter operations	Configure for the Timer operations
Gate (G)	Timer0 or Timer1 will be in RunMode when TRX bit of TCON register is high.	Timer0 or Timer1 will be in RunMode when TRX bit of TCON register is high and INT0 or INT1 is high.

Bit Details	00	01	10	11
M1 M0	This is for Mode 0. (8-bit timer/counter, with 5-bit pre-scaler)	This is Mode 1. (16-bit timer/counter)	This is Mode 3 (8-bit auto reload-timer/counter)	This is Mode 3 (The function depends on Timer0 or Timer1)

The Gate bit will be high when the timer or counter is in mode 0 to 2.

Examples

To configure the Timer0 as 16-bit event counter and Timer1 as 8-bit auto reload counter, we can use the bit pattern 0 0 1 0 0 1 0 1. It is equivalent to 25H. If we want to program the TMOD register with this bit pattern, we can use this instruction:

```
MOVTMOD, #25H
```

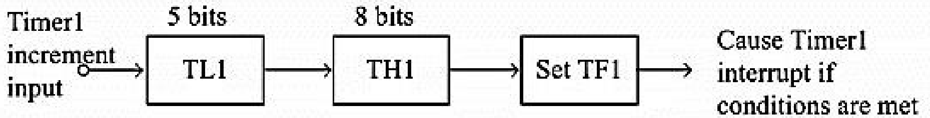
The above instruction is executed, then the timer/counter will be controlled by the software. To configure the system as hardware controlled mode, then the gate bits will be 1. So the bit patterns will be 1 0 1 0 1 1 0 1 = ADH

we can use this instruction:

```
MOVTMOD, #0ADH
```

Mode 0 of Timer/Counter

The Mode 0 operation is the 8-bit timer or counter with a 5-bit pre-scaler. So it is a 13-bit timer/counter. It uses 5 bits of TL0 or TL1 and all of the 8-bits of TH0 or TH1.



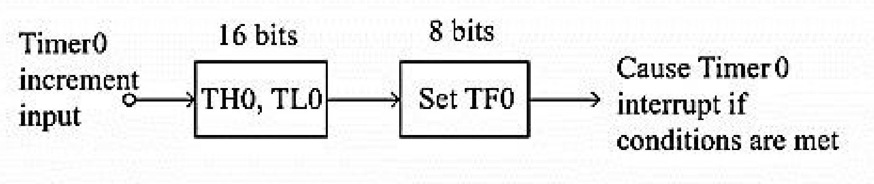
In this example the Timer1is selected, in this case, every 32 (25)event for counter operations or 32 machine cycles for timer operation, the TH1 register will be incremented by 1. When the TH1overflows from FFH to 00H, then the TF1 of TCON register will be high, and it stops the timer/counter. So for an example, we can say that if the TH1 is holding F0H, and it is in timer mode, then TF1will be high after 10H * 32 = 512 machine cycles.

```
MOVTMOD, #00H
MOVTH1, #0F0H
MOVIE, #88H
SETB TR1
```

In the above program, the Timer1 is configured as timer mode 0. In this case Gate = 0. Then the TH1 will be loaded with F0H, then enable the Timer1 interrupt. At last set the TR1 of TCON register, and start the timer.

Mode 1 of Timer/Counter

The Mode 1 operation is the 16-bit timer or counter. In the following diagram, we are using Mode 1 for Timer0.



In this case every event for counter operations or machine cycles for timer operation, the TH0– TL0 register-pair will be incremented by 1. When the register pair overflows from FFFFH to 0000H, then the TF0 of TCON register will be high, and it stops the timer/counter. So for an example, we can say that if the TH0 – TL0 register pair is holding FFF0H, and it is in timer mode, then TF0 will be high after 10H = 16 machine cycles. When the clock frequency is 12MHz, then the following instructions generate an interrupt 16 μs after Timer0 starts running.

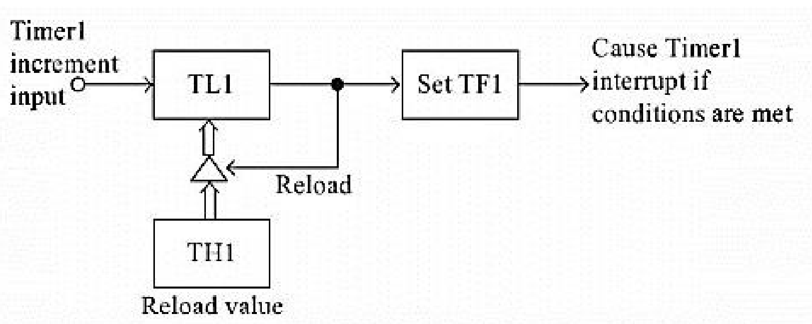
```
MOVTMOD, #01H
```

```
MOVTL0, #0F0H
MOVTH0, #0FFH
MOVIE, #82H
SETB TR0
```

In the above program, the Timer0 is configured as timer mode 1. In this case Gate = 0. Then the TL0 will be loaded with F0H and TH0 is loaded with FFH, then enable the Timer0 interrupt. At last set the TR0 of TCON register, and start the timer.

Mode 2 ofTimer/Counter

The Mode 2 operation is the 8-bit auto reload timer or counter. In the following diagram, we are using Mode 2 for Timer1.



In this case every event for counter operations or machine cycles for timer operation, the TL1register will be incremented by 1. When the register pair overflows from FFH to 00H, then the TF1 of TCON register will be high, also theTL1 will be reloaded with the content of TH1 and starts the operation again.

So for an example, we can say that if the TH1 and TL1 register both are holding F0H and it is in timer mode, then TF1 will be high after 10H= 16 machine cycles. When the clock frequency is 12MHz this happens after 16 μs, then the following instructions generate an interrupt once every 16 μs after Timer1 starts running.

```
MOVTMOD, #20H
MOVTL1, #0F0H
MOVTH1, #0F0H
MOVIE, #88H
SETB TR1
```

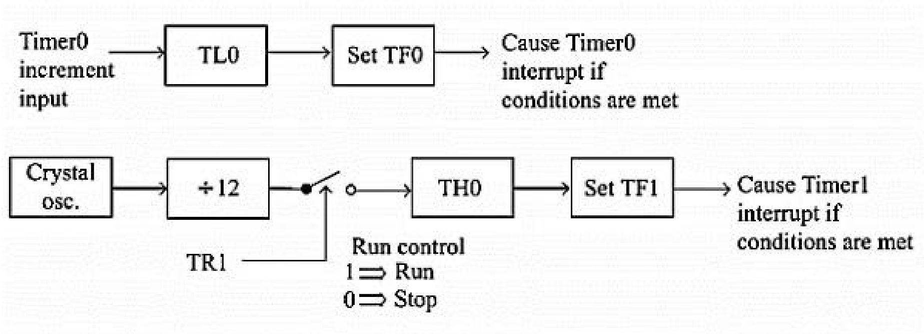
In the above program, the Timer1 is configured as timer mode 2. In this case Gate = 0. Then the TL1 and TH1 are loaded with F0H. then enable the Timer1 interrupt. At last set the TR1 of TCON register, and start the timer.

Timer1 in mode 2 generates the desired baud rate when the serial port is working on Mode 1 or 3.

Mode 3 of Timer/Counter

Mode 3 is different for Timer0 and Timer1. When the Timer0 is working in mode 3, the TL0 will be used as an 8-bit timer/counter. It will be controlled by the standard Timer0 control bits, T0 and INT0 inputs. The TH0 is used as an 8-bit timer but not the counter. This is controlled by Timer1

Control bit TR1. When the TH0 overflows from FFH to 00H, then TF1 is set to 1. In the following diagram, we can Timer0 in Mode 3.



When the Timer1 is working in Mode 3, it simply holds the count but does not run. When Timer0 is in mode 3, the Timer1 is configured in one of the mode 0, 1 and 2. In this case, the Timer1 cannot interrupt the microcontroller. When the TF1 is used by TH0 timer, the Timer1 is used as Baud Rate Generator.

The meaning of gate bit in Timer0 and Timer1 for mode 3 is as follows

It controls the running of 8-bit timer/counter TL0 as like Mode 0, 1, or 2. The running of TH0 is controlled by TR1 bit only. So the gate bit in this mode for Timer0 has no specific role.

The mode 3 is present for applications requiring an extra 8-bit timer/counter. In Mode 3 of Timer0, the 8051 has three timers. One 8-bit timer by TH0, another8-bit timer/counter by TL0, and one 16-bit timer/counter by Timer1.

If the Timer0 is in mode3, and Timer1 is working on either 0, 1 or 2, then the gun control of the Timer1 is activated when the gate bit is low or INT1 is high. The run control is deactivated when the gate is high and INT1 is low.

8051 INTERRUPTS PROGRAMMING

* There are 6 interrupts in 8051.

1. Reset
2. Interrupt for Timer 0
3. Interrupt for Timer 1
4. External hardware interrupt 0 (INT0)
5. External hardware interrupt 1 (INT1)
6. Serial communication interrupt (RI & TI)

Interrupt Vector Table for 8051

Interrupt	ROM location	Pin No.
Reset	0000H	9
INT0	0003H	P3.2 (12)
INT1	0013H	P3.3 (13)
Timer 0 interrupt (TF0)	000BH	–
Timer 1 interrupt (TF1)	001BH	–
Serial COM interrupt (RI & TI)	0023H	–

Enabling and Disabling an interrupt:

IE register (Interrupt Enable Register) is responsible for enabling and disabling the interrupts.

D7							D0
EA	–	ET2	ES	ET1	EX1	ET0	EX0

EA = 0, disables all interrupts.

EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

ET2 :- Enables or disables timer 2 overflow interrupt (only applicable for 8952)

ES – Enables or disables the serial port interrupt.
ET1 – Enables or disables timer 1 overflow interrupt.
EX1 – Enables or disables External interrupt 1.
ET0 – Enables or disables timer 0 overflow interrupt.
EX0 – Enables or disables External interrupt 0.

Programming External Hardware Interrupts :-

To make INT0, INT1 edge-triggered interrupts, bit of TCON register must be programmed.

D_7							D_0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

IT0, IT1 : Interrupt 0, 1 type control bits.

If 0, low-level triggered interrupt
If 1, edge-triggered interrupt.

IE0, IE1 : Interrupt-in-service flags. (Edge flag)

These bits are set by CPU when an external interrupt edge is detected. Cleared when interrupt is processed.

Interrupt Priority in 8051 :-

Interrupt Priority upon reset :

Highest Priority ↓ Lowest Priority

Interrupt		Flag
External interrupt 0	–	INT0
Timer interrupt 0	–	TF0
External interrupt 1	–	INT1
Timer interrupt 1	–	TF1
Serial Communication	–	RI & TI

6. **Setting Interrupt Priority with the IP register:**

We can alter the sequence of the default priority table by means of IP register.

IP (Interrupt Priority Register)

D_7 ... D_0

-	-	PT2	PS	PT1	PX1	PT0	PX0

When any of the above bit is set high, the corresponding interrupt will be given the highest priority.

When two or more bits are set to high, interrupts are serviced according to the sequence of the table.

Eg: a) Program the IP register to assign the highest priority to INT1.

Solution:

MOV IP, # 00000100B (or)

MOV IP, # 04H (or)

SETB IP.2

Eg. b) Show the order of interrupt priority if INT0, INT1 & TF0 are activated at the same time.

INT1

INT0

TF0

Eg. c) Show the instruction to enable the serial interrupt, timer 0 interrupt, & ext. h/w int 1.

MOV IE, # 10010110B (or)

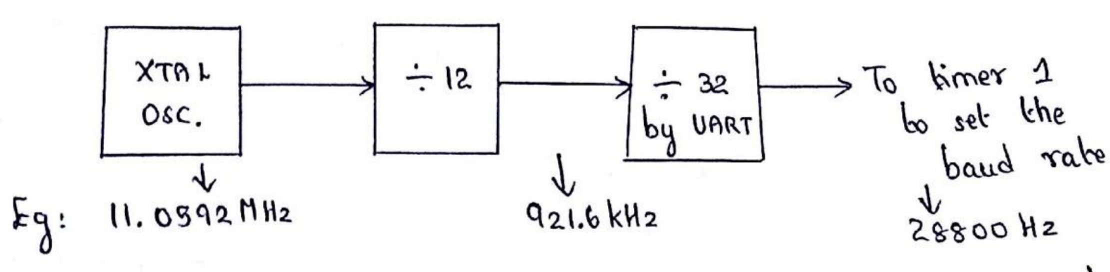
MOV IE, # 96H

Eg. d) Disable all the interrupts with a single instruction

CLR IE.7

8051 - SERIAL COMMUNICATION PROGRAMMING

* For serial data communication, the byte of data must be converted to serial bits using shift register, then it can be transmitted over a single data line.
* Baud Rate is the number of bits transmitted per second.
* The commonly used PC Baud rates are 1200, 2400, 4800, 9600, 19200.
* Different baud rates can be achieved in 8051 with the help of timer 1 in mode 2.



* To get the desired baud rate, TH1 must be loaded with the following values:

Baud Rate	TH1 (Hex)	TH1 (Decimal)
9600	FD	-3
4800	FA	-6
2400	F4	-12
1200	E8	-24

Serial Data Buffer (SBUF) Register:

* SBUF is an 8-bit register.
* The data to be transmitted serially via TxD line, must be placed in SBUF.
* The serial data received via RxD line, will be placed in SBUF.

Scanned by

To access SBUF:

Eg:

```
MOV SBUF, A
MOV A, SBUF
MOV SBUF, #'D'
```

Serial Port Control Register (SCON):-

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0, SM1 :

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

SM2 : Enables the multiprocessor capability of 8051 (Make it 0)

REN : Receive Enable

TB8 : Transmitted 9th bit

RB8 : Received 9th bit

TI : Transmit Interrupt Flag

After finishing the transfer of 8-bit character, TI flag is raised high.

RI : After receiving a 8-bit character, RI flag is raised high.

Scanned by

Programming the 8051 to transfer data serially:

1. TMOD register is loaded with 20_H, indicating the use of timer 1 in mode 2 to set the baud rate.
2. TH1 is loaded with appropriate value to set the baud rate.
3. SCON register is loaded with 50_H indicating serial mode 1.
4. Timer 1 has to be started. (SETB TR1)
5. TI is cleared. (CLR TI)
6. The character byte has to written into SBUF.
7. TI flag bit is monitored (JNB TI, label)
8. To transfer the next character, repeat from step 5.

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

```
        MOV   TMOD, #20H
        MOV   TH1, #FAH
        MOV   SCON, #50H
        SETB  TR1
AGAIN:  MOV   SBUF, #"A"
HERE:   JNB   TI, HERE
        CLR   TI
        SJMP  AGAIN
```

Programming the 8051 to receive data serially:

1. TMOD register is loaded with 20_H, indicating the use of timer 1 in mode 2 to set the baud rate.
2. TH1 is loaded with appropriate value to set the baud rate.

Scanned by

3. SCON register is loaded with 50_H indicating serial mode 1.

4. Timer 1 has to be started. (SETB TR1)

5. RI is cleared. (CLR RI)

6. Monitor RI flag bit continuously.

7. When RI flag is raised, SBUF has the received byte.

8. To receive the next character, go to step 5.

Eg: Program 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.

```
        MOV   TMOD, #20H
        MOV   TH1, #FAH
        MOV   SCON, #50H
        SETB  TR1
HERE:   JNB   RI, HERE
        MOV   A, SBUF
        MOV   P1, A
        CLR   RI
        SJMP  HERE
```

Doubling the baud rate in 8051:

Baud rate can be doubled by setting the SMOD bit of PCON register.

PCON (Power Control Register)

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SMOD	-	-	-	GF1	GF0	PD	IDL

If SMOD = 1, baud rate will be doubled.

Eg:

TH1	SMOD = 0	SMOD = 1
FD_H	9600	19,200
FA_H	4800	9,600

Scanned by

STEPPER MOTOR INTERFACING

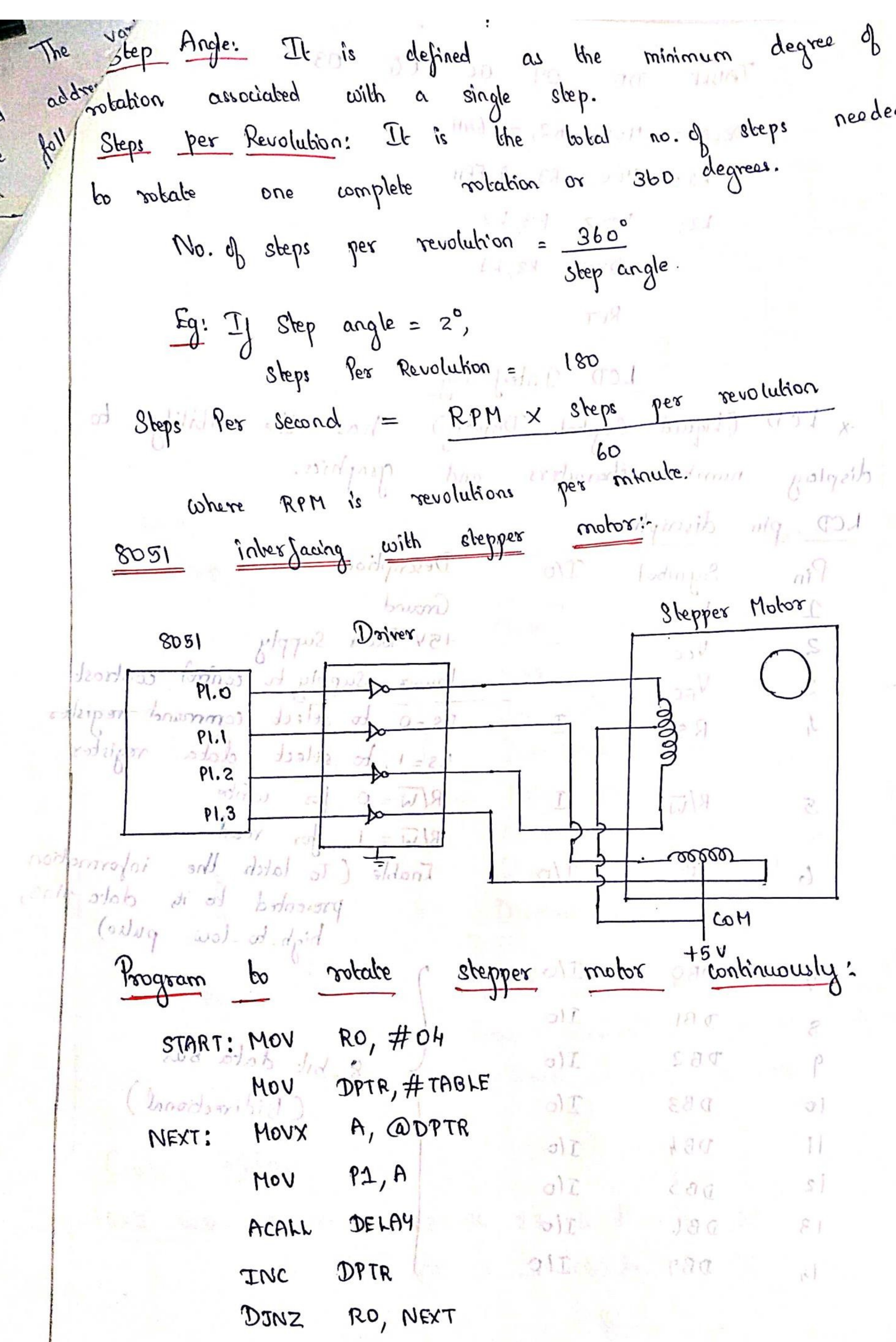
* Stepper Motor is a widely used device that translates electrical pulses into mechanical movement.
* Stepper Motor is used in applications such as disk drives, dot matrix printers, robotics etc. for position control.
* stepper motor has a permanent magnet rotor surrounded by a stator.

* Most motors have 4 stator windings referred as a 4-phase stepper motor.
* When a sequence of power is applied to stator, rotor will rotate.

Normal 4-step sequence:

Step #	Winding A	B	C	D
1	1	0	0	1
2	1	1	0	0
3	0	1	1	0
4	0	0	1	1

Clockwise (↓ steps 1 to 4) Counter-clockwise (↑ steps 4 to 1)



Step Angle: It is defined as the minimum degree of rotation associated with a single step.

Steps per Revolution: It is the total no. of steps needed to rotate one complete rotation or 360 degrees.

$$\text{No. of steps per revolution} = \frac{360^\circ}{\text{step angle}}$$

Eg: If Step angle = 2°,

Steps Per Revolution = 180

$$\text{Steps Per Second} = \frac{\text{RPM} \times \text{steps per revolution}}{60}$$

where RPM is revolutions per minute.

8051 interfacing with stepper motor:

Program to rotate stepper motor continuously:

```
START: MOV   R0, #04
       MOV   DPTR, #TABLE
NEXT:  MOVX  A, @DPTR
       MOV   P1, A
       ACALL DELAY
       INC   DPTR
       DJNZ  R0, NEXT
       SJMP  START
```

```
TABLE: DB    09    0C    06    03

DELAY:  MOV   R2, # 64H
   L1:  MOV   R3, # FFH
   L2:  DJNZ  R3, L2
        DJNZ  R2, L1
        RET
```

LCD Interfacing

* LCD (Liquid Crystal Display) has the ability to display numbers, characters and graphics.

LCD pin discriptions:

Pin	Symbol	I/O	Description
1	V_{SS}	–	Ground
2	V_{CC}	–	+5V Power Supply
3	V_{EE}	–	Power Supply to control contrast
4	RS	I	RS = 0, to select command register RS = 1, to select data register
5	$R/\overline{W}$	I	$R/\overline{W} = 0$ for write $R/\overline{W} = 1$ for read
6	E	I/O	Enable (To latch the information presented to its data pins, high-to-low pulse)
7	DB0	I/O	8-bit data bus (Bidirectional)
8	DB1	I/O	
9	DB2	I/O	
10	DB3	I/O	
11	DB4	I/O	
12	DB5	I/O	
13	DB6	I/O	
14	DB7	I/O	

PROGRAMMING 8051 TIMERS

* The 8051 has 2 timers/counters. (Timer 0 & Timer 1).
* They can be used either as timers to generate a time delay or as counters to count events happening outside the microcontroller.

Basic Registers of the timers:

The 16-bit register of Timer 0 (T0) and Timer 1 (T1) can be accessed as two separate 8-bit registers : Low byte (TL) and High byte (TH).

Timer 0 register:

TH0								TL0							
D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Timer 1 register:

TH1								TL1							
D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

TMOD (Timer Mode) Register:

MSB ... LSB

GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
Timer 1				Timer 0			

M1, M0 : Mode bits :

M_1	M_0	Mode	Operating Mode
0	0	0	13-bit timer
0	1	1	16-bit timer
1	0	2	8-bit timer

C/$\overline{T}$: If 0, used as a timer. (Clock source is the crystal frequency of 8051)

If 1, used as a counter. (Clock source is the pulse outside the 8051). (Pin P3.4 and Pin P3.5)

GATE: If GATE = 0, software instructions start and stop the timers. (SETB and CLR).

If GATE = 1, hardware means start and stop the timers. (Pin P3.2 and P3.3)

TCON (Timer Control Register):-

MSB | | | | | | | LSB

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1 : Timer 1 Overflow flag : This is set when Timer 1 overflows.

TF0 : Timer 0 Overflow flag: This flag is set when Timer 0 overflows.

TR1 : Timer 1 Run : This flag has to be set to start the timer1. (SETB TR1)

TR0 : Timer 0 Run : This flag has to be set to start the timer 0. (SETB TR0)

IE1 : External interrupt 1 edge flag.

IE0 : External interrupt 0 edge flag.

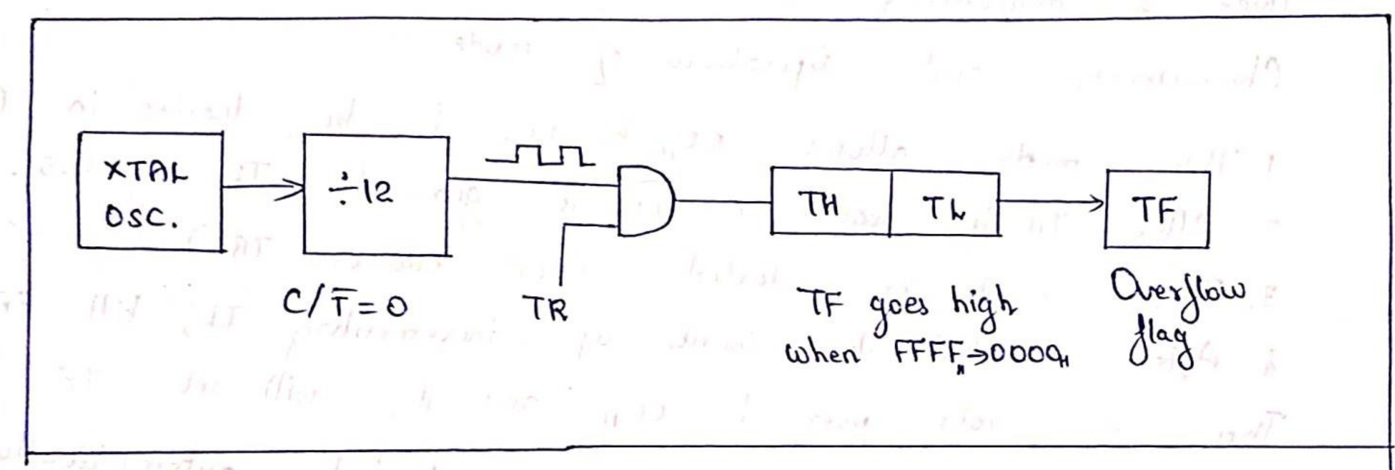
IT1 : Interrupt 1 type control bit.

IT0 : Interrupt 0 type control bit

Mode 1 Programming :-

Characteristics and Operations of mode 1: (16-bit timer mode)

1. This mode allows 0000_H to $FFFF_H$ to be loaded in TL & TH.
2. After loading TL & TH, timer must be started by SETB TR0 or TR1.
3. After it is started, it starts to count-up. It counts up till $FFFF_H$. Then it rolls over to 0000_H and it will set the flag bit, TF, high.
4. In order to repeat the process, TH & TL must be reloaded with the original value and TF must be reset to 0.



Eg: Write an 8051 ALP to create a square wave of 50% duty cycle on the P1.5 bit. Use Timer 0 to generate the time delay.

```
        MOV   TMOD, #01H    ; Select Timer 0 in mode-1
HERE:   MOV   TL0, #F2H     ; Load count in TL & TH
        MOV   TH0, #FFH
        CPL   P1.5          ; Complement P1.5
        ACALL DELAY
        SJMP  HERE          ; Load TL & TH again
```

Delay using timer 0:

```
DELAY :   SETB   TR0          ; Start the Timer 0
AGAIN :   JNB    TF0, AGAIN   ; Monitor TF until it rolls
          CLR    TR0          ; Stop Timer 0
          CLR    TF0          ; Clear TF
          RET
```

Mode 0 Programming: (13_bit timer)

Mode 0 is exactly like Mode 1, except that it is a 13_bit timer. This mode allows 0000_H to $1FFF_H$ to be loaded in TL & TH.

Mode 2 Programming: (8_bit Timer)

Characteristics and Operations of mode 2:

1. This mode allows 00_H to FF_H to be loaded in TH.
2. After TH is loaded, a copy is given to TL by 8051.
3. Timer must be started. (SETB TR0 or TR1)
4. After started, it counts up, incrementing TL, till FF_H. Then it rolls over to 00_H and it will set TF high.
5. When TF goes high, TL is reloaded automatically with the value kept in TH. To repeat the process, clear TF.

XTAL OSC → ÷12

C/T̄ = 0

TR0

TL0 → TF0

Overflow flag.

Reload

TH0

TF0 goes high when FF → 0

The main application of Mode 2 is setting the baud rate in serial communication.

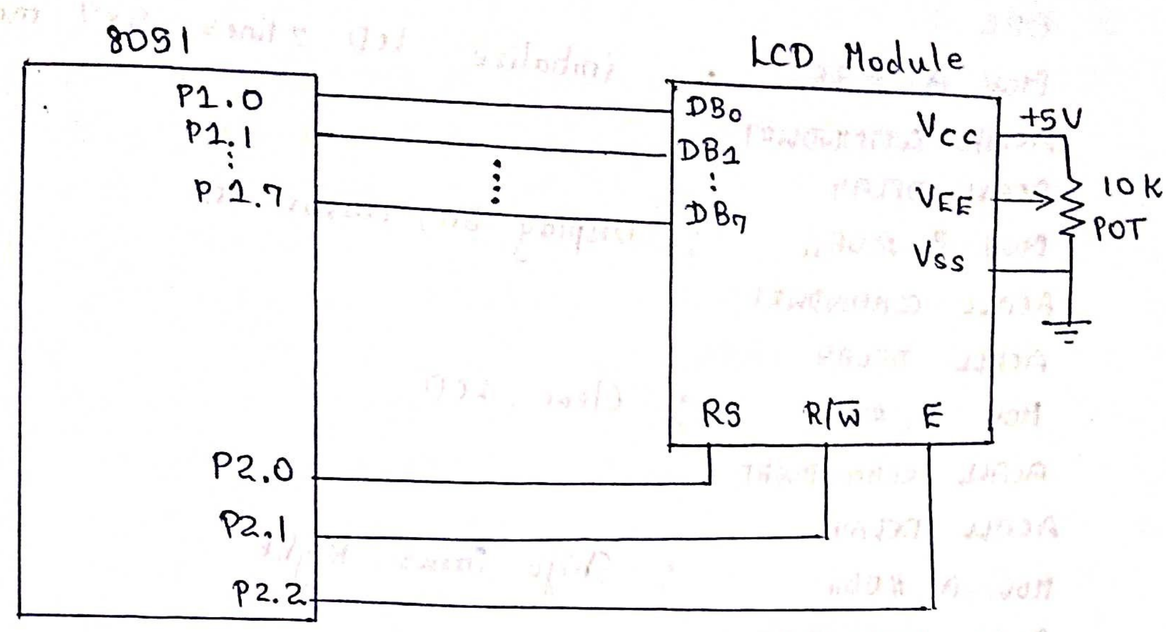
LCD Interfacing

* LCD (Liquid Crystal Display) has the ability to display numbers, characters and graphics.

LCD pin discriptions:

Pin	Symbol	I/O	Description
1	V_{SS}	–	Ground
2	V_{CC}	–	+5V Power Supply
3	V_{EE}	–	Power Supply to control contrast
4	RS	I	RS = 0, to select command register RS = 1, to select data register
5	$R/\overline{W}$	I	$R/\overline{W} = 0$ for write $R/\overline{W} = 1$ for read
6	E	I/O	Enable (To latch the information presented to its data pins, high-to-low pulse)
7	DB0	I/O	8-bit data bus (Bidirectional)
8	DB1	I/O	
9	DB2	I/O	
10	DB3	I/O	
11	DB4	I/O	
12	DB5	I/O	
13	DB6	I/O	
14	DB7	I/O	

LCD connection with 8051:



LCD Command Codes:

CODE (HEX)		Command to LCD Instruction Register
1	–	Clear display screen
2	–	Return home
4	–	Decrement cursor
6	–	Increment Cursor
5	–	Shift display right
7	–	Shift display left
8	–	Display off, cursor off
9	–	Display off, cursor on
	⋮	
80	–	Force cursor to beginning of 1st line
C0	–	Force cursor to beginning of 2nd line
38	–	2 lines & 5x7 matrix

Cursor address:

16x2 LCD: 80 81 82 83 84 85 86 through 8F

C0 C1 C2 C3 C4 C5 C6 through CF

Program to display 'NO' using LCD :-

```
        ORG
        MOV A, #38H         ; initialize LCD 2 lines, 5x7 matrix
        ACALL COMMNDWRT
        ACALL DELAY
        MOV A, #0EH         ; Display on, cursor on
        ACALL COMMNDWRT
        ACALL DELAY
        MOV A, #01H         ; Clear LCD
        ACALL COMMNDWRT
        ACALL DELAY
        MOV A, #06H         ; Shift Cursor Right
        ACALL COMMNDWRT
        ACALL DELAY
        MOV A, #84H         ; Cursor at line 1, position 4.
        ACALL COMMNDWRT
        ACALL DELAY
        MOV A, #'N'         ; Display letter 'N'
        ACALL DATAWRT
        ACALL DELAY
        MOV A, #'O'         ; Display letter 'O'
        ACALL DATAWRT
        ACALL DELAY
AGAIN:  SJMP AGAIN

COMMNDWRT:  MOV P1, A
            CLR P2.0   -> RS = 0
            CLR P2.1   -> R/W̄ = 0
            SETB P2.2  -> E = 1  } High-to-Low
            CLR P2.2   -> E = 0  }
            RET
DATAWRT:    MOV P1, A
            SETB P2.0  -> RS = 1
            CLR P2.1   -> R/W̄ = 0
            SETB P2.2
            CLR P2.2
            RET
DELAY:      MOV R3, #50H
HERE2:      MOV R4, #255D
HERE:       DJNZ R4, HERE
```

Interfacing DAC (Digital to Analog Converter)

DAC is a device used to convert digital pulses to analog signals.

8051 — P1.0, P1.1, P1.2, P1.3, P1.4, P1.5, P1.6, P1.7 → DAC 808 (V_{cc} = +5V; $V_{ref}(+)$ via 1.5k, with 1k to +5V; $V_{ref}(-)$ via 2.5k to ground; I_{out} → V_{out} = 0 - 10V, R = 5K POT)

Program to generate a stair-step ramp:

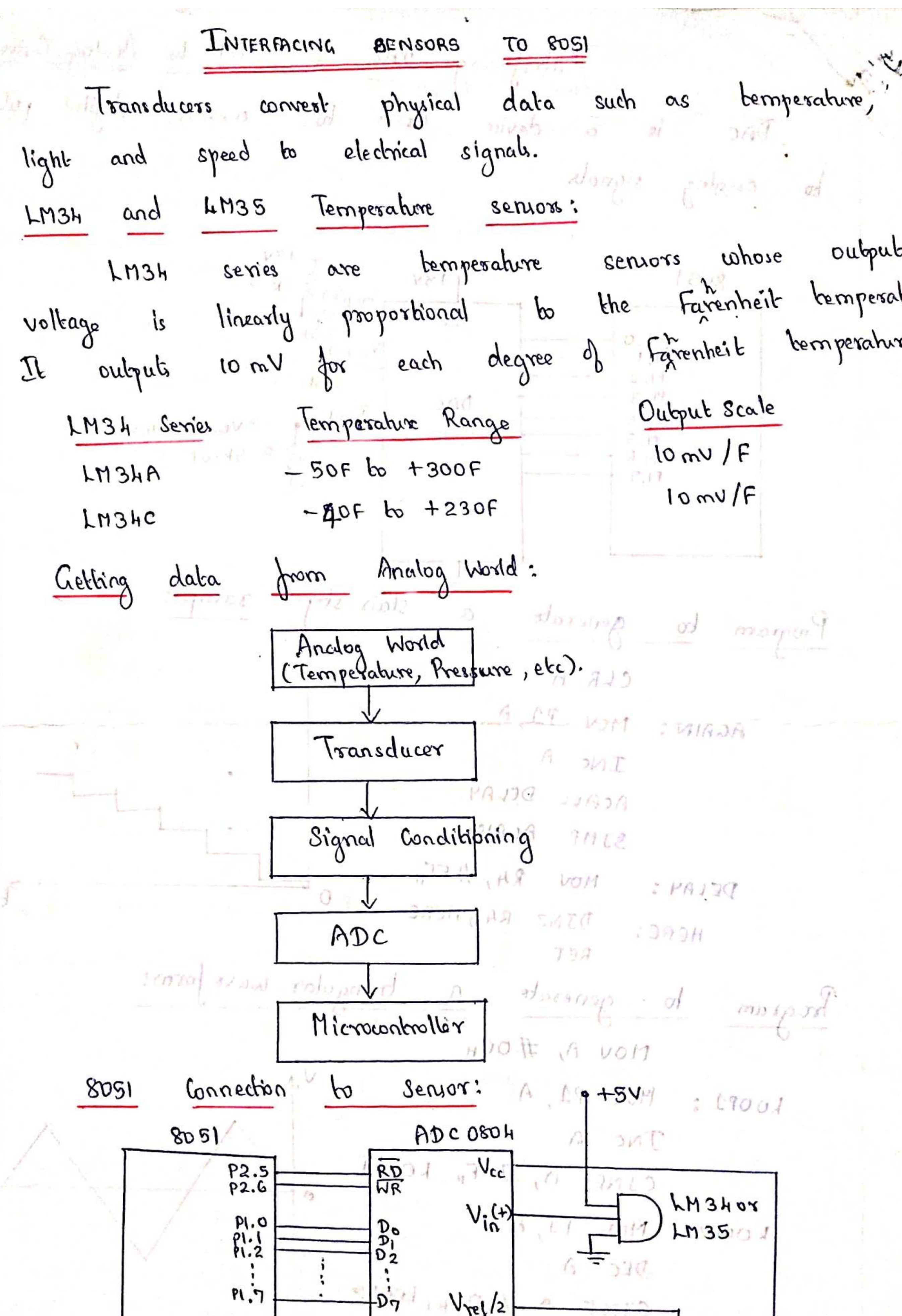
```
        CLR A
AGAIN:  MOV P1, A
        INC A
        ACALL DELAY
        SJMP AGAIN
DELAY:  MOV R4, #FFH
HERE:   DJNZ R4, HERE
        RET
```

(Graph: V vs t, stair-step ramp starting at 0)

Program to generate a triangular-wave form:

```
        MOV A, #00H
LOOP1:  MOV P1, A
        INC A
        CJNE A, #FFH, LOOP1
LOOP2:  MOV P1, A
        DEC A
        CJNE A, #00H, LOOP2
        SJMP LOOP1
```

(Graph: V vs t, triangular waveform)



INTERFACING SENSORS TO 8051

Transducers convert physical data such as temperature, light and speed to electrical signals.

LM34 and LM35 Temperature sensors:

LM34 series are temperature sensors whose output voltage is linearly proportional to the Farenheit temperature. It outputs 10 mV for each degree of Farenheit temperature.

LM34 Series	Temperature Range	Output Scale
LM34A	−50F to +300F	10 mV / F
LM34C	−40F to +230F	10 mV/F

Getting data from Analog World:

8051 Connection to Sensor:

Comparison of Microprocessor, Microcontroller

What is a Microprocessor?

A microprocessor is just a CPU combined with one or multiple Integrated Circuits (ICs). It has not any ROM, RAM, and other instruments. The main operation center of a microprocessor contains registers, ALU, and a control unit. Microprocessors are categorized based on the data size in which ALU applies.

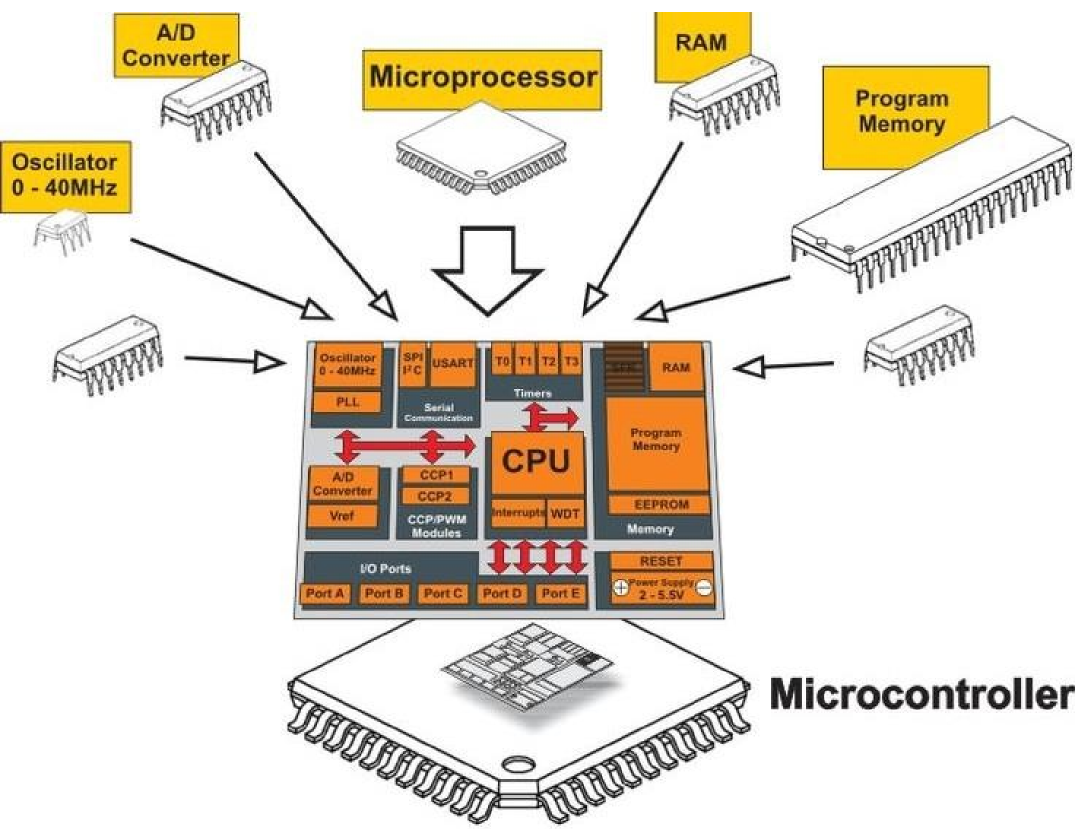
The performance of a microprocessor is based on additional circuits of peripherals to run. Microprocessors are not designed for multiple tasks but they are used where tasks are tricky and complex such as games running, improvement of software, and other programs that need high capacity for memory and where input and output are mixed. It may be introduced as the heart of a computer circuit. Some common examples of microprocessors are I3, I5, and Pentium.

What is a Microcontroller?

A microcontroller is like a simple computer designed on a single IC. It includes CU, ALU, ROM, timer, processor core, RAM, I/O connectors, register, and counters modeled to run several

tasks. Microcontrollers are normally employed in applications and projects that need direct monitoring by users. Because it contains all the parts required in its simple chip, it does not require any additional sections to apply its task.

As a result, microcontrollers are commonly used in fixed systems and the main companies of manufacturing microcontrollers are using them in the embedded applications. The microcontroller can be introduced as the center of an embedded circuit. Some common examples of the popular and economic systems are AVR and 8051 series form.



Microprocessor	Micro Controller
Read-Only Memory (ROM) · Read-Write Memory · Microprocessor · System Bus · Serial Interface · Timer · I/O Port	Microcontroller · Read-Only Memory · Read-Write Memory · Timer · I/O Port · Serial Interface
Microprocessor is heart of Computer system.	Micro Controller is a heart of embedded system.
It is just a processor. Memory and I/O components have to be connected externally	Micro controller has external processor along with internal memory and i/O components
Since memory and I/O has to be connected externally, the circuit becomes large.	Since memory and I/O are present internally, the circuit is small.
Cannot be used in compact systems and hence inefficient	Can be used in compact systems and hence it is an efficient technique
Cost of the entire system increases	Cost of the entire system is low
Due to external components, the entire power consumption is high. Hence it is not suitable to used with devices running on stored power like batteries.	Since external components are low, total power consumption is less and can be used with devices running on stored power like batteries.
Most of the microprocessors do not have power saving features.	Most of the micro controllers have power saving modes like idle mode and power saving mode. This helps to reduce power consumption even further.
Since memory and I/O components are all external, each instruction will need external operation, hence it is relatively slower.	Since components are internal, most of the operations are internal instruction, hence speed is fast.
Microprocessor have less number of registers, hence more operations are memory based.	Micro controller have more number of registers, hence the programs are easier to write.
Microprocessors are based on von Neumann model/architecture where program and data are stored in same memory module	Micro controllers are based on Harvard architecture where program memory and Data memory are separate
Mainly used in personal computers	Used mainly in washing machine, MP3 players

PIC microcontroller was developed in the year 1993 by microchip technology. The term PIC stands for Peripheral Interface Controller. Initially this was developed for supporting PDP computers to control its peripheral devices, and therefore, named as a peripheral interface device. These microcontrollers are very fast and easy to execute a program compared with other microcontrollers. PIC Microcontroller architecture is based on Harvard architecture. PIC microcontrollers are very popular due to their ease of programming, wide availability, easy to interfacing with other peripherals, low cost, large user base and serial programming capability (reprogramming with flash memory), etc.

We know that the microcontroller is an integrated chip which consists of CPU, RAM, ROM, timers, and counters, etc. In the same way, PIC microcontroller architecture consists of RAM, ROM, CPU, timers, counters and supports the protocols such as SPI, CAN, and UART for interfacing with other peripherals. At present PIC microcontrollers are extensively used for industrial purpose due to low power consumption, high performance ability and easy of availability of its supporting hardware and software tools like compilers, debuggers and simulators.

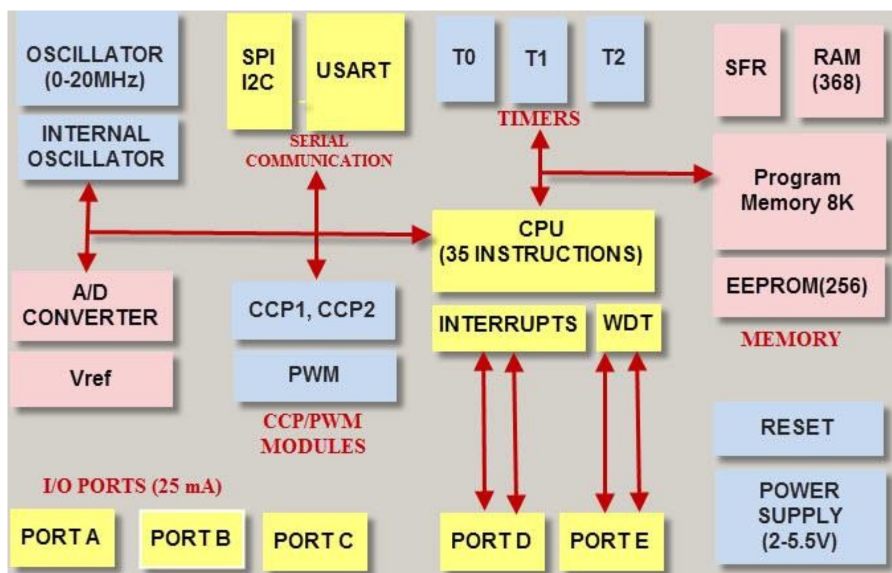
What is a PIC Microcontroller?

PIC (Programmable Interface Controllers) microcontrollers are the worlds smallest microcontrollers that can be programmed to carry out a huge range of tasks. These microcontrollers are found in many electronic devices such as phones, computer control systems, alarm systems, embedded systems, etc. Various types of microcontrollers exist, even though the best are found in the GENIE range of programmable microcontrollers. These microcontrollers are programmed and simulated by a circuit-wizard software.

Every PIC microcontroller architecture consists of some registers and stack where registers function as Random Access Memory(RAM) and stack saves the return addresses. The main features of PIC microcontrollers are RAM, flash memory, Timers/Counters, EEPROM, I/O Ports, USART, CCP (Capture/Compare/PWM module), SSP, Comparator, ADC (analog to digital converter), PSP(parallel slave port), LCD and ICSP (in circuit serial programming) The 8-bit PIC microcontroller is classified into four types on the basis of internal architecture such as Base Line PIC, Mid Range PIC, Enhanced Mid Range PIC and PIC18

Architecture of PIC Microcontroller

The PIC microcontroller architecture comprises of CPU, I/O ports, memory organization, A/D converter, timers/counters, interrupts, serial communication, oscillator and CCP module which are discussed in detailed below.



Architecture of PIC Microcontroller

CPU (Central Processing Unit)

It is not different from other microcontrollers CPU and the PIC microcontroller CPU consists of the ALU, CU, MU and accumulator, etc. Arithmetic logic unit is mainly used for arithmetic operations and to take logical decisions. Memory is used for storing the instructions after processing. To control the internal and external peripherals, control unit is used which are connected to the CPU and the accumulator is used for storing the results and further process.

Memory Organization

The memory module in the PIC microcontroller architecture consists of RAM (Random Access Memory), ROM (Read Only Memory) and STACK.

Random Access Memory (RAM)

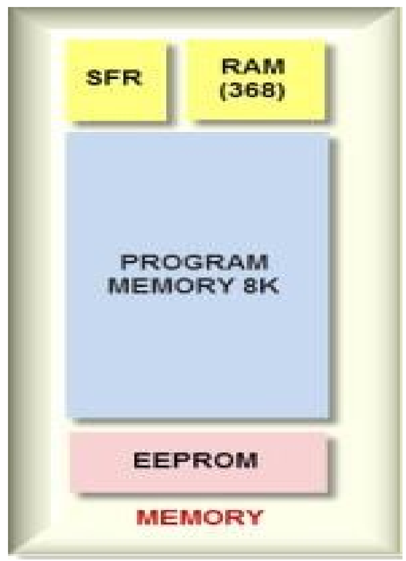
RAM is an unstable memory which is used to store the data temporarily in its registers. The RAM memory is classified into two banks, and each bank consists of so many registers. The RAM registers are classified into two types: Special Function Registers (SFR) and General Purpose Registers (GPR).

- *General Purpose Registers (GPR)*

These registers are used for general purpose only as the name implies. For example, if we want to multiply two numbers by using the PIC microcontroller. Generally, we use registers for multiplying and storing the numbers in other registers. So these registers don’t have any special function,- CPU can easily access the data in the registers.

- *Special Function Registers*

These registers are used for special purposes only as the name SFR implies. These registers will perform according to the functions assigned to them , and they cannot be used as normal registers. For example, if you cannot use the STATUS register for storing the data, these registers are used for showing the operation or status of the program. So, user cannot change the function of the SFR; the function is given by the retailer at the time of manufacturing.



Memory Organization

Read Only Memory (ROM)

Read only memory is a stable memory which is used to store the data permanently. In PIC microcontroller architecture, the architecture ROM stores the instructions or program, according to the program the microcontroller acts. The ROM is also called as program memory, wherein the user will write the program for microcontroller and saves it permanently, and finally the program is executed by the CPU. The microcontrollers performance depends on the instruction, which is executed by the CPU.

Electrically Erasable Programmable Read Only Memory (EEPROM)

In the normal ROM, we can write the program for only once we cannot use again the microcontroller for multiple times. But, in the EEPROM, we can program the ROM multiple times.

Flash Memory

Flash memory is also programmable read only memory (PROM) in which we can read, write and erase the program thousands of times. Generally, the PIC microcontroller uses this type of ROM.

Stack

When an interrupt occurs, first the PIC microcontroller has to execute the interrupt and the existing process address. Then that is being executed is stored in the stack. After completing the execution of the interrupt, the microcontroller calls the process with the help of address, which is stored in the stack and get executes the process.

I/O Ports

- The series of PIC16 consists of five ports such as Port A, Port B, Port C, Port D & Port E.
- Port A is an 16-bit port that can be used as input or output port based on the status of the TRISA (Tradoc Intelligence Support Activity) register.
- Port B is an 8- bit port that can be used as both input and output port.

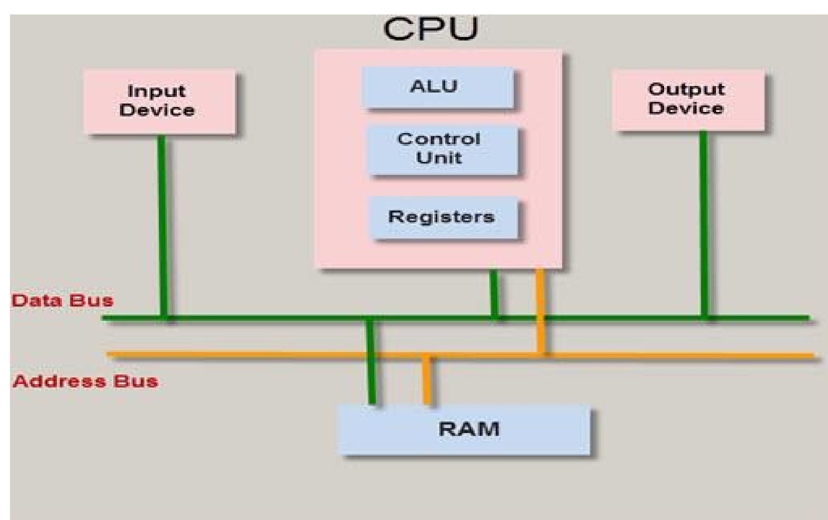
- Port C is an 8-bit and the input of output operation is decided by the status of the TRISC register.
- Port D is an 8-bit port acts as a slave port for connection to the microprocessor BUS.
- Port E is a 3-bit port which serves the additional function of the control signals to the analog to digital converter.

BUS

BUS is used to transfer and receive the data from one peripheral to another. It is classified into two types such as data bus and address.

Data Bus: It is used for only transfer or receive the data.

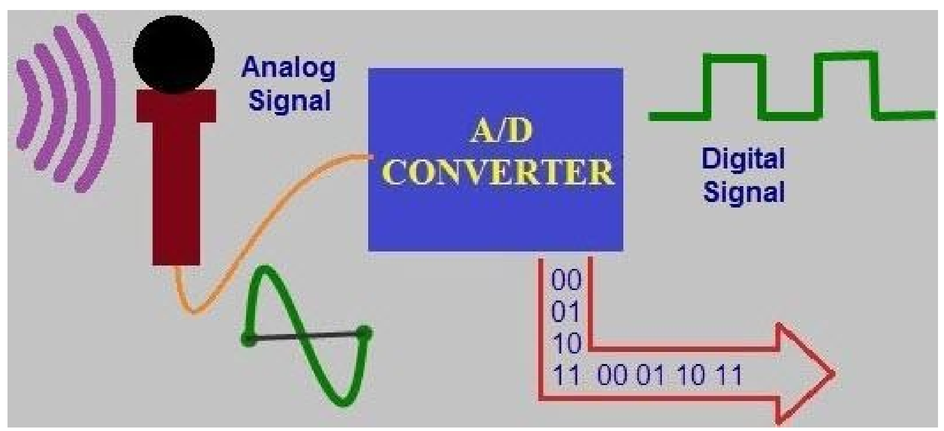
Address Bus: Address bus is used to transmit the memory address from the peripherals to the CPU. I/O pins are used to interface the external peripherals; UART and USART both are serial communication protocols which are used for interfacing serial devices like GSM, GPS, Bluetooth, IR , etc.



BUS

A/D converters

The main intention of this analog to digital converter is to convert analog voltage values to digital voltage values. A/D module of PIC microcontroller consists of 5 inputs for 28 pin devices and 8 inputs for 40 pin devices. The operation of the analog to digital converter is controlled by ADCON0 and ADCON1 special registers. The upper bits of the converter are stored in register ADRESH and lower bits of the converter are stored in register ADRESL. For this operation, it requires 5V of an analog reference voltage.



A/D CONVERTER

Timers/ Counters

PIC microcontroller has four timer/counters wherein the one 8-bit timer and the remaining timers have the choice to select 8 or 16-bit mode. Timers are used for generating accuracy actions, for **example, creating specific time delays between two operations.**

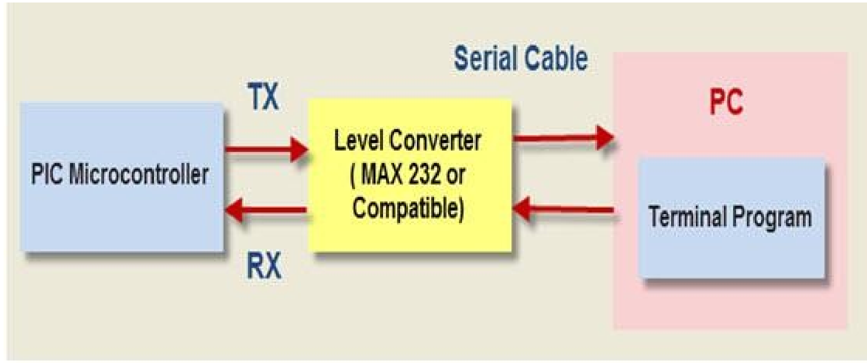
Interrupts

PIC microcontroller consists of 20 internal interrupts and three external interrupt sources which are associated with different peripherals like ADC, USART, Timers, and so on.

Serial Communication

Serial communication is the method of transferring data one bit at a time sequentially over a communication channel.

- USART: The name USART stands for Universal synchronous and Asynchronous Receiver and Transmitter which is a serial communication for two protocols. It is used for transmitting and receiving the data bit by bit over a single wire with respect to clock pulses. The PIC microcontroller has two pins TXD and RXD. These pins are used for transmitting and receiving the data serially.
- SPI Protocol: The term SPI stands for Serial Peripheral Interface. This protocol is used to send data between PIC microcontroller and other peripherals such as SD cards, sensors and shift registers. PIC microcontroller support three wire SPI communications between two devices on a common clock source. The data rate of SPI protocol is more than that of the USART.
- I2C Protocol: The term I2C stands for Inter Integrated Circuit , and it is a serial protocol which is used to connect low speed devices such as EEPROMS, microcontrollers, A/D converters, etc. PIC microcontroller support two wire Interface or I2C communication between two devices which can work as both Master and Slave device.



Serial Communication

Oscillators

Oscillators are used for timing generation. Pic microcontroller consist of external oscillators like RC oscillators or crystal oscillators. Where the crystal oscillator is connected between the two oscillator pins. The value of the capacitor is connected to every pin that decides the mode of the operation of the oscillator. The modes are crystal mode, high-speed mode and the low-power mode. In case of RC oscillators, the value of the resistor & capacitor determine the clock frequency and the range of clock frequency is 30KHz to 4MHz.

CCP module

The name CCP module stands for capture/compare/PWM where it works in three modes such as capture mode, compare mode and PWM mode.

- Capture Mode: Capture mode captures the time of arrival of a signal, or in other words, when the CCP pin goes high, it captures the value of the Timer1.

- Compare Mode: Compare mode acts as an analog comparator. When the timer1 value reaches a certain reference value, then it generates an output.

- PWM Mode: PWM mode provides pulse width modulated output with a 10-bit resolution and programmable duty cycle.

PIC Microcontroller Applications

The PIC microcontroller projects can be used in different applications, such as peripherals, audio accessories, video games, etc. For better understanding of this PIC microcontroller, the following project demonstrates PIC microcontroller's operations.

ARM Based Microcontrollers

ARM based Microcontrollers (MCU) contain a 32-bit wide data bus. They are the brain of an embedded system, a computer scaled down to a single compact chip for managing a specific operation, and are highly integrated single chips with a processor, memory, I/O peripherals, timer/counter, and communication ports all contained within. The Processor present in the MCU is the core/CPU which decides

the functioning of an MCU. An ARM MCU is developed by ARM Holdings that contains an ARM processor core developed based on Advanced RISC Machine (ARM) architecture with32-bit RISC (reduced instruction set computer) instruction set in it. ARM cores include Cortex, SecurCore, and Mali.

ARM Architecture

The ARM architecture processor is an advanced reduced instruction set computing [RISC] machine and it's a 32bit reduced instruction set computer (RISC) microcontroller. It was introduced by the Acron computer organization in 1987. This ARM is a family of microcontroller developed by makers like ST Microelectronics, Motorola, and so on. The ARM architecture comes with totally different versions like ARMv1, ARMv2, etc., and, each one has its own advantage and disadvantages.

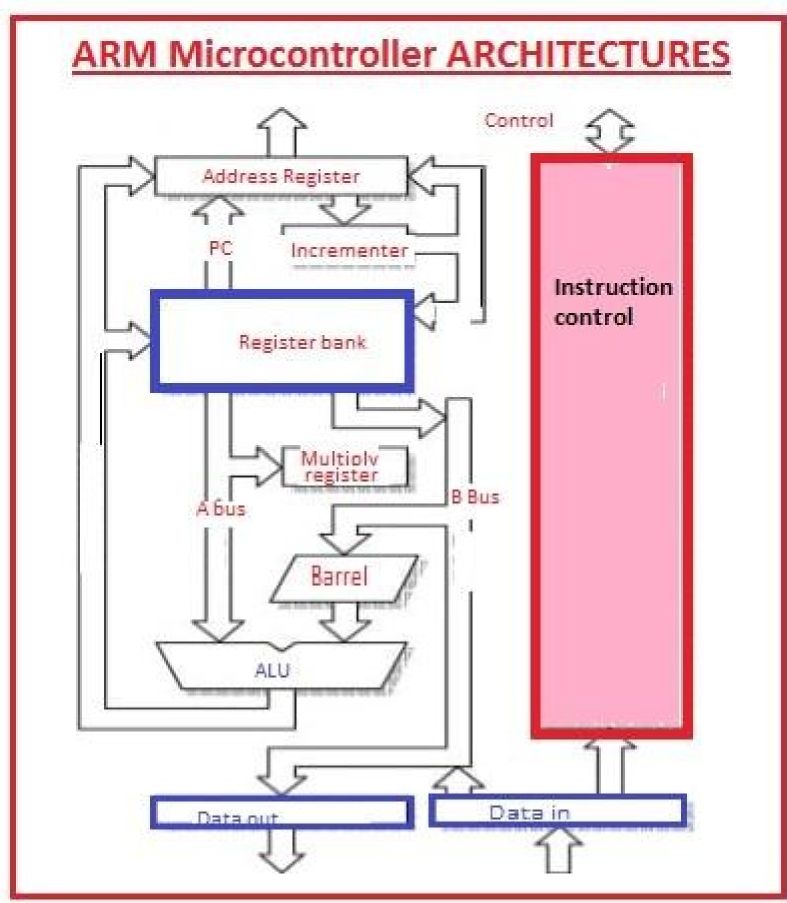
The ARM microcontroller most commonly used controller in different types of embedded projects and in different types of industrial projects it uses due to different types of advantages over other controllers and modern structures.

Due to less cost and better operation, it is mostly used in different types of projects such as the different categories of control systems, wireless circuits, sensing devices, and in automobiles. This controller is a memory of different types of central processing units and is based on RISC. It is available in thirty-two-bit and sixty-four-bit configurations. As its processor is based on RISC which is a large operating speed and executes the least number of commands. In today's post, we will have a detailed look at its working, operation, pinouts, and some other related factors. So let's get started with *Introduction to* ***ARM Microcontroller.***

Introduction to ARM Microcontroller

- The **ARM** stands for Advanced RISC Machine and it is based on the RISC architecture which is a commonly used computer configuration.
- It is a thirty-two-bit module that was created by Acron computers in 1987.
- There are different types of MCU manufacturers that create this board are ST Microelectronics and Motorola.
- This module has different categories such as ARMv1, ARMv2, and each module provides its own features.
- As it used RISC or a reduced instruction set which helps to reduce the physical dimension of the integrated circuit so it uses a small number of transistors for its creation.
- As this model exists in small size so different devices such as mobile phones, tablets, and other handheld devices comprise of this board.
- For execution, fetching, decoding, different types of commands three-stage pipeline is used.

- The processor of this board uses thumb commands based on the thumb two techniques, so it decreases the memory needed for the program and makes sure a higher density of coding.
- As this model comprises thirty two-bit architecture which provides better performance of the execution of commands.



Features of ARM Microcontroller

- These are some important features of this controller which are described here with the details.
- This board comprises of thirty two bit central processing unit which is high speed.
- It comprises of the three-stage pipeline.
- This board uses the thumb 2 technique.
- This module is compatible with the different types of tools and RTOS.
- It is compatible with the sleep mode of operation.

- It has the ability to control different types of software

The ARM Architecture

- Arithmetic Logic Unit
- Booth multiplier
- Barrel shifter
- Control unit
- Register file

This article covers the below mentioned components.

The ARM processor conjointly has other components like the Program status register, which contains the processor flags (Z, S, V and C). The modes bits conjointly exist within the program standing register, in addition to the interrupt and quick interrupt disable bits; Some special registers: Some registers are used like the instruction, memory data read and write registers and memory address register.

Priority encoder: The encoder is used in the multiple load and store instruction to point which register within the register file to be loaded or kept .

Multiplexers: several multiplexers are accustomed to the management operation of the processor buses. Because of the restricted project time, we tend to implement these components in a very behavioral model. Each component is described with an entity. Every entity has its own architecture, which can be optimized for certain necessities depending on its application. This creates the design easier to construct and maintain.

ARM Microcontroller Register Modes

An ARM micrcontroller is a load store reducing instruction set computer architecture means the core cannot directly operate with the memory. The data operations must be done by the registers and the information is stored in the memory by an address. The ARM cortex-M3 consists of 37 register sets wherein 31 are general purpose registers and 6 are status registers. The ARM uses seven processing modes to run the user task.

- USER Mode
- FIQ Mode
- IRQ Mode
- SVC Mode

- UNDEFINED Mode
- ABORT Mode
- Monitor Mode

ARM Microcontroller Register Modes

USER Mode: The user mode is a normal mode, which has the least number of registers. It doesn’t have SPSR and has limited access to the CPSR.

FIQ and IRQ: The FIQ and IRQ are the two interrupt caused modes of the CPU. The FIQ is processing interrupt and IRQ is standard interrupt. The FIQ mode has additional five banked registers to provide more flexibility and high performance when critical interrupts are handled.

SVC Mode: The Supervisor mode is the software interrupt mode of the processor to start up or reset.

Undefined Mode: The Undefined mode traps when illegal instructions are executed. The ARM core consists of 32-bit data bus and faster data flow.

THUMB Mode: In THUMB mode 32-bit data is divided into 16-bits and increases the processing speed.

THUMB-2 Mode: In THUMB-2 mode the instructions can be either 16-bit or 32-bit and it increases the performance of the ARM cortex –M3 microcontroller. The ARM cortex-m3 microcontroller uses only THUMB-2 instructions.

Some of the registers are reserved in each mode for the specific use of the core. The reserved registers are

- Stack Pointer (SP).
- Link Register (LR).
- Program Counter (PC).

- Current Program Status Register (CPSR).
- Saved Program Status Register (SPSR).

The reserved registers are used for specific functions. The SPSR and CPSR contain the status control bits which are used to store the temporary data. The SPSR and CPSR register have some properties that are defined operating modes, Interrupt enable or disable flags and ALU status flag. The ARM core operates in two states 32-bit state or THUMBS state.

Arithmetic Logic Unit (ALU)

The ALU has two 32-bits inputs. The primary comes from the register file, whereas the other comes from the shifter. Status registers flags modified by the ALU outputs. The V-bit output goes to the V flag as well as the Count goes to the C flag. Whereas the foremost significant bit really represents the S flag, the ALU output operation is done by NORed to get the Z flag. The ALU has a 4-bit function bus that permits up to 16 opcode to be implemented.

Booth Multiplier Factor

The multiplier factor has 3 32-bit inputs and the inputs return from the register file. The multiplier output is barely 32-Least Significant Bits of the merchandise. The entity representation of the multiplier factor is shown in the above block diagram. The multiplication starts whenever the beginning 04 input goes active. Fin of the output goes high when finishing.

Booth Algorithm

Booth algorithm is a noteworthy multiplication algorithmic rule for 2’s complement numbers. This treats positive and negative numbers uniformly. Moreover, the runs of 0’s or 1’s within the multiplier factor are skipped over without any addition or subtraction being performed, thereby creating possible quicker multiplication. The figure shows the simulation results for the multiplier test bench. It’s clear that the multiplication finishes only in16 clock cycle.

Barrel Shifter

The barrel shifter features a 32-bit input to be shifted. This input is coming back from the register file or it might be immediate data. The shifter has different control inputs coming back from the instruction register. The Shift field within the instruction controls the operation of the barrel shifter. This field indicates the kind of shift to be performed (logical left or right, arithmetic right or rotate right). The quantity by which the register ought to be shifted is contained in an

immediate field within the instruction or it might be the lower 6 bits of a register within the register file.

The shift_val input bus is 6-bits, permitting up to 32 bit shift. The shifttype indicates the needed shift sort of 00, 01, 10, 11 are corresponding to shift left, shift right, an arithmetic shift right and rotate right, respectively. The barrel shifter is especially created with multiplexers.

Control Unit

For any microprocessor, control unit is the heart of the whole process and it is responsible for the system operation,so the control unit design is the most important part within the whole design. The control unit is sometimes a pure combinational circuit design. Here, the control unit is implemented by easy state machine. The processor timing is additionally included within the control unit. Signals from the control unit are connected to each component within the processor to supervise its operation.

Application of ARM Microcontroller

- This board is used in different types of techniques used in space and aerospace.
- Different types of medical devices such as MRI machines, computed tomography scanners, ultrasound machines.
- It used in different types of accelerators, nuclear reactors, and X-ray machines.

5.1 PROGRAMMING 8051 TIMERS

8051 TIMERS

8051 has two timers/Counters(TIMER 0 and TIMER 1). They can be used as timers to generate time delays or as event counters.

BASIC REGISTERS OF THE

TIMER1.TIMER REGISTERS

It is a16 bit register and can be accessed as 8 bit registers say Timer High(THx) and Timer Low (TLx),These registers can be accessed as any other

registerslike A,B,R0 etc.,

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

Figure 5.1.1 Timer Registers

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazidi,Janice Gillispie Mazidi, Rolin McKinlay]

2. TMOD (Timer Mode) Register

TMOD Register is an 8-bit register used to control the mode of operation of both timers. The high four bits (bits 4 through 7) relate to Timer 1whereas the low four bits (bits 0 through 3) perform the exact same functions, but for timer 0. In each case, the lower two bits are used to set the timer mode and upper two bits to specify the operationas shown in Figure 5.1.2.

(MSB) (LSB)

GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
Timer 0				Timer 1			

GATE Gating control when set. Timer/Counter "x" is enabled only while "$\overline{INTx}$" pin is high and "TRx" control bit is set. When cleared Timer "x" is enabled whenever "TRx" control bit is set.

C/$\overline{T}$ Timer or Counter selector cleared for Timer operation (input from internal system clock). Set for Counter operation (input from "Tx" input pin).

M1	M0	Operating Mode
0	0	8-bit Timer/Counter "THx" with "TLx"s 5-bit prescaler.
0	1	16-bit Timer/Counter "THx" with "TLx" are cascaded; there is no prescaler.
1	0	8-bit auto-reload Timer/Counter "THx" holds a value which is to be reloaded into "TLx" each time it overflows.
1	1	(Timer 0) TL0 is an 8-bit Timer/Counter controlled by the standard Timer 0 control bits. TH0 is an 8-bit timer only controlled by Timer 1 control bits.
1	1	(Timer 1) Timer/Counter 1 stopped.

Figure 5.1.2 TMOD Register

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazidi,Janice Gillispie Mazidi, Rolin McKinlay, pg.no.241]

In upper or lower 4 bits, first bit is a GATE bit. Every timer has a means ofstarting and stopping. Some timers do this by software, some by hardware, and some haveboth software and hardware controls. The hardware way of starting and stopping the timerby an external source is achieved by making GATE=1 in the TMOD register and if GATE=0 then we do start and stop the timers by programming.

The second bit is C/T bit and is used to decide whether a timer is used as a time delay generator or an event counter. If this bit is 0 then it is used as a timer and if itis 1 then it is used as a counter.

In upper or lower 4 bits, the last bits third and fourth are known as M1 and M0 respectively. These are used to select the timer mode.

TIMER'S CLOCK FREQUENCY AND ITS PERIOD

In 8051-based system, the crystal oscillator has a frequency of 11.0592 MHz whenC/T bit of TMOD is 0. Each machine cycle is made up of 12 clock cycles.

Hence for a single machine cycle, the frequency becomes 1/12 × 11.0529 MHz = 921.6 KHz. For a single machine cycle, the time taken is T = 1/921.6 KHz = 1.085 us, so the oscillator takes 1.085us for completing a single machine cycle.

MODES OF OPERATION:

MODE 1:

It is a 16-bit timer; therefore it allows values from 0000 to FFFFH to be loaded into the timer's registers TL and TH as shown in Figure 5.1.3. After TH and TL are loaded with a 16-bit initial value, the timer must be started. We can do it by "SETB TR0" for timer 0 and "SETB TR1" for timer 1. After the timer is started, it starts count up until it reaches its limit of FFFFH. When it rolls over from FFFF to 0000H, it sets higha flag bit called TFx (timer flag). This timer flag can be monitored. When this timer flagis raised, one option would be stop the timer with the instructions "CLR TR0" or CLR TR1 for timer 0 and timer 1 respectively. Again, it must be noted that each timer flag TF0for timer 0 and TF1 for timer1. After the timer reaches its limit and rolls over, in order torepeat the process the registers TH and TL must be reloaded with the original value and TF must be reset to 0.

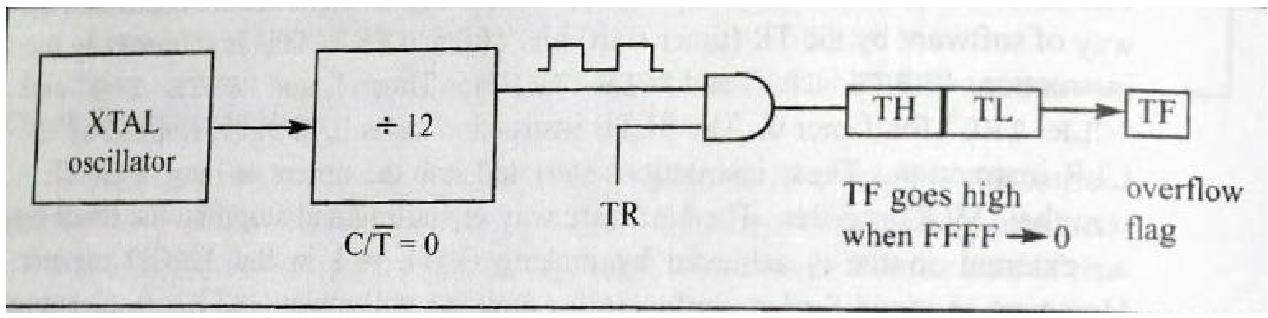


Figure 5.1.3 Timer in Mode 1

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazidi,Janice Gillispie Mazidi, Rolin McKinlay, pg.no.244]

MODE 0 :

Mode 0 is exactly same like mode 1 except that it is a 13-bit timer instead

of 16-bit. The 13- bit counter can hold values between 0000 to 1FFFH in TH-TL.

MODE 2:

It is an 8 bit timer that allows only values of 00 to FFH to be loaded into the

timer's register TH as shown in Figure 5.1.4. After THx is loaded with 8 bit value, the 8051 gives a copy of it to TLx. Then the timer must be started. It is done by the instruction"SETB TR0" for timer 0 and "SETB TR1" for timer1. This is like mode 1.

After timer is started, it starts to count up by incrementing the TLx register.It counts up until it reaches its limit of FFH. When it rolls over from FFH to 00, it sets high the TFx (timer flag). If we are using timer 0, TF0 goes high; if using TF1 then TF1is raised. When TLx register rolls from FFH to 00 and TF is set to 1, TLx is reloaded automatically with the original value kept by the THx register. To repeat the process, wemust simply clear TFx and let it go without any need by the programmer to reload the original value. This makes mode 2 auto reload, in contrast in mode 1 in which programmer has to reload THx and TLx.

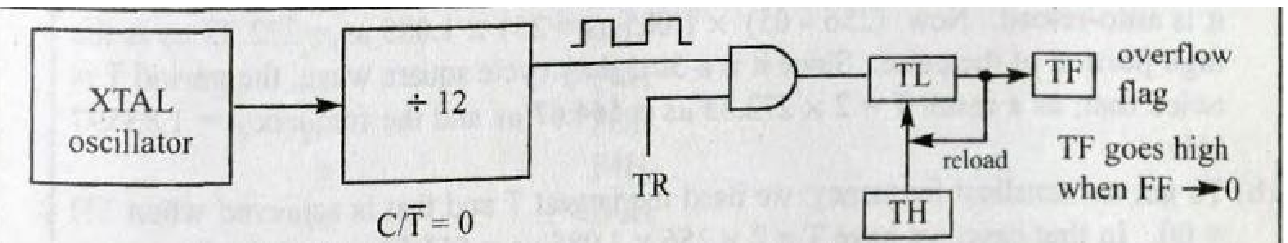


Figure 5.1.4 Timer in Mode 2

[Source: "The 8051Microcontroller and Embedded Systems: Using Assembly and C" by Mohamed Ali Mazidi,Janice Gillispie Mazidi, Rolin McKinlay, pg.no.251]

MODE 3:

Mode 3 is also known as a split timer mode. Timer 0 and 1 may be programmed to be in mode 0, 1 and 2 independently of similar mode for other timer. Thisis not true for mode 3; timers do not operate independently if mode 3 is chosen for timer 0. Placing timer 1 in mode 3 causes it to stop counting; the control bit TR1 and the timer1 flag TF1 are then used by timer 0.

TIMER PROGRAMMING

MODE 1 PROGRAMMING:

It is a 16 bit Timer mode.

STEPS TO PROGRAM IN MODE 1:

1. Load the TMOD value register with mode and timer0 or 1.
2. Load registers TLx and THx with initial count corresponding to delay.
3. Start the timer.
4. Continuously monitor the timer flag (TFx) with the "JNB TFx,target" instruction tosee if it is raised, if it is raised(TFx=1) then get out of the loop.

5. Stop the timer.
6. Clear the TFx flag for the next round.
7. Go back to Step 2 to load THx and TLx again.

STEPS TO CALCULATE COUNT TO LOAD INTO THX-TLX TO GENERATE DESIRED DELAY

(Assume XTAL = 11.0592 MHz)

Steps for finding the TH, TL registers values

1. Divide the desired time delay by 1.085 us
2. Perform 65536 – n, where n is the decimal value we got in Step 1
3. Convert the result of Step 2 to hex, say we get yyxx. yyxx is the initial hex value to beloaded into the timer's register.
4. Set TL = xx and TH = yy

Example:

Generate Delay =10ms with Clock frequency= 11.0592 MHz, using Timer0 in mode1.

Solution:

Given:

Time delay=10ms
Clock frequency=11.0592 MHz

Step 1: Divide the desired time delay by 1.085 usCount =10 ms/ 1.085 us=9216

Step 2: Perform 65536 – n

65536-9216=56320= DC00H

Step 3: Set TL = xx and TH = yy

Here xx=DC and yy=00, Hence, TH0=DC and TL0=00.

Following is the assembly code program to generate a delay of 10ms.

```
MOV TMOD, #01               ;Timer 0, mode 1, 16-bitmode
HERE: MOV TL0, #00          ;TL0=0, the low byte
MOV TH0,#0DCH               ;TH0=DC, the high byte
SETB TR0                    ;Start timer 0
AGAIN: JNB TF0,AGAIN        ;Monitor
timer flag 0CLR TR0         ;Stop the
timer 0
CLR TF0                     ;Clear timer 0 flag
```

Program to generate a square wave of 5 kHz frequency on pin P1.0, clock frequency

=11.0592 MHz

Given:

Square wave frequency=5

kHz Clock

frequency=11.0592 MHz

Step 1: Calculate the

Time delayT=1/f=1/5 kHz

=0.2 ms

T=0.2 ms which is the period of

square waveT/2 =0.2/2=0.1 ms delay

for high and low

Step 2: Divide the desired time delay by

1.085 usCount=0.1ms/1.085 us = 921

Step 3: Perform 65536 – n

TH0-TL0= 65536-921=64615=FC67H
Step 4: Set TL = xx and TH = yy

Here xx=FC and yy=67. Hence,TH0=FCh,TL0=67h
MOV TMOD,#01

AGAIN: MOV TL1,#67H

MOV TH1,#0FCH

SETB TR1 Timer 0, mode 1, 16-bitmode

;TL1=67, low byte of timer

;TH1=FC, the high byte

;Start timer 1 BACK: JNB TF1,BACK ;until

timer rolls over

CPL P1.0 ; compliment P1.0

CLR TR1 ;Stop the timer 1

CLR TF1 ;Clear timer 1 flag

SJMP **AGAIN** ;Reload timer

STEPS FOR GENERATING DELAY IN MODE 2

1. Select timer in TMOD register indicating which timer (timer 0 or timer 1) is to be used,and the timer mode (mode 2) to be selected
2. THx register loaded with initial count value
3. Start timer
4. Continuously monitoring the timer flag (TFx) with the JNB TFx,target instruction tosee whether TFx is ‘1’ (high) . Get out of the loop when TF goes high
5. Clear the TFx flag
6. Go back to Step4, since mode 2 is auto-reload

Toggle LED connected at P1.0 with 5 microsec delay using timer1 and mode 2

```
MOV TMOD,#20                  ;Timer 1 mode 2, 8-bit auto
reloadAGAIN: MOV TH1,#-5  ;TL1=256-5, low byte of
timer SETB TR1                ;Start timer 1
BACK: JNB TF1, BACK          ;until timer rolls over
CPL P1.0                      ; compliment P1.0 toggle LED
CLR TF1                       ;Stop the timer 1
```

Step 4: Set TL = xx and TH = yy

Here xx=FC and yy=67. Hence,TH0=FCh,TL0=67h

MOV TMOD,#01 **AGAIN**: MOV TL1,#67HMOV TH1,#0FCH

SETB TR1

;Timer 0, mode 1, 16-bitmode

;TL1=67, low byte of timer

;TH1=FC, the high byte

;Start timer 1 BACK: JNB TF1,BACK ;until

```
SJMP BACK                     ; loop
```

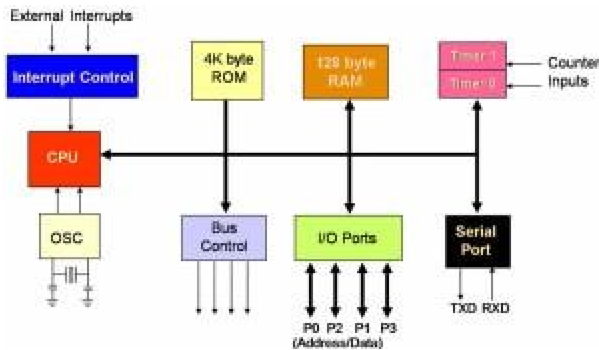
Interfacing Devices

Interfacing can be defined as transferring data between microcontrollers and interfacing peripherals such as sensors, keypads, microprocessors, analog to digital converters or ADC, LCD displays, motors, external memories, even with other microcontrollers, some other interfacing peripheral devices and so on or input devices and output devices. These devices that are interfacing with 8051 microcontroller are used for performing special tasks or functions are called as interfacing devices.

Interfacing is a technique that has been developed and being used to solve many composite problems in circuit designing with appropriate features, reliability, availability, cost, power consumption, size, weight, and so on. To facilitate multiple features with simple circuits, microcontroller is interfaced with devices such as ADC, keypad, LCD display and so on.

Major Electronic Peripherals Interfacing to Microcontroller 8051

Interfacing is one of the important concepts in microcontroller 8051 because the microcontroller is a CPU that can perform some operation on a data and gives the output. However to perform the operation we need an input device to enter the data and in turn output device displays the results of the operation. Here we are using keyboard and LCD display as input and output devices along with the microcontroller.

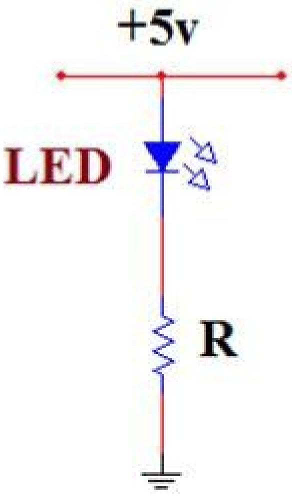


Microcontroller 8051 Peripheral devices

Interfacing is the process of connecting devices together so that they can exchange the information and that proves to be easier to write the programs. There are different type of input and output devices as for our requirement such as LEDs, LCDs, 7segment, keypad, motors and other devices.
Here is given some important modules interfaced with microcontroller 8051.

1. LED Interfacing to Microcontroller:

Description:
LEDs are most commonly used in many applications for indicating the output. They find huge range of applications as indicators during test to check the validity of results at different stages. They are very cheap and easily available in a variety of shape, color and size.

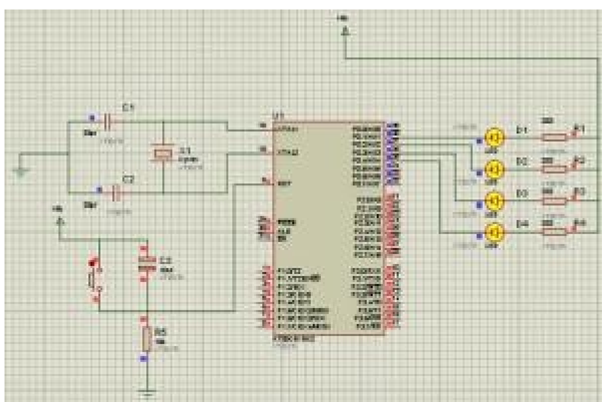


Light Emitting Diode

The principle of operation of LEDs is very easy. A simple LEDs also servers as a basic display devices, it On and OFF state express meaning full information about a device. The common available LEDs have a 1.7v voltage drop that means when we apply above 1.7V, the diode conducts. The diode needs 10mA current to glow with full intensity.
The following circuit describes "how to glow the LEDs".

LEDs can be interfaced to the microcontroller in either common anode or common cathode configuration. Here the LEDs are connected in common anode configuration because the common cathode configuration consumes more power.

Circuit Diagram

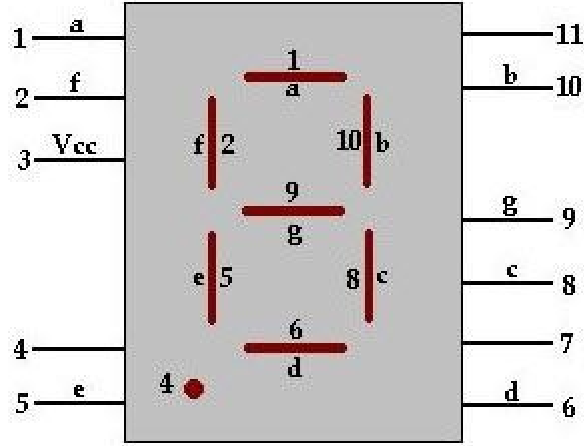


LED Interfacing to Microcontroller

2. 7-Segment Display interfacing circuit

Description:
A Seven segment display is the most basic electronic display. It consists of eight

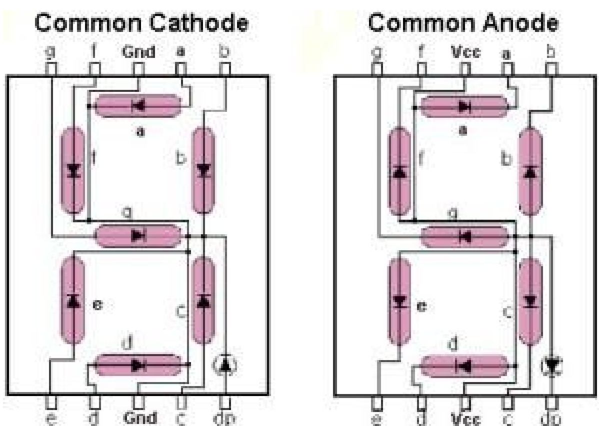
LEDs which are associated in a sequence manner so as to display digits from 0 to 9 when proper combinations of LEDs are switched on. A 7-segment display uses seven LEDs to display digits from 0 to 9 and the 8th LED is used for dot. A typical seven segment looks likes as shown in figure below.



7-Segment Display

The 7-segment displays are used in a number of systems to display the numeric information. They can display one digit at a time. Thus the number of segments used depends on the number of digits to display. Here the digits 0 to 9 are displayed continuously at a predefined time delay.

The 7-segment displays are available in two configurations which are common anode and common cathode. Here common anode configuration is used because output current of the microcontroller is not sufficient enough to drive the LEDs. The 7-segment display works on negative logic, we have to provide logic 0 to the corresponding pin to make on LED glow.



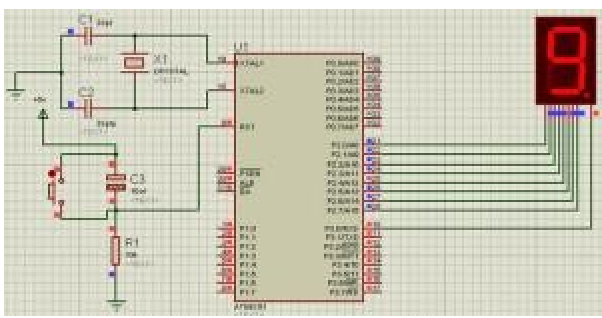
7-Segment Display Configurations

The following table shows the hex values used to display the different digits.

Digits	a	b	c	d	e	f	g	Hex Code
0	0	0	0	0	0	0	1	0x40
1	1	0	0	1	1	1	1	0xF9
2	0	0	1	0	0	1	0	0x24
3	0	0	0	0	1	1	0	0x30
4	1	0	0	1	1	0	0	0x19
5	0	1	0	0	1	0	0	0x12
6	0	1	0	0	0	0	0	0x02
7	0	0	0	1	1	1	1	0xF6
8	0	0	0	0	0	0	0	0x00
9	0	0	0	1	1	0	0	0x10

7-Segment Display Table

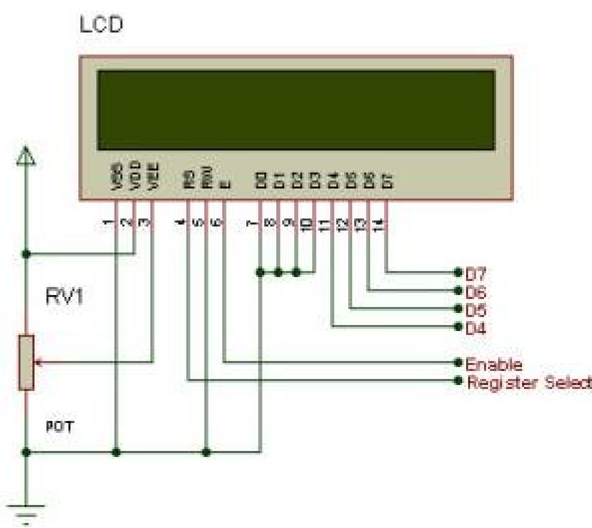
Circuit Diagram



7-Segment Display interfacing

3. LCD Interfacing to Microcontroller

LCD stands for liquid crystal display which can display the characters per line. Here 16 by 2 LCD display can display 16 characters per line and there are 2 lines. In this LCD each character is displayed in 5*7 pixel matrix.



LCD Display

LCD is very important device which is used for almost all automated devices such as washing machines, an autonomous robot, power control systems and other devices. This is achieved by displaying their status on small display modules like 7-seven segment displays, multi segment LEDs etc. The reasons

being, LCDs are reasonably priced, easily programmable and they have a no limitations of displaying special characters.
It consists of two registers such as command/instruction register and data register.

The command/instruction register stores the command instructions given to the LCD. A command is an instruction which is given to the LCD that perform a set of predefined tasks like initializing, clearing the screen, setting the cursor posing, controlling display etc.

The data register stores the data to be displayed on LCD. The data is an ASCII value of the characters to be displayed on the LCD.

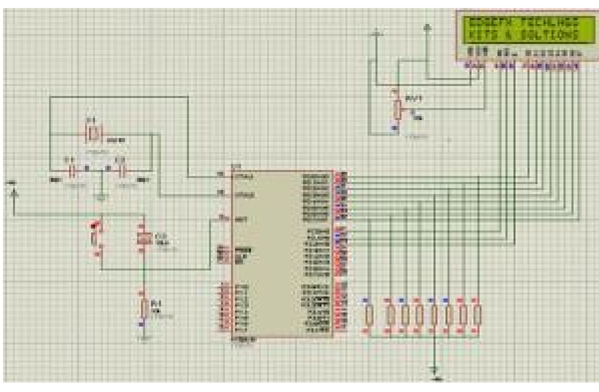
Operation of LCD is controlled by two commands. When RS=0, R/W=1 it reads the data and when RS=1, R/W=0, it writes (print) the data.

LCD uses following command codes:

Hex Rode	Command to LCD Instruction Register
1	clear screen display
2	Return home
4	decrement cursor
6	increment cursor
E	display ON, cursor ON
80	force the cursor to the begining of the 1st line
c0	force cursor to the begining of the 2nd line
38	Use 2 lines and 5*7 matrics

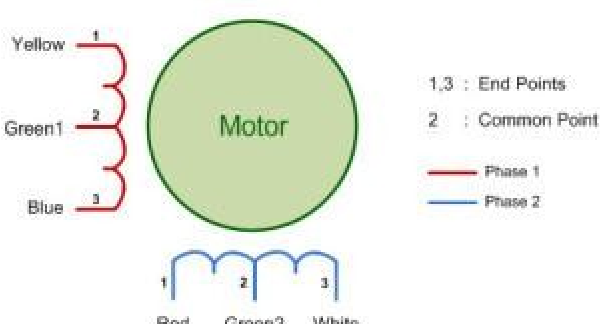
LCD Display Commands

Circuit Diagram:



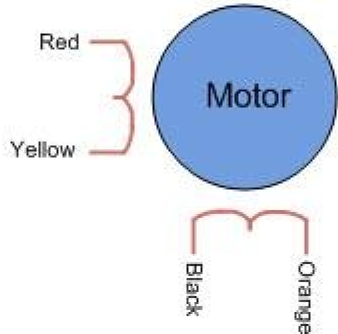
LCD Interfacing to Microcontroller

4. Stepper motor interfacing circuit



Unipolar Stepper Motor

A stepper motor is one of the most commonly used motor for precise angular movement. The advantage of using a stepper motor is that the angular position of the motor can be controlled without any feedback mechanism. The stepper motors are widely used in industrial and commercial applications. They are also commonly used as in drive systems such as robots, washing machines etc.



Bipolar Stepper Motor Bipolar Stepper Motor

Stepper motors can be unipolar or bipolar and here we are using unipolar stepper motor. The unipolar stepper motor consists of six wires out of which four are connected to coil of the motor and two are common wires. Each common wire is connected to a voltage source and remaining wires are connected to the microcontroller.

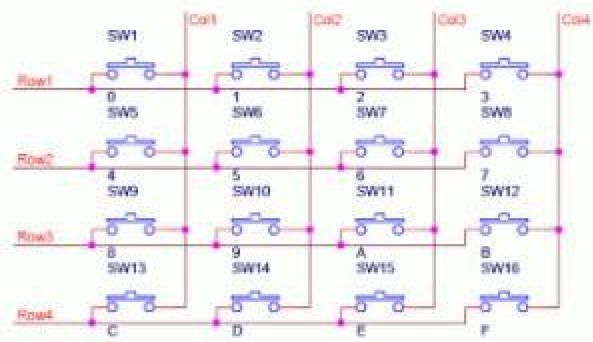
5. Matrix keypad interfacing to 8051

Description:

Matrix Keypad

Keypad is a widely used input device with lot of applications such as telephone, computer, ATM, electronic lock etc. A keypad is used to take input from the user for further processing. Here a 4 by 3 matrix keypad consisting of switches arranged in rows and columns is interfaced to the microcontroller. A 16 by 2 LCD is also interfaced for displaying the output.

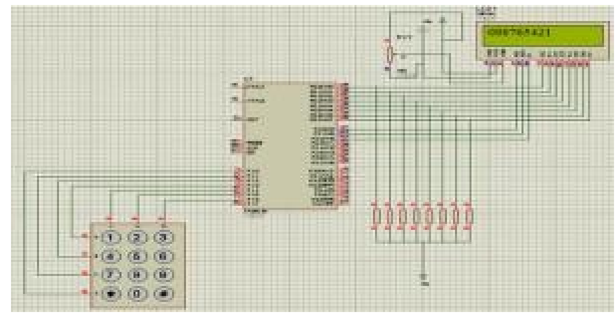
The interfacing concept of keypad is very simple. Every number of keypad is assigned two unique parameters that are row and column (R, C). Hence every time a key is pressed the number is identifying by detecting the row and column numbers of keypad.



Keypad Internal Diagram

Initially all the rows are set to zero ('0') by the controller and columns are scanned to check if any key is pressed. In case of no key is pressed the output of all columns will be high ('1').

Circuit Diagram



Matrix keypad interfacing to 8051

We hope we have been able to provide ample knowledge about the basic yet important interfacing circuits of microcontroller 8051. These are the most basic circuits required in any embedded system application and we hope we have provided you with a good revision.
A further query or feedback related to this topic is welcome to be mentioned in the comment section below.

Photo Credits

- Microcontroller 8051 Peripheral devices by aninditadhikary
- 7-Segment Display by electronicsteacher
- 7-Segment Display Configurations by thelearningpit
- LCD Display by bp.blogspot
- Unipolar &Bipolar Steppers by engineersgarage
- Matrix Keypad by vetco
- Keypad Internal Diagram by bp.blogspot