GRT INSTITUTE OF ENGINEERING AND TECHNOLOGY, TIRUTTANI - 631209

Approved by AICTE, NewDelhi Affiliated to Anna University, Chennai

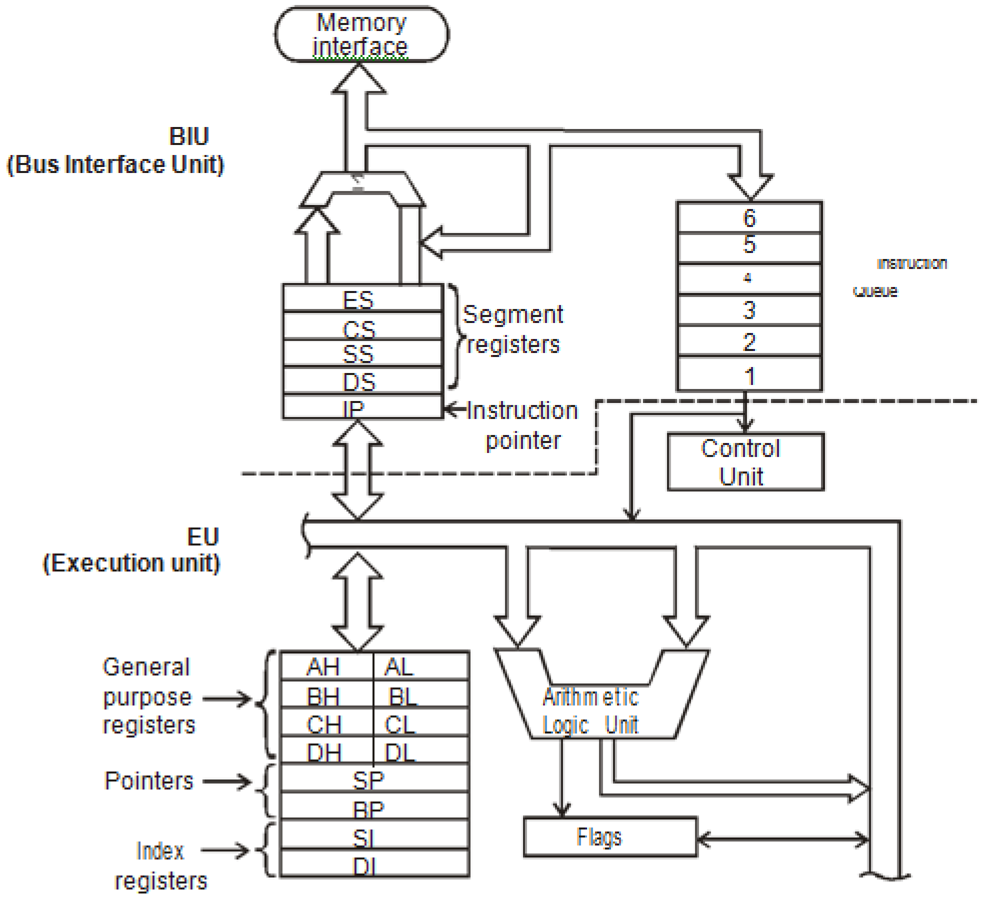
EC 8691 - MICROPROCESSOR AND MICROCONTROLLER

HANDOUTS

UNIT-1

THE 8086 MICROPROCESSOR

ARCHITECTURE OF 8086



1. Bus Interface Unit (BIU)
2. Execution Unit (EU)

The BIU and EU function independently. The BIU interfaces the 8086 to the outside world. The BIU fetches instructions, reads data from memory and ports, and writes data to memory and I/O ports.

EU receives program instruction codes and data from the BIU, executes these instructions and stores the results either in the general registers or output them through the BIU. EU has no connections to the system buses. It receives and outputs all its data through the BIU.

The BIU contains

1. Segment registers
2. Instruction pointer
3. Instruction queue

The EU contains

1. ALU
2. General purpose registers
3. Index registers
4. Pointers
5. Flag register

General Purpose Registers

1. Accumulator register (AX)
2. Base register (BX)
3. Count register (CX)
4. Data register (DX)

(i) Accumulator register

It consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL contains the low-order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation.

(ii) Base register

BX register usually contains a data pointer used for based, based indexed or register indirect addressing.

(iii) Count register

Count register can be used as a counter in string manipulation and shift/rotate instructions.

(iv) Data register

Data register can be used as a port number in I/O operations.

Segment Registers

Most of the registers contain data/instruction offsets within 64 KB memory segment. There are four different 64 KB segments for instructions, stack, data and extra data. The segment registers are:

1. Code segment (CS)
2. Stack segment (SS)

3. Data segment (DS)
4. Extra segment (ES)

Pointer Registers

(i) Stack Pointer (SP)

It is a 16-bit register pointing to program stack.

(ii) Base Pointer (BP)

BP register is usually used for based, based indexed or register indirect addressing.

Index Registers

(i) Source Index (SI)

SI is used for indexed, based indexed and register indirect addressing and string manipulation instructions.

(ii) Destination Index (DI)

DI is used for indexed, based indexed and register indirect addressing.

Instruction Pointer (IP)

It is a 16-bit register. The operation is same as the program counter. The IP register is updated by the BIU to point to the address of the next instruction.

Flag register

- It is a 16-bit register containing nine 1-bit flags:
- Six status or condition flags (OF, SF, ZF, AF, PF, CF)
- Three control flags (TF, DF, IF)

Instruction Set

8086 instruction set consists of the following instructions:

- Data moving instructions.
- Arithmetic instructions - add, subtract, increment, decrement, convert byte/word and compare.
- Logic instructions - AND, OR, exclusive OR, shift/rotate and test.
- String manipulation instructions - load, store, move, compare and scan for byte/ word.

INTERRUPTS AND INTERRUPT SERVICE ROUTINE

Interrupts

A signal to the processor to halt its current operation and immediately transfer control to an interrupt service routine is called as interrupt. Interrupts are triggered either by hardware, as when the keyboard detects a key press, or by software, as when a program executes the INT instruction.

- There are also interrupt functions that work with disk drive and other hardware. They are called as **software interrupts**.
- Interrupts are also triggered by different hardware, these are called **hardware interrupts**.
- To make a software interrupt there is an INT instruction, it has very simple syntax: INT *value*.
  Where value can be a number between 0 to 255 (or 00 to FF H).

Interrupt Service Routines (ISRs)

ISR is a routine that receives processor control when a specific interrupt occurs. The 8086 will directly call the service routine for 256 vectored interrupts without any software processing.

Interrupt vector table:

It is a table maintained by the operating system. It contains addresses (vectors) of current interrupt service routine. When an interrupt occurs, the CPU branches to the address in the table that corresponds to the interrupt's number.

When an interrupt occurs, regardless of source, the 8086 does the following:

1. The CPU pushes the flags register onto the stack.
2. The CPU pushes a far return address (segment:offset) onto the stack, segment value first.
3. The CPU determines the cause of the interrupt (i.e., the interrupt number) and fetches the four byte interrupt vector from address 0 **:** vector x 4 (0:0, 0:4, 0:8 etc)
4. The CPU transfers control to the routine specified by the interrupt vector table entry.

AVAILABLE INTERRUPT	3FF H 3FC H	TYPE 255 POINTER: (AVAILABLE)
		. .
	084 H	TYPE 33 POINTER: (AVAILABLE)
	080 H	TYPE 32 POINTER: (AVAILABLE)
RESERVED INTERRUPT	07F H	TYPE 31 POINTER: (AVAILABLE)
		. .
	014 H	TYPE 5 POINTER: (RESERVED)
DEDICATED INTERRUPT	010 H	TYPE 4 POINTER: OVERFLOW
	00C H	TYPE 3 POINTER: 1-BYTE INT INSTRUCTION
	008 H	TYPE 2 POINTER: NON MASKABLE
	004 H	TYPE 1 POINTER: SINGLE STEP
	000 H	TYPE 0 POINTER: DIVIDE ERROR

Types of Interrupts

1. Hardware Interrupt - External uses INTR and NMI
2. Software Interrupt - Internal - from INT or INTO
3. Processor Interrupt - Traps and 10 Software Interrupts

E*xternal* - generated outside the CPU by other hardware (INTR, NMI)

Internal - generated within CPU as a result of an instruction or operation (INT, INTO, Divide Error and Single Step)

Dedicated Interrupts

Divide Error Interrupt (Type 0)

This interrupt occurs automatically following the execution of DIV or IDIV instructions when the quotient exceeds the maximum value that the division instructions allow.

Single Step Interrupt (Type 1)

This interrupt occurs automatically after execution of each instruction when the Trap Flag (TF) is set to 1. It is used to execute programs one instruction at a time, after which an interrupt is requested.

Non Maskable Interrupt (Type 2)

It is the highest priority hardware interrupt that triggers on the positive edge. This interrupt occurs automatically when it receives a low-to-high transition on its NMI input pin. This interrupt cannot be disabled or masked. It is used to save program data or processor status in case of system power failure.

Breakpoint Interrupt (Type 3)

This interrupt is used to set break points in software debugging programs.

Overflow Interrupt (Type 4)

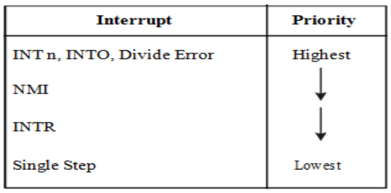
This interrupt is initiated by INTO (Interrupt on Overflow) instruction. It is used to check overflow condition after any signed arithmetic operation in the system.

Software Interrupts (INT n)

The software interrupts are non maskable interrupts. They are higher priority than hardware interrupts. The software interrupts are called within the program using the instruction INT n. Here 'n' means *value* and is in the range of 0 to 255. These interrupts are useful for debugging, testing ISRs and calling procedures.

Hardware Interrupts

INTR and NMI are called hardware interrupts. INTR is maskable and NMI is non-maskable interrupts. INTR interrupts (Type 0 -255) can be used to interrupt a program execution.



Interrupt	Priority
INT n, INTO, Divide Error	Highest
NMI	↓
INTR	↓
Single Step	Lowest

<u>ADDRESSING MODES OF 8086</u>

ADDRESSING MODES

An addressing mode is the way the 8086 identifies the operands for the instruction. All instructions that access the data use one or more of the addressing modes.

The memory address of an operand consists of two components:

- Starting address of the memory segment
- Offset

Register Addressing Mode

Both source and destination operands are registers. The operand sizes must match. MOV *destination, source*

Examples:

MOV AL, AH

MOV AX, BX

Immediate Addressing Mode

The data operand is supplied as part of the instruction. The immediate operand can only be a source.

Examples:

MOV CH, 3A H
MOV DX, 0C1A5 H

Direct Addressing Mode

One of the operands is a memory location, given by a constant offset.

In this mode the 16 bit effective address (EA) is taken directly from the displacement field of the instruction.

Examples:

MOV [1234 H], AX
MOV [3BD2 H], DL

Register Indirect Addressing Mode

One of the operands is a memory location, with the offset given by one of the BP, BX, SI, or DI registers.

Example:

MOV [BX], CL

Base Addressing Mode

In this mode EA is obtained by adding a displacement (signed 8 bit or unsigned 16 bit) value to the contents of BX or BP. The segment registers used are DS and SS.

Offset (EA) = [BX or BP + 8-bit or 16-bit displacement]

Example:

MOV AX, [BP + 200]

Indexed Addressing Mode

The operand's offset is the sum of the content of an index register SI or DI and an 8-bit or 16-bit displacement.

Example:

MOV [DI] + 4, AH

Offset (EA) = [SI or DI + 8-bit or 16-bit displacement]

Based Indexed Addressing Mode

In this mode, the EA is computed by adding a base register (BX or BP), an index register (SI or DI) and a displacement (unsigned 16 bit or sign extended 8 bit)

Offset (EA) = [BX or BP] + [SI or DI] + 8-bit or 16-bit displacement

Example:

MOV AX, [BX + SI + 1234 H]
MOV CX, [BP][SI] + 4

String Addressing Mode

The instruction is a string instruction, which uses index registers implicitly to access memory.

Example:

MOVS B

ASSEMBLER DIRECTIVES

An assembler is a program which translates an assembly language program into machine language program. An assembler directive is a statement to give direction to the assembler to perform the task of assembly process.

The assembler directives control organization of the program and provide necessary information to the assembler to understand assembly language programs to generate machine codes. They indicate how an operand or a section of a program is to be processed by the assembler. An assembler supports directives to define data, to organize segments, to control procedures, to define macros etc. An assembly language program consists of two types of statements: Instructions and Directives. The instructions are translated to machine codes by the assembler, whereas the directives are not translated to machine codes.

Some assembler directives are,

1. Borland Turbo Assembler (TASM)
2. IBM Macro Assembler (MASM)
3. Intel 8086 Macro Assembler (ASM)
4. Microsoft Macro Assembler

1. ASSUME

The ASSUME directive enables error-checking for register values.

It is used to inform the assembler the names of the logical segments, which are to be assigned to the different segments used in an assembly language program

Format:

ASSUME *segregister***:***name* [[**,** *segregister***:***name*]]...

ASSUME *dataregister***:***type* [[**,** *dataregister***:***type*]]...

ASSUME *register***:ERROR** [[**,** *register***:ERROR**]]...

ASSUME [[*register***:**]] **NOTHING** [[**,** *register***:NOTHING**]]...

Examples:

ASSUME CS : CODE

ASSUME DS : DATA

2. DB (Define Byte)

It can be used to define data like **BYTE**.

Format:

Name of the Variable *DB* *Initial values*

Example:

```
WEIGHTS                     DB   18, 68, 45
```

3. DW (Define Word)

It can be used to define data like **WORD** (2 bytes).

Format:

Name of the Variable   *DW*   *Initial values*

Example:

```
SUM                         DW   4589
```

4. DD (Define Double Word)

It can be used to define data like **DWORD** (4 bytes).

Format:

Name of the Variable   *DD*   *Initial values*

Example:

```
NUMBER                      DD   12345678
```

5. DQ (Define Quad Word)

It can be used to define data like **QWORD** (8 bytes).

Format:

Name of the Variable   *DQ*   *Initial values*

Example:

```
TABLE                       DQ   1234567812345678
```

6. DT (Define Ten Bytes)

It can be used to define data like **TBYTE** (10 bytes).

Format:

Name of the Variable   *DT*   *Initial values*

Example:

```
AMOUNT                      DT    12345678123456781234
```

7. END (End of program)

It marks the end of a program module and, optionally, sets the program entry point to *address*.

Format:

END *[[address]]*

Example:

```
END label
```

8. ENDP (End Procedure)

It marks the end of procedure.

name previously begun with

PROC.

Format:

*name***ENDP**

Example:

```
CONTROL PROC FAR
.
.
.
CONTROL ENDP
```

9. ENDM (End Macro)

It terminates a macro or repeat

block.

Format:

ENDM

Example:

```
CODE    MACRO
        .
        .
        .
        ENDM
```

10. ENDS (End of Segment)

It marks the end of segment, structure, or union *name* previously begun with SEGMENT, STRUCT, UNION, or a simplified segment directive.

Format:

name **ENDS**

Example:

```
CODE    SEGMENT
        .
        .
        .
CODE    ENDS
```

11. EQU (Equate)

It assigns numeric value of *expression or text* to *name*. The *name* cannot be redefined
later.

Format:

```
name        EQU   expression
name        EQU   <text>
```

Example:

```
CLEAR_CARRY       EQU   CLC
```

12. EVEN (Align on Even memory Address)

The EVEN directive tells the assembler to increment the location counter to the next even address if it is not already at an even address.

Format:

```
EVEN
```

Example:

```
SALES            DB
EVEN
DATA_ARRAY DW    100   DUP (?)
```

13. EXTRN (External)

It is used to tell the assembler that the names or labels following the directive are in some other assembly module. This directive defines one or more external variables, procedures, labels, or symbols called *name* whose type is *type*.

Format:

EXTRN *name* **:** *type*

Example:

```
EXTRN            MULTIPLIER :          WORD
```

UNIT-2

8086 SYSTEM BUS STRUCTURE

SIGNALS OF 8086

Address / Data Bus (AD_{15}–AD_0)

The multiplexed Address/ Data bus acts as address bus during the first part of machine cycle (T1) and data bus for the remaining part of the machine cycle.

Address/Status (A_{19}/S_6, A_{18}/S_5, A_{17}/S_4, A_{16}/S_3)

During T1 these are the four most significant address lines for memory operations. During I/O operations these lines are LOW. During memory and I/O operations, status information is available on these lines during T2, T3, T_{WAIT}, T4.

Bus High Enable/Status (BHE /S_7)

During T1 the bus high enable signal ($\overline{BHE}$) should be used to enable data onto the most significant half of the data bus, pins D_{15}±D_8.
$\overline{BHE}$ is LOW during T1 for read, write, and interrupt acknowledge cycles when a byte is to be transferred on the high portion of the bus. The S_7 status information is available during T2, T3, and T4.

Read (RD)

This signal is used to read data from memory or I/O device which reside on the 8086 local bus.

Ready

If this signal is low the 8086 enters into WAIT state. The READY signal from memory/ IO is synchronized by the 8284A clock generator to form READY. This signal is active HIGH.

Interrupt Request (INTR)

It is a level triggered maskable interrupt request. A subroutine is vectored via an interrupt vector lookup table located in system memory. It can be internally masked by software resetting the interrupt enable bit. INTR is internally synchronized. This signal is active HIGH.

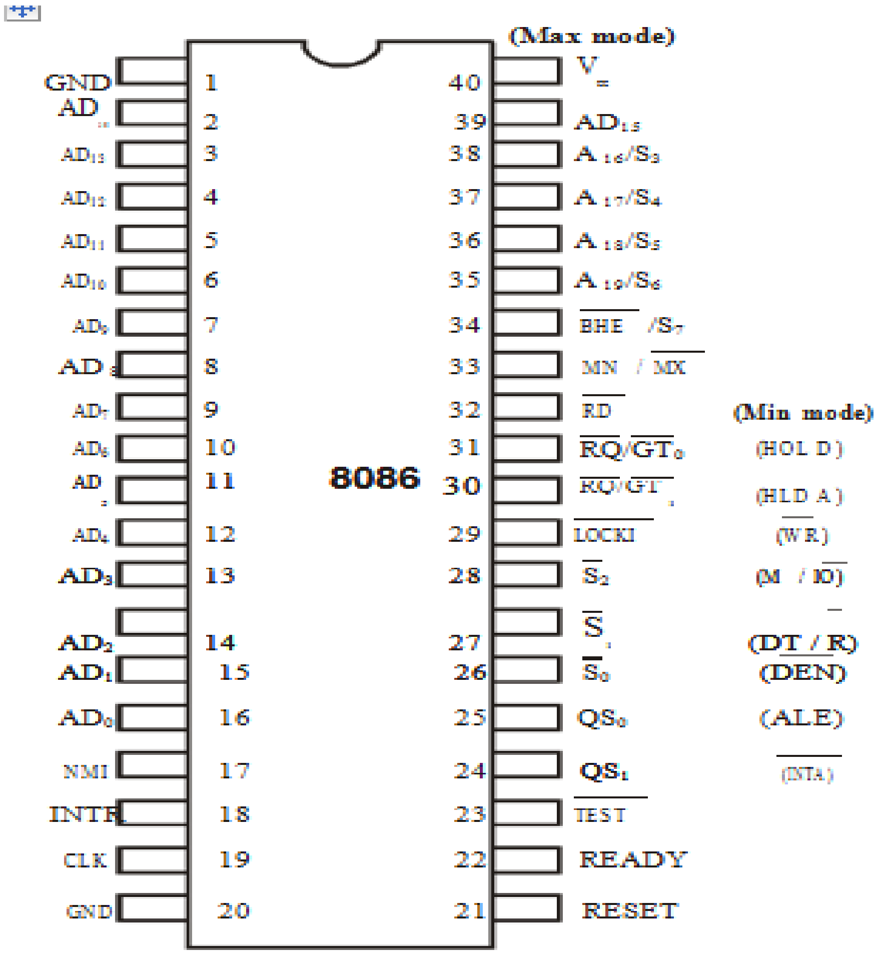
$\overline{TEST}$ - This input is examined by the "Wait" instruction. If the TEST input is LOW execution continues, otherwise the processor waits in an ``Idle'' state. This input is synchronized internally during each clock cycle on the leading edge of CLK.

Non-Maskable Interrupt (NMI)

It is an edge triggered input which causes a type 2 interrupt. NMI is not maskable internally by software. A transition from LOW to HIGH initiates the interrupt at the end of the current instruction.

Reset

This signal is used to reset the 8086. It causes the processor to immediately terminate its present activity. The signal must be active HIGH for at least four clock cycles. It restarts execution when RESET returns LOW.



Clock (CLK)

This signal provides the basic timing for the processor and bus controller. The clock frequency may be 5 MHz or 8 MHz or 10 MHz depending on the version of 8086.

Vcc

It is a +5V power supply pin.

Ground (GND)

Two pins (1 and 20) are connected to ground ie, 0 V power supply.

Minimum/Maximum (MN/ $\overline{\text{MX}}$)

This pin indicates what mode the processor is to operate in. The 8086 can be configured in either minimum mode or maximum mode using this pin.

Minimum Mode Signals

MEMORY / IO (M/ $\overline{\text{IO}}$)

It is used to distinguish a memory access from an I/O access. M = HIGH, I/O = LOW.

WRITE($\overline{\text{WR}}$)

It indicates that the processor is performing a write memory or write I/O cycle, depending on the state of the M/ $\overline{\text{IO}}$ signal.

Interrupt Acknowledge ($\overline{\text{INTA}}$)

This signal indicates recognition of an interrupt request. It is used as a read strobe for interrupt acknowledge cycles.

Address Latch Enable (ALE)

This signal is used to demultiplex the AD_0-AD_{15} into A_0-A_{15} and D_0-D_{15}. It is a HIGH pulse active during T1 of any bus cycle.

Data Transmit/Receive ($DT/\overline{R}$)

This signal desires to use a data bus transceiver (8286/8287). It is used to control the direction of data flow through the transceiver. A high signal on this pin indicates that 8086 is transmitting the data and low indicates that 8086 is receiving the data.

Data Enable($\overline{DEN}$)

This signal informs the transceivers (8286/8287) that the 8086 is ready to send or receive data.

Hold

This signal indicates that another master (DMA or processor) is requesting the host 8086 to handover the system bus.

Hold Acknowledge (HLDA)

On receiving HOLD signal 8086 outputs HLDA signal HIGH as an acknowledgement.

Maximum Mode Signals

Maximum mode operation differs from minimum mode in that some of the control signals must be externally generated. This requires additional circuitry, however, a chip -the 8288 bus controller- designed for this purpose is available.

Status ($\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$)

These three status signals indicate the type of machine cycle used.

Request/Grant ($\overline{RQ}/\overline{GT_0}$, $\overline{RQ}/\overline{GT_1}$)

These pins are used by other local bus masters to force RQ / GT_1 the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with RQ / GT_0 having higher priority than RQ / GT_1.

LOCK

This signal indicates that other system bus masters are not to gain control of the system bus while LOCK is active LOW. The LOCK signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction. This signal is active LOW.

Queue Status (QS_1, QS_0)

The queue status is valid during the CLK cycle after which the queue operation is performed. QS1 and QS0 provide status to allow external tracking of the internal 8086 instruction queue.

MULTIPROCESSOR CONFIGURATIONS

Multiprocessor

A multiprocessor system will have two or more processors that can execute instructions or perform operations simultaneously.

Basic Multiprocessor Configurations

1. Co processor configuration
2. Closely coupled configuration
3. Loosely coupled configuration

COPROCESSOR CONFIGURATION

In coprocessor configuration both the CPU (8086) and external processor (Math Co-processor 8087) share entire memory and I/O sub system. They also share same bus control logic and clock generator. 8086 is the master and 8087 is the slave.

Coprocessors add instructions to the instruction set. An instruction to be executed by the coprocessor is indicated by an escape (ESC) prefix or instruction.

1. The 8086 fetches the instructions.
2. The coprocessor monitors the instruction sequence and captures its own instructions.
3. The ESC is decoded by the CPU and coprocessor simultaneously.
4. The CPU computes the 20 bit address of memory operand and does a dummy read. The coprocessor captures the address of the data and obtains control of the bus to load or store as needed.
5. The coprocessor sends BUSY (high) to the TEST pin.
6. The CPU goes to the next instruction and if this is an 8086 instruction, the CPU and coprocessor execute in parallel.
7. If another coprocessor instruction occurs, the 8086 must wait until BUSY goes low ie, TEST pin become active. To implement this, a WAIT instruction is put in front of most 8087 instructions by the Assembler.
8. The WAIT instruction does the operations ie, wait until the TEST pin is active.

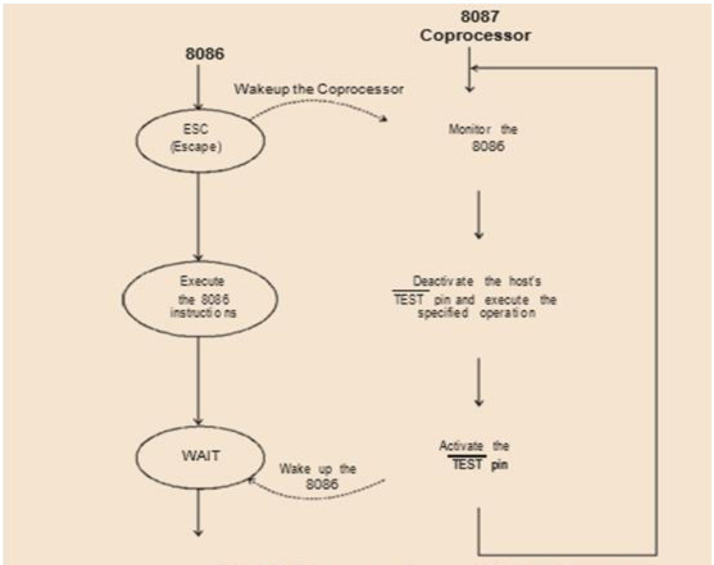

8087 Coprocessor
8086
Wakeup the Coprocessor
ESC (Escape)
Monitor the 8086
Execute the 8086 instructions
Deactivate the host's $\overline{\text{TEST}}$ pin and execute the specified operation
WAIT
Wake up the 8086
Activate the $\overline{\text{TEST}}$ pin

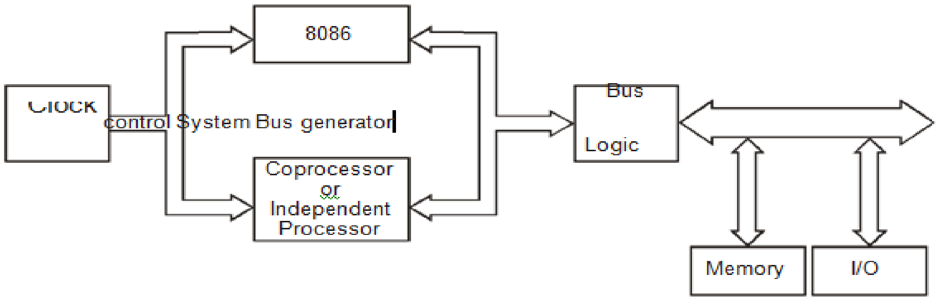
Fig. 2.17. Interaction between 8086 and Coprocessor

CLOSELY COUPLED CONFIGURATION

Coprocessor and closely coupled configurations are similar in that both the 8086 and the external processor (8089) share :

- Memory
- I/O system
- Bus and Bus control logic
- Clock generator

The main difference between coprocessor and closely coupled configuration is, no special instruction WAIT or ESC is used. The communication between 8086 and independent processor is done through memory space.


8086
Clock
control System Bus generator
Coprocessor or Independent Processor
Bus Logic
Memory
I/O

LOOSELY COUPLED CONFIGURATION

In loosely coupled configuration a number of modules of 8086 can be interfaced through a common system bus to work as a multiprocessor system. Each module in the loosely coupled configuration is an independent microprocessor based system with its own clock source, and its own memory and I/O devices interfaced through a local bus.

Advantages

1. Better system throughput by having more than one processor.
2. The system can be expanded in modular form. Each processor is an independent unit and normally on a separate PC board. One can be added or removed without affecting the others in the system.
3. A failure in one module normally does not affect the breakdown of the entire system and faulty module can be easily detected and replaced.

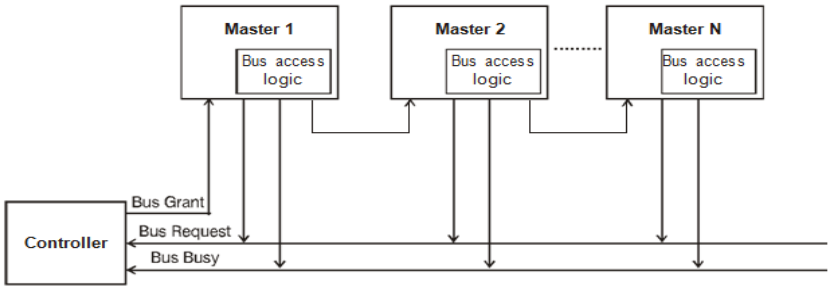
Bus allocation schemes:

i. Daisy chaining
ii. Polling method
iii. Independent Priority

Daisy Chaining

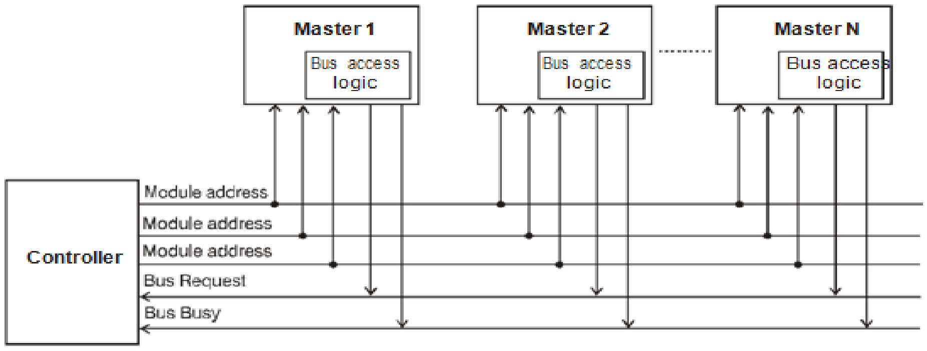
In daisy chaining method all masters make use of the same line for bus request. In response to a bus request, the controller sends a bus grant if the bus is free. The bus grant signal serially propagates through each master until it encounters the first one that is requesting access to the bus.

But failure of any one master causes the whole system to fail and arbitration is slow due to the propogation delay of bus grant signal is proportional to the number of masters.



Polling Method

In polling method, the controller sends address of device to grant bus access. The number of address lines required is depend on the number of masters connected in the system. For example, if three masters are connected in the system, one address line is required. In response to a bus request, controller generates a sequence of master addresses. When the requesting master recognizes the address, it activates the busy line and begins to use the bus. The priority can be changed by altering the polling sequence stored in the controller. If one module fails entire system does not fail.



Independent Priority

In the independent priority scheme each master has a separate pair of bus request (BRQ) and bus grant (BGR) lines and each pair has a priority assigned to it. The built in priority decoder within the controller selects the highest priority request and asserts the corresponding bus grant signal. Synchronization of clocks must be performed once a master is recognized. Master will receive a common clock from one side and pass it to the controller which will derive a clock for transfer.

MAXIMUM MODE CONFIGURATION OF 8086

Maximum Mode

For maximum mode of operation MN/$\overline{MX}$ pin is grounded.

Pins 24 to 31 have the following functions:

Pins 24 & 25	QS1, QS0 (Instruction Queue Status)
Pins 26, 27 & 28	S0, S1,S2(Status signals)
Pin 29	Lock

Pin 30 & 31 : $\overline{RQ}$ / $\overline{GT_1}$, $\overline{RQ}$ / $\overline{GT_0}$ (Request / Grant)

Request / Grant lines are used for local bus priority control. Other processors ask the CPU through these lines to release the local bus.

The important signals are :

$\overline{MRDC}$ - Memory Read Command

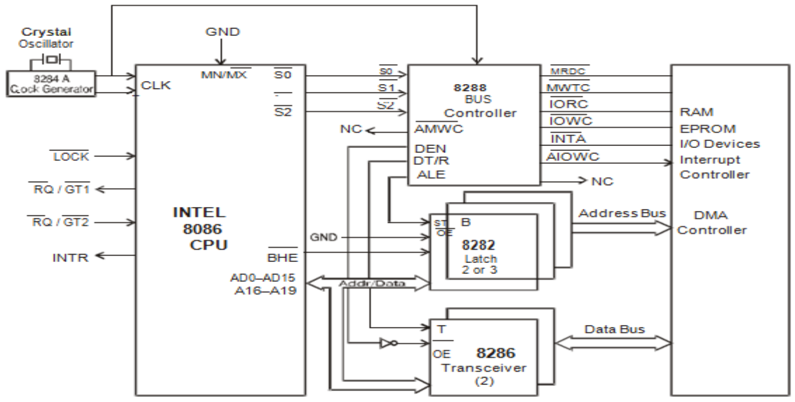
$\overline{MWTC}$ - Memory Write Command

IORC - I/O Read Command

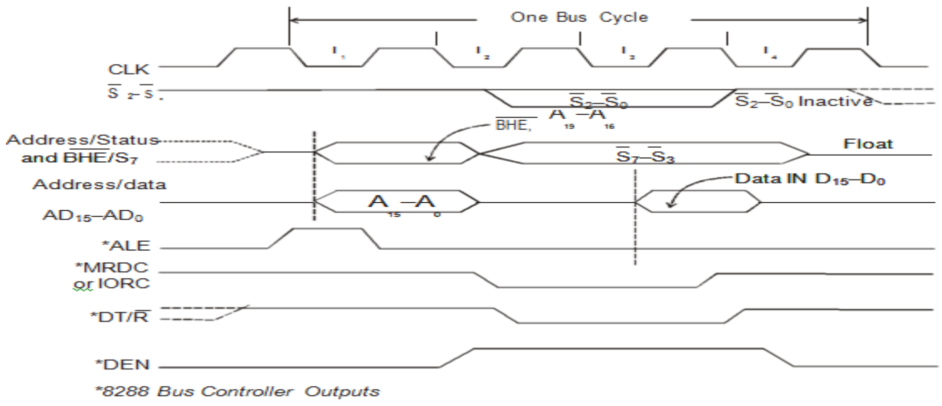
IOWC - I/O Write Command

AMWC - Advanced Memory Write Command

AIOWC - Advanced I/O Write Command



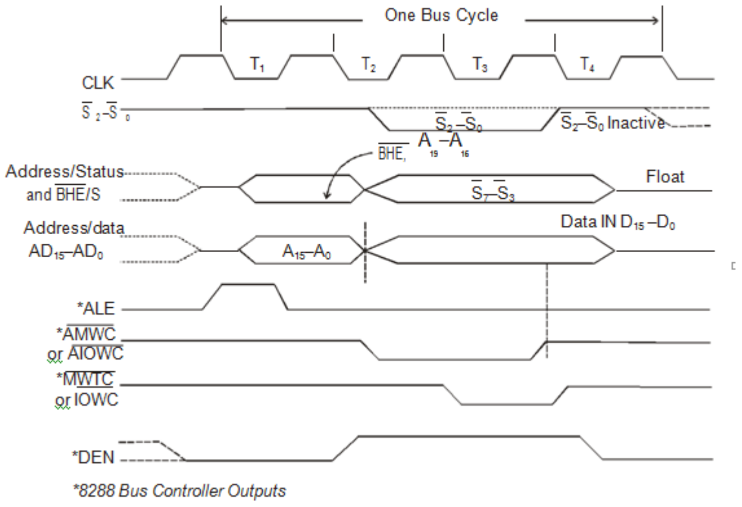
Read cycle



Write cycle

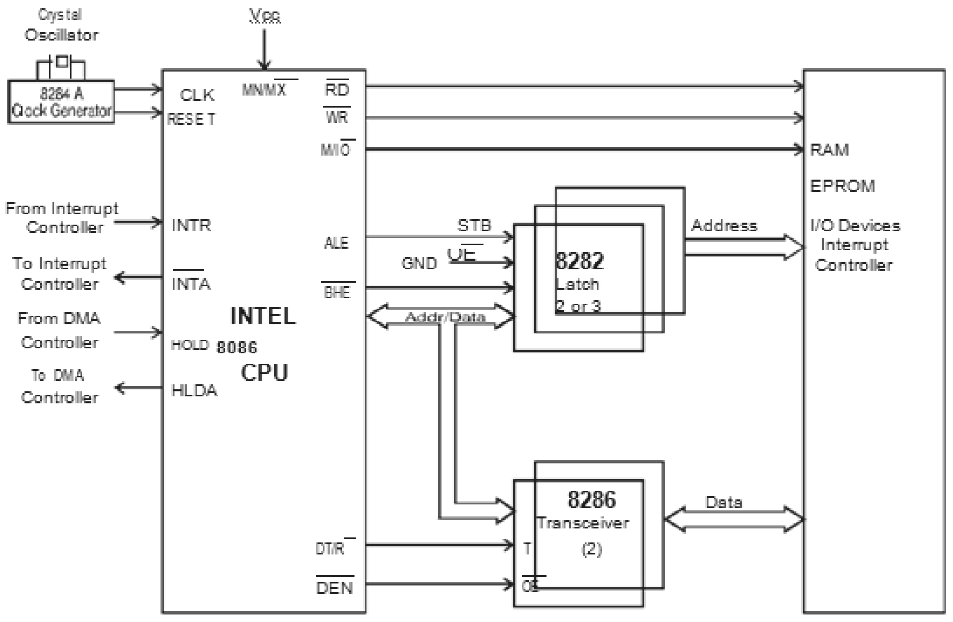
- The status signals S_2 , S_1 and S_0 are set at the begining of bus cycle. When $S_2 = S_1 = S_0 = 1$, (inactive-passive), the bus controller will output a pulse on its **ALE** and apply a required signal to its DT/ R pin at **T**$_1$.
- At T_2, the bus controller will set DEN = 1, therefore transceiver is enabled. For an input, bus controller will activate MRDC or IORC . These signals are activated until T_4. For an output, it will activate AMWC or AIOWC . These signals are activated from T_2 to T_4 and MWTC or IOWC is activated from **T**$_3$ to **T**$_4$.

- The status signals S_2 , S_1 and S_0 are remain active during **T**$_1$ and **T**$_2$ and become inactive during T_3 and T_4.



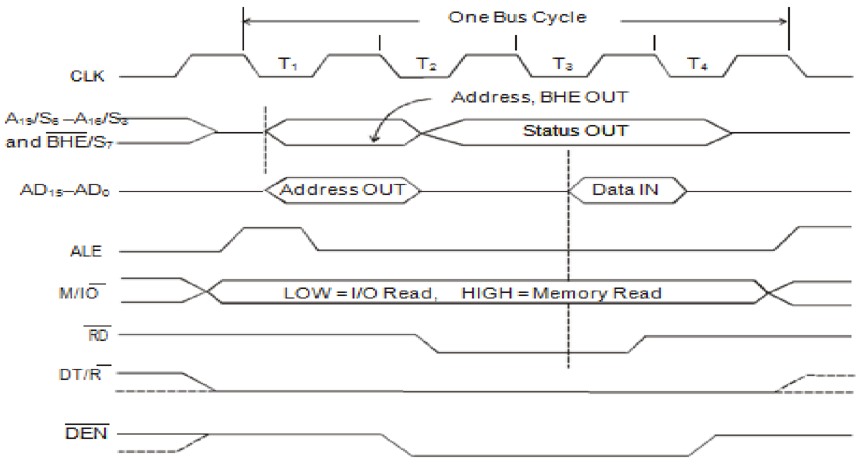
MINIMUM MODE OF OPERATION

For minimum mode of operation MN/ MX is connected to V_{CC} (+5 volts). All control signals for controlling memory and I/O devices are generated inside the 8086 microprocessor. In this mode , peripheral devices can be used with the microprocessor without any special consideration.



Read Operation :

- The Read cycle begins in T_1 with the assertion of the address latch enable (ALE) signal and also M/ IO signal. During the negative going edge of this signal, the valid address is latched on this bus.
- The BHE and **A_0** Signals address low, high or both bytes. From T_1 to T_4 the M/ IO signal indicates memory or I/O operation. At T_2, the address is removed from the bus and is sent to the output. The bus is then tristated.
- The RD signal is activated in **T_2**. This signal causes the addressed device to enable its data bus drivers.
- After RD goes low, the valid data is available on the data bus. After the data is accepted by the processor, RD is raised high at the beginning of **T_4**.
- At T_2 DEN is lowered to enable transceiver. At **T_4** DEN is raised to disable the transceiver.



Write Operation

- The Write cycle begins in T_1 with the assertion of the ALE signal. The M/ IO signal is asserted to indicate a memory or I/O operation.
- At T_2, after sending the address in T_1, the processor sends the data to be written to the addressed location.
- The data on the bus remains until middle of T_4 state.
- The WR signal becomes active at the beginning of **T_2**.

- The $\overline{BHE}$ and **A_0** signals are used to select the proper type of memory or I/O to be read or written.
- At T_2 DEN is lowered to enable transceiver. At **T_4** it is raised to disable the transceiver.

SYSTEM BUS STRUCTURE

Microprocessor is processing device of every computing device. It needs to communicate with outer world. It needs to communicate with input devices to get data, it needs to communicate with memory to process data according to instructions written in memory and finally it needs to communicate with output devices to display the output on output devices. To communicate with external world, microprocessor make use of buses.

System bus is a single computer bus that connects the major components of a computer system. It consists of data bus, address bus and control bus.

(i) Data Bus

- It is used for the exchange of data between the processor, memory and peripherals.
- It is bi-directional so that it allows data flow in both directions.
- The width of the data bus can differ for every microprocessor.
- When the microprocessor issues the address of the instruction, it gets back the instruction through the data bus. When it issues the address of the data, it loads the data through the data bus.

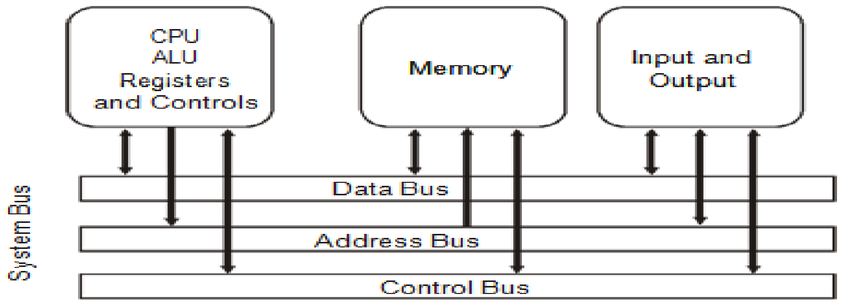
(ii) Address Bus

- The address bus contains the connections between the microprocessor and memory or output devices that carry the signals relating to the addresses which the CPU is processing at that time, such as the locations that the CPU is reading from or writing to.
- It is unidirectional.
- The width of the address bus corresponds to the maximum addressing capacity of the bus, or the largest address within memory that the bus can work with.

Maximum address capacity = 2^n (n=address lines). Address bus may be multiplexed with data bus.

(iii) Control Bus

- The control bus carries the signals relating to the control and coordination of the various activities across the computer, which can be sent from the control unit within the CPU.
- Microprocessor uses control bus to process data, that is what to do with the selected memory location. This is a dedicated bus, because all timing signals are generated according to control signal. Some control signals are Read, Write and Opcode fetch etc.



I/O PROGRAMMING

I/O programming discuss the ways in which information can be transferred between input-output devices or mass storage devices and the CPU or memory. The three modes of transfer of device data,commands and status are,

(i) Programmed I/O

(ii) Interrupt driven I/O

(iii) DMA transfer

PROGRAMMED I /O

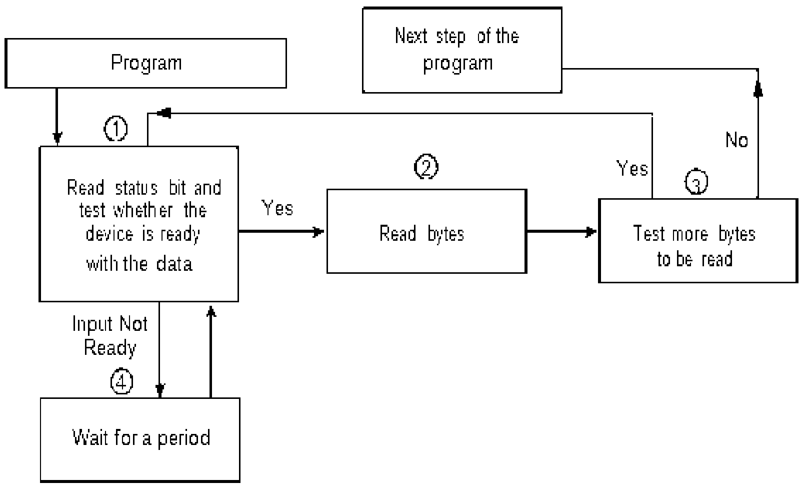
Programmed I/O consists of continually examining the status of an interface and performing an I/O operation with the interface when its status indicates that it has data to be input or its data-out buffer register is ready to receive data from the CPU.

(i) Read input in programmed I/O mode

Each input is read after first testing whether the device is ready with the input (a state reflected by a bit in a status register) or whether the device input buffer is not empty. The program waits for the ready status by repeatedly testing the status bit and till all targeted bytes are read from the input device. The program is in busy state only after the device gets ready else in wait state.

(ii) Output write in programmed I/O mode

Each output written after first testing whether the device is ready to accept the bytes at its output register or output buffer is empty. The program waits for the ready status by repeatedly testing the status bit and till all the targeted bytes are written to the device. The program in busy state only after the device gets ready else wait state.


Program
Next step of the program
①
Read status bit and test whether the device is ready with the data
Yes
②
Read bytes
Yes
③
No
Test more bytes to be read
Input Not Ready
④
Wait for a period

Interrupt driven I /O

There are several ways of combining with interrupt I/O, some involving only software, some only hardware, and some a combination of the two. They are,

i.) Polling
ii.) Daisy chaining
iii.) Interrupt priority management hardware

Polling

Polling is the most common and simplest method of I/O control. It requires no special hardware and all I/O transfers are controlled by the CPU programme. Polling is a synchronous mechanism, by which devices are serviced in sequential order.

Polling is constantly testing a port to see if data is available. i.e, the CPU polls (asks) the port if it has data available or if it is capable of accepting data. Polling notifies the part of the computer containing the I/O interface that a device is ready to be read or otherwise handled but does not indicate which device. The interrupt controller must poll (send a signal out to) each device to determine which one made the request.

The polling technique has the following limitations:

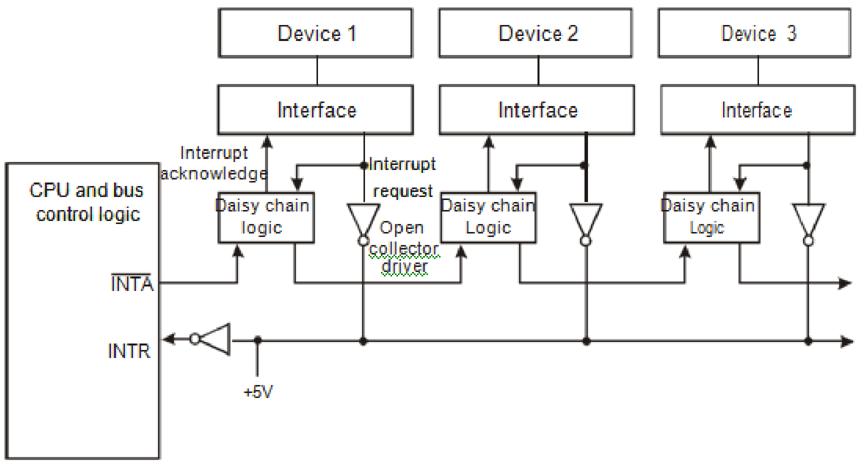
1. It is wasteful of the processors time, as it needlessly checks the status of all devices all the time.
2. It is inherently slow, as it checks the status of all I/O devices before it comes back to check any given once again.
3. When fast devices are connected to a system, polling may simply not be fast enough to satisfy the minimum service requirements.

4. Priority of the device is determined by the order in the polling loop, but it is possible to change it via software.

Daisy chaining

It is a simple hardware means of attaining a priority scheme. It consists of associating
a logic circuit with each interface and passing the interrupt acknowledge signal through these circuits as shown in Fig.2.14. The priority of an interface is determined by its position on the daisy chain. The closer it is to the CPU the higher its priority.

This is significantly faster than a pure software approach. A daisy chain is used to identify the device requesting service. Daisy chaining is used for level sensitive interrupts, which act like a wired 'OR' gate. Any requesting device can take the interrupt line low, and keep it asserted low until it is serviced.



Interrupt priority management hardware

A more flexible hardware priority arrangement can be held by designing a programmable interrupt priority management circuit and including it in the bus control logic. This is the fastest system. The duty is placed on the requesting device to request the interrupt, and identify itself. The identity could be a branching address for the desired interrupt-handling routine.

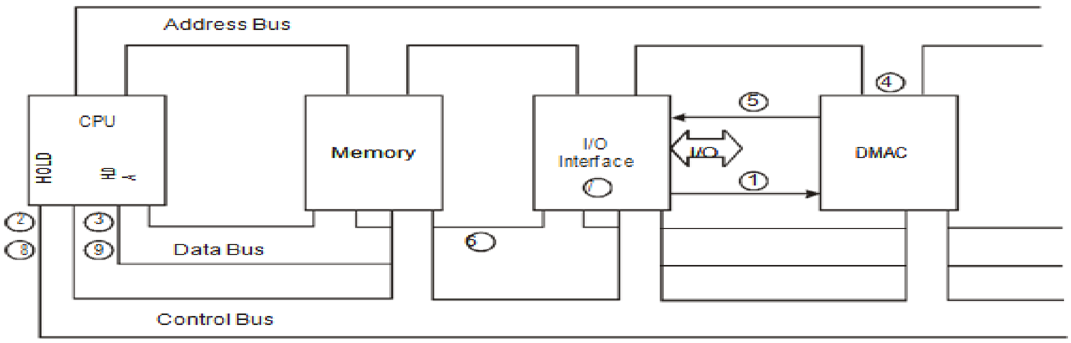
If the device just supplies an identification number, this can be used in conjunction with a lookup table to determine the address of the required service routine. Response time is best when the device requesting service also supplies a branching address.

Direct Memory Access Block Transfer

A DMA controller allows devices to transfer data to or from the system memory without the intervention of the processor.

During any given bus cycle, one of the system components connected to the system bus is given control of the bus. This component is said to be the master during that cycle and the component it is communicating with is said to be the slave. The CPU with its bus control logic is normally the master, but other specially designed components can gain control of the bus by sending a bus request to the CPU. After the current bus cycle is completed the CPU will return a bus grant signal and the component sending the request will become the master.

Taking control of the bus for a bus cycle is called cycle stealing. Just like the bus control logic, a master must be capable of placing addresses on the address bus and directing the bus activity during a bus cycle.



UNIT-3

I/O INTERFACING

INTERFACING LED DISPLAYS TO8086

LED's are most important display device. The I/O device cannot be connected directly to the microprocessor, it can be connected through a latch IC.

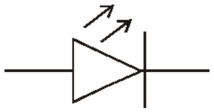
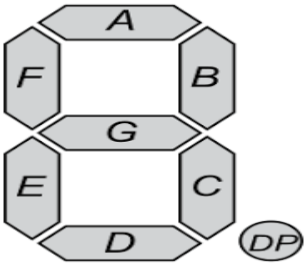


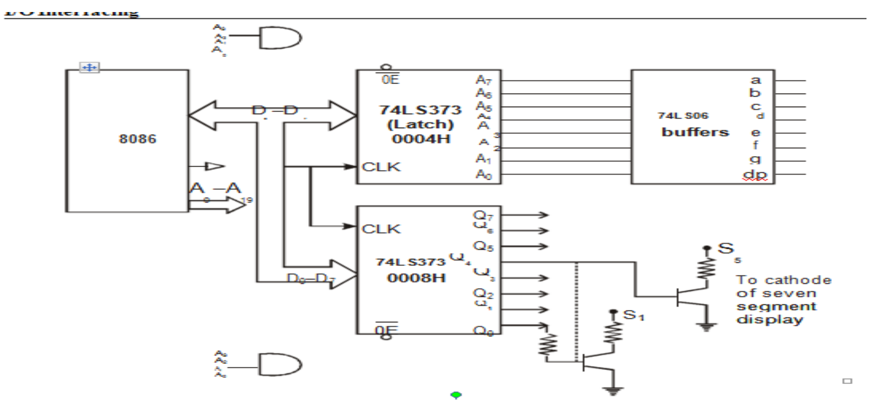
Fig. A single LED

Seven segment LED display

The most important application of LED is to display alphanumeric characters. The seven LED's are arranged as a segment and selectively lighted up to display the alphanumeric characters.



The LED modules can be used in the common anode or common cathode configuration. The cathode is grounded in common cathode type and give '1' to light up the particular segments. By selectively lighting up the segments, the alphanumeric characters are displayed.



Assembly language program for LED

```
            MOV AX, 2000H   : Initialize pointer
            MOV DS,
            AX              : code table DS : BX
            MOV BX, 0000H
NEXT:   MOV AL, 00H         : get 1st number from table
            MOV DH,
            AL
            MOV CL, 05H     : count for display
            MOV DL, E1H     : selection code for 1st display
AGAIN :     XLAT
            OUT 04H,        : out the code for the first number to port
            AL              04H
            MOV AL,
            DL              : Get to enabled display code
            OUT 08H,
            AL              : select 1st display
            ROL DL          : decide code for selecting next
            INC DH          : display the next number
            MOV AL, DH      : get next number to be displayed
            LOOP AGAIN      : Repeat five times
            JMP NEXT        : continue the procedure
```

ALP for rotating the word 'HELLO' from right to left displayed on LED.

```
MESG DB 'H', 'E', 'L','L','O'
REEP :MOV AL, 18 H          ; command to shift display left.
CALL COMMAND
CALL DELAY
LEA BX, MESG                ; point to the characters
MOV CX, 5                   ; CX = number of characters
REPEAT : MOV AL, [BX]       ; move to AL the character
CALL DAT
CALL DELAY_ROLL
INC BX
LOOP REPEAT                 ; repeat this until CX = 0
JMP REEP                    ; infinite loop for continuous rotation
```

The DELAY_ROLL procedure is used for deciding the rate at which rotation occurs.

DIGITAL TO ANALOG (D/A) INTERFACE

The digital to analog converters (DAC) convert binary numbers into their analog equivalent voltages or currents. Several techniques are employed for digital to analog conversion.

i. Weighted resistor network
ii. R-2R ladder network
iii. Current output D/A converter

The DAC find applications in areas like digitally controlled gains, motor speed control, programmable gain amplifiers, digital voltmeters, panel meters, etc. DAC have many applications besides those where they are used with a microcomputer. In a compact disk
audio player for example a 14-or16-bit DAC is used to convert the binary data read off the disk by a laser to an analog audio signal. Most speech synthesizer integrated circuits contain a DAC to convert stored binary data words into analog audio signals.

Resolution: It is a change in analog output for one LSB change in digital input.

$(1/2^n)*V_{ref}$

1/256*5 V=39.06 mV (since *n*=8 for 8-bit DAC)

Settling time: It is the time required for the DAC to settle for a full scale code change.

DAC 0800 8-bit Digital to Analog converter

DAC0800 is a monolithic 8-bit DAC manufactured by National semiconductor.
It has settling time around 100ms.
It can operate on a range of power supply voltage 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.

Resolution of the DAC is 39.06mV.

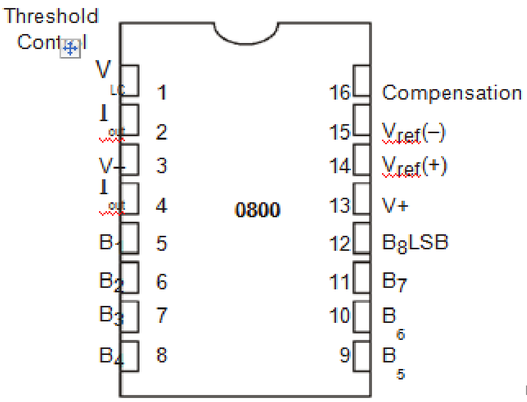
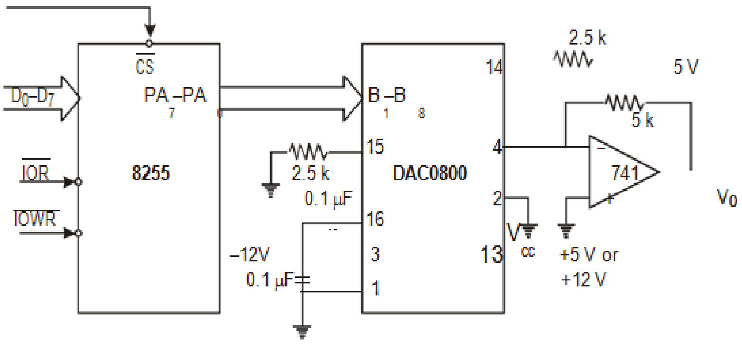


Fig.3.18. Pin Diagram of DAC 0800

Interfacing of DAC0800 with 8086



The V_{ref+} should be tied to +5 V to generate a wave of +5V amplitude. The required frequency of the output is 500 Hz, i.e. the period is 2 ms. Assuming the wave to be generated is symmetric, the waveform will rise for 1 ms and fall for 1 ms.

```
ASSUME   CS : CODE
CODE     SEGMENT
START :  MOV AL,80 H      ; Initialise 8255 ports
         OUT CWR, AL      ; suitably.
         MOV AL, 00H      ; Start rising ramp from
                          ; 0V by sending 00H to
BACK :   OUT Port A, AL   DAC.
         INC AL           ; Increment ramp till 5V
         CMP AL, FFH      ; compare with FFH
         JB BACK          ; If it is FFH then
                          ; Output it and start the
BACK1 :  OUT Port A, AL   falling
         DEC AL           ; ramp by decrementing the
         CMP AL, 00       ; counter till it reaches
         JA BACK1         ; zero. Then start again
         JMP BACK         ; for the next cycle.
CODE     ENDS
         END START
```

ANALOG TO DIGITAL INTERFACE

The digital equipment cannot directly accept the analog signals such as voltage or current. These signals are need to be converted into digital form by using suitable device called Analog to Digital converter (ADC). An ADC is essential in a microprocessor based system as the microprocessor can only handle digital data, though the real-world signals are in analog form.

The resolution of an ADC refers to the number of bits in the output binary word. An 8-bit converter for example has a resolution of 1 part in 256. An important specification for an ADC is its conversion time. This is simply the time it takes the converter to produce a valid output binary code for an applied input voltage. The high speed converter has a short conversion time.

ADC is treated as an input device by the microprocessor that sends an initialising signal to the ADC to start the analog to digital data conversation process. The start of conversion signal is a pulse of a specific duration. The process of analog to digital conversion is a slow process, and the microprocessor has to wait for the digital data till the conversion is over. After the conversion is over, the ADC sends end of conversion (EOC) signal to inform the microprocessor that the conversion is over and the result is ready at the output buffer of the ADC. These tasks of issuing an SOC pulse to ADC, reading EOC signal from the ADC and reading the digital output of the ADC are carried out by the CPU using 8255 I/O ports. Mostly, 8255 is used for interfacing the ADC with a microprocessor.

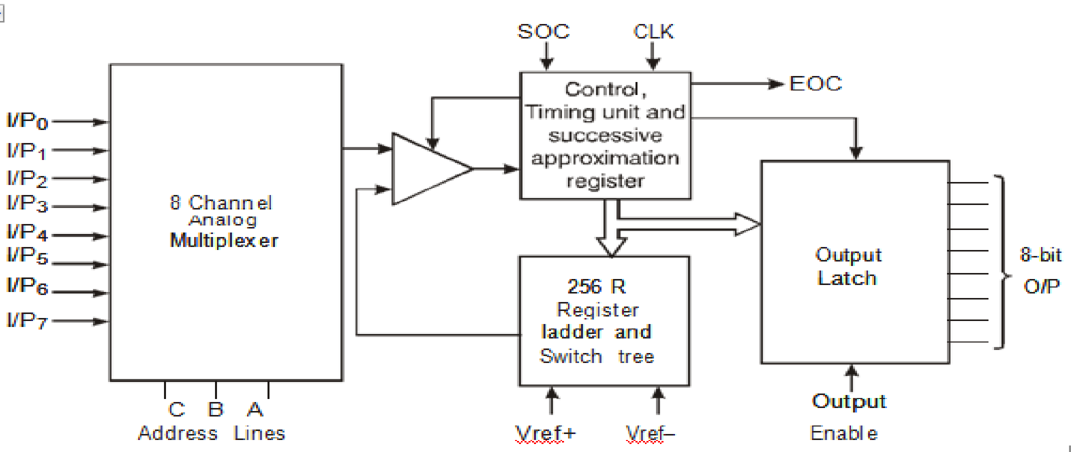
The time taken by the ADC from SOC to the EOC signal is called as the conversion delay of the ADC. Parallel converter or flash converter, successive approximation and dual slope integration techniques are the most popular techniques used in the integrated ADC chips. The ADC interfacing consists of the following steps.

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

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

ADC 0808/0809

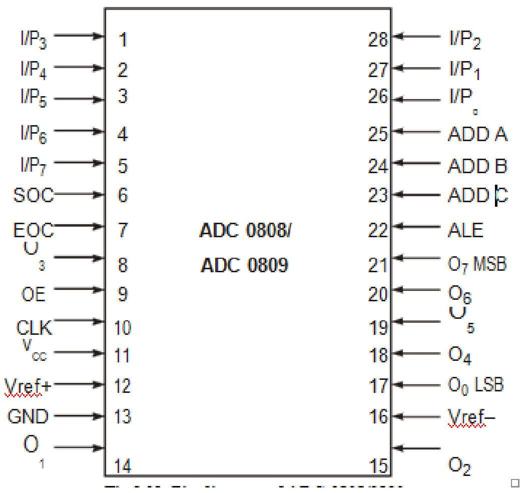
The analog to digital converter chips 0808 and 0809 are 8-bit CMOS, successive approximation converters. Successive approximation technique is one of the fast techniques for analog to digital conversion. The conversion delay is 100s at a clock frequency of 640 kHz, which is quite low as compared to other converters. These converters do not need any external zero or full scale adjustments as they are already taken care of by internal circuits. These converters internally have a 3:8 analog multiplexer so that at a time eight different analog inputs can be connected to the chips. Out of these eight inputs only one can be selected for conversion by using address lines ADD A, ADD B and ADD C, as shown.



SOC
CLK
EOC
I/P0
I/P1
I/P2
I/P3
I/P4
I/P5
I/P6
I/P7
8 Channel Analog Multiplexer
Control, Timing unit and successive approximation register
256 R Register ladder and Switch tree
Output Latch
8-bit O/P
C B A
Address Lines
Vref+
Vref–
Output Enable

These are unipolar analog to digital converters, i.e. they are able to convert only positive analog input voltages to their digital equivalents. These chips do not contain any internal sample and hold circuit. If one needs a sample and hold circuit for the conversion of fast, signals into equivalent digital quantities, it has to be externally connected at each of the analog inputs.

PIN DIAGRAM :



INTERRUPT CONTROLLER

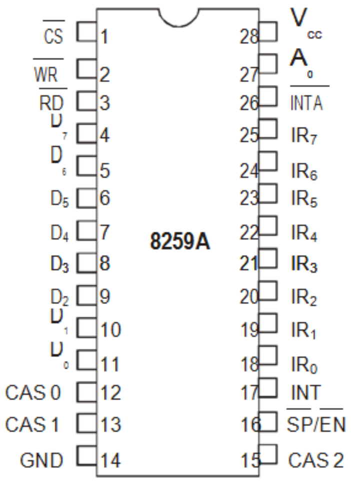
1. The 8259A programmable interrupt controller extends the hardware interrupt facility provided in a microprocessor.
2. It manages up to 8 vectored priority interrupts for a processor.
3. It has built-in features for expandability to other 8259A's (up to 64 vectored priority interrupts).
4. It is programmed by the system's software as an I/O peripheral.

Features of 8259 A

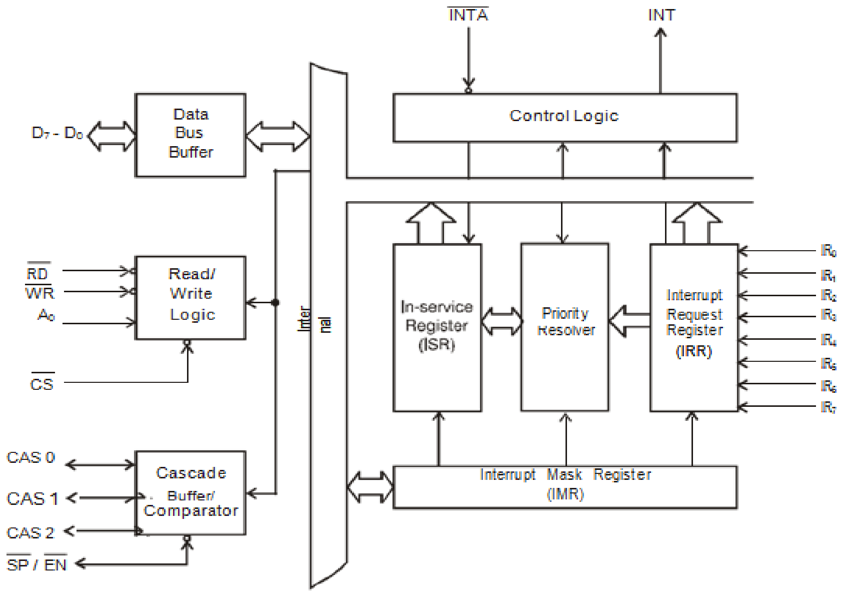
- It can manage 8 priority interrupts.
- By cascading 8259s it is possible to get 64 priority interrupts.
- It can be programmed to accept either the level triggered or the edge triggered interrupt request.

- Reading of interrupt request register (IRR) and in-service register (ISR) through data bus.

Pin Diagram of 8259A:



Block Diagram of 8259A



8259 Consists of the following blocks :

- Control logic unit
- In-Service Register (ISR)
- Priority Resolver
- Read/Write logic
- Data Bus buffer
- Cascade buffer
- Interrupt Request Register (IRR)
- Interrupt Mask Register (IMR).

1. Interrupt Request Register (IRR)

The interrupts at the IR input lines are handled by two registers in cascade, the Interrupt Request Register (IRR) and the In-Service Register (ISR). The IRR is used to store all the interrupt levels which are requesting service.

2. In-Service Register (ISR)

The ISR is used to store all the interrupt levels which are being serviced.

3. Priority Resolver

This logic block determines the priorities of the bits set in the IRR. The highest priority is selected and strobed into the corresponding bit of the ISR during INTA pulse.

4. Interrupt Mask Register (IMR)

The IMR stores the bits which mask the interrupt lines to be masked. The IMR operates on the IRR. Masking of a higher priority input will not affect the interrupt request lines of lower quality.

5. Data Bus Buffer

This 3-state, bidirectional 8-bit buffer is used to interface the 8259A to the system data bus. Control words and status information are transferred through the Data Bus Buffer.

6. Read/Write Control Logic

The function of this block is to accept output commands from the Microprocessor. It contains the Initialization Command Word (ICW) registers and Operation Command Word (OCW) registers which store the various control formats for device operation. This function block also allows the status of the 8259A to be transferred onto the data bus.

7. Cascade Buffer/Comparator

This block is used to expand the number of interrupt levels by cascading two or more 8259s.

This function block stores and compares the IDs of all 8259A's used in the system. The associated three I/O pins (CAS0-2) are outputs when the 8259A is used as a master and are inputs when the 8259A is used as a slave. As a master, the 8259A sends the ID of the interrupting slave device onto the CAS0±2 lines.

8. Control Logic

This block has two pins INT and INTA.

INT (Interrupt)

This output goes directly to the CPU interrupt input. The voltage level on this line is designed to be fully compatible with the 8080A, 8085A and 8086 input levels.

INTA (Interrupt Acknowledge)

INTA pulses will cause the 8259A to release vectoring information onto the data bus.

The format of this data depends on the system mode of the 8259A.

The various modes of operation of the 8259 are:

- Fully nested mode
- Special fully nested mode.
- Special mask mode
- Buffered mode
- Poll command mode
- Cascade mode with master or slave selection
- Automatic end-of-interrupt mode

1. Fully Nested Mode

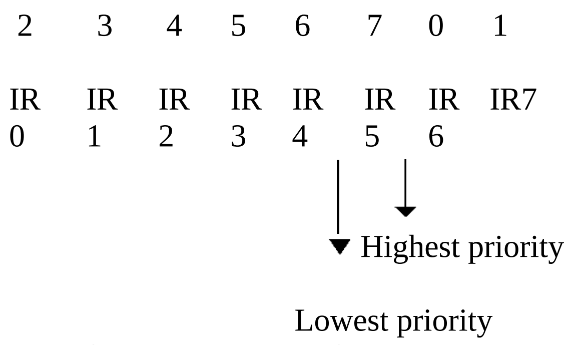
This mode is entered after initialization unless another mode is programmed. The interrupt requests are ordered in priority from 0 through 7 (0 highest). When an interrupt is acknowledged the highest priority request is determined and its vector placed on the bus.

2. Automatic End of Interrupt (AEOI) Mode

If AEOI = 1 in ICW4, then the 8259A will operate in AEOI mode continuously until reprogrammed by ICW4. In this mode the 8259A will automatically perform a non-specific EOI operation at the trailing edge of the last interrupt acknowledge pulse.

3. Automatic Rotation (Equal Priority Devices)

In some applications there are a number of interrupting devices of equal priority. In this mode a device after being serviced, receives the lowest priority. So a device requesting an interrupt will have to wait. In the worst case until each of 7 other devices are serviced at most once.



4. Specific Rotation (Specific Priority)

The programmer can change priorities by programming the bottom priority and thus fixing all other priorities; i.e., if IR4 is programmed as the lowest priority device, then IR5 will have the highest one.

5. Special Mask Mode

In the special mask mode, when a mask bit is set in OCW1, it inhibits further interrupts at that level and enables interrupts from all other levels (lower as well as higher) that are not masked. Thus, any interrupts may be selectively enabled by loading the mask register.

6. Poll Command

In poll mode the INT output functions as it normally does. The microprocessor should ignore this output. This can be accomplished either by not connecting the INT output or by masking interrupts within the microprocessor, thereby disabling its interrupt input. Service to devices is achieved by software using a poll command.

7. Special Fully Nest Mode

This mode will be used in the case of a big system where cascading is used, and the priority has to be conserved within each slave. In this case the fully nested mode will be programmed to the master.

8. Buffered Mode

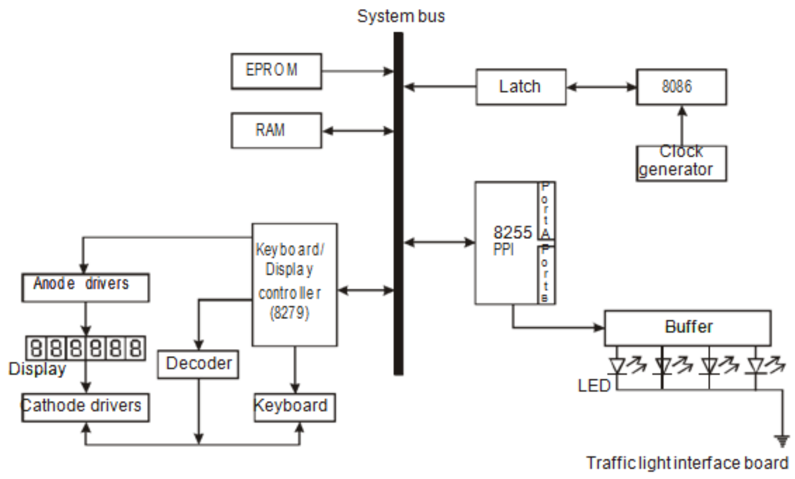
When the 8259A is used in a large system where bus driving buffers are required on the data bus and the cascading mode is used, there exists the problem of enabling buffers.

TRAFFIC LIGHT CONTROLLER

The traffic light control is implemented in road crossings using a microprocessor system by automatically switching ON/OFF of traffic lights in the desired sequence.

The traffic light system consists of EPROM for program storage, RAM. The keyboard is provided for manual control. Seven segment LED display is provided to display the direction of traffic.

The Red (stop), Yellow (wait) and Green (go) LED's are switched ON or OFF in a desired sequence using microprocessor system. The LED are interfaced to the bus through buffer and ports of 8255. The switching sequence is given in look up table. The microprocessor can give the codes for switching the lights for schedule-I and after a time delay, it output the code for next schedule.



Program to interface traffic light with 8086

```
MOV BX, LOOKUP
MOV CX, 0008H
MOV AL, BX
OUT DX, AL
INC BX
MOV AL, BX
MOV DX, PORT A
OUT DX, AL
INC BX
MOV AL, [BX]
MOV DX, PORT B
OUT DX, AL
CALL DELAY
INC BX
LOOP NEXT
JMP START
PUSH CX
MOV CX, 0005H
REPEAT: MOV DX, OFFFFH
DEC DX
JNZ LOOP2
LOOP REPEAT
POP CX
RET
LOOKUP :        80H ,21H, 09H, 10H, 00H (SOUTH)
                0CH, 09H, 80H, 00H (EAST)
                64H, 08H, 00H, 04H (NORTH)
                24H, 03H, 02H, 00H (WEST)
                END
```

TIMER 8254

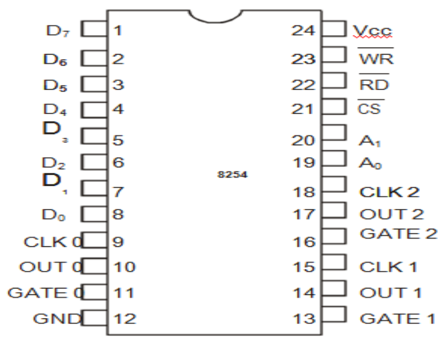
Features of 8254

- 8254 is a Timer/Counter
- It is designed to solve the common timing control problems in microcomputer system design.
- Compatible with all Intel and most other microprocessors It can be operated at count rates upto 10 MHz
- Six programmable counter modes and all modes are software programmable. Three independent 16-bit counters

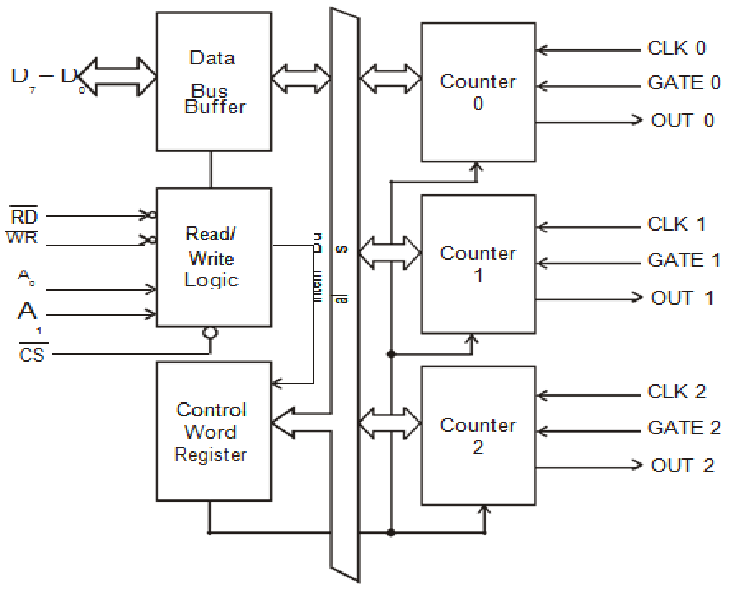
Applications of 8254

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

PIN DIAGRAM



BLOCK DIAGRAM



Data Bus Buffer

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

Read/Write Logic

- The Read/Write logic accepts inputs from the system bus and generates control signals for the other functional blocks of the 8254.
- A
  1 and A0 select one of the three counters or the control word register to be read from/written into.

Control Word Register

- The control word register is selected by the Read/Write logic when A_1, A_0=11.
- If the CPU then does a write operation to the 8254, the data is stored in the control word register and is interpreted as a control word used to define the operation of the counters.
- The control word register can only be written to; status information is available with the Read-Back command.

Counter 0, Counter 1, Counter 2

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

Operational Modes of 8254

The 8254 can operate in six operating modes. They are

Mode 0: Interrupt On Terminal Count
Mode 1: Hardware Retriggerable One-Shot
Mode 2: Rate Generator
Mode 3: Square Wave Mode
Mode 4: Software Triggered Strobe
Mode 5: Hardware Triggered Strobe

Mode 0: Interrupt On Terminal Count

- Mode 0 is typically used for event counting.
- After the control word is written, OUT is initially low, and will remain low until the counter reaches zero. OUT then goes high and remains high until a new count or a new Mode 0 control word is written into the counter.

GATE = 1 enables counting;
GATE = 0 disables counting. GATE has no effect on OUT.

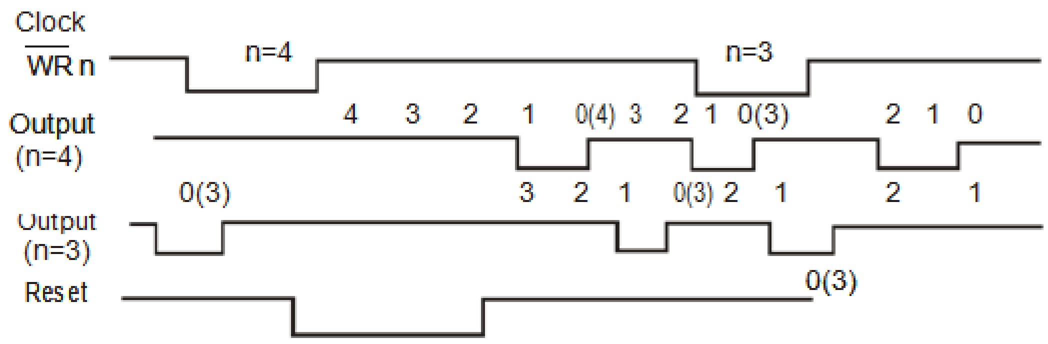
Mode 1: Hardware Retriggerable One-Shot

- OUT will be initially high. OUT will go low on the CLK pulse following a trigger to begin the one-shot pulse, and will remain low until the counter reaches zero.

- OUT will then go high and remain high until the CLK pulse after the next trigger. Thus generateing a one-shot pulse.After writing the control word and initial count, the counter is armed.

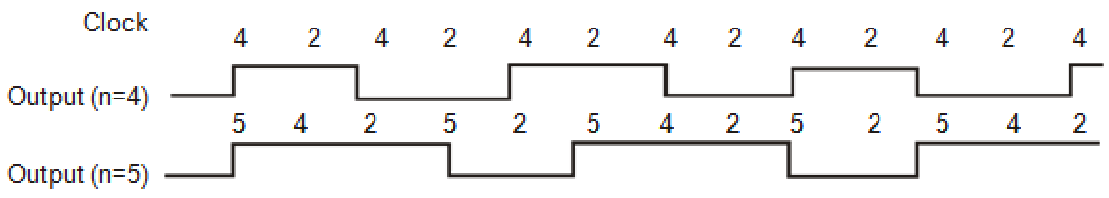
Mode 2: Rate Generator

- This mode functions like a divide-by-N counter.
- It is typically used to generate a real time clock interrupt.
- OUT will initially be high. When the initial count has decremented to 1, OUT goes low for one CLK pulse. OUT then goes high again, the counter reloads the initial count and the process is repeated.



Mode 3: Square Wave Mode

- Mode 3 is typically used for baud rate generation.
- Mode 3 is similar to Mode 2 except for the duty cycle of OUT. OUT will initially be high. When half the initial count has expired, OUT goes low for the remainder of the count.
- Mode 3 is periodic; the sequence above is repeated indefinitely. An initial count of N results in a square wave with a period of N CLK cycles.
- Mode 3 is implemented as follows:



Even counts:

OUT is initially high. The initial count is loaded on one CLK pulse and then is decremented by two on succeeding CLK pulses. When the count expires OUT changes value and the counter is reloaded with the initial count. The above process is repeated indefinitely.

Odd counts:

For odd counts, OUT will be high for (N +1)/2 counts and low for (N - 1)/2 counts.

Mode 4: Software Triggered Strobe

- The output goes high on setting the mode. After terminal count, the output goes low for one clock period and then goes high again.

- In this mode the OUT is initially high; it goes low for one clock period at the end of the count. The count must be reloaded for subsequent outputs.

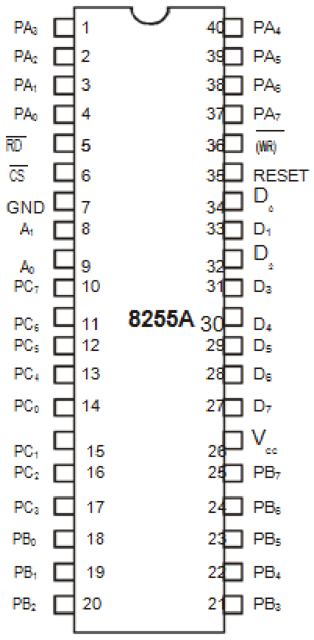
Mode 5: Hardware Triggered Strobe

- This mode is similar to mode 4, but a trigger at the gate initiates the counting.
- This mode is similar to mode 4, except that it is triggered by the rising pulse at the gate. Initially the OUT is high and when the gate pulse is triggered from low to high, the count begins, at the end of the count, the OUT goes low for one clock period.

8255 A - PROGRAMMABLE PERIPHERAL INTERFACE

- The 8255A (PPI) interfaces peripheral I/O devices to the microcomputer system bus.
- It is programmable by the system software.
- Peripheral devices: Printers, keyboards, displays, floppy disk controllers, CRT controllers, machine tools, D-to-A and A-to-D converters, etc are connected to the microprocessor through the 8255A port pins.
- It has a 3-state bi-directional 8-bit buffer which interfaces the 8255A to the system data bus.
- It has 24 programmable I/O Pins.

PIN DIAGRAM



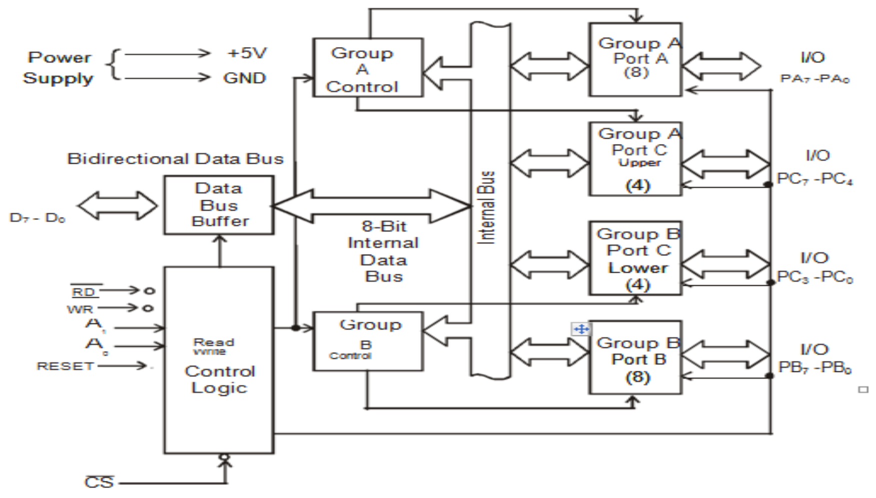
Block Diagram of 8255A

The 8255 consists of Four sections namely

1. Data Bus Buffer
2. Read/Write Control Logic
3. Group A Control
4. Group B Control

Data Bus Buffer

This is a tri-state, bi-directional data bus used to interface the internal data bus of 8255A to the system data bus of 8085.



Read/Write Control Logic

- This block controls the Chip Selection (CS), Read (RD) and Write (WR) operations.
- It consists of A_0 and A_1 signals which are generally connected to the CPU address lines A_0 and A_1 respectively.

Group A Control and Group B Control

To execute peripheral data transfer, three 8-bit ports are provided in 8255A i.e. ports A, B and C. For the purpose of programming 8255A, these ports are grouped as follows.

Group A : Port A and Most Significant Bits (MSB) of Port C ($PC_4 – PC_7$)

Group B : Port B and Least Significant Bits (LSB) of Port C ($PC_0 – PC_3$)

Port A: One 8-bit data output latch/buffer and one 8-bit input latch buffer.

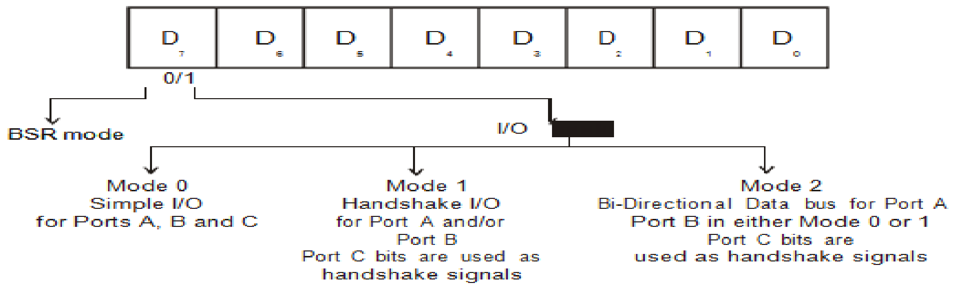
Port B: One 8-bit data input/output latch/buffer.

Port C: One 8-bit data output latch/buffer and one 8-bit data input buffer. This port can be divided into two 4-bit ports and it can be used for the control signal outputs and status signal inputs in conjunction with ports A and B.

Operating Modes

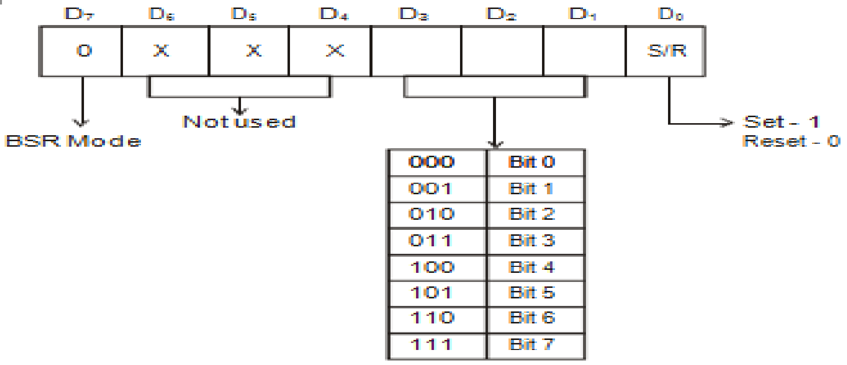
1. Bit Set/Reset (BSR) mode
2. I/O Mode

D_7 bit of control word decides the type of mode. If it is “1”, I/O mode is selected. If it is “0”, BSR mode is selected.



BSR (Bit Set/Reset) Mode

This mode is applicable only for Port C. A control word with bit D_7 = 0 is recognized as BSR control word. This control word can set or reset a single bit in the Port C. The control word format is given below.

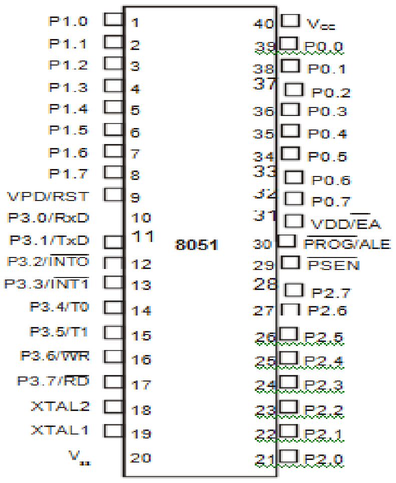


000	Bit 0
001	Bit 1
010	Bit 2
011	Bit 3
100	Bit 4
101	Bit 5
110	Bit 6
111	Bit 7

UNIT-4

MICROCONTROLLER

I/O PINS OF 8051



V$_{cc}$

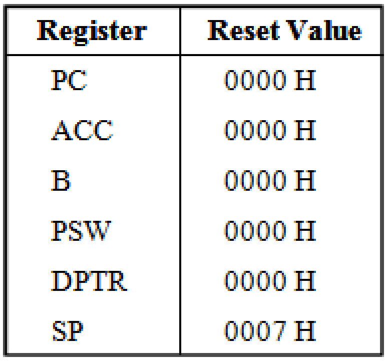
Pin 40 of 8051 provides supply voltage to the chip from the +5V voltage source.

GND

Pin 20 is the ground for the chip,

RST

Pin 9 is the RESET pin. This is referred as a power-on reset. When activating (high pulse) this pin the microcontroller will reset and all values in the registers to be lost.



Register	Reset Value
PC	0000 H
ACC	0000 H
B	0000 H
PSW	0000 H
DPTR	0000 H
SP	0007 H

EA (External Access)

This input pin is connected to either V$_{cc}$ or GND. The 8051 family members has ON - CHIP ROM to store programs. EA pin (31) is connected to **V**$_{cc}$ in 8051. For 8031 family members, there is no ON-CHIP ROM, code is stored on an external ROM and is fetched by 8031. Here, EA pin is connected to GND to indicate that the code is stored externally.

PSEN (Program Store Enable)

This is an output pin and pin number is 29. In an 8051 - based system in which an external ROM holds the program code, this pin is connected to the OE pin of the ROM.

ALE (Address Latch Enable)

This is an output pin (30) and is active high. When connecting an 8051 to external memory, port 0 provides both address and data. In otherwords, 8051 multiplexes address and data through port 0 to save pins. If ALE = 0, port 0 provides data (D0 - D7). If ALE=1, it has address (A_0- A_7).

XTAL 1 and XTAL 2

The 8051 has an on - chip oscillator and it requires an external clock to return it. The quartz crystal oscillator (12 MHz / 20 MHz) is connected to XTAL 1 (pin 19) and XTAL 2 (pin 18) with two 30pF capacitors

b. Draw and explain the architecture of 8051 with neat diagram. (13)

The intel 8051 contains two separate buses for both program and data. So, it has two distinctive memory spaces of 64K x 8 size for both program and data. It is based on an 8 bit central processing unit with an 8 bit accumulator and another 8 bit B register as main processing blocks. Other portions of the architecture include few 8 bit and 16 bit registers and 8 bit memory locations. It has some amount of data RAM built in the device for internal processing. This area is used for stack operations and temporary storage of data. 8051 is supported with on-chip peripheral functions like I/O ports, Timers / Counters, serial communication port. Fig. 4.2 shows the block diagram of the 8051.

The features of the 8051 are :

- 8 bit CPU with registers A (the accumulator) and B
- 16 bit Program Counter (PC) and Data Pointer (DPTR)
- 8 bit Program Status Word (PSW)
- 64K Program memory address space
- 64K Data memory address space
- 128 bytes of on chip data memory
- 32 I/O pins for four 8 bit ports : Port 0, Port 1, Port 2, Port 3
- Two 16 bit timers / counters : T_0 and T_1
- Full duplex UART : SBUF

Central processing unit

The CPU is the brain of the microcontrollers reading user's programs and executing the expected task as per instructions stored there in. It's primary elements are an Accumulator (ACC), B register (B), Stack pointer (SP), Program counter (PC), Program status word (PSW), Data pointer register (DPTR) and few more 8 bit registers.

Accumulator

The accumulator performs arithmetic and logic functions on 8 bit input variables. Arithmetic operations include basic addition, subtraction, multiplication and division. Logical operations are AND, OR XOR as well as rotate, clear, complement etc. Apart from all the above, accumulator is responsible for conditional branching decisions and provides a temporary place in a data transfer operations within the device.

B Register

B register is used in multiply and divide operations. During execution B register either keeps one of the two inputs and then retains a portion of the result. For other instructions it is used as general purpose register.

Stack Pointer

Stack Pointer (SP) is an 8 bit register. This pointer keeps track of memory space where the important register information are stored when the program flow gets into executing a subroutine. The stack portion may be placed in anywhere in the onchip RAM. But normally SP is initialized to 07H after a device reset and grows up from the location 08H. The SP is automatically incremented or decremented for all PUSH or POP instructions and for all subroutine calls and returns.

Program Counter

The Program Counter (PC) is the 16 bit register giving address of next instruction to be executed during program execution and it always points to the program memory space.

Data Pointer Register

The Data Pointer Register (DPTR) is the 16 bit addressing register that can be used to fetch any 8 bit data from the data memory space. When it is not being used for this purpose, it can be used as two eight bit registers, DPH and DPL.

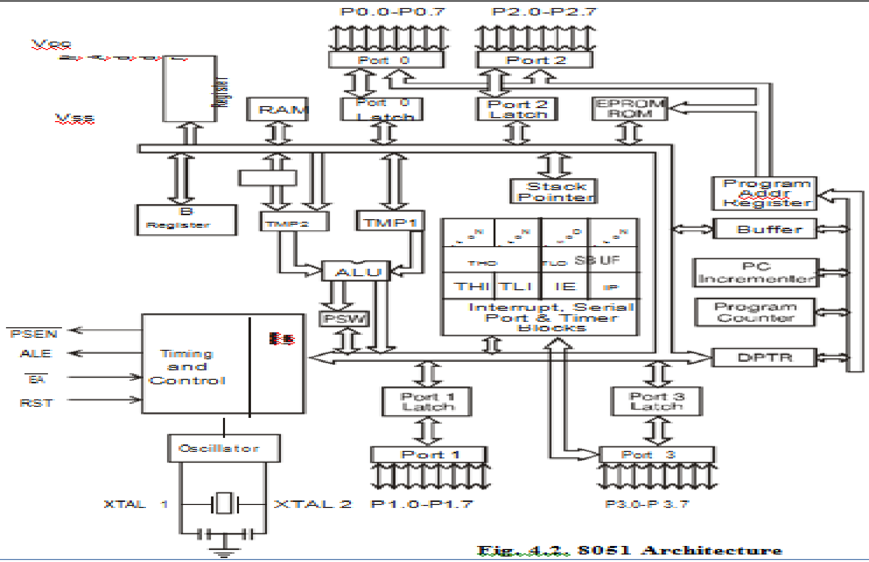
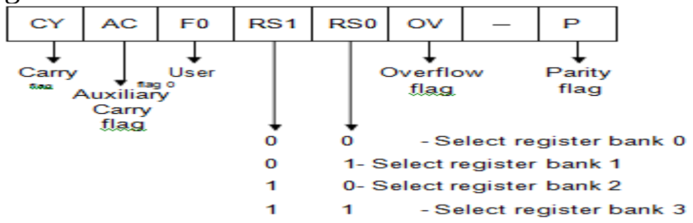


Fig. 4.2. 8051 Architecture

Program Status Word

The Program Status Word (PSW) keeps the current status of the arithmetic and logic operations in different bits. The 8051 has four math flags that respond automatically to the outcomes of arithmetic and logic operations and 3 general purpose user flags that can be set 1 or cleared to 0 by the programmer as desired. The math flags are carry (C), auxiliary carry (AC), overflow (OV) and parity (P). User flags are named flag 0 (F0), Register bank select bits RS0 and RS1.


CY
AC
F0
RS1
RS0
OV
—
P
Carry flag
Auxiliary Carry flag
User flag 0
Overflow flag
Parity flag
0 0 - Select register bank 0
0 1- Select register bank 1
1 0- Select register bank 2
1 1 - Select register bank 3

Input / Output Ports

8051 has 32 I/O pins configured as 4 eight bit parallel ports (P0, P1, P2 and P3). Each pin can be used as an input or as an output under the software control. These I/O pins can be accessed directly by memory instructions during program execution to get require flexibility.

These port lines can be operated in different modes and all the pins can be made to do many different tasks apart from their regular I/O function executions. Port 0 used as a multiplexed address / data bus.

Any instruction that accesses external program memory will output the higher order byte (A8 - A15) on Port 2 during read cycle. Port 1 and Port 3 are available for standard I/O functions. Port 3 pins has the additional functions : 2 external interrupt lines, 2 counter inputs, 2 serial port data lines and 2 timing control storbe lines.

Timers / Counters

8051 has two 16 bit Timers / Counters, T0 and T1 capable of working in different modes. Each consists of a 'HIGH' byte and a 'LOW' byte which can be accessed under software. There is a mode control register (TMOD) and a control register (TCON) to configure these timers / counters in number of ways.

Serial Port

The 8051 has a high speed full duplex serial port which is software configurable in 4 basic modes :

Shift register mode
Standard UART mode
Multiprocessor mode
9 bit UART mode.

Interrupts

The 8051 has five interrupt sources : One from the serial port (RI / TI) when a transmission or reception operation is executed : two from the timers (TF0, TF1) when overflow occurs and two come from the two input pins INT0, INT1.

Oscillator and Clock

The 8051 generates the clock pulses by which all internal operations are synchronized. Pins XTAL 1 and XTAL 2 are provided for connecting a resonant network to form an oscillator. A quartz crystal is used for oscillator. The crystal frequency is the basic internal clock frequency of the microcontroller.

PORTS AND CIRCUITS OF 8051

Port 0 (P0.0 - 0.7)

Port 0 is used for both address and data bus (AD_0 – AD_7). When the microcontroller chip is connected to an external memory, Port 0 provides both address and data. ALE pin indicates if Port 0 has address or data.

When ALE = 0, Port 0 provides data (D_0 – D_7)
= 1, Port 0 provides address (A_0 – A_7)

ALE is used for demultiplexing address and data with the help of a latch.

Port 1 (P1.0 - P1.7)

Port 1 pins are used as input or output. To make port 1 as an input port, write 1 to all its 8 bits. To make port 1 as output port, write 0 to all its 8 bits. Thus port 1 pins have no dual functions.

Port 2 (P2.0 - P2.7)

Port 2 pins are used as input / output pins similar in operation to port 1. The alternate use of port 2 is to supply a high order address byte (A_8 – A_{15}) when the microcontroller is connected to external memory.

Port 3 (P3.0 - P3.7)

Port 3 pins are used as input or output.

Pin	Function
P3.0 - RXD	Serial data input
P3.1 - TXD	Serial data output
P3.2 - $\overline{\text{INT0}}$	External interrupt 0
P3.3 - $\overline{\text{INT1}}$	External interrupt 1
P3.4 - T0	External timer 0 input
P3.5 - T1	External timer 1 input
P3.6 - $\overline{\text{WR}}$	External memory write pulse
P3.7 - $\overline{\text{RD}}$	External memory read pulse

ADDRESSING MODES

The addressing modes are the ways of accessing data in register or in memory or be provided as an immediate value. The 8051 mnemonics are written with the destination address named first, followed by the source address.

The following addressing modes are used to access data :

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

Immediate Addressing Mode

When a source operand is a constant rather than a variable, then the constant can be embedded into the instruction itself. This kind of instructions take two bytes and first one specifies the opcode and second byte gives the required constant. The operand comes immediately after the opcode. The mnemonic for immediate data is the pound sign (#). This addressing mode can be used to load information into any of the registers including DPTR register.

Examples :

MOV A, # 18H

$\boxed{A} \leftarrow 18H$

MOV B, # 65H

$\boxed{B} \leftarrow 65H$

MOV DPTR, #2040H

DPL $\leftarrow$ 40H

DPH $\leftarrow$ 20H

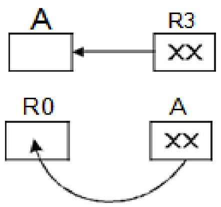
Register Addressing Mode

Register addressing accesses the eight working registers (R_0 - R_7) of the selected register bank. The least significant three bits of the instruction opcode indicate which register is to be used for the operation.

Examples :

MOV A, R3

MOV R0, A



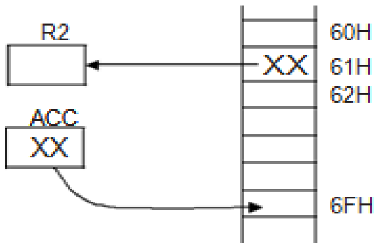
Direct Addressing Mode

In the direct addressing mode, all 128 bytes of internal RAM and the SFRs may be addressed directly using the single - byte address assigned to each RAM location and each SFR. Internal RAM uses address from 00H to 7FH to address each byte.

Examples

MOV R2, 61H

MOV 6F H, A



Register Indirect Addressing Mode

In this mode a register is used as a pointer to the data. If the data is inside the CPU, only registers R0 and R1 are used for this purpose. When R0 and R1 hold the addresses of RAM locations, they must be preceded by the "@" sign.

Indexed Addressing Mode

Only the program memory can be accessed by this mode. This mode is intended for reading lookup tables in the program memory. A 16 bit base register (DPTR or PC) points to the base of the lookup tables and accumulator carries the constant indicating table entry number.

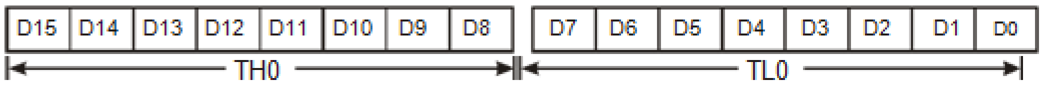
MOVC A, @A + DPTR : The contents of A are added to the DPTR to form the 16 bit address of the needed data. 'C' means code.

TIMER AND COUNTER REGISTERS

- Timers are used to generate a time delay.
- Counters are used to count events happening outside the microcontroller.
- Many microcontroller applications require the counting of external events [frequency of a pulse train, generation of internal time delays].
- The 8051 has two timers/ counters, T0 and T1. T0 and T1 may be programmed to count internal clock pulses, acting as a timer or may be programmed to count external pulses acting as a counter.

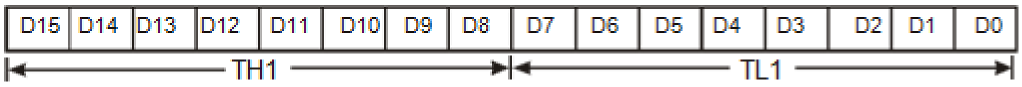
Registers of the Timer 0

The 16 bit-register of Timer 0 is divided into two 8 - bit registers called the Timer 0 Low byte (TLO) and the Timer 0 High byte (THO).



Registers of the Timer 1

The 16 bit register of Timer 1 is divided into TL1 (Timer 1 Low byte) and TH1 (Timer 1 High byte).



Control Registers

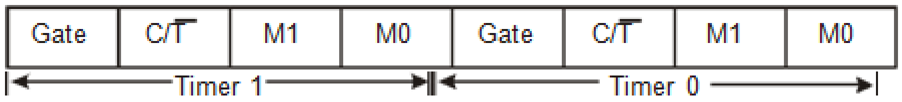
The Timer / Counter action is controlled by

(i) TMOD (Timer Mode Control Special Function Register)

(ii) TCON (Timer Control Special Function Register)

TMOD Register

TMOD register is used to set the various timer operation modes. TMOD is dedicated solely to the two timers and can be considered to be two duplicate 4-bit registers, each of which controls the action of one of the timers. TMOD register is not bit addressable.



Gate:

- It is the OR gate enable bit, which controls RUN/STOP of timer 1/0.
- Timer/Counter is enabled while TR 1/0 in TCON is set and signal on external interrupt INT1/0 pin is high.
- Cleared to 0 by program to enable timer to run if bit TR1/0 in TCON is set.

C/$\overline{T}$: (Clock/Timer)

This bit in the TMOD register is used to decide whether the timer is used as a timer or an event counter.

If C/$\overline{T}$ = 0, it is used as a timer for time delay generation.

If C/$\overline{T}$ =1, it is used as a counter by counting pulses from external input pin 3.5 (T1) or 3.4 (T0).

M1:

Timer/Counter operating mode select bit 1.
Set/cleared by program to select mode.

M0:

Timer/Counter operating mode select bit 0.
Set/cleared by program to select mode.

TCON Register

Timer counter (TCON) special function register has control bits and flags for the timers in the upper nibble and control bits and flags for the external interrupts in the lower nibble.

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT1

TF1:

- Timer T overflow flag.
- Set by hardware when timer/ counter 1 overflows.
- Cleared when processor vectors to execute ISR (Interrupt Service Routine) located at program address 00 1 BH.

TR1:

- Timer 1 run control unit.
- Set to 1 by program (software) to enable timer to count.
- Cleared to 0 by program to halt (off) the timer

TF0:

- Timer 0 overflow flag.
- Set by hardware when timer/ counter 0 overflows. Cleared when processor vectors to execute ISR located at program address 000BH.

UNIT -5

INTERFACING MICROCONTROLLER

STEPPER MOTOR INTERFACING WITH 8051

A stepper motor is a widely used device that translates electrical pulses into mechanical movement. In applications such as disk drives, dot matrix printers and robotics the stepper motor is used for position control. Every stepper motor has a permanent magnet rotor surrounded by four stator windings, that are paired with a center-tapped common. The center tap allows a change of current direction in each of two coils when a winding is grounded, thereby resulting in a polarity change of the stator. The stepper motor shaft runs in a fixed repeatable increment which allows one to move it to a precise position. This repeatable fixed movement is possible as a result of basic magnetic theory where poles of the same polarity repel and opposite poles attract. The direction of the rotation is dictated by the stator poles. The stator poles are determined by the current sent through the wire coils. As the direction of the current is changed, the polarity is also changed causing the reverse motion of the rotor As the sequence of power is applied to each stator winding, the rotor will rotate. There are several used sequences where each has a different degree of precision.

The drive circuitry for a stepper motor is shown in Fig.5.20. The driver must be able to handle the current and voltage involved. Furthermore, because the load is inductive, the drivers must be protected against the inductive voltage surge which occurs when a transistor switch tries to open up and discontinue the current flow through a winding of the stepper. The drivers include clamping diodes to alleviate this problem. Now when one of the drivers turns off, its diode permits the current which was flowing through the motor winding to continue, dying away at a rate determined by the L/R ratio of the motor winding. The voltage on the output of the driver is driven by the motor winding only slightly above +5V, by an amount equal to the voltage drop across the diode. This protects the driver.

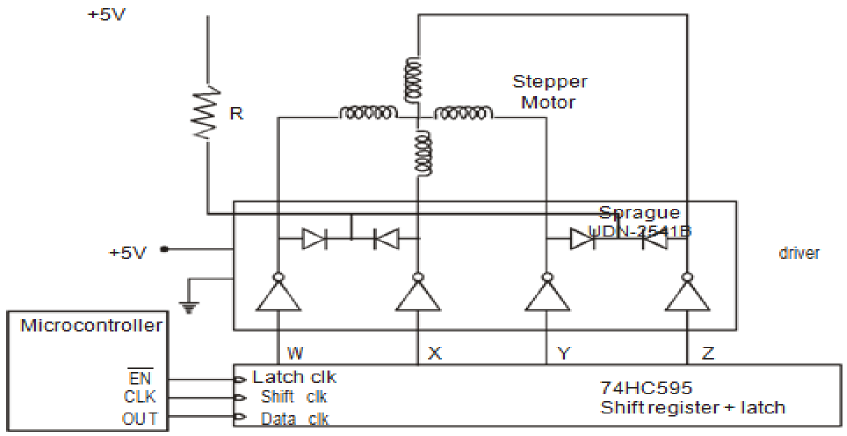


Fig 5.20 Drive circuitry for a stepper motor

The movement of the stepper motor with a single step is depends on the internal construction of the motor, in particular the number of teeth on the stator and the rotor. The step angle is the minimum degree of rotation associated with a single step. Various motors have different step angles. Table 5.3 shows some step angles for various motors.

Steps per revolution = Total number of steps needed to rotate one complete rotation or 360 degrees.

$$\text{Steps per second} = \frac{\text{RPM} \times \text{Steps per revolution}}{60}$$

WAVEFORM GENERATION:

- Steps to generate sine wave on 8051 microcontroller.
- Generate digital values of sine wave on a port that is 8 bit binary value.
- Convert that digital value into analog value to take that 8 bit output on 1 pin.
- Generated sine wave is in steps hence to obtain a pure sine wave, pass it through low pass filter. Thus by remove high frequency part, obtain smoother sine wave.
- First, generate digital values for sine wave. For this example take 16 points in 1 cycle. Thus 1 value will hold for 1/16th of 360 degree. Hence use sine(360 * (i/16)) where i runs from 0 to 15.
- This will cover 16 equally spaced points in one cycle. Place this cycle in while (1) loop so that will get continuous sine wave.
- In a cycle of sine wave, half cycle is positive and remaining half cycle is negative. Since microcontroller cannot have negative voltage, will shift sine wave to half of maximum value.
- As maximum value is 255 for 8 bits, half of it is 127.5.Thus digital value to be assigned to port is 127.5 + 127.5 * sine(360*(i/16)) where i runs from 0 to 15. Here minimum value is 127.5 - 127.5 = 0 and maximum value is 127.5 + 127.5 = 255

- Hence sine wave will be between 0 and 255 and which can be assigned to port. Since most of the values will come in fraction, have to round figure to assign integer value.

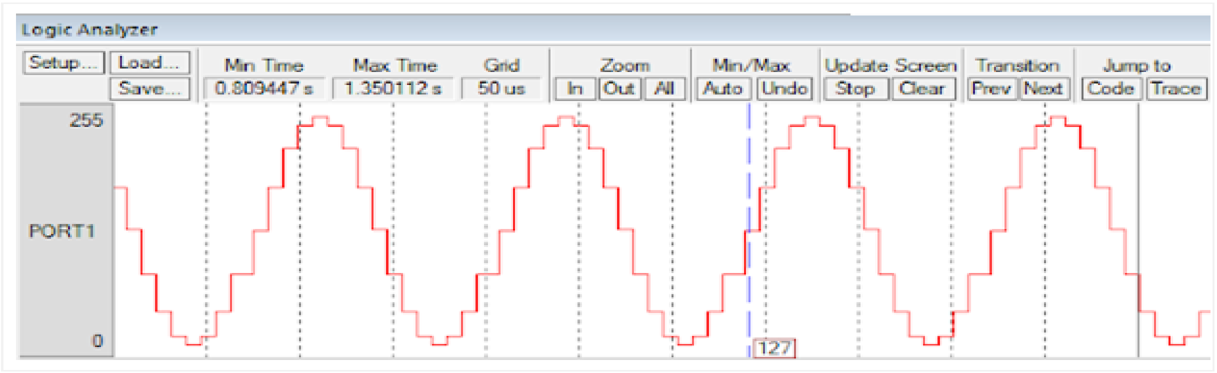
Program:

```
#include<reg51.h>

int main(void)
{
  //Digital values of sine wave
  unsigned char x[16]={127,176,218,245,255,245,218,176,128,79,37,10,0,10,37,79};

  unsigned char i;


  while(1)
  {
    for(i=0;i<16;i++)
    {
      P1=x[i];
    }
  }
}
```



SERIAL COMMUNICATION IN 8051

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

TMOD Register

GATE	C/T	M1	M0	GATE	C/T	M1	M0
0	0	1	0	0	0	0	0

If M_1 M_0 = 10, 8 bit Auto - reload counter

2. The TH 1 is loaded with one of the values in Table to set the baud rate for serial data transfer.

Baud rate	TH 1 (Decimal)	TH 1 (Hex)
9600	–3	FD
4800	–6	FA
2400	–12	F4
1200	–24	E8

- TR 1 is set to start Timer 1.
- TI is cleared by the "CLR TI" instruction.
- The character byte to be transferred serially is written into the SBUF registers.

- The TI flag bit is monitored with the use of the instruction “JNB TI, XX ” to see if the character has been transferred completely.
- To transfer next character, go to step 5.

Program

Write an ALP to transfer letter ‘E’ serially at 4800 baud continuously.

Solution:

```
          MOV      TMOD, # 20 H      ;   Timer 1, Mode 2 (Auto-reload)
          MOV      TH 1 #-6          ;   4800 baud rate
          MOV      SCON, # 50 H      ;  8-bit, 1 stop, 1 start, REN enabled
          SET B    TR 1              ;   Start Timer 1
LOOP 1:   MOV      SBUF, # ‘E’       ;   Letter ‘E’ to be transferred
LOOP 2:   JNB      TI, LOOP2         ;  Wait for the last bit
          CLR      TI                ;   Clear TI for next character
          SJMP     LOOP 1            ;   Go to Loop 1 for sending ‘E’
```

Programming the 8051 to receive data serially

- The first 4 steps are as same in programming to transfer data serially.
- RI is cleared with “CLR RI “ instruction.
- The RI flag bit is monitored with the use of the instruction “JNB RI, XX” to see if the character has been received yet.
- When RI is raised, SBUF has the byte. Its contents are moved into a safe place.
- To receive the next character, go to step 5.

Program

Write an ALP to receive bytes of data serially and put them in Port 2. Set the baud rate at 2400, 8 bit data and 1 stop bit.

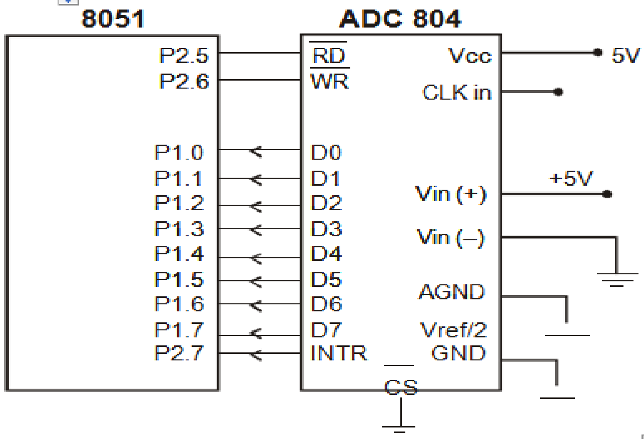
Solution:

```
             MOV    TMOD, # 20 H   ;   Timer 1, mode 2
             MOV    TH 1, # F4 H   ;  For 2400 baud TH1=-12 (F4 H)
             MOV    SCON, # 50 H   ;  8-bit, 1 stop, REN enabled
             SET B  TR 1           ;   Start Timer 1
LOOP 1;  JNB        RI, LOOP 1     ;   Wait for character to come in
             MOV    A, SBUF        ;   Save incoming byte in A
             MOV    P2, A          ;   Send to Port 2
             CLR    RI             ;  Get ready to receive next byte
             SJMP   LOOP 1         ;   Go to Loop 1, to keep getting data.
```

ADC INTERFACING

- ADCs are used to convert the analog signals to digital numbers so that the microcontroller can read them.
- ADC [like ADC 0804 IC] works with +5 volts and has a resolution of 8 bits.
- Conversion time is defined as the time taken to convert the analog input to digital (binary) number. The conversion time varies depending upon the clock signals; it cannot be faster than 110 µs .
- Analog input is given to the pins V_{in} (+) and V_{in} (-).
- V_{in} (-) is connected to ground.
- Digital output pins are D_0 - D_7. D_7 is the MSB and D_0 is the LSB.

- There are two pins for ground, analog ground and digital ground. Analog ground is connected to the ground of the analog V_{in} and digital ground is connected to the ground of the V_{CC} pin.
- The following steps are followed for data conversion :
- Make chip select (CS) = 0 and send a low - to - high pulse to pin WR to start the conversion.
- Keep monitoring the INTR pin. If INTR is low, the conversion is finished and go to the next step. If INTR is high, keep polling until it goes low.
- After the INTR has become low, we make CS = 0 and send a high- to-low pulse to the RD pin to get the data out.



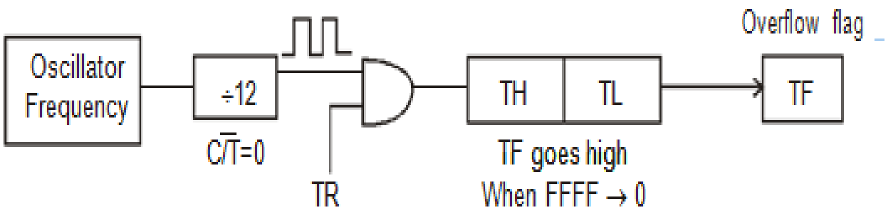
The program presents the concept to monitor the INTR pins and bring an analog input into register A. Then call a hex - to - ASCII conversion and data display subroutines continuously.

- P2.6 = WR (start conversion needs to low - to - high pulse)
- P2.7 = INTR, when low, end - of - conversion
- P2.5 = RD (a high-to-low will read the data from ADC chip)
- P1.0 - P1.7 = D_0 - D_7 of ADC 804

<u>TIMERS/COUNTERS IN 8051</u>

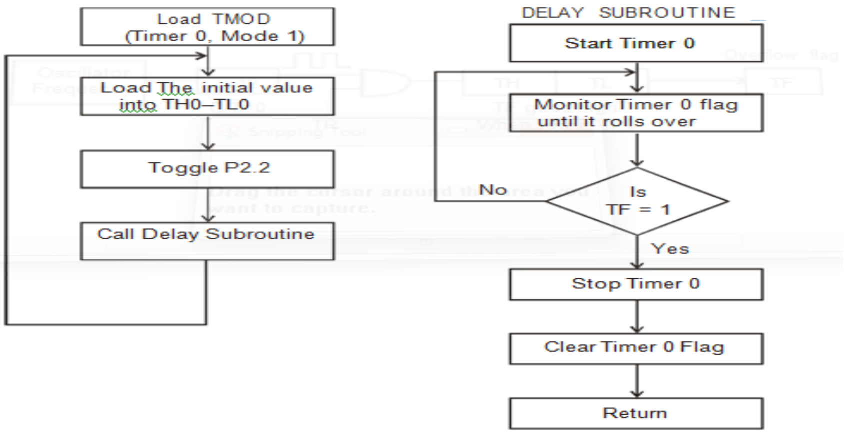
Operations of mode 1:

- It allows values of 0000 H to FFFF H to be loaded into the timer's registers TL and TH.
- After TH and TL are loaded with a 16 - bit initial value, the timer must be started.
- This is done by "SET B TR0" for Timer 0 and "SET B TR1" for Timer 1.
- After the timer is started, it starts to count up. It counts up until it reaches its limit of FFFF H. When it rolls over from FFFF H to 0000H, it sets high a flag bit called TF (Timer Flag). This timer flag can be monitored. When this timer flag is raised, one option would be to stop the timer with the instructions "CLR TR0" or "CLR TR1" for Timer 0 and Timer 1 respectively.
- After the timer reaches its limit and rolls over to repeat the process the registers TH and TL must be reloaded with the original value and TF must be reset to 0



Procedure for timer 0

- Load the TMOD value register indicating which timer (Timer 0 or Timer 1) is to be used and which timer mode (0 or 1) is selected.
- Load registers TL and TH with initial count values.
- Start the Timer.
- Keep monitoring the timer flag (TF). When TF becomes high get out of the loop.
- Stop the timer.
- Clear the TF flag for the next round.



```
                MOV      TMOD, # 01
                MOV      TL0, # 0F2 H
    LOOP 1:     MOV      TH0, # 0FF H
                CPL      P2.2
                ACALL    DELAY
                SJMP     LOOP 1
    DELAY:      SET      TR0
    LOOP 2:     JNB      TF0, LOOP2
                CLR      TR0
                CLR      TF0
                RET
```

TMOD register :

Timer 1				Timer 0				
GATE	C/T	M1	M0	GATE	C/T	M1	M0	
0	0	0	0	0	0	0	1	=01 H

Timer 0, Mode 1 –16 bit Timer mode.

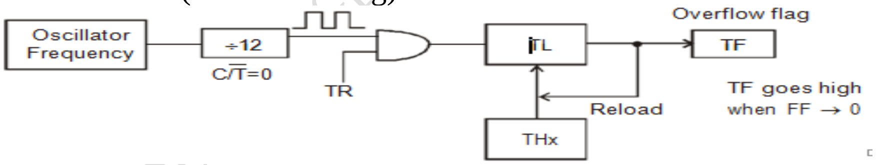
Program :

```
            CLR      P1.4
            MOV      TMOD,    #01 H
LOOP 1:     MOV      TL0,     #0B0 H
            MOV      TH0,     #3C H
            SET B    P1.4
            SET B    TR0
LOOP2:      JNB      TF0, LOOP2
            CLR      TR0
            CLR      TF0
            CLR      P1.4
            LJMP     LOOP1
```

Mode 2 Programming

Operations of Mode 2:

- Mode 2 allows only values of 00 H to FF H to be loaded into the timer's register TH.
- After TH is loaded with the 8 bit value, the 8051 gives a copy of it to TL. Then the timer must be started. This is done by "SET B TR0" for Timer 0 and "SET B TR 1" for Timer 1.
- After the timer is started, it started it starts to count up by incrementing the TL register. It counts up until it reaches its limit of FFH. When it rolls over from FFH to 00H, it sets high the timer flag (TF) TF0 is raised for Timer 0 and TF 1 is raised for Timer 1.
- When the TL register rolls from FF H to 00 H and TF is set to 1, TL is reloaded automatically with the original value kept by the TH register. To repeat the process clear TF (anti - reloading).



Procedure

- Load the TMOD value register indicating which timer (Timer 0 or 1) is to be used and select the timer mode 2.
- Load the TH registers with the initial count value.
- Start the timer.
- Keep monitoring the timer flag (TF) with "JNB TFx" instruction. When TF becomes high get out of the loop.
- Clear the TF flag
- Go back to step 4, since Mode 2 is auto - reload.

TMOD register :

Timer 1				Timer 0				
GATE	C/T	M1	M0	GATE	C/T	M1	M0	
0	0	1	0	0	0	0	0	=20 H

Timer 1, Mode 2 – Auto reload

Program :

```
            MOV     TMOD, # 20H
            MOV     TH1, #6
            SETB    TR1
LOOP:       JNB     JF1, LOOP
            CPL     P1.3
            CLR     TF 1
            SJMP    LOOP
```

COUNTER PROGRAMMING

- When C/T = 1, the timer is used as a counter and gets its pulses from outside the 8051. The counter counts up as pulses are fed from pins T0 (Timer 0 input) and T1 (Timer 1 input). These two pins belong to port 3. For Timer 0, when C/T = 1 pin 3.4 provides the clock pulse and counter counts up for each clock pulse coming from that pin.
- Similarly for Timer 1, when C/T = 1 each clock pulse coming in from pin 3.5 makes the counter countup.

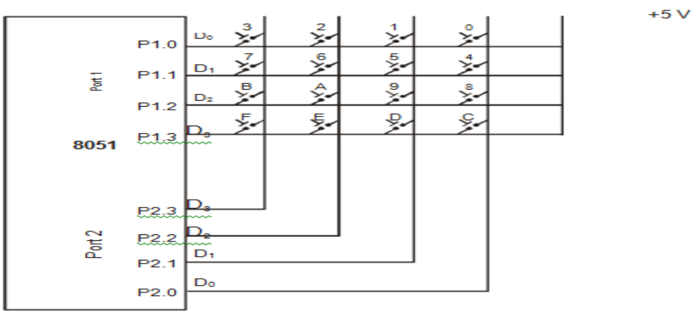
  P3.4 - T0 - Timer/Counter 0 external input

  P3.5 - T1 - Timer/Counter 1 external input
- In counter mode, the TMOD, TH and TL registers are the same as for the timer. Counter programming also same as timer programming.

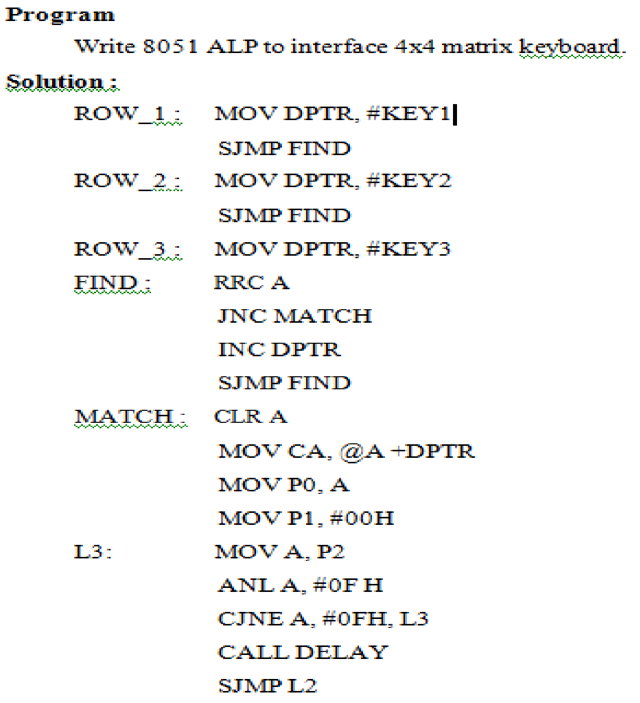
```
            MOV     TMOD, # 0110 0000 B    ;  Counter 1, Mode 2, C/T̄ =1
            MOV     TH 1, # 00 H           ;  Clear TH 1
            SET B   P3.5                   ;  Make T1 input
LOOP 1:     SET B   TR 1                   ;  Start the counter
LOOP 2:     MOV     A, TL 1                ; Get copy of count TL 1
            MOV     P2, A                  ;  Display it on Port 2
            JNB     TF 1, LOOP 2           ; Goto Loop 2 if TF = 0
            CLR     TR1                    ;  Stop the counter 1
            CLR     TF 1                   ;  Make TF = 0
            SJMP    LOOP 1                 ;  Jump to Loop 1.
```

KEYBOARD INTERFACING

- The rows are connected to an output port and the columns are connected to an input port.
- When a key is pressed, a row and a column make a contact, otherwise there is no connection between rows and columns.
- If all the rows are grounded and a key is pressed, one of the columns will have 0 since the key pressed provides the path to ground.
- If no key has been pressed, reading the input port will yield 1s for all columns since they are connected to Vcc.



- If any key is pressed, the columns are scanned again and again until one of them has a 0 on it.
- After the key press detection, it waits 20 milli seconds for the bounce and then scans the columns again.
- After 20 ms delay, the key is still pressed, it goes to detect which row it belongs to. To detect the row it grounds one row at a time, reading the columns each time.
- If all columns are high, the pressed key cannot belong to that row. Therefore it grounds the next row and continues until it finds the row the key press belongs to.
- After finding the row, it sets up the starting address for the look-up table holding the ASCII codes for that row and goes to the next stage to identify the key.
- Now it rotates the column bits, one bit at a time into the carry flag and checks if it is low.
- When carry flag is zero, it pulls out the ASCII code for that key from look-up table; otherwise it increments the pointer to point to the next element of the look-up table.

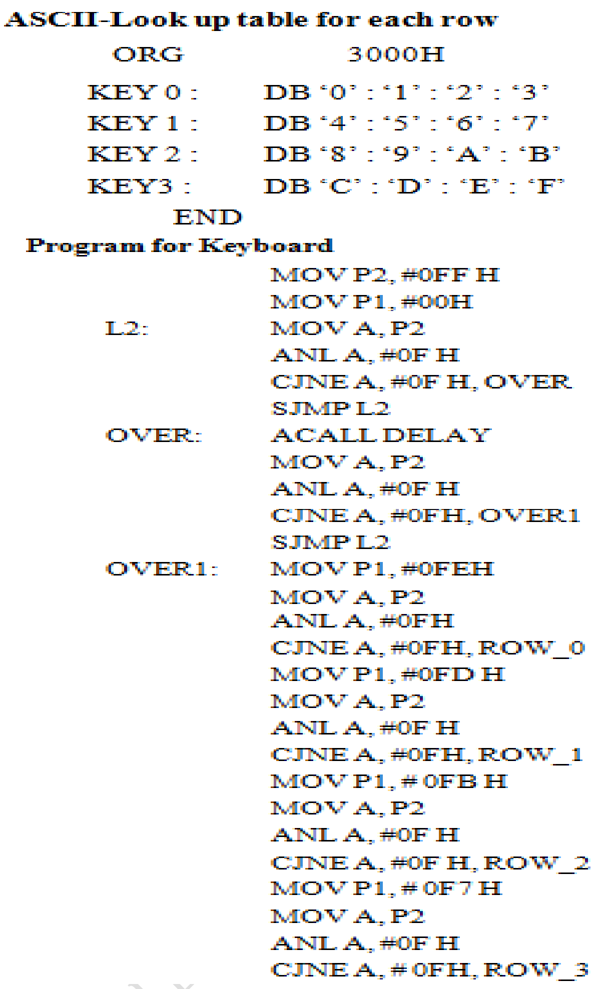


Program

Write 8051 ALP to interface 4x4 matrix keyboard.

Solution :

```
ROW_1 :     MOV DPTR, #KEY1
            SJMP FIND
ROW_2 :     MOV DPTR, #KEY2
            SJMP FIND
ROW_3 :     MOV DPTR, #KEY3
FIND :      RRC A
            JNC MATCH
            INC DPTR
            SJMP FIND
MATCH :     CLR A
            MOV CA, @A +DPTR
            MOV P0, A
            MOV P1, #00H
L3:         MOV A, P2
            ANL A, #0F H
            CJNE A, #0FH, L3
            CALL DELAY
            SJMP L2
```



ASCII-Look up table for each row

```
            ORG           3000H
KEY 0 :     DB '0' : '1' : '2' : '3'
KEY 1 :     DB '4' : '5' : '6' : '7'
KEY 2 :     DB '8' : '9' : 'A' : 'B'
KEY3 :      DB 'C' : 'D' : 'E' : 'F'
        END
```

Program for Keyboard

```
            MOV P2, #0FF H
            MOV P1, #00H
L2:         MOV A, P2
            ANL A, #0F H
            CJNE A, #0F H, OVER
            SJMP L2
OVER:       ACALL DELAY
            MOV A, P2
            ANL A, #0F H
            CJNE A, #0FH, OVER1
            SJMP L2
OVER1:      MOV P1, #0FEH
            MOV A, P2
            ANL A, #0FH
            CJNE A, #0FH, ROW_0
            MOV P1, #0FD H
            MOV A, P2
            ANL A, #0F H
            CJNE A, #0FH, ROW_1
            MOV P1, # 0FB H
            MOV A, P2
            ANL A, #0F H
            CJNE A, #0F H, ROW_2
            MOV P1, # 0F7 H
            MOV A, P2
            ANL A, #0F H
            CJNE A, # 0FH, ROW_3
```

EXTERNAL MEMORY INTERFACING

- When the data is located in the code space of 8051, MOVC instruction is used to get the data, where 'C' stands for code.
- When the data memory space must be implemented externally, MOVX instruction is used, where 'X' stands for external.

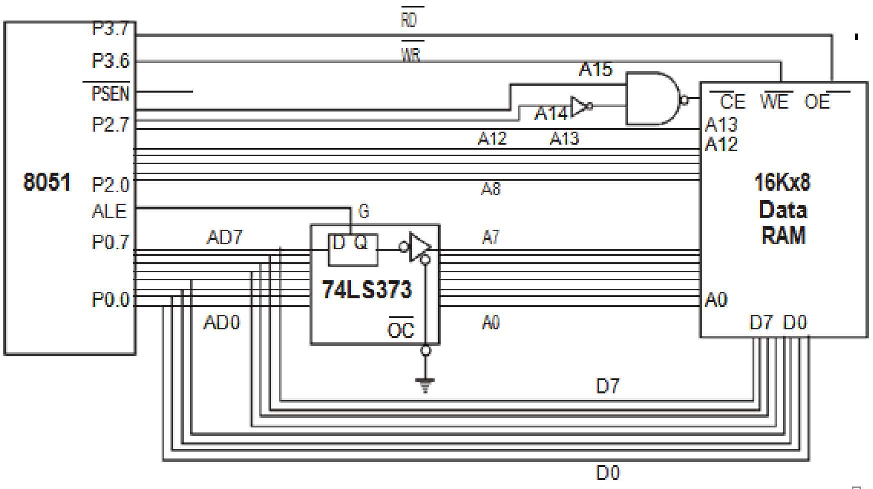
External data RAM interfacing

- To connect the 8051 to an external SRAM, we must use both RD (P3.7) and WR (P3.6).
- In writing data to external data RAM, the instruction "MOVX @DPTR, A" is used, where the contents of register A are written to external RAM whose address is pointed to by the DPTR register.

Program:

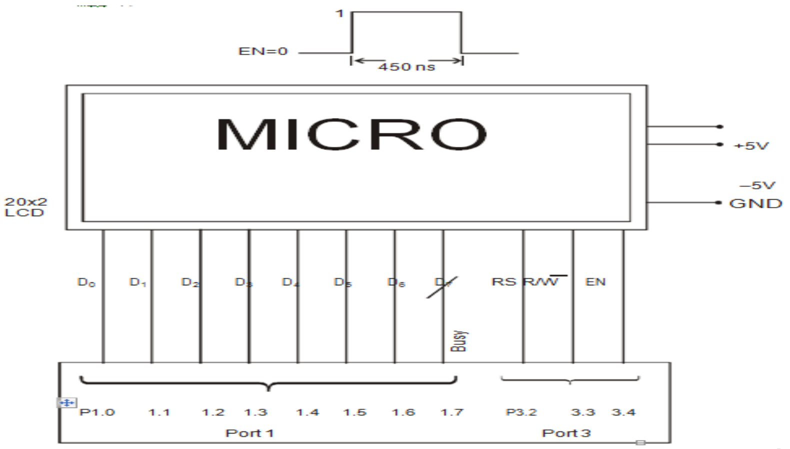
Write a program to read 200 bytes of data from Port 1 and save the data in external RAM starting at RAM location 5000H.

```
RAMDATA  EQU    5000H
COUNT    EQU    200
         MOV    DPTR, # RAMDATA ; pointer to external NV-RAM
         MOV    R3, #COUNT       ; counter
AGAIN :  MOV    A, P1            ; read data from P1
         MOVX   @DPTR, A         ; save it external NV-RAM
         ACALL  DELAY            ; wait before next sample
         INC    DPTR             ; next data location
         DJNZ   R3, AGAIN        ; until all are read
HERE :   SJMP   HERE             ; stay here when finished
```



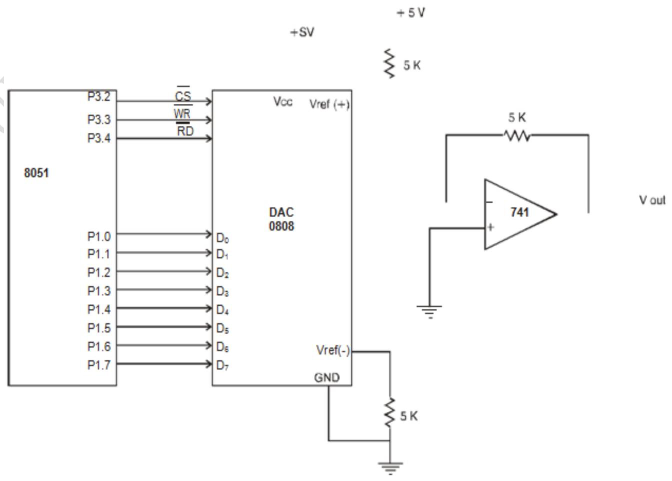
LCD INTERFACING

- The various types of LCD displays are, 16x2, 20x1, 20x2, 20x4, 40x2 and 40x4 LCDs. 16x2 LCD means that it having two lines, 16 characters per line.
- The 8 bit data pins (D_0–D_7) are used to send information tot he LCD or read the contents of the LCD's internal registers.
- The data lines are connected to Port 1. Register Select (RS),
- Read/Write (R/W) and Enable (EN) plans are connected to Port 3.



- There are two important registers are available inside the LCD. They are (i) instruction command register, (ii) data register.
- The RS pin is used to select the register. If RS=0, the instruction command code register is selected, allowing the user to send a command. If RS=1, the data register is selected, allowing the user to send data to be displayed on the LCD.
- R/W pin is used to write information to the LCD or read information from it. EN (enable) pin is used to latch information presented to its data pins.
- When data is supplied to data pins, a high-to-low pulse must be applied to EN pin in order for the LCD to latch in the data present at the data pins.
- This pulse must be a minimum of 450 ns.
- If RS=0 and R/W = 0
When busy flag (D_7)=1, the LCD is busy and will not accept any new information.
- When busy flag (D_7) = 0, the LCD is ready to receive new information.

DIGITAL TO ANALOG CONVERTERS



- The digital - to - analog converter (DAC) is used to convert digital pulses to analog signals.

The methods of creating a DAC are:

- Binary weighted
- R/2R ladder.
- Mostly R/2R method with DAC 0808 (MC 1408) is used since it can achieve a much higher degree of precision. Port 1 furnishes the digital byte to be converted to an analog Voltage and port 3 controls the conversion process.
- In DAC 0808, the digital inputs are converted to current. The total courrent provided by the I_{out} pin is a function of the binary numbers at the $D_0 – D_7$ inputs of DAC and the reference current I_{ref}.

$$I_{out} = I_{ref}\left(\frac{D_7}{2}+\frac{D_6}{4}+\frac{D_5}{8}+\frac{D_4}{16}+\frac{D_3}{32}+\frac{D_2}{64}+\frac{D_1}{128}+\frac{D_0}{256}\right)$$

Where $I_{ref} = 2$ mA.