Department of Information Technology

(V SEM IT)

EC 8691-MICROPROCESSORS AND MICROCONTROLLERS

QUESTION BANK

? MARKS & 16 MARKS

PART A - TWO MARKS

UNIT-I

1. What is pipelining?

Fetching the next instruction while the current instruction executes is called pipelining. 8086 is the first instruction fetch pipeline processor.

2. What are the signals involved in memory bank selection in 8086 microprocessor?

Entire memory is divided into two memory banks: $bank_0$ and $bank_1$. $Bank_0$ is selected only when A_0 is zero and $Bank_1$ is selected only when BHE is zero .A_0 is zero for all even addresses. So $bank_0$ is usually referred as *even addressed memory bank. BHE* is used to access higher order memory bank, referred to as *odd addressed memory* bank.

3. What is the use of MN/MX signals in 8086?

It is used to operate the microprocessor in two operating modes i.e. maximum and minimum mode. The minimum mode is used for small systems with a single processor and maximum mode is for medium size to large systems, which include two or more processors.

4. List the advantages of using segment registers in 8086.

- It allows the memory addressing capacity to be 1MB even though the address associated with individual instruction is only 16-bit.
- It facilitates use of separate memory areas for program, data and stack.
- It allows the program to be relocated which is very useful in multiprogramming.

5. Explain the BHE and LOCK signals of 8086

BHE (Bus High Enable): Low on this pin during first part of the machine cycle indicates that at least one byte of the current transfer is to be made on higher byte AD15-AD8.

LOCK: This signal indicates that an instruction with a LOCK prefix is being executed and the bus is not to be used by another processor.

6 .Mention the function of SI and DI registers.

SI: Source Index. It is used to hold the index value of source operand (data) for string instructions. DI: Destination Index It is used to hold the index value of destination operand (data) for string instructions .It is used for single stepping through a program.

7. What is meant by software interrupt in 8086?

The software interrupts are program instructions. These instructions are inserted at desired locations in a program. While running a program, if a software interrupt is encountered then the processor executes an interrupt service routine (ISR).

8. What is the function of TF, DF and IF in 8086?

TF: It is used for single stepping through a program. In the mode, the 8086 generates an internal interrupt after execution of each instruction.

DF: It is used to set direction in string operation.

IF: It is used to receive external maskable interrupts through INTR pin. Clearing IF, disable these interrupts.

9. What is the operation carried out When 8086 executes the instruction MOVSW & MOVSB ?

MOVSW – Move String Word.This instruction transfers a word from the ource string (addressed by SI) to the destination string (addressed by DI) and update SI and DI to point to the next string element.
MOVSB – Move String Byte. [[DI]] & [[SI]] Move 8 bit data from memory location addressed by SI segment in DS location to addressed by DI in segment ES. If DF (Direction Flag) = 0, SI is incremented by 1. DI is decremented by 1.

10. Differentiate between Memory mapped I/O and I/O mapped I/O.

Memory mapped I/O	**I/O mapped I/O**
It is treated as memory location	It is not treated as memory location
No special instructions are needed to Access	It requires special instructions like IN, OUT to access I/O devices
Microprocessor can access 1 MB memory locations or I/O ports	Microprocessor can access 64 KB memory locations or I/O ports
It requires 20 address lines	It requires 16 address lines
MEMR, MEMW signals can be used to access I/O devices	IOR , IOW signals are used
It is suitable for small system	It is suitable for large system

11. What is USART? Define Baud rate.

USART is an integrated circuit. It is a programmable device , its function and specifications for serial I/O can be determined by writing instructions in its internal registers .

Baud rate

The number of symbols per second i.e rate at which the bits are transmitted is called Baud rate.

Baud rate = 1/Time for a bit cell.

12. Give the various modes of 8254 timer?

Mode 0: Interrupt or terminal count

Mode 1: Rate generator

Mode 3: Square wave generator

Mode 4: Software triggered strobe

Mode 5: Hardware triggered strobe

13 .What is microcontroller?

Microcontroller incorporates all the features that are found in microprocessor with the added features of in-built ROM, RAM, Parallel I/O, Serial I/O, counters and clock circuit to make a micro computer system on its own.

14 . What are the alternate functions of Port 3 in 8051 microcontroller? Name the interrupts of 8051 microcontroller.

P3.0-RXD P3.1-TXD P3.2-INT0 P3.3-INT1 P3.4-T0 P3.5-T1 P3.6-WR P3.7-RD

Interrupts:

External interrupt-0, External interrupt-1, Timer-0 interrupt, Timer-1 interrupt, and serial port interrupt

15. What is the function of SM2 bit present in SCON register in 8051?

SM2 e n a b l e s the multiprocessor communication f e a t u r e in modes 2 and 3. If SM2 = 1, RI will not be activated if the received 9th data bit (RB8) is 0.In mode 1, if SM2 = 1, RI will not be activated if a valid stop bit was not received. In mode 0, SM2 should be 0.

16. If a 12 Mhz crystal is connected with 8051, how much is the time taken for the count in timer 0 to get incremented by one?

Baud rate = oscillator frequency/12

$$= (12 \times 10^6) / 12 = 1 \times 10^6 \text{ Hz}$$

$T = 1/f = 1/(1 \times 10^6) = 1\ \mu\text{sec}$

17. What is the function of DPTR register and TMOD register?

The data pointer register (DPTR) is the 16 bit address register that can be used to fetch any 8 bit data from the data memory space. When it is not b e i n g used for this purpose, it can be used as two eight bit registers , DPH and DPL.

TMOD (timer mode) register is used to set the various timer operation m od e s . TMOD is dedicated t o the two timers (Timer0 and Timer1) and can be considered to be two duplicate 4 bit registers, each of which controls the action of one of the timers.

18 . What are the register banks in 8051 microcontroller & give applications of microcontroller?.

34 general purpose or working registers in which A and B hold results of math and logical operations. The other 32 are arranged as part of the internal RAM in 4 banks: bank 0, bank 1, bank 2 and bank 3, each of eight registers.

Applications:

- Industrial control (process control)
  Motor speed control (stepper motor control)
  Peripheral devices (printer)
  Stand alone devices (color Xerox machine)
  Automobile applications (power steering)
  Home applications (washing machine)

19 .What are the bits available in TMOD register?

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

GATE: Gating control when set

C/T : Timer or counter selection; 1= counter, 0= Timer.

20. How does 8051 differentiate between the external and internal program memory?

EXTERNAL PROGRAM MEMORY	INTERNAL PROGRAM MEMORY
EA pin is high	EA pin is grounded
PSEN signal is activated	PSEN is grounded
8051 can address up to 64 KB of External	4KB of internal program memory is available
Accessible by only direct and indirect Addressing	Accessible by all addressing modes

21. What happens in power down mode of 8051 microcontroller?

The memory locations of power down RAM can be maintained through a separate small battery backup supply so that the content of these RAM can be preserved during power failure conditions.

22. Give steps to program 8051 for serial data transfer.

The 8051 has a serial data communication circuit that uses register S BU F to hold data. Register SCON controls data communication, register PCON controls data rates, and p i n s RXD (P3.0) and TXD (P3.1) connect to serial data network.

23. How does the status of EA pin affect the access to internal and external program memory?

EA- Enable Interrupt bit. Cleared to 0 by program to disable all interrupts, set to 1 to permit individual interrupts to be enabled by their enabled bits . It is set to access data from external memory or else it is grounded for internal memory operations.

24. Give the priority level of the interrupt sources.

Interrupt source Priority within a level

IE0 (External INT0)

TF0(Timer 0)

IE 1 (External INT 1)

TF 1 (Timer 1)

PART-B

PART B – 16 MARKS

1. (a).Draw the 8086 functional block diagram and explain its architectural features.

Architecture diagram of 8086 Microprocessor

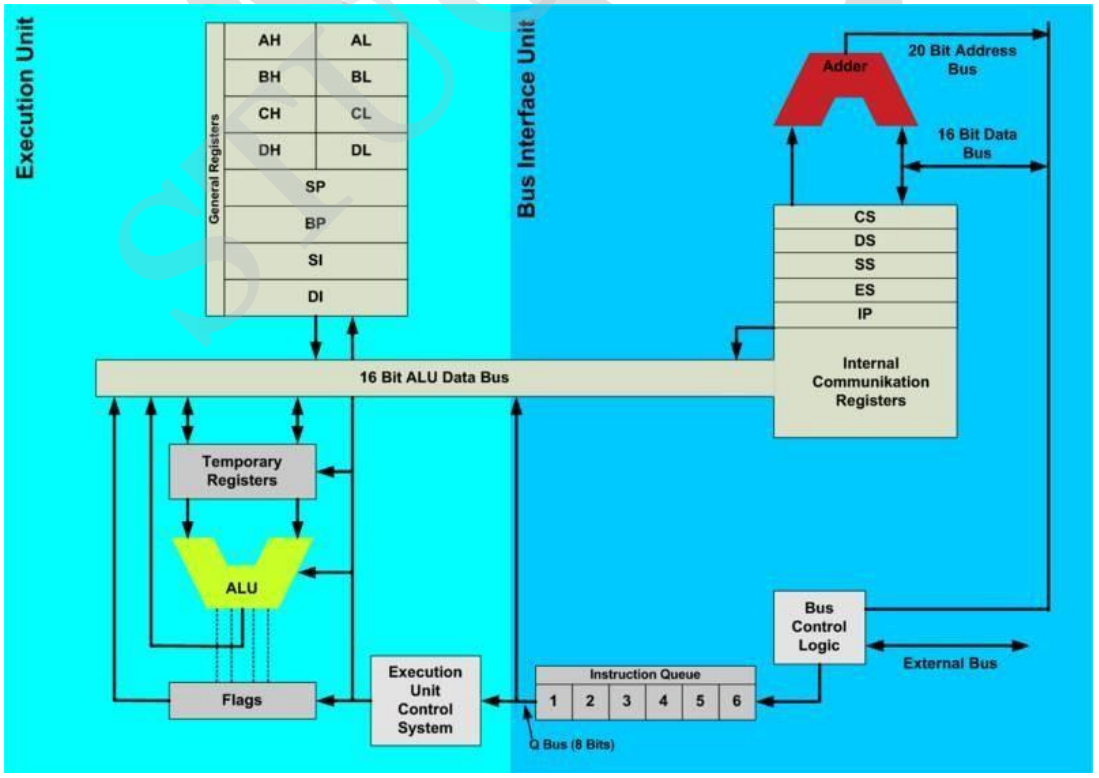
The 8086 has pipelined architecture. In pipelined architecture the processor can have number of functional units and the execution time of functional units can be overlapped. Each functional unit works independently most of the time. internal Functions of the 8086 processor was partitioned logically into two processing

- BusInterface Unit(BIU)

1. Segment Registers
2. Instruction Pointer
3. Instruction Queue

- ExecutionUnit(EU)

1. Arithmetic Logical Unit(ALU)
2. General Purpose Register
3. Index Register
4. Pointers Registers
5. Flags



Bus Interface Unit

The bus interface unit is the 8086's interface to the outside world. It provides a full 16-bit bidirectional data bus and 20-bit address. The bus interface unit is responsible for performing all external bus operations. Specifically, it has the following functions:

- Instruction Fetch
- Instruction Queueing
- Operand Fetch & Storage
- Address Relocation
- Bus Control.

The BIU is also responsible for generating bus control signals such as those for memory read or write and I/O read or write. These signals are needed for control of the circuits in the memory and I/O subsystems.

The instruction queue is a FIFO group of register. The size of queue is 6 bytes. The BIU fetches instruction code from memory and store in queue. The EU fetches instruction code from the queue.

Execution Unit

The execution unit is responsible for decoding and executing all instructions. It consists of an ALU, status and control flags, eight general-purpose registers, temporary registers, and queue control logic.

The EU extracts instructions from the top of the queue in the BIU, decodes them, generates operand addresses if necessary, passes them to the BIU and requests it to perform the read or write bus

cycles to memory or I/O, and performs the operation specified by the instruction on the operands. During execution of the instruction, EU tests the status and control flags and updates them based on the results of executing the instruction. If the queue is empty, the EU waits for the next instruction byte to be fetched and shifted to the top of the queue.

When the EU executes a branch or jump instruction, it transfers control to a location corresponding to another set of sequential instructions. Whenever this happens, the BIU automatically resets the queue and then begins to fetch instructions from this new location to refill the queue.

Category	Size(bits)	Register Name
General	16	AX, BX, CX & DX
General	08	AH, AL, BH, BL, CH, CL, DH & DL
Pointer	16	SP & BP
Index	16	SI & DI
Segment	16	CS, DS, ES & SS

Flag	16	Flag Register (FR)

8086 Registers by Category

Arithmetic Logic Unit(ALU)

The 8086 is a true 16-bit machine which means that the ALU (Arithmetic Logic Unit) is designed to work with 16-bit numbers and the data bus is also 16bits wide. It is used to perform all arithmetic and logical operation. Based on the result of operation it will modify the content of flag register.

General Register

All general registers of the 8086 microprocessor can be used for arithmetic and logic operations. The general registers are:

1. Accumulator [AX] −→ Register AX consists of 2 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case 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.

2. Base register [BX] −→ Register BX consists of 2 8-bit registers BL and BH, which can be c ombined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.

3. Count register [CX] −→ Register CX consists of 2 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low-order byte of the word, and CH contains the high-order byte. Count register can be used as a counter in string manipulation and shift/rotate instructions.

4. Data register [DX] −→ Register DX consists of 2 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX. When combined, DL register contains the low-order byte of the word, and DH contains the high-order byte. Data register can be used as a port number in I/O operations. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

Pointer Register

1. Stack Pointer (SP) −→ The stack pointer is used in instruction which use stack, i.e. PUSH, POP, CALL, RET, etc. It always points to a location in memory know as the stack top.
However, the complete address is formed by adding the content of stack segment after 4 bit left-shift, to the stack pointer.

2. Base Pointer (BP) −→ The chief purpose of this register is to provide indirect access to data in stack memory. It may also be used for general data storage. And also used for based, based indexed or register indirect addressing.

Index Register

1. Source Index (SI) −→ The source index register may be used for general data storage. However, main purpose is to store offset in case of indexed, base indexed and relative base indexed addressing modes. It stores the offset value of source data in case of *string instruction*.

2. Destination Index (DI) −→ The destination index register may be used for general data storage.

However, main purpose is to store offset in case of indexed, base indexed and relative base indexed addressing modes. It stores the offset value of destination data in case of *string instruction*.

Instruction Pointer

This register is also referred as *program counter*. As the name suggests, it is used for the calculation of actual memory address of instruction. It stores the offset for the instruction. During an instruction fetch, IP contents are added to the *code segment* register contents after bit left-shift.

Segment register of 8086.

Intel introduced the concept of memory segmentation in the 8086. In memory segmentation, memory is divided into a number of parts called *segment*.

In the 8086, 1MB physical memory is divided into four segments: CODE Segment, DATA Segment, and STACK Segment & EXTRA Segment. Each segment has memory space of 64KB. Each segment is addressed be a 16-bit segment register.

Segment registers are programmable. It is used to design multitasking and multiuser system using 8086. The program code and data for each task/user can be stored in sperate segments, so the program execution can be switched from one task/user to another task/user by changing the content of segment registers. Starting address of the segment is known as *Segment base* (or) *Base address*.

The CPU 8086 is able to address 1MB of physical memory. This complete memory can be divided into 16 segments, each of 64KB in size. The physical address of 8086 is 20-bits wide. However, its register and memory locations which contain logical address are just 16-bits wide. Hence we go for segmentation.

Segmentation

1. Code segment (CS) −→CS is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by

   instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions.

2. Stack segment (SS) −→SS is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction.

3. Data segment (DS) −→ DS is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, and DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions.

4. Extra segment (ES) −→ ES is a 16-bit register containing address of 64KB segment, usually with program data. By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. ES register can be changed directly using POP and LES instructions.

Properties of Segmentation

- The address of the segments may be assigned as 0000H to F000H.
- The OFFSET address values are from 0000H to FFFFH so that the physical addresses range from 00000H to FFFFFH.
- It is non-overlapping segments. In some cases, however, segments may be overlapping.

Advantages of memory segmentation

- Allow the memorycapacity to be 1Mb even though the addresses associated with the individual instructions are only 16 bits wide.
- Allow the placing of code, data and stack portions of the same program in different parts of memory, for the data and code protection.
- Facilitate the use of separate memory areas for the program, its data and the stack.
- Permit a program and/or its data to be put into different areas of memory each time the program is executed.
- Multitasking becomes easy.

1. (b) Explain about addressing mode of 8086.

Immediate Addressing

ABCDH

Register Addressing

This type instruction will specifies the name of the register which holds the data to be operated by the instruction. Ex: MOV AX,BX / MOV CL,DL

Direct Addressing

The instruction will specifies the address which holds the data be operated by the instruction. Ex: MOV DH,[08]H / MOV CX,[A2B3]H

Register Indirect Addressing

This instruction will specifies the name of the register which holds the address of the operand. Ex: MOV CX,[BX] / MOV [SI],AX

Based Addressing

In this addressing mode BX or BP register is used to hold a base value for OFFSET. Then a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction. Ex: MOV AX,[BX+28H] / MOV [BP+1357H],DX

Indexed Addressing

Here the index register is used to hold an index value for memory data. And a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction. Ex: MOV [SI+1432H],CX/ MOV BX,[DI+ABCDH]

Relative Based Index Addressing

Here the index register and base register is used with signed 8-bit or unsigned 16-bit displacement to specify the operand. EX: MOV [SI+BP+0123H],CX / MOV DX,[DI+BX+A143H]

String Addressing

It is used in string operation. SI used to hold the source address and DI used to hold the destination address of operand. Ex: MOVS/ MOVSB

Direct I/O

It is used to access data from standard I/O mapped device or ports. Here the 8-bit or 16-bit port address is directly specified in the instruction. Ex: IN AL,09H / OUT 1010,[AX]

address is stored in DX register. Ex: OUT [DX],AX / IN AX,[DX]

Relative Addressing

Here the effective address of a program instruction is specified relative to instruction pointer (IP) by an 8-bit signed displacement or 16-bit displacement. EX:JZ 410AH

Stack Addressing

This addressing mode used in stack operation. In this SS→ used as segment base and SP→ used as OFFSET value. EX: PUSH CX / POP BX

2.(a) Explain about data transfer and arithmetic Instruction set of 8086. Data Transfer Instruction

Theinstruction set of8086microprocessor includes a variety ofinstruction to transfer data/address into registers, memory locations and I/O ports. The various mnemonics used for data transfer instructions are MOV, XCHG, PUSH, POP, IN, OUT, etc., and they perform any one of the following operations:

- Copy the content of a register to another register.
- Copy the content of a register to memory or vice-versa.
- Load the immediate operand to memory/register.
- Copythe content of a register/memoryto segment register(excluding CS register) or vice-versa.
- Exchange the content of two registers or register and memory.
- Copy the content of accumulator to port or vice-versa.
- Load effective address in segment registers.

The data transfer instruction generally involves two operands: source and destination operand. The both operand should be of same size (byte or word). *Moving the content of 8-bit register to 16-bit register/memory or vice-versa is illegal*. The source can be a register or a memory or an immediate data. The destination can be a register or a memory location. The source and destination can't refer to memory in the sameinstruction.

So, copying the content of one memory location to another memory location in a single instruction is not possible except PUSH instruction. The data transfer instruction (except POPF and SAHF) does not affect the flags.

Examples:

- ***MOV*** → The move instruction moves the data between the specified register and to memory or vice-

inatio n, Source

1. MOV BL, 05H → Move immediate data 05H to BL register.
2. MOV DS,AX → Move the content of register pair AX to data segment register DS.
3. MOV CX,[SI] → Move the content of memory locations pointed by SI register to CX Register pair.

4. MOV [DI], BX → Move the content of BX register pair the memory location pointed by DI registers.

- ***LDS*** → The load DS instruction moves a first word from memory to register pair and second word to DS register. *Syntax*: LDS Register pair, Memory

Example

```
ORG 1000H MOV AX,M RET
M DW 4321H DW 5678H END
```

Before Inst.
AX is 3456H, DS is 9765H

After Inst.
AX is set to 4321H, DS is set to 5678H

- ***LEA*** → The Load Effective Address instruction moves a address of memory (offset) to register pair. *Syntax*: LEA Register pair, Memory

Example

```
ORG 100H
LEA AX,M ; AX = offset of M
HLT
M dw 4321H END
```

Before Inst.
AX is 3456H

After Inst.
AX is set to 4321H

- ***XCHG*** → This instruction exchanges the contents of two specified registers or a register and a memory location. *Syntax*: XCHG Register, Register/Memory

Example

```
ORG 100H
MOV AX,4567H
MOV BX,8456H
XCHG AX,BX
HLT
```

Before Inst.
AX is 4567H and bx is 8456H

After Inst.
AX is set to 8456H and bx is set to 4567H

- ***XLAT*** → This translate instruction is used to move the byte of data from memory to AL register. Where memory address is calculated by AL,BX and DS registers. i.e ((AL)+(BX)+(DS*10)→(AL). *Syntax* : XLAT

Example

```
ORG 100H MOV
DS,0300H MOV
BX,0100H MOV
AL,0DH HLT
```

Execution of XLAT→*Replaces the contents of AL by the contents of memory location with physical address*
*PA=(DS)*10 +(BX) +(AL)= 03000H + 0100H + 0DH = 0310DH*
Thus(0310DH)→(AL) Assuming this memory locations contains 52H. This value is placed in AL. That is (AL)=52H .

- ***PUSH*** → This instruction pushes the contents of the specified register/memory location on to the stack. The stack pointer is decremented by 2, after each execution of the instruction. The actual current stack-top is always occupied by the previously pushed data.

  Hence, the push operation decrements SP by two and then stores the two byte contents of the operand onto the stack. The higher byte is pushed first and then the lower byte. *Syntax*: PUSH Register/Memory.

- ***POP*** → This instruction loads the specified register/memory location with the contents of the memory location of which the address is formed using the current stack segment and stack pointer as usual. The stack pointer is incremented by 2. The POP instruction serves exactly

  opposite to the PUSH instruction. *Syntax*: POP Register/Memory.

- ***IN*** →This instruction is used for reading an input port. The address of the input port may be direct or indirect. AL and AX are the allowed destinations for 8-bit and 16-bit input operation. DX is the only register used to specify the port address. If the port address is of 16 bits it must be in DX. *Syntax* : IN AX,Port Address

  Example

IN AL,03H	*This instruction reads data from an 8-bit port whose address is 03H & stores it in AL.*
IN AX,DX	*This instruction reads data from an 16-bit port whose address is in DX & stores it in AX .*
HLT	

- ***OUT***→This instruction is used for writing to an output port. The address of the output port may be direct or indirect. AL and AX are the allowed source for 8-bit and 16-bit output oper- ation. DX is the only register used to specify the port address.

  If the port address is of 16 bits it must be in DX. The data to an odd addressed port is trans- ferred on D−8-D_{15} while that to an even addressed port is transferred on D_0-D_7. *Syntax*: OUT Port Address,AX.

  Example

OUT 04H,AL	*This instruction writes 8-bit data from AL to port address 04H . This instruction*
OUT DX,AX	*writes 16-bit data from AX to port address is specified in DX .*
HLT	

- ***LAHF*** → Load AH from lower byte of flag. This instruction loads the AH register with the lower byte of the flag register. This command may be used to observe the status of all the condition code flags(except over flag) at a time. *Syntax*: LAHF

- ***SAHF*** → Store AH to lower byte of flag register. This instruction sets or resets the condition code flags(except over flag) in the lower byte of the flag register depending upon the correspond- ing bit position in AH. If a bit in AH is 1, the flag corresponding to the bit position is set, else it is reset. *Syntax*: SAHF

- ***PUSHF*** → The push flag instruction pushes the flag register on to stack; first the upper byte and then the lower byte is pushed on to it. The SP is decremented by 2, for each push operation. *Syntax*: PUSHF
- ***POPF*** → The pop flag instruction loads the flag register from memory addressed by SP and SS. The SP is incremented by 2, for each push operation. *Syntax*: POPF

Arithmetic Instruction

The arithmetic instructions are used to perform the following operation:

- Addition or subtraction of binary, BCD or ASCII data.
- Multiplication or division of signed or unsigned binary data.
- Increment or decrement or comparison of binary data.

The arithmetic instruction generally involve two operands: source and destination operand. The source can be a register or a memory or an immediate data. The destination can be a register or a memory location. The result of arithmetic operation is stored in destination register or memory except in case of comparison. In comparison the result is used to modify the flags and then the result is discarded.

The source and destination can't refer to memory in the same instruction. In double operand arithmetic instruction except division, the source and destination operand should be of same size, either both the operand sizeshould be byte or word.

In all arithmetic instructions employing immediate addressing mode, if the immediate operand/data is 8-bit and the size of register/memory is 16-bit then the 8-bit immediate operand is sign extended to 16-bit. The processor use the result of arithmetic operation to alter the flags.

Examples:

- ***ADD*** → This instruction adds an immediate data or contents of a memory location or a register to the contents of another register or memory location. The result is stored in destination register. All the condition code flags are affected, depending upon the result. *Syntax*: ADD Destination, Source

1. ADD AX,0124H → Add immediate data to AX register - Immediate Addressing
2. ADD [2000H],0124H → Add immediate data to memory content - ImmediateAddressing
3. ADD AX,BX → ADD the content of AX to BX - Register Addressing
4. ADD AX,[SI] → ADD the content of memory locations pointed by SI to AX - Register Indirect Addressing - Indexed Addressing

5. ADD BX,[1234H] → ADD the content of BX to content of the OFFSET memory location 1234H in data segment - Direct Addressing
6. ADD 0124H → Add immediate data to AX - Implicit Addressing - Destination AX Implicit

- ***ADC*** → Add with Carry. This instruction adds an immediate data or contents of a memory location or a register to the contents of another register or memory location along with the carry flag bit to the result. The result is stored in destination register. All the condition code flags are affected, depending upon the result. *Syntax* : ADC Destination, Source

- ***SUB*** → This instruction subtract an immediate data or contents of a memory location or a register from the contents of another register or memory location. The result is left in the destination register. All the condition code flags are affected, depending upon the result. *Syntax*: SUB Destination, Source

1. SUB AX,0124H → Subtract immediate data from AX register - Immediate Addressing
2. SUB [2000H],0124H → Subtract immediate data from memory content - Immediate Ad-dressing
3. SUB AX,BX → Subtract the content of BX from AX - Register Addressing
4. SUB AX,[BX] → Subtract content of memory locations pointed by SI from AX - Register Indirect Addressing - Based Addressing
5. SUB BX,[1234H] → Subtract the content of BX from content of the OFFSET memory location 1234H in data segment - Direct Addressing

- ***SBB*** → Subtract with Borrow. This instruction subtract an immediate data or contents of a memory location or a register from the contents of another register or memory location along with the carry flag bit to the result. The result is lift in the destination register. All the condition code flags are affected, depending upon the result. *Syntax*: SBB Destination, Source
- ***INC***→ This instruction increases the contents of the specified register or memory location by 1. All the condition code flag are affected *except* the carry flag(CF).*Syntax*: INC Register/Memory

1. INC AX → Increment the content of AX by 1 - Register Addressing
2. INC [SI] → Increment the content of memory locations pointed by SI - Register Indirect Addressing - Indexed Addressing
3. INC [1234H] →Increment the content of OFFSET memory location 1234H in data segment - Direct Addressing

- ***DEC*** → This instruction subtract 1 from the contents of the specified register or memory location. All the condition code flag are affected *except* the carry flag(CF).*Syntax*: DEC Reg- ister/Memory

1. DEC AX → Decrement the content of AX by 1 - Register Addressing
2. DEC [SI] → Decrement the content of memory locations pointed by SI - Register Indirect Addressing - Indexed Addressing
3. DEC [1234H] → Decrement the content of OFFSET memory location 1234H in data segment - Direct Addressing

- ***CMP*** → This instruction compares the source operand, which may be an immediate data or contents of a memory location or a register with a destination operand that may be the contents of another register or memory location. For comparison, same like subtraction i.e it subtract the source content from destination content but does not store the result anywhere. All the condition code flags are affected, depending upon the result. *Syntax* : CMP Destination, Source

1. If both of operands are equal, zero flag is set.
2. If the source operand is greater than the destination operand carry flag is set or else, carry flag is reset.

- ***MUL***→ This instruction multiplies an unsigned byte or word by the contents of AL. The unsigned byte or word may be in any one of the general purpose registers or memory location. The MSB of the result is stored in DX, while LSB stored in AX. All the flags are modified depending upon the result. If the most significant bit or word of the result is '0' CF & OF both will be set.

Syntax: MUL Register/Memory

1. MUL BH ; (AX) ← (AL) * (BH) - Register Addressing
2. MUL CX ; (DX) (AX) ← (AX) * (CX) - Register Addressing
3. MUL WORD PTR [SI] ; (DX) (AX) ← (AX) * ([SI]) - Register Indirect Addressing - Indexed Addressing
4. MUL BYTE PTR [BX] ; (AX) ← (AL) * ([BX]) - Register Indirect Addressing - Based Addressing.

- ***IMUL***→ This instruction multiplies a signed byte in source operand by a signed byte in AL or a signed word in source operand by a signed word in AX. The source can be a general purpose register, memory operand, index register or base register, but it can't be an immediate data.In

32- bit MSB stored in DX and LSB stored in AX. The AF, PF, SF and ZF flags are undefied after IMUL. If AX and DX contain parts of 16 and 32-bit result respectively, CF & OF both will be set.In 8-bit, the unused higher bits of the result are filled by sign bit and CF, AF are cleared. *Syntax* :IMUL Register/Memory

- ***DIV***→ This instruction divide an unsigned word or double word by a 16-bit or 8-bit operand.

1. The dividend must be in AX for 16-bit operation and divisor may be specified using any one of the addressing modes except immediate. The result **Quotient**→AL while the **Remainder**→AH. If the result is too big to fir in AL, type)(divide by zero) and an interrupt is generated.
2. In 32-bit, the higher word should be in DX & lower word should be in AX. The divisor may be specified any addressing mode. The result **Quotient**→AX while the **Remainder**→DX.
3. This instruction does not affect any flag.*Syntax*: DIV Register/Memory

- ***IDIV*** →This instruction performs the same operation as the DIV, but with signed operands.

The results are stored similarly in DIV. If the result is too big to fit in destination register divide by 0 interrupt is generated. All the flags are undefined after IDIV instruction. *Syntax* : IDIV Register/Memory

- ***DAA***→This instruction is used to convert the result of the addition of two packed BCD numbers

to a valid BCD number. The result has to be only in AL. If the lower nibble is greater than 9, after addition or if AF is set, it will add 06H to the lower nibble in AL. After adding 06H in the lower nibble of AL, if the upper of AL is greater than 9 or if carry flag is set, DAA instruction adds 60H to AL.

MOV AL, 053H ; (AL) ← 053H - Immediate Addressing MOV BL, 029H ; (BL) ← 029H - Immediate Addressing ADD AL,BL

; (BL) ← (AL) + (BL)
DAA ; Content of AL adjusted to BCD HLT

- ***DAS*** → This instruction converts the result of subtraction of two packed BCD numbers to a

valid BCD number. The subtraction has to be in AL only. If the lower nibble of AL is greater than 9, this instruction will subtract 06H from lower nibble of AL. If the result of subtraction sets the carry flag or if upper nibble is greater than 9, it subtracts 60H from AL. This instruction modifies the AF, CF, SF, PF and ZF flags. The OF is undefined after DAS.

MOV AL, 053H ; (AL) ← 053H - Immediate Addressing MOV

```
BL, 029H ; (BL) ← 029H - Immediate Addressing SUB AL,BL
; (BL) ← (AL) - (BL)
DAS ; Content of AL adjusted to BCD HLT
```

- ***AAA***→The ASCII Adjust After Addition. This instruction is executed after an ADD instruction that adds two ASCII coded operands to give a byte of result in AL. The AAA instruction converts the resulting contents of AL to unpacked decimal digits.

1. After the addition, the AAA instruction examines the lower 4 bits of AL to check whether it contains a valid BCD number in the range 0-9 and AF is 0. Then AAA sets four higher bits of AL is 0. And AH is cleared before addition.
2. Else lower digit is 0-9 and AF is set, 06h is added to AL. The upper bits of AL is cleared and AH is incremented by 1.
3. If the value in the lower nibble of AL is greater than 9 then the AL is incremented by 06H, AH is incremented by 1, the AF & CF flags are set to 1, and the higher 4 bits of AL are cleared to 0. The remaining flags are unaffected.

- ***AAS*** →The ASCII Adjust AL After Subtraction. This instruction corrects the result in AL register after subtracting two unpacked ASCII operands. The result is in unpacked decimal format.

1. If the lower 4 bits of AL register are greater than 9 or if the AF flag is 1, the AL is decre- mented by 6 and AH register is decremented by 1, the CF & AF are set to 1. Otherwise, the CF and AF are set to 0, the result needs no correction.
2. As a result, the upper nibble of AL is 00 and the lower nibble may be any number from 0 to 9. The procedure is similar to the AAA instruction except for the subtraction of 06H from AL. AH is modified as difference of the previous contents(usually zero) of AH and the borrow for adjustment.

- ***AAM*** →The ASCII Adjust After Multiplication. This instruction converts the product available in AL into unpacked BCD format. In this instruction higher nibbles of the multiplication operands should be 0. The multiplication of such operands is carried out using MUL instruction. The result of multiplication is available in AX.

- ***AAD*** →The **ASCII Adjust Before Division**. This instruction converts two unpacked BCD digits in AH & AL to the equivalent binary number in AL. This Adjustment must be made before dividing the two unpacked BCD digits in AX by an unpacked BCD byte. PF, SF, ZF are modified while AF, CF, OF are undefined, after the execution of the instruction AAD.

- ***NEG*** → Negate. This instruction forms 2's complements of the specified destination in the instruction. For obtaining 2's complement, it subtract the contents of destination from zero. The result is stored back in the destination operand which may be a register or a memory location. If OF is set, it indicates that the operation could not be completed successfully. This instruction affects all the condition code fags. *Syntax* : NEG Register/Memory

- ***CBW*** → This instruction converts a signed byte to a signed word. In other words, it copies the sign bit of a byte to be converted to all the bits in the higher byte of the result word. The byte to be converted

must be in AL. The result will be in AX. It does not affect any flag.

- ***CWD*** → This instruction copies the sign bit of AX to all the bits of the DX register. This operation is to be done before signed division. It does not affect any flag.

2(b). Explain about assembler directives.

There are some instruction in the ASM which are not a part of processor instruction set. The assembler directives are the instructions to the assembler, linker & loader regarding the program being assembled. It also referred as *"PSEUDO-OPERATIONS", "ASSEMBLER DIRECTIVES"*. The assembler directives are used to specify start and end of a program, attach value to variables, allocate storage locations to input/output data, to define start and end of segment, procedure, macro, etc.,.

The assembler directive control the generation of machine code and organisation of the program. But no machine codes are generated for assembler directives. The words defined in this section are directions to the assembler, not instruction for the INTEL 8086. The assembler directives described here are those for the INTEL 8086 Macro Assembler (ASM86), the Borland Turbo Assembler (TASM) & the IBM Macro Assembler (MASM). Some of the assembler directives that can be used for 8086 assembly language development are explained below.

1. ***ASSUME*** → This directive is used to tell the assembler the name of the logical segment it should use for a specified segment. The statement ASSUME CS:CODE, for example, tells the assembler that the instructions for a program are in a logical segment named CODE. The statement ASSUME DS:DATA tells the assembler that for any program instruction which refers to the data segment, it should use logical segment called DATA. If, for example, the assembler reads the statement MOV AX,[BX] after it reads this ASSUME, it will know that the memory location referred to by [BX] is in the logical segment DATA.We should tell the assembler what to assume for any segment we use in a program.

2. ***DB-Define Byte*** → This directive is used to declare a byte-type variable, or to set aside one or more storage location of type byte in memory. The statement CURRENT_TEMP DB 42H, for example, tells the assembler to reserve 1 byte of memory and to put the value 42H in that memory location when the program is loaded into RAM to be run. NAME_STD DB 'KAMAL' reserve 5 byte of memory and initialize with ASCII codes for letters in KAMAL.
3. ***DW-Define Word*** → It is used to tell the assembler to define a variable of type word or reserve storage locations of type word in memory.
4. ***DD-Define Doubleword*** → It is used to declare a variable of type doubleword or to reserve memory location which can be accessed as type doubleword.
5. ***DQ-Define Quadword*** → It is used to tell the assembler to declare a variable 4 words in length or to reserve 4 words of storage in memory.

6. ***DT-Define Ten Bytes*** → It is used to the assembler to define a variable which is 10 bytes in length or to reserve 10 bytes of storage in memory.

7. ***END-End Program*** → The END directive is put after the last statement of a program to tell the assembler that this is the end of the program module. The assembler will ignore any statement after an END directive, so we make sure to use only one END directive at the very end of our program module.

8. ***ENDP-End Procedure*** → It is used along with the name of the procedure to indicate the end of a procedure to the assembler. This directive, together with the procedure directive. PROC, is used to "bracket" a procedure.

9. ***ENDS-End Segment*** → It is used with name of a segment to indicate the end of that logical segment. ENDS is used with the segment directive to "bracket" a logical segment containing instruction or data.

10. ***EQU-Equate*** → It is used to give a name to some value or symbol. Each time the assembler finds the given name in the program, it will replace the name with the value or symbol you equated with that name.

11. ***EVEN-Align On Even Memory Address*** → As the assembler assembles a section of data declarations or instruction statements, it uses a location counter to keep track of how many bytes it is from the start of a segment at any time. The EVEN directive tells the assembler to increment the location counter to the nest even address if it is not alreadyat aneven address.

12. ***EXTRN*** → The EXTRN directive is used to tell the assembler that the names or labels following the directive are in some other assembly module.

13. ***GLOBAl-Declare Symbols as PUBLIC or EXTRN*** → It can be used in place of a PUBLIC directive or in place of an EXTRN directive. For a name or symbol defined in the current assembly module, the GLOBAL directive is used to make the symbol available to other modules. The statement GLOBAL DIVISOR, for example, makes the variable DIVISOR public so that it can be accessed from other assembly modules.

14. ***GROUP-Group-Relates Segments*** → It is used to tell the assembler to group the logical segments named after the directive into one logical group segment. This allows the contents of all the segments to be accessed from the same group segment base. The assembler sends a message to the linker and/or locator telling it to link the segments so that the segments are physically in the same 64-Kbyte segment.

15. ***INCLUDE-Include Source Code From File*** → This directive is used to tell the assembler to insert a block of source code from the named file into the current source module. This shortens the source code. An alternative is to use the editor block commands to copy the file into he current source module.

16. ***LABEL*** → As the assembler assembles a section of data declarations or instruction statements, it uses a location counter to keep track of how many bytes it is from the start of a segment

at any time. The LABEL directive is used to give a name to the current value in the location counter. The LABEL directive must be followed by a term which specifies the type you want associated with that name. If the label is going to be used as the destination for a jump or a call, then the label must be specified as type nearor type far.

17. ***LENGTH-Not Implemented In IBM MASM*** → Length is an operator which tells the assembler to determine the number of elements in some named data item, such as a string or an array.

18. ***NAME*** → It is used to give a specific name to each assembly module when programs consisting of several modules are written. The Statement NAME PC_BOARD, for example, might be used to name an assembly module which contains the instructions for controlling a printed- circuit-board-making machine.

19. ***OFFSET*** → It is an operator which tells the assembler to determine the offset or displacement of a named data item(variable)or procedure from the start of the segment which contains it. This operator is usually used to load the offset of a variable into a register so that the variable can be accessed with one of the indexed addressing modes.

20. ***ORG-Originate*** → As the assembler assembles a section of data declarations or instruction statements, it uses a location counter to keep track of how many bytes it is from the start of a segment at any time. The location counter is automatically set to 0000H when the assembler starts reading a segment. The ORG directive allows you to set the location counter to a desired value at any point in the program. A '$' is often used to symbolically represent the current value of the location counter. The '$' actually represents the next available byte location where the assembler can put a data or code byte.

21. ***PROC-Procedure*** → It is used to identify the start of a procedure. The PROC directive follows a name we gave the procedure. After the PROC directive, the term near or the term far is used to specify the type of the procedure.

22. ***PTR-Pointer*** → It is used to assign a specific type to a variable or to a label. It is necessary to do this in any instruction where the type of the operand is not clear. When the assembler reads the instruction INC[BX], for example, it will not know whether to increment the byte pointed to by BX or to increment the word pointed to by BX. We use the PTR operator to clarify how we want the assembler to code the instruction. The statement INC BYTE PTR[BX] tells the assembler that we want to increment the byte pointed to by BX. The statement INC WORD PTR [BX] tells the assembler that we want to increment the word pointed to by BX.

23. ***PUBLIC*** → Large programs are usually written as several separate modules. Each module is individually assembled, tested and debugged. When all the modules are working correctly, their object code files are linked together to form the complete program. In order for the modules to link together correctly, any variable name or label referred to in other modules must be declared

public in the module in which it is defined. The public directive is used to tell the assembler that a specified name or label will be accessed from other modules.

24. ***SEGMENT*** → It is used to indicate the start of a logical segment. Preceding the segment directive is the name we want to give the segment. The statement CODE SEGMENT, for example, indicates to the assembler the start of a logical segment called CODE. The SEGMENT and ENDS directives are used to 'bracket' a logical segment containing code or data.

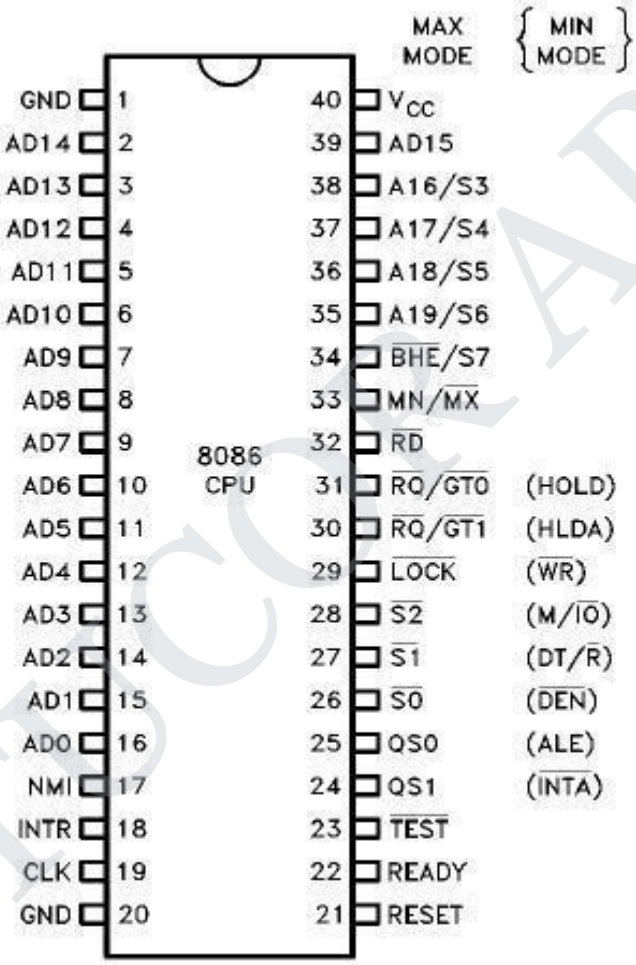
25. ***SHORT*** → It is used to tell the assembler that only a 1-byte displacement is needed to code a Jump instruction. If the jump destination is after the jump instruction in the program, the assembler will automatically reserve 2 bytes for the displacement. UsingSHORT operator save
1 byte of memory by telling the assembler that it needs to reserve only 1 byte for this particular jump. In order for this to work, the destination must be in the range of -128 bytes to +127 bytes from the address of the instruction after the jump.

26. ***TYPE*** → The TYPE operator tells the assembler to determine the type of a specified variable.
The assembler actually determines the number of bytes in the type of the variable. For a byte- type variable, the assembler will give a value of 1. For a word-type variable, the assembler will give a value of 4. The TYPE operator can be used in an instruction such as ASS BX,TYPE WORD_ARRAY, where we want to increment BX to point to the next word in an array of words.

27. ***ALIGN*** → Forces the assembler to align the next segment at an address divisible by specified divisor.
28. ***CODE*** → Provides shortcut in definition of CODE segment.
29. ***DATA*** → Provides shortcut in definition of DATA segment.
30. ***DUP*** → Used to initialize several locations.
31. ***NOTE*** → Name and labels referred to external in one module must be declared public.
32. ***MACRO*** → Used to represent start of the macro programming.
33. ***ENDM*** → Used to represent end of the macro programming.
34. ***.MODEL*** → Used to provide shortcut in defining segments.
35. ***PAGE*** → Used to control the format of listing of ASM.
36. ***STACK*** → Used for stack programming.
37. ***TITLE*** → Used to causes a title for a program.
38. ***FAR*** → Used to declare the procedure as far which assigns a far address.
39. ***NEAR*** → Used to declare a procedure as near which assigns a near address.
40. ***THIS*** → Used with EQU directive to set a label to a byte, word or double word.

3.(a) Explain about 8086 signals.

The Intel 8086 high performance 16-bit CPU is available in three clock rates: 5, 8 and 10 MHz. The CPU is implemented in N-Channel, depletion load, silicon gate technology (HMOS- III), and packaged in a 40-pin CERDIP or plastic package. The 8086 operates in both single processor and multiple processorconfigurations to achieve high performance levels.



The following pin function descriptions are for 8086 systems in either minimum or maximum mode. The "Local Bus" in these descriptions is the direct multiplexed bus interface connection to the 8086 (without regard to additional bus buffers).

8086 Common Signal to Minimum/Maximum Mode AD15-AD0 - (*Pin Number: 2-16,39 & Type: I/O)*

ADDRESS DATA BUS → These lines constitute the time multiplexed memory/IO address (T1), and data (T2, T3, TW, T4) bus. A0 is analogous to BHE for the lower byte of the data bus, pins D7-D0.

It is LOW during T1 when a byte is to be transferred on the lower portion of the bus in memory or I/O operations. Eight-bit oriented devices tied to the lower half would normally use A0 to condition chip select functions. (See BHE.) These lines are active HIGH and float to 3-state OFF during interrupt acknowledge and local bus "hold acknowledge".

A19/S6 - A16/S3 - (*Pin Number: 35-38 & Type: O*)

ADDRESS/STATUS → 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, TW, T4. The status of the interrupt enable FLAG bit (S5) is updated at the beginning of each CLK cycle. A17/S4 and A16/S3 are encoded as shown. This information indicates which relocation register is presently being used for data accessing. These lines float to 3-state OFF during local bus "hold acknowledge."

A17/S4	A16/S3	Characteristics
0(LOW)	0	Alternate Data
0	1	Stack
1(HIGH)	0	Code or None
1	1	Data
S6 is 0 (LOW)	-	-

8086 Status Signal Characteristics (S3-S6)

BH E/S7 - (*Pin Number: 34 & Type: O*)

BUS HIGH ENABLE/STATUS → During T1 the bus high enable signal (BHE) should be used to enable data onto the most significant half of the data bus, pins D15-D08. Eight-bit oriented devices tied to the upper half of the bus would normally use BHE to condition chip select functions. 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 S7 status information is available during T2, T3, and T4.

The signal is active LOW, and floats to 3-state OFF in "hold". It is LOW during T1 for the first interrupt acknowledge cycle.

BH E	A0	Characteristics
0	0	Whole word
0	1	Upperbyte from/to oddaddress
1	0	Lower byte from / to even address
1	1	None

Table 2.3: 8086 *BH E* Signal Characteristics

RD - (*Pin Number: 32 & Type: O*)

READ → Read strobe indicates that the processor is performing a memory or I/O read cycle, de-

pending on the state of the S2 pin. This signal is used to read devices which reside on the 8086 local bus. RD is active LOW during T2, T3 and TW of any read cycle, and is guaranteed to remain HIGH in T2 until the 8086 local bus has floated. This signal floats to 3-state OFF in "hold acknowledge".

READY - (*Pin Number: 22 & Type: I*)

Ready is the acknowledgement from the addressed memory or I/O device that it will complete the data transfer. The READY signal from memory/IO is synchronized by the 8284A Clock Generator to form READY. This signal is active HIGH. The 8086 READY input is not synchronized. Correct operation is not guaranteed if the setup and hold times are not met.

INTR - (*Pin Number: 18 & Type: I*)

INTERRUPT REQUEST → It is a level triggered input which is sampled during the last clock cycle of each instruction to determine if the processor should enter into an interrupt acknowledge operation. A subroutine is vectored to 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.

$\overline{TEST}$ **- (*Pin Number: 23 & Type: I*)**

This input is examined by the "Wait" instruction. If the *T EST* 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.

NMI - (*Pin Number: 17 & Type: I*)

NON-MASKABLE INTERRUPT → An edge triggered input which causes a type 2 interrupt. A subroutine is vectored to via an interrupt vector lookup table located in system memory. NMI is not maskable internally by software. A transition from LOW to HIGH initiates the interrupt at the end of the current instruction. This input is internally synchronized.

RESET - (*Pin Number: 21 & Type: I*)

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, as described in the Instruction Set description, when RESET returns LOW. RESET is internally synchronized.

CLK - (*Pin Number: 19 & Type: I*)

CLOCK → It provides the basic timing for the processor and bus controller. It is asymmetric with a 33% duty cycle to provide optimized internal timing.

V*C C* - (*Pin Number: 40 & Type: -*)→ +5V power supply pin.
GND - (*Pin Number: 1,20 & Type: -*)→ Ground.

$\overline{\text{MN}}$/*MX* - (*Pin Number: 33 & Type: I*)

MINIMUM/MAXIMUM → Indicates what mode the processor is to operate in. The Two modes are discussed in the following sections.

Maximum Mode Signals

The following pin function descriptions are for the 8086/8288 system in maximum mode (i.e., $\overline{\text{MN}}$/*M X* =V*SS*). Only the pin functions which are unique to maximum mode are described; all other pin functions are as described above.

RQ/GT 0**,** *RQ/GT* 1 **- (*Pin Number: 30,31 & Type: I/O*)**

REQUEST/GRANT → These pins are used by other local bus masters to force 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. *RQ/GT* pins have internal pull-up resistors and may be left unconnected. The request/grant sequence is as follows (refer figure2.9: REQUEST/GRANT SEQUENCE TIMING(MAXIMUM MODE ONLY))

1. A pulse of 1 CLK wide from another local bus master indicates a local bus request ("hold") to the 8086 (pulse 1).
2. During a T4 or T1 clock cycle, a pulse 1 CLK wide from the 8086 to the requesting master (pulse 2), indicates that the 8086 has allowed the local bus to float and that it will enter the "hold acknowledge" state at the next CLK. The CPU's bus interface unit is disconnected logically from the local bus during "hold acknowledge".
3. A pulse 1 CLK wide from the requesting master indicates to the 8086 (pulse 3) that the "hold" request is about to end and that the 8086 can reclaim the local bus at the next CLK. Each master-master exchange of the local bus is a sequence of 3 pulses. There must be one dead CLK cycle after each bus exchange. Pulses are active LOW.

Request/Grant sequence timing (Maximum mode only)

If the request is made while the CPU is performing a memory cycle, it will release the local bus during T4 of the cycle when all the following conditions are met:

- Request occurs on or before T2.
- Current cycle is not the low byte of a word (on an odd address).
- Current cycle is not the first acknowledge of an interrupt acknowledge sequence.
- A locked instruction is not currently executing.

If the local bus is idle when the request is made the two possible events will follow:

- Local bus will be released during the next clock.
- A memory cycle will start within 3 clocks. Now the four rules for a currently active memory cycle apply with condition number 1 already satisfied.

$S2, \overline{S1}, \overline{S0}$ - **(*Pin Number: 26-28 & Type: O*)**

STATUS→ These signals are active during T4, T1, and T2 and is returned to the passive state (1, 1, 1) during T3 for during TW when READY is HIGH. This status is used by the 8288 Bus Controller to generate all memory and I/O access control signals. Any change by $S2$, $S1$ or $S0$ during T4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T3 or TW is used to indicate the end of a bus cycle. These signals float to 3-state OFF in "hold acknowledge". These status lines areencoded as shown .

$S2$	$S1$	$S0$	**Characteristics**
0(LOW)	0	0	Interrupt Acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1(HIGH)	0	0	Code Access
1	0	1	Read Memory
1	1	0	Write Memory
1	1	1	Passive

LOCK - **(*Pin Number: 29 & Type: O*)**

This output indicates that other system bus masters are not to gain control of the system bus while *LOC K* is active LOW. The *LOC K* signal is activated by the "LOCK" prefix instruction and remains active until the completion of the next instruction. This signal is active LOW, and floats to 3-state OFF in "hold acknowledge".

QS1, QS0 - (*Pin Number: 24,25 & Type: O*)

QUEUE STATUS → 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 (refer Table 2.5: 8086 QUEUE Status Signal Characteristics).

QS1	QS0	Characteristics
0(LOW)	0	No Operation
0	1	First Byte of Opcode from queue
1(HIGH)	0	Empty the queue
1	1	Subsequent byte from queue

Table 2.5: 8086 QUEUE Status Signal Characteristics

Minimum Mode Signals

The following pin function descriptions are for the 8086 in minimum mode (i.e., MN/*MX* =V*C C*). Only the pin functions which are unique to minimum mode are described; all other pin functionsare as described above.

M/*IO*$\overline{\ }$ - (*Pin Number: 28 & Type: O*)

Memory/IO →This signal logically equivalent to S2 in the maximum mode. It is used to distinguish a memory access from an I/O access. M/*IO* becomes valid in the T4 preceding a bus cycle and remains valid until the final T4 of the cycle (M=HIGH, IO=LOW). M/*IO* floats to 3-state OFF in local bus "hold acknowledge".

W R$\overline{\ }$ - (*Pin Number: 29 & Type: O*)

WRITE → This signal indicates that the processor is performing a write memory or write I/O cycle, depending on the state of the M/*I O* signal. *W R* is active for T2, T3 and TW of any write cycle. It is active LOW, and floats to 3-state OFF in local bus "hold acknowledge".

INT A$\overline{\ }$ - (*Pin Number: 24 & Type: O*)

Used as a read strobe for interrupt acknowledge cycles. It is active LOW during T2, T3 and TW ofeach interrupt acknowledge cycle.

ALE - (*Pin Number: 25 & Type: O*)

ADDRESS LATCH ENABLE →Provided by the processor to latch the address into the 8282/8283 address latch. It is a HIGH pulse active during T1 of any bus cycle. Note that ALE is never floated.

DT/*R* -$\overline{\ }$ (*Pin Number: 27 & Type: O*)

DATA TRANSMIT/RECEIVE → Needed in minimum system that desires to use an 8286/8287 data bus transceiver. It is used to control the direction of data flow through the transceiver. Logically DT/*R* is equivalent to *S*1 in the maximum mode, and its timing is the same as for M/*I O*. (T=HIGH, R=LOW.) This signal floats to 3-state OFF in local bus "hold acknowledge".

DEN –**(*Pin Number: 27 & Type: O*)**

DATA ENABLE → Provided as an output enable for the 8286/8287 in a minimum system which uses the transceiver. *DEN* is active LOW during each memory and I/O access and for *INTA* cycles. For a read or *INTA* cycle it is active from the middle of T2 until the middle of T4, while for a write cycle it is active from the beginning of T2 until the middle of T4. DEN floats to 3-state OFF in local bus"hold acknowledge".

HOLD, HLDA - (*Pin Number: 31,30 & Type: I/O*)

It indicates that another master is requesting a local bus "hold." To be acknowledged, HOLD must be active HIGH. The processor receiving the "hold" request will issue HLDA (HIGH) as an acknowledge- ment in the middle of a T4 or T*i* clock cycle. Simultaneous with the issuance of HLDA the processor will float the local bus and control lines. After HOLD is detected as being LOW, the processor will
LOWer the HLDA, and when the processor needs to run another cycle, it will again drive the local bus and control lines. Hold acknowledge (HLDA) and HOLD have internal pull-up resistors.

The same rules as for RQ/GT apply regarding when the local bus will be released. HOLD is not an asynchronous input. External synchronization should be provided if the system cannot otherwise guarantee the setuptime.

3.(b)Explain about procedure and macro.

Procedures

When a group of instructions are to be used several times to perform a same function in a program, then we can write them as a separate subprogram called procedure or subroutine. Whenever required the procedures can be called in a program using CALL instruction.The procedures are written and assembled as separate program modules and stored in memory. When a procedure is called in the main program, the program control is transferred to procedure and after executing the procedure the program control is transferred back to main program. In 8086, the instruction CALL is used to call a procedure in the main program and the instruction RET is used to return the control to main program.

Syntax

Declare	Call A Procedure
Procedure_name PROC — Statement — RET	— — CALL Procedure_name — —

Procedure Syntax

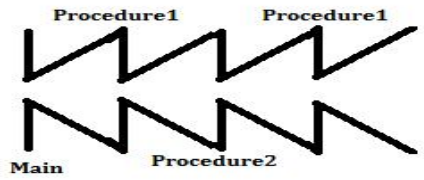
Types Procedure Based on CALL

The 8086 has 2 types of call instruction intra (within a segment) & inter (outside a seg- ment).Based on the call any one of the RET instruction is used near or far. This near & far procedure works like call instruction i.e storing of CS and IP andect.,.

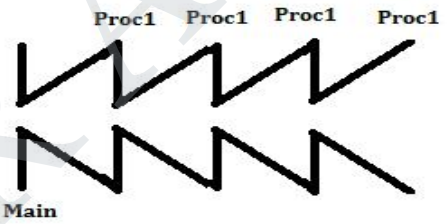
1. FAR Procedure → Outside a Segment
2. NEAR Procedure → Within a segment

Types Procedure Based on Execution Flow

1. *Reentrant Procedure* → Procedure1 calls procedure2 and procedure2 again calls procedure1. This type of call is said to be "REENTRANT" procedure.
2. *Recursive Procedure* → A recursive procedure is a procedure which by call itself. Here the no.of calls is said to be depth of procedure.



Reentrant Procedure



Recursive Procedure

Advantages

- The machine codes for the group of instructions in the procedure has to be pun in memory only once.
- We canpass the parameters usingregisters/memory.

Disadvantages

- The procedure are the need of stack, and the overhead time required to call the procedure and return to the calling program.
- It uses a keyword CALL to call the procedure.

Macros

When a group of instruction are to be used several times to perform a same function in a program and they are too small to be written as a procedure, then they can be defined as a *MACRO*. A macro is a small group of instructions enclosed by the assembler directives MACRO & ENDM.

The macro are identified by their name and usually defined at the start of a program. The macro is called by its name in the main program. Whenever the macro is called in the program, the assembler will insert the defined group of instruction in place of the call.In other words the macro call is like short hand expression which tells the assembler, *"Every time a macro name in the program, replace it with the group of instruction defined as macro"*. Actually the assembler generate machine codes for the group of instructions whenever it is called in the main program.

Syntax

Declare	Call A Macro
macro_name MACRO ___ Statement ___ ENDM	___ ___ macro_name ___ ___

MacroSyntax

Advantages

- The machine codes is generated every time when macro is called. So execution time is reduced compare to procedure.
- Parameters are passed as part of statement.

Disadvantages

- Required more memory compare to procedure.

Comparison of Procedure & Macro

Procedure	Macro
1. Accessed by CALL & RET mechanism during program execution.	1. Accessed during assembly with name given to macro when defined.
2. Machine code for instruction are stored in memory once.	2. Machine codes are generated for instruction in the macro each time it is called.
3. Parameters are passed in register, memory locations or stack.	3. Parameters are passed as part of statement which calls Macro.
4. Required more to execute compare to macro due to CALL and RET keyword.	4. Execution time is minimum compare to procedure because program control is not transferred.
5. Required minimum amount of memory compare to macro.	5. Required more memory compare to procedure. Because machine code generated every time macro is called.

4(a). Explain about IO processor

8089 is an IO processor, designed to work with Intels x86 family of processors. It communicate with the host processors using a memory table, which contains the details of the task to be executed which is prepared by the host to allot a task to the IOP. The IOP reads the memory table to get the details of the allotted task. This memory table has an address of a program written in 8089 instructions, know as a channel program. It can fetch and execute its instruction on its own and after completing the task; it either interrupts the CPU or maintain the busy flag in the memory based table.

8089 operated in tightly coupled or loosely coupled configuration. I n tightly coupled, the 8089 shares the system bus and memory with the host using its *RQ*/*GT* pins. In loosely coupled, 8089 has its own local bus and communicates with the host using arbiter and controller. In loosely coupled *RQ*/*GT* pin of 8089 may be used to communicate with other 8089.

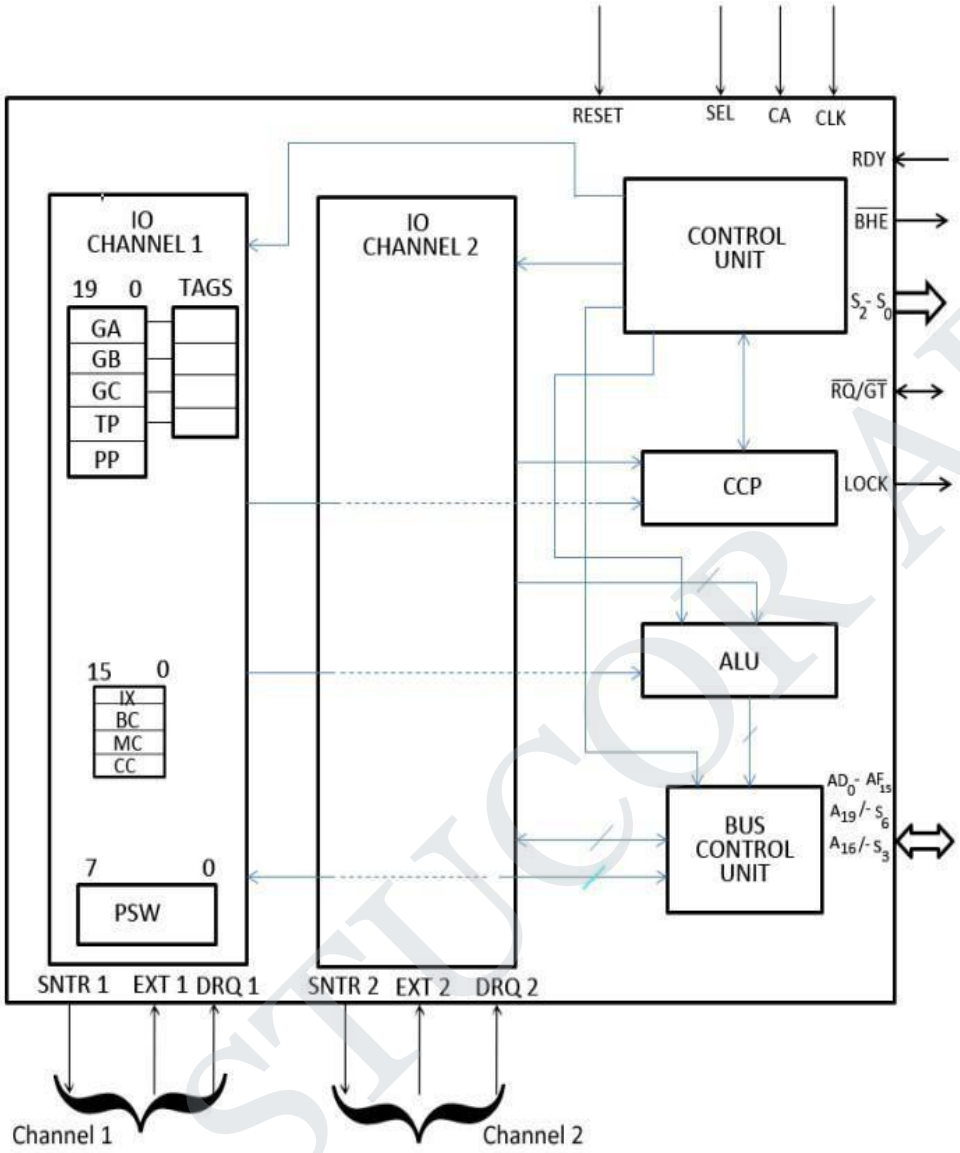
The 8089 IOP uses only 16 address line so address range up to 64Kb. The 8089 handled IO devices need not have the same bus it can even handle 8-bit IO devices.

The 8089 has two internal IO channels which can be programmed independently, to handle two separate IO tasks for the host CPU. The Common ALU is shared by both the channels. The common control unit derives the control signals required for the operation of the IOP channels. The bus interface control unit handles all the busactivities.

The Channel Control Pointer (CCP) available for programmers. It automatically gets loaded with the 20-bit address of a memory table for the channel. The address table for channel 2 is calculated by adding 8 to the contents of CCP. After preparing the memory table; the host asserts Channel Attention (CA) signal and simultaneously selects one of the two channels using the SEL line. The SEL line usually connected to the A0 line of the host CPU.

The two channels are identical in their organization and may be used interchangeably for each other. Each of the channel has two sets of registers i.e., pointers and registers. The pointers are 20-bit registers normally used to address memory, while the registers are 16-bit general purpose data registers. Each of the pointer, except PP, has a tag bit assigned with each of them. This bit indicates whether the 20-bit register contains is to be used or the lower 16-bit register content is to be used as the pointer.

The registers GA, GB, GC, BC, IX and MC can also be used as general purpose registers in the channel program, ff they are not used as pointers for any special function. The memory operands can be accessed using one of the pointers. GA and GB can be used for source and destination pointers respectively for DMA operations. The DQR and EXT pins are used for data transfer control and



8089 Architecture Diagram

Operation termination signals using during DMA operations. The SINTR pins are used for data transfer control and operation termination signals during DMA operations.The SINTR pins are used by the channels either to inform the CPU that the previous operation is over or to ask for its attention or interference if required, before the completion of the task. The task pointer (TP) stores the address of the next instruction to be executed and is equivalent to the PC in a CPU. It also has a tag bit for indicating whether the next instruction is stored in the system or I/O s p a c e . The parameter pointer (PP) is not programmable by the user, but is automatically filled by the 8089 while initializing a task.Each Channel also has an 8-bit program status register which contains the current channel status. This status concerns such things as the source and destination address widths, channel activity, interrupt control and servicing, bus load limit and priority. The PSW is not user accessible directly, but can be modified usingchannel commands.

The most important feature of the IOP is its ability to perform DMA operations with a great flexibility and number of options. The direct transfer between an 8-bit peripheral and a 16-bit destination or source is possible with 8089. These option are programmed using the channel control register that contains flags for the transfer type, transfer mode, synchronized control, source/destination indicator, lock control, chaining control, single transfer mode and termination control.

4 (b). Explain in details about closely coupled configuration and loosely coupled configuration. Write about multiprocessor configuration.

Coprocessors and closely coupled configurations are similar - both the CPU and the external processor share:

Memory

I/O system

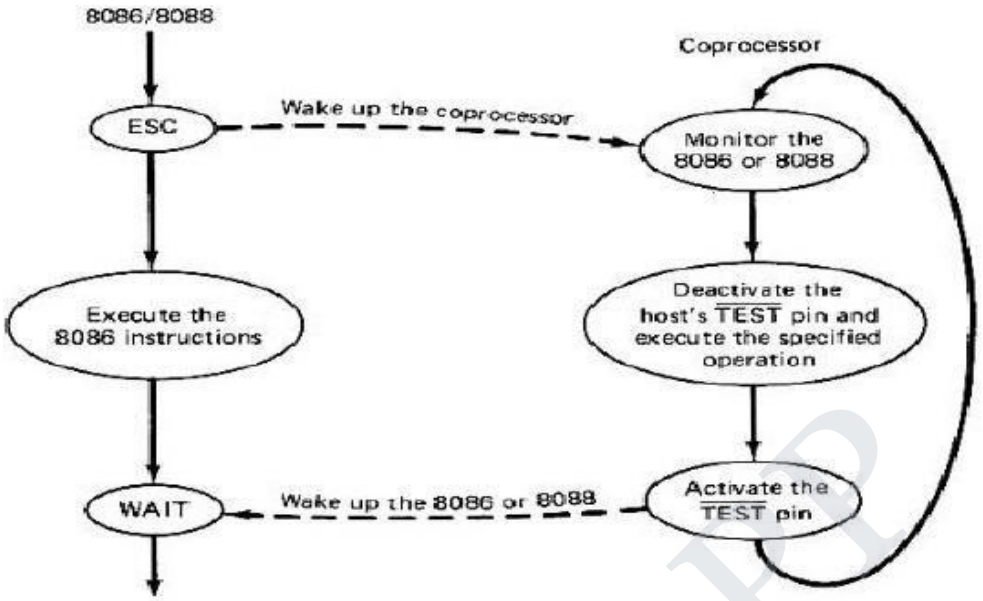
Bus & bus control logic

Clock generator

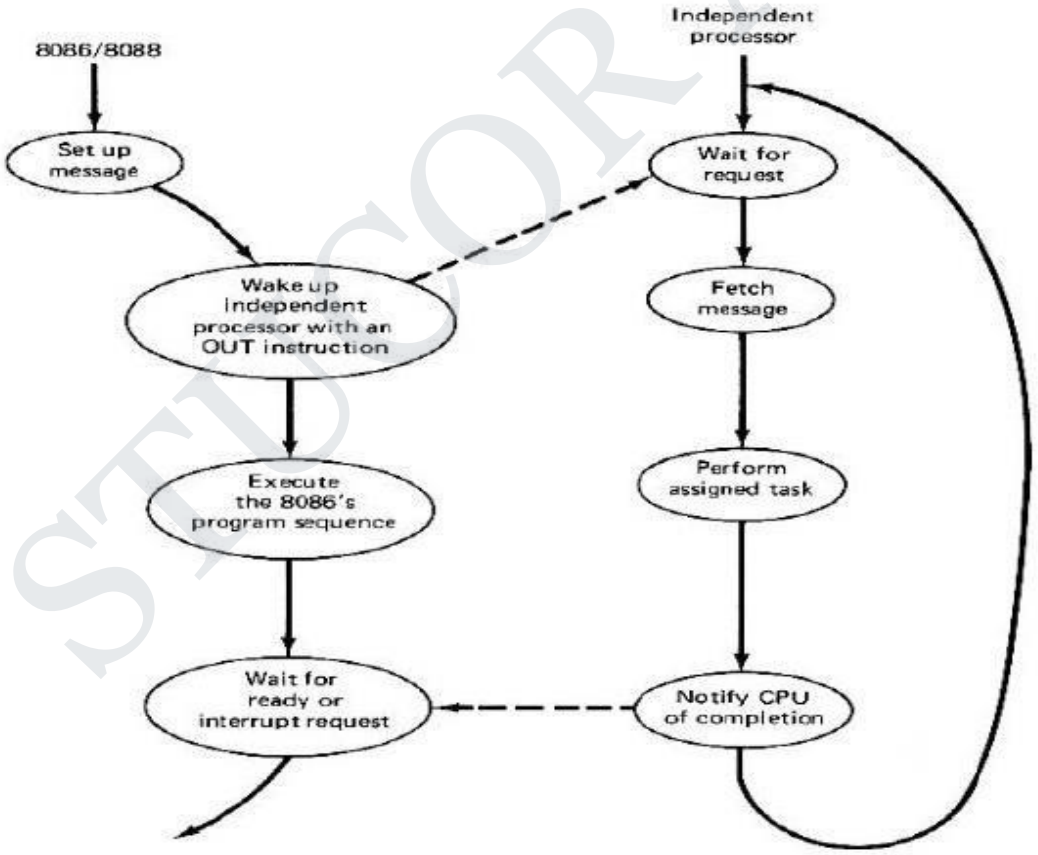
Coprocessor Configuration:

WAIT instruction allows the processor to synchronize itself with external hardware, eg., waiting for 8087 math co-processor. The CPU executes WAIT waiting state.

TEST input is asserted (low), the waiting state is completed and execution will resume. ESC instruction: ESC opcode, operand, opcode : immediate value recognizable to a coprocessor as an instruction opcode .Operand: name of a register or a memory address (in any mode) when the CPU executes the ESC instruction, the processor accesses the memory operand by placing the address on the address bus. If a coprocessor is configured to share the system bus, it will recognize the ESC instruction and therefore will get the opcode and the operand .



Synchroniztion on between the 8086 and its coprocessor.



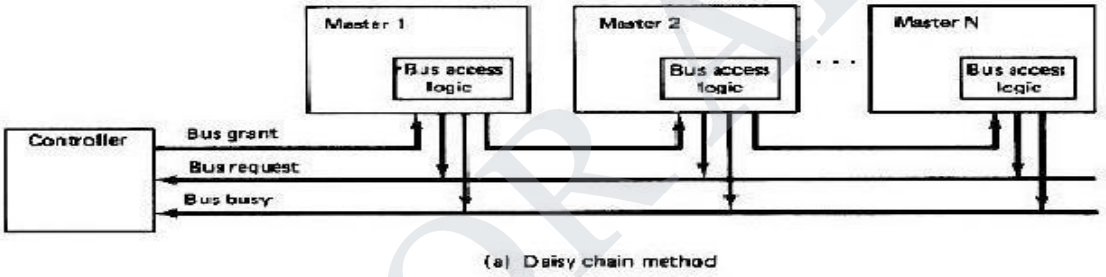
A loosely coupled configuration provides the following advantages:

1. High system throughput can be achieved by having more than one CPU.
2. The system can be expanded in a modular form. Each bus master module is an independent unit and normally resides on a separate PC board. Therefore, a bus master module can be added or removed without affecting the other modules in the system.

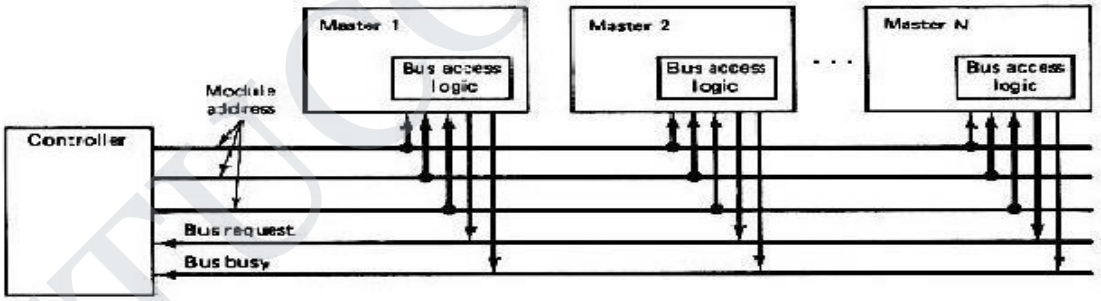
3. A failure in one module normally does not cause a breakdown of the entire system and the faulty module can be easily detected and replaced.

4. Each bus master may have a local bus to access dedicated memory or I/O devices so that a greater degree of parallel processing can be achieved. More than one bus master module may have access to the shared system bus Extra bus control logic must be provided to resolve the bus arbitration problem. The extra logic is called bus access logic and it is its responsibility to make sure that only one bus master at a time has control of the bus.

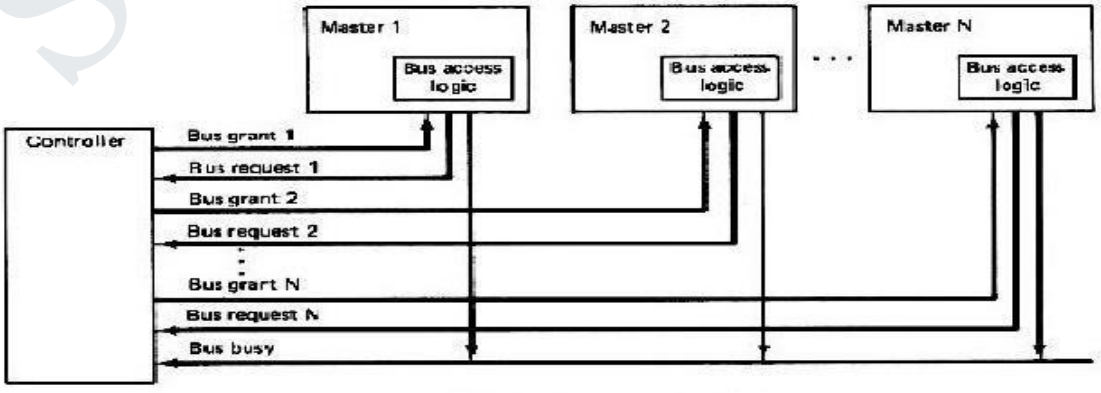
1. Daisy chaining.
2. Polling.
3. Independent requesting



(a) Daisy chain method



(b) Polling method



(c) Independent requests method

5.(a) Explain the function of Programmable Peripheral Interface (8255)

- The parallel input-output port chip 8255 is also called as programmable peripheral input-output port. The Intel's 8255 is designed for use with Intel's 8- bit, 16-bit and higher capability microprocessors. It has 24 input/output lines which may be individually programmed in two groups of twelve lines each, or three groups of eight lines. The two groups of I/O pins are named as Group A and Group B. Each of these two groups contains a subgroup of eight I/O lines called as 8-bit port and another subgroup of four lines or a 4-bit port. Thus Group A contains an 8-bit port A along with a 4-bit port. C upper.
- The port A lines are identified by symbols PA0-PA7 while the port C lines are identified as PC4-PC7. Similarly, Group B contains an 8-bit port B, containing lines PB0-PB7

and a 4-bit port C with lower bits PC0- PC3. The port C upper and port C lower can be used in combination as an 8-bit port C.

- Both the port C are assigned the same address. Thus one may have either three 8-bit I/O ports or two 8-bit and two 4-bit ports from 8255. All of these ports can function independently either as input or as output ports. This can be achieved by programming the bits of an internal register of 8255 called as control word register (CWR).

• The internal block diagram and the pin configuration of 8255 are shown in fig.

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

• The signal description of 8255 are briefly presented as follows :

- PA7−PA0: These are eight port A lines that acts as either latched output or buffered input lines depending upon the control word loaded into the control word register.

• PC7−PC4 : Upper nibble of port C lines. They may act as either output latches or input buffers lines.

• This port also can be used for generation of handshake lines in mode 1 or mode 2.

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

• PB0−PB7 : These are the eight port B lines which are used as latched output lines or buffered input lines in the same way as port A.

- RD : This is the input line driven by the microprocessor and should be low to indicate read operation to 8255.
- WR : This is an input line driven by the microprocessor. A low on this line indicates write operation.

- CS : This is a chip select line. If this line goes low, it enables the 8255 to respond to RD and WR signals, otherwise RD and WR signal areneglected.
- A1−A0 : These are the address input lines and are driven by the microprocessor. These lines A1-A0 with RD, WR and CS from the following operations for8255.

These address lines are used for addressing any one of the four registers, i.e. three ports and a control word register as given in table below.

- In case of 8086 systems, if the 8255 is to be interfaced with lower order data bus, the A0 and A1 pins of 8255 are connected with A1 and A2 respectively.
- D0−D7 : These are the data bus lines those carry data or control word to/from the microprocessor.
- RESET : A logic high on this line clears the control word register of 8255. All ports are set as input ports by default after reset.

Block Diagram of 8255

- It has a 40 pins of 4 groups.

1. Data bus buffer
2. Read Write control logic
3. Group A and Group B controls
4. Port A, B and C

- Data bus buffer: This is a tristate bidirectional buffer used to interface the 8255 tosystem databus. Data is transmitted or received by the buffer on execution of inputor output instruction by the CPU.
- Control word and status information are also transferred through this unit.
- Read/Write control logic: This unit accepts control signals (RD, WR) and also inputs from address bus and issues commands to individual group of control blocks (Group A, Group B).
- It has the following pins.

a) CS – Chip select : A low on this PIN enables the communication between CPU and 8255.

b) RD (Read) – A low on this pin enables the CPU to read the data in the ports or the status word through data bus buffer.

c) WR (Write) : A low on this pin, the CPU can write data on to the ports or on to the control register through the data bus buffer.

d) RESET: A high on this pin clears the control register and all ports are set to the input mode

e) A0 and A1 (Address pins): These pins in conjunction with RD and WR pins control the selection of one of the 3 ports.

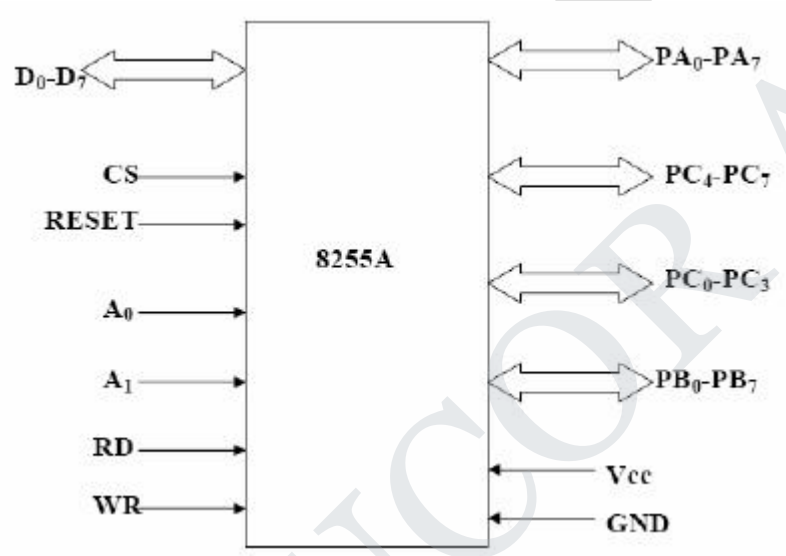
- Group A and Group B controls : These block receive control from the CPU and issues commands to their respective ports.
- Group A - PA and PCU (PC7 –PC4)

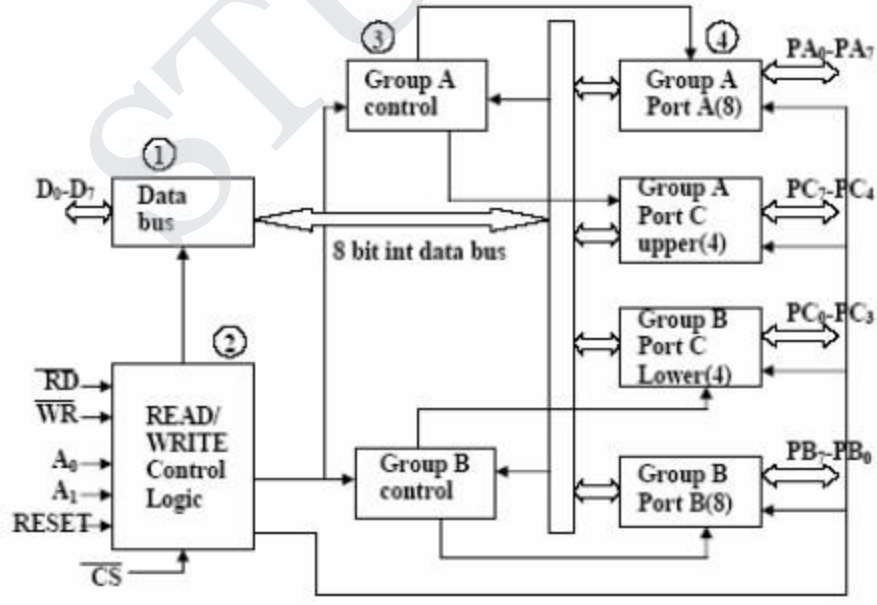
• Group B - PCL (PC3 – PC0)Control word register can only be written into no read operation of the CW register is allowed.

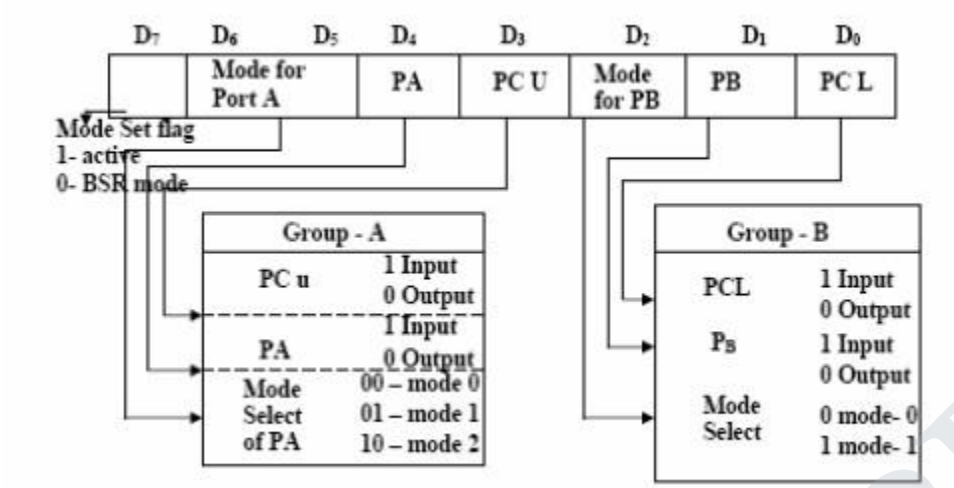
a) Port A: This has an 8 bit latched/buffered O/P and 8 bit input latch. It can be programmed in 3 modes – mode 0, mode 1, mode 2.

b) Port B: This has an 8 bit latched / buffered O/P and 8 bit input latch. It can be programmed in mode 0, mode 1.

c) Port C : This has an 8 bit latched input buffer and 8 bit out put latched/buffer. This port can be divided into two 4 bit ports and can be used as control signals for port A and port B. it can be programmed in mode 0.







Modes of Operation of 8255

- There are two basic modes of operation of 8255. I/O mode and Bit Set-Reset mode (BSR).
- In I/O mode, the 8255 ports work as programmable I/O ports, while in BSR mode only port C (PC0-PC7) can be used to set or reset its individual port bits.
- Under the I/O mode of operation, further there are three modes of operation of 8255, so as to support different types of applications, mode 0, mode 1 and mode 2.
- BSR Mode: In this mode any of the 8-bits of port C can be set or reset depending on D0 of the control word. The bit to be set or reset is selected by bit select flags D3, D2 and D1 of the CWR as given in table.

- **I/O Modes :**

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

- The salient features of this mode are as listed below:

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

- All these modes can be selected by programming a register internal to 8255 known as CWR.
- The control word register has two formats. The first format is valid for I/O modes of operation, i.e. modes 0, mode 1 and mode 2 while the second format is valid for bit set/ reset (BSR) mode of operation.

b) **Mode 1:** (Strobed input/output mode) In this mode the handshaking control the input and output action of the specified port. Port C lines PC0-PC2, provide strobe or handshake lines for port B. This group which includes port B and PC0-PC2 is called as group B for Strobed data input/output. Port C lines PC3-PC5 provide strobe lines for port A. This group

including port A and PC3-PC5 from group A. Thus port C is utilized for generating handshake
signals. The salient features of mode 1 are listed as follows:

1. Two groups – group A and group B are available for strobed data transfer.
2. Each group contains one 8-bit data I/O port and one 4-bit control/data port.
3. The 8-bit data port can be either used as input and output port. The inputs and outputs both are latched.
4. Out of 8-bit port C, PC0-PC2 are used to generate control signals for port B and PC3-PC5 are used to generate control signals for port A. the lines PC6, PC7 may be used as independent data lines.

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

Input Control Signal Definitions (Mode 1):

- STB (Strobe input) – If this lines falls to logic low level, the data available at 8- bit input port is loaded into input latches.
- IBF (Input buffer full) – If this signal rises to logic 1, it indicates that data has been loaded into latches, i.e. it works as an acknowledgement. IBF is set by a low on STB and is reset by the rising edge of RD input.
- INTR (Interrupt request) – This active high output signal can be used to interrupt the CPU whenever an input device requests the service. INTR is set by a high STB pin and a high at IBF pin. INTE is an internal flag that can be controlled by the bit set/reset mode of either PC4(INTEA) or PC2(INTEB) as shown in fig.

- INTR is reset by a falling edge of RD input. Thus an external input device can be request the service of the processor by putting the data on the bus and sending the strobe signal.

Output Control Signal Definitions (Mode 1) :

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

Mode 2 (Strobed bidirectional I/O): This mode of operation of 8255 is also called as strobed bidirectional I/O. This mode of operation provides 8255 with an additional features for communicating with a peripheral device on an 8-bit data bus. Handshaking signals are provided to maintain proper data flow and synchronization between the data transmitter and receiver. The interrupt generation and other functions are similar to mode 1

- In this mode, 8255 is a bidirectional 8-bit port with handshake signals. The RD and WR

- signals decide whether the 8255 is going to operate as an input port or output port.

• The salient features of Mode 2 of 8255 are listed as follows:

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

• **Control signal definitions in mode 2:**

- INTR – (Interrupt request) As in mode 1, this control signal is active high and is used to interrupt the microprocessor to ask for transfer of the next data byte to/from it. This signal is used for input (read) as well as output (write) operations.
- OBF (Output buffer full) – This signal, when falls to low level, indicates that the CPU has written data to port A.

• ACK (Acknowledge) This control input, when falls to logic low level, acknowledges that the previous data byte is received by the destination and next byte may be sent by the processor. This signal enables the internal tristate buffers to send the next data byte on port A.

- INTE1 (A flag associated with OBF) This can be controlled by bit set/reset mode with PC6.

• Control signals for input operations :

- STB (Strobe input) A low on this line is used to strobe in the data into the input latches of 8255.
- IBF (Input buffer full) When the data is loaded into input buffer, this signal rises to logic '1'. This can be used as an acknowledge that the data has been received by the receiver.

• The waveforms in fig show the operation in Mode 2 for output as well as input port.

• Note: WR must occur before ACK and STB must be activated before RD.

5.(b). Explain the function of KEYBOARD/DISPLAY INTERFACE (8279)

- Intel's 8279 is a general purpose keyboard display controller that simultaneously drives the display of a system and interfaces a keyboard with the CPU, leaving it free for its routine task.

Architecture and Signal Descriptions of 8279

• The keyboard display controller chip 8279 provides:

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

• Fig shows the functional block diagram of 8279 followed by its brief description.

- I/O Control and Data Buffers : The I/O control section controls the flow of data to/ from the 8279. The data buffers interface the external bus of the system with internal bus of 8279.

•The I/O section is enabled only if CS is low. The pins A0, RD and WR select the command, status or data read/write operations carried out by the CPU with 8279.

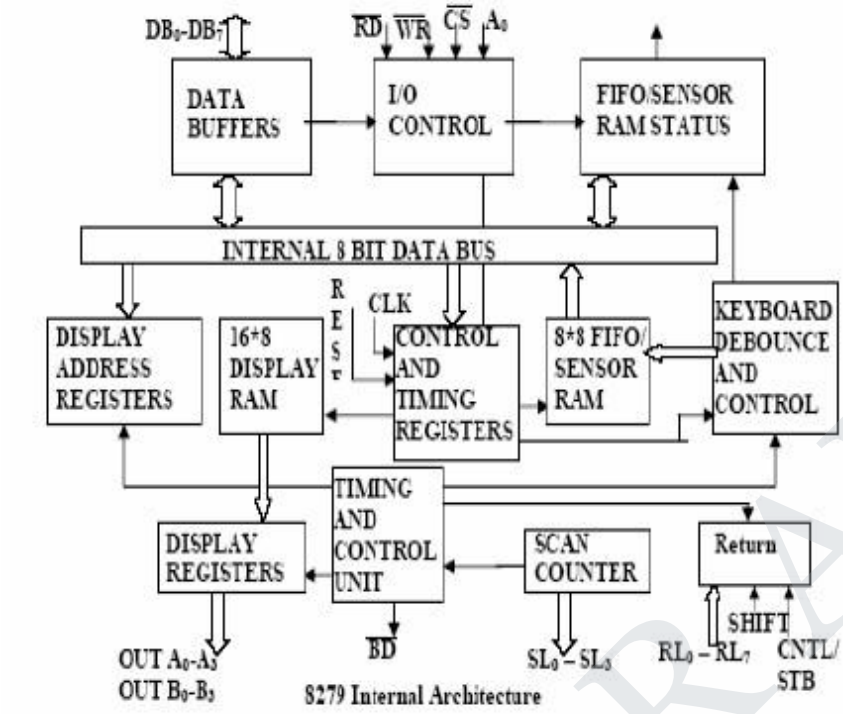
- Control and Timing Register and Timing Control : These registers store the keyboard and display modes and other operating conditions programmed by CPU.

The registers are written with A0=1 and WR=0. The Timing and control unit controls the basic timings for the operation of the circuit. Scan counter divide down the operating frequency of 8279 to derive scan keyboard and scan display frequencies.

- Scan Counter : The scan counter has two modes to scan the key matrix and refresh the display. In the encoded mode, the counter provides binary count that is to be externally decoded to provide the scan lines for keyboard and display (Four externally decoded scan lines may drive upto 16 displays). In the decode scan mode, the counter internally decodes the least significant 2 bits and provides a decoded 1 out of 4 scan on SL0-SL3(Four internally decoded scan lines may drive upto 4 displays). The keyboard and display both are in the same mode at a time.
- Return Buffers and Keyboard Debounce and Control: This section for a key closure row wise. If a key closer is detected, the keyboard debounce unit debounces the key entry (i.e. wait for 10 ms). After the debounce period, if thekey continues to be detected. The code of key is directly transferred to the sensor RAM along with SHIFT and CONTROL key status.

- **FIFO/Sensor RAM and Status Logic**: In keyboard or strobed input mode, this block acts as 8-byte first-in-first-out (FIFO) RAM. Each key code of the pressed key is entered in the order of the entry and in the mean time read by the CPU, till the RAM become empty.
- The status logic generates an interrupt after each FIFO read operation till the FIFO is empty. In scanned sensor matrix mode, this unit acts as sensor RAM. Each row of the sensor RAM is loaded with the status of the corresponding row of sensors in the matrix. If a sensor changes its state, the IRQ line goes high to interrupt the CPU.

- **Display Address Registers and Display RAM** : The display address register holds the address of the word currently being written or read by the CPU to or from the display RAM. The contents of the registers are automatically updated by 8279 to accept the next data entry by CPU.



8279 Internal Architecture

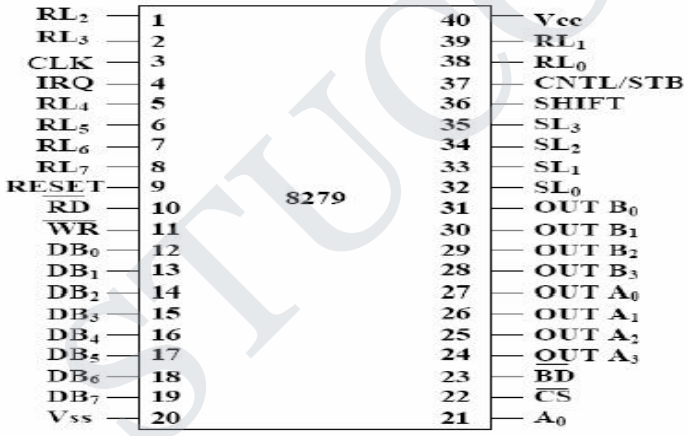
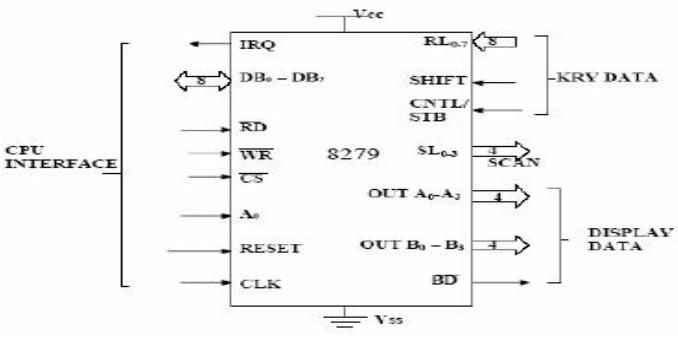


Fig. 5.38 : 8279 Pin Configuration



- The signal discription of each of the pins of 8279 as follows :
- DB0-DB7 : These are bidirectional data bus lines. The data and command words to and from the CPU are transferred on these lines.
- CLK : This is a clock input used to generate internal timing required by 8279.
- RESET : This pin is used to reset 8279. A high on this line reset 8279. After resetting 8279, its in sixteen 8-bit display, left entry encoded scan, 2-key lock out mode. The clock prescaler is set to 31.

- CS : Chip Select – A low on this line enables 8279 for normal read or write operations. Other wise, this pin should remain high.
- A0 : A high on this line indicates the transfer of a command or status information .A low on this line indicates the transfer of data. This is used to select one of the internal registers of 8279.
- RD, WR (Input/Output) READ/WRITE – These input pins enable the data buffers to receive or send data over the data bus.
- IRQ : This interrupt output lines goes high when there is a data in the FIFO sensor RAM. The interrupt lines goes low with each FIFO RAM read operation but if the FIFO RAM further contains any key-code entry to be read by the CPU, this pin again goes high to generate an interrupt to the CPU.
- Vss, Vcc : These are the ground and power supply lines for the circuit.
- SL0-SL3-Scan Lines : These lines are used to scan the key board matrix and display digits. These lines can be programmed as encoded or decoded, using the mode control register.
- RL0 - RL7 - Return Lines : These are the input lines which are connected to one terminal of keys, while the other terminal of the keys are connected to the decoded scan lines. These are normally high, but pulled low when a key is pressed.
- SHIFT : The status of the shift input lines is stored along with each key code in FIFO, in scanned keyboard mode. It is pulled up internally to keep it high, till it is pulled low with a key closure.
- BD – Blank Display : This output pin is used to blank the display during digit switching or by a blanking closure.
- OUT A0 – OUT A3 and OUT B0 – OUT B3 – These are the output ports for two 16*4 or 16*8 internal display refresh registers. The data from these lines is synchronized with the scan lines to scan the display and keyboard. The two 4-bit ports may also as one 8-bit port.
- **CNTL/STB- CONTROL/STROBED I/P Mode** : In keyboard mode, this lines is used as a control input and stored in FIFO on a key closure. The line is a strobed lines that enters the data into FIFO RAM, in strobed input mode. It has an interrupt pull up. Modes of Operation of 8279
- The modes of operation of 8279 are as follows :

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

- Input (Keyboard) Modes : 8279 provides three input modes. These modes are as

follows:

1. Scanned Keyboard Mode : This mode allows a key matrix to be interfaced using either encoded or decoded scans. In encoded scan, an 8*8 keyboard or in decoded scan, a 4*8 keyboard can be interfaced. The code of key pressed with SHIFT and CONTROL status is stored into the FIFO RAM.
2. Scanned Sensor Matrix : In this mode, a sensor array can be interfaced with 8279 using either encoded or decoded scans. With encoded scan 8*8 sensor matrix or with decoded scan 4*8 sensor matrix can be interfaced. The sensor codes are stored in the CPU addressable sensor RAM.
3. Strobed input: In this mode, if the control lines goes low, the data on return lines, is stored in the FIFO byte by byte.

- **Output (Display) Modes** : 8279 provides two output modes for selecting the display options. These are discussed briefly.

1. **Display Scan** : In this mode 8279 provides 8 or 16 character multiplexed displays those can be organized as dual 4- bit or single 8-bit display units.
2. **Display Entry** : (right entry or left entry mode) 8279 allows options for data entry on the displays. The display data is entered for display either from the right side or from the left side.

Keyboard Modes

i. **Scanned Keyboard mode with 2 Key Lockout** : In this mode of operation, when a key is pressed, a debounce logic comes into operation. During the next two scans, other keys are checked for closure and if no other key is pressed the first pressed key is identified.

- The key code of the identified key is entered into the FIFO with SHIFT and CNTL status, provided the FIFO is not full, i.e. it has at least one byte free. If the FIFO does not have any free byte, naturally the key data will not be entered and the error flag is set.The lines is pulled down with a key closer.
- If FIFO has at least one byte free, the above code is entered into it and the 8279 generates an interrupt on IRQ line to the CPU to inform about the previous key closures. If another key is found closed during the first key, the keycode is entered in FIFO.
- If the first pressed key is released before the others, the first will be ignored. A key code is entered to FIFO only once for each valid depression, independent of other keys pressed along with it, or released before it.
- If two keys are pressed within a debounce cycle (simultaneously), no key is recognized till one of them remains closed and the other is released. The last key, that remains depressed is considered as single valid key depression.

ii. **Scanned Keyboard with N-Key Rollover** : In this mode, each key depression is treated independently. When a key is pressed, the debounce circuit waits for 2 keyboards scans and then checks whether the key is still depressed. If it is still depressed, the code is entered in FIFO RAM.

Any number of keys can be pressed simultaneously and recognized in the order, the keyboard scan recorded them. All the codes of such keys are entered into FIFO.

In this mode, the first pressed key need not be released before the second is pressed. All the keys are sensed in the order of their depression, rather in the order the keyboard scan senses them, and independent of the order of their release.

iii. Scanned Keyboard Special Error Mode : This mode is valid only under the **NKey rollover mode**. This mode is programmed using end interrupt / error mode set command. If during a single debounce period (two keyboard scans) two keys are found pressed, this is considered a simultaneous depression and an error flag is set.

- This flag, if set, prevents further writing in FIFO but allows the generation of further interrupts to the CPU for FIFO read. The error flag can be read by reading the FIFO status word. The error Flag is set by sending normal clear command with CF = 1.

iv. Sensor Matrix Mode : In the sensor matrix mode, the debounce logic is inhibited. The 8-byte FIFO RAM now acts as 8 * 8 bit memory matrix. The status of the sensor switch matrix is fed directly to sensor RAM matrix. Thus the sensor RAM bits contains the row-wise and column wise status of the sensors in the sensor matrix.

- The IRQ line goes high, if any change in sensor value is detected at the end of a sensor matrix scan or the sensor RAM has a previous entry to be read by the CPU.

The IRQ line is reset by the first data read operation, if AI = 0, otherwise, by issuing the end interrupt command. AI is a bit in read sensor RAMword.

Display Modes

- There are various options of data display. For example, the command number of characters can be 8 or 16, with each character organised as single 8-bit or dual 4-bit codes. Similarly there are two display formats.
- The first one is known as left entry mode or type writer mode, since in a type writer the first character typed appears at the left-most position, while the subsequent characters appear successively to the right of the first one. The other display format is known as right entry mode, or calculator mode, since in a calculator the first character entered appears at the rightmost position and this character is shifted one position left when the next characters is entered.
- Thus all the previously entered characters are shifted left by one position when a new characters is entered.

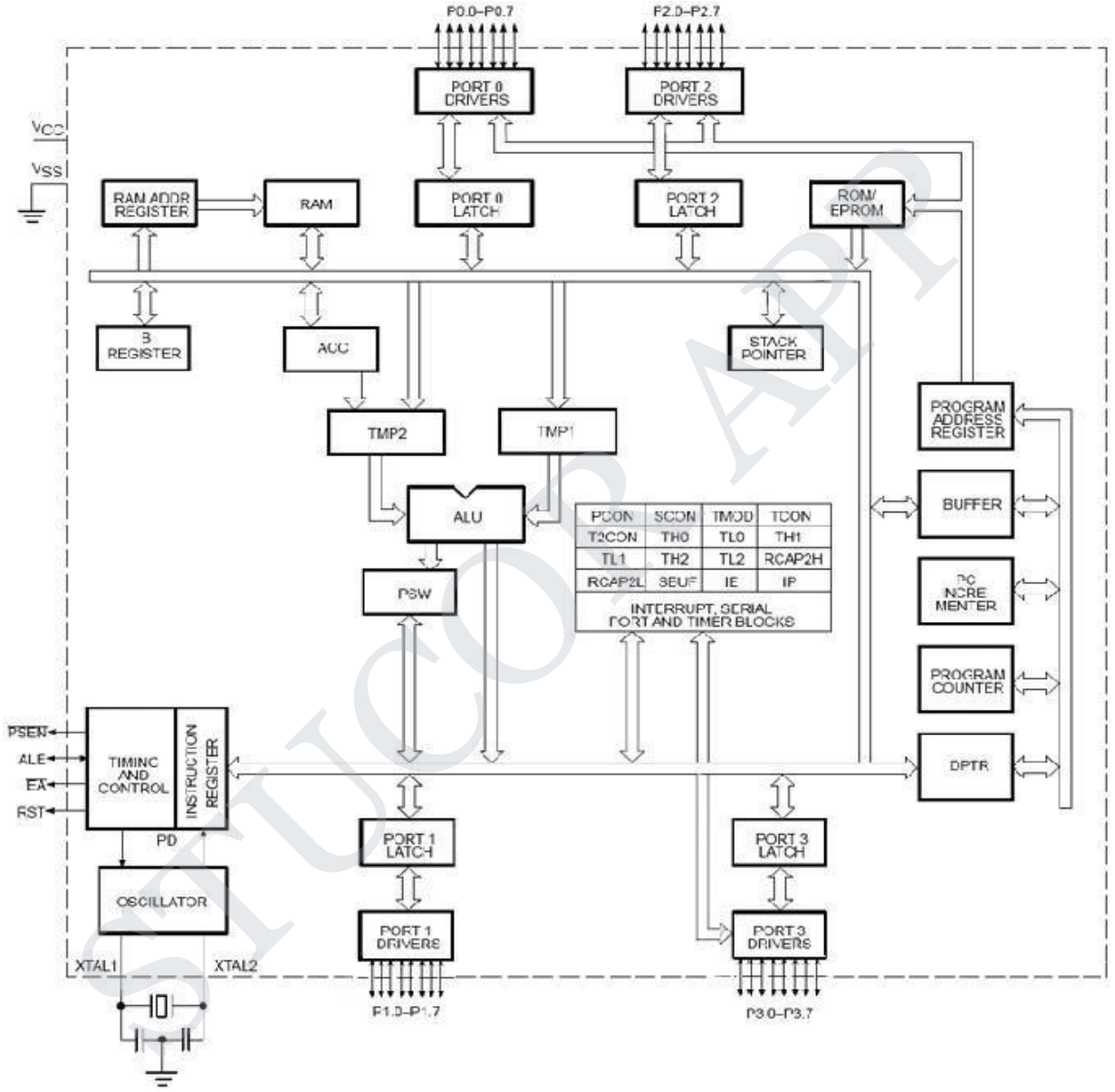
i. **Left Entry Mode** : In the left entry mode, the data is entered from left side of the display unit. Address 0 of the display RAM contains the leftmost display characters and address 15 of the RAM contains the right most display characters.

It is just like writing in our address is automatically updated with successive reads or writes. The first entry is displayed on the leftmost display and the sixteenth entry on the rightmost display. The seventeenth entry is again displayed at the leftmost display position.

ii. **Right Entry Mode** : In this right entry mode, the first entry to be displayed is entered on the rightmost display. The next entry is also placed in the right most display but after the previous display is shifted left by one display position. The leftmost characters is shifted out of that display at the seventeenth entry and is lost, i.e. it is pushed out of the display RAM.

6(a). Explain 8051 architecture.

The INTEL 8051 is the 8-bit single chip 40 pin microcontroller and are available in N-channel metal oxide silicon (NMOS)and complementary metal oxide silicon (CMOS) construction. It is de- signed for use in sophisticated real -time instruction and industrial control. It operates with 12MHz clock and single+5V power supply.



8051 Microcontroller Architecture Diagram

Accumulator

The accumulator register (ACC or A) acts as an operand register, in case of some instructions. This may either be implicit or specified in the instruction. The Accumulator, as its name suggests, is used as a general register to accumulate the results of a large number of instructions. The ACC register has been allotted an address in the on-chip special function register bank.

The Register Bank (R-registers)

The R-registers are a set of eight registers that are named R0, R1, etc. up to and including R7. These registers are used as auxiliary registers in many operations. To continue with the above example, perhaps you are adding 10 and 20. The original number 10 may be stored in the Accumulator whereas the value 20 may be stored in, say, register R4. To process the addition with the help of following instruction: **ADD A, R4**

After executing this instruction the Accumulator will contain the value 30. The R- registers are very important auxiliary, or "helper", registers. The Accumulator alone would not be very useful. The R-registers are also used to temporarily store values. For example, lets say we want to add the values in R1 and R2 together and then subtract the values of R3 and R4. One way to do this would be:

→ MOV A,R3;Move the value of R3 into the accumulator
→ ADD A,R4 ;Add the value of R4
→ MOV R5,A ;Store the resulting value temporarily in R5
→ MOV A,R1 ;Move the value of R1 into the accumulator
→ ADD A,R2 ;Add the value of R2
→ SUBB A,R5 ;Subtract the value of R5 (which now contains R3 + R4)

As we used R5 to temporarily hold the sum of R3 and R4. Of course, this is not the most efficient way to calculate (R1+R2) - (R3 +R4) but it does illustrate the use of the R- registers as a way to store values temporarily.

B Register

The B-register is very similar to the Accumulator in the sense that it may hold an 8-bit (1- byte) value. This register is only used by two 8051 instructions: MUL AB and DIV AB. Thus, if we want to quickly and easily multiply or divide A-register by another number, we have to store the other number in B-register and make use of these two instructions. In other instruction, it may just be used as a scratch pad. This register is considered as a special function register.

Program Status Word (PSW)

The PSW Special Function Register (SFR) contains the carry flag, the auxiliary carry flag, the overflow flag, and the parity flag it may set or reset based on the operation of microcontroller. Additionally, the PSW register contains the register bank select flags which are used to select which of the R-register banks are currently used.

Programming Tip **:** when we write an interrupt handler routine, it is a very good idea to always save the PSW SFR on the stack and restore it when our interrupt is complete. Many 8051 instructions modify the bits of PSW. If our interrupt routine does not guarantee that PSW is the same upon exit as it was upon entry, our program is bound to behave rather erratically and unpredictably–and it will be tricky to debug since the behavior will tend not to make any sense.

CY	AC	F0	RS1	RS0	OV	---	P

- CY : Carry Flag.
- AC : Auxiliary Carry Flag.
- F0 : Flag 0 (available for user).
- RS1 : Register Select 1.
- RS0 : Register Select 0.
- OV : Arithmetic Overflow Flag.
- P : Accumulator Parity Flag.

RS1	RSO	Register Bank	Address
0	0	0	00h - 07h
0	1	1	08h - 0Fh
1	0	2	10h - 17h
1	1	3	18h - 1Fh

Format of PSW

ALU

The arithmetic and logical unit performs 8-bit arithmetic and logical operations over the operands held by the temporary registers TEMP1 and TEMP2. Users can not access these temporary registers. Based on the result of ALU the flags in PSW may be modified.

Program Counter

The Program Counter (PC) is 16-bit register which tells the 8051 where the next instruction to execute is found in memory. When the 8051 is initialized PC always starts at 0000h and is incremented each time an instruction is executed. It is important to note that PC is not always incremented by one. Since some instructions require 2 or 3 bytes the PC will be incremented by 2 or 3 in these cases. The Program Counter is special in that there is no way to

directly modify its value.

Data Pointer (DPTR)

It is the 8051s only user-accessible 16-bit (2-byte) register. As the name suggests, it is used to point to data. It is used by a number of commands which allow the 8051 to access external memory. When the 8051 accesses external memory it will access external memory at the address indicated by DPTR. While DPTR is most often used to point to data in external memory, many programmers often take advantage of the fact that its the only true 16-bit register available.

Data Pointer High (DPH)/Data Pointer Low (DPL)- The SFRs DPL and DPH work together to represent a 16-bit value called the Data Pointer. Since it is an unsigned two-byte integer value, it can represent values from 0000H to FFFFH (0 through 65,535 decimal).

Programming Tip **:** DPTR is really DPH and DPL taken together as a 16-bit value. In reality, we almost always have to deal with DPTR one byte at a time. For example, to push DPTR onto the stack we must first push DPL and then DPH. We can't simply push DPTR onto the stack. Additionally, there is an instruction to "increment DPTR." When we execute this instruction, the two bytes are operated upon as a 16-bit value. However, there is no instruction that decrements DPTR. If we wish to decrement the value of DPTR, we must write our own code to do so.

Stack Pointer (SP)

This 8-bit wide register is incremented before the data is stored onto the stack using push or call instruction. This register contains 8-bit stack top address. The stack may defined anywhere in the on-chip 128 byte RAM i.e the stack refers to an area of internal RAM that is used in conjunction with certain opcodes to store and retrieve data quickly. After reset, the SP register is initialised to 07H.

After each write to stack operation, the 8-bit contents of the operands are stored onto the stack, after incrementing the SP register by one. Thus if SP contains 18H, the forthcoming PUSH operation will store the data at address 19H in the internal RAM. The SP content will be incremented to 19H. As data is retrieved from the stack, the byte is read from the stack, and then the SP decremented to point to the next available byte of stored data. The 8051 stack is not a top-down data structure, like other INTEL processors. This register has also been allotted an address in the special function
register bank.

tack & Stack Pointer Microcontroller(μC)	tack & Stack Pointer Microprocessor(μP)
Stack pointer is a 8-bit register	Stack pointer is a 16-bit register
Stack is a group of internal RAM memory location	Stack is a group of memory location
When data is to be placed on a stack, the SP is incremented (↑)	When data is to be placed on a stack, the SP is decremented (↓)
When data is to be retrieved, the SP is decremented (↓)	When data is to be retrieved, the SP is incremented (↑)
Stack capacity is limited. (The program- mer is to be careful to limit its growth - 80 locations only i.e., 80 bytes)	Stack capacityis higher

μC Stack Vs μP Stack

Port 0 to 3 Latches and Drivers

These four latches and drivers are allotted to each of the four on-chip I/O ports. These latches have been allotted addresses in the special function register bank. Using the allotted addresses, the user can communicate with these ports. These are identified as Port0, Port1, Port2 and Port3.

Serial Data Buffer

The serial data buffer internally contains two independent registers. One of them is a transmit buffer which is necessarily a parallel-in serial-out register. The other is called receive buffer which is a serial-in parallel-out register. Loading a byte to the transmit buffer initiates serial transmission of that byte. The serial data buffer is identified as SBUF and is one of the special function registers. If a byte is written to SBUF, it initiates serial communication and if the SBUF is read, it reads received serial data.

Timer Register

These two 16-bit registers can be accessed as their lower and upper bytes. For example, TL0 represents the lower byte of the timing register 0, while TH0 represents higher bytes of the timing register 0. Similarly, TL1 and TH1 represents lower and higher bytes of timing register 1. All these registers can be accessed using the four addresses allotted to them which lie in the special function registers.

Control Registers

The special function register IP, IE, TMOD, TCON, SCON and PCON contain control and status information for interrupts, timers/controle and serial ports. All of these registers have been allotted addressed in the special function register.

Timing & Control Unit

This unit derives all the necessary timing and control signals required for the internal operation of the circuit. It also derives control signals required for controlling the external system bus.

Oscillator

This circuit generates the basic timing clock signal for the operation of the circuit using crystal oscillator.

Instruction Register

This register decodes the opcode of an instruction to be executed and gives information to the timing and control unit to generate necessary signals for the execution of the instruction.

EPROM & Program Address Register

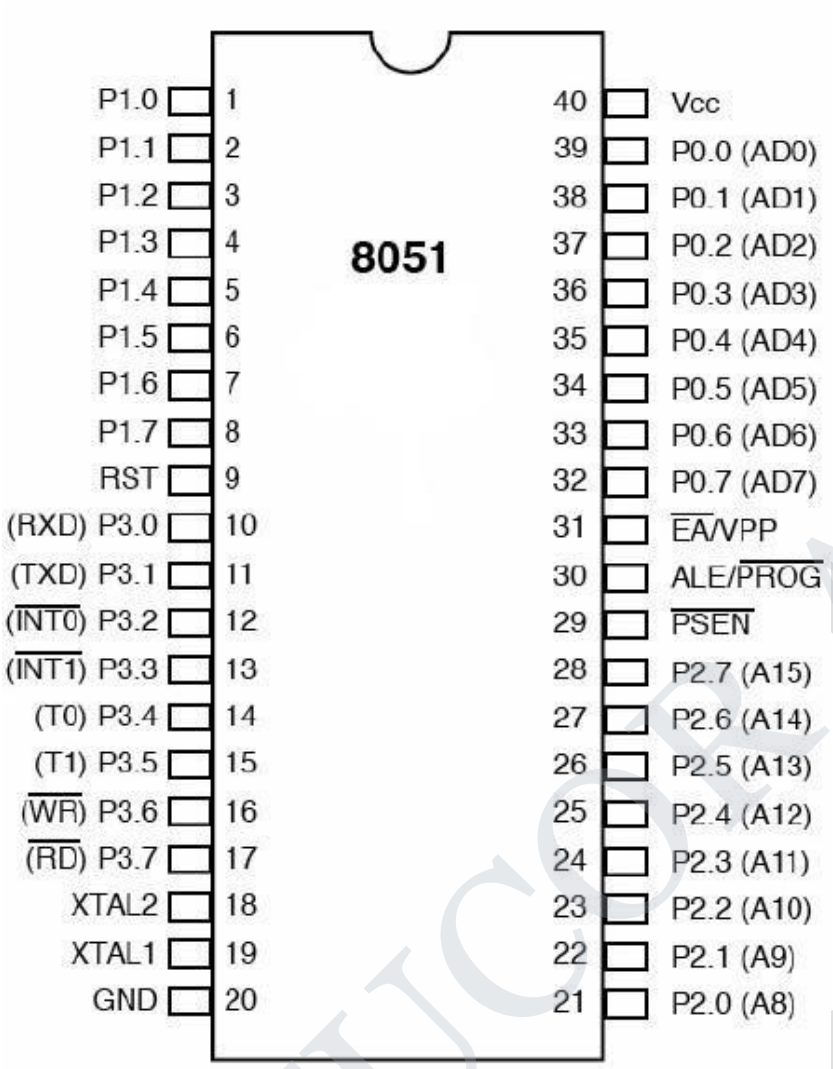
These blocks provide an on-chip EPROM and a mechanism to internally address it.

RAM & RAM Address Register

These clocks provide internal 1287 bytes of RAM and a mechanism to address it internally.

Special Function Registers (SFR)-Register Bank

This is a set of special function registers, which can be addressed using their respective addresses which lie in the range 80H to FFH.



V*CC*: This is a +5V power supply voltage pin.

V*SS*: This is a return pin for the supply.

RESET: The reset input pin resets the 8051, only when it goes high for two or more machine cycles. For a proper reinitialization after reset, the clock must be running.

ALE/PROG: The address latch enable output pulse indicates that the valid address bits are avail- able on their respective pins. This ALE signal is valid only for external memory accesses. Normally, the ALE pulses are emitted at a rate of one-sixth of the oscillator frequency. This pin acts as program pulse input during on-chip EPROM programming. ALE may be used for external timing or clocking purpose. One ALE pulse is skipped during each to external data

memory.

EA/V*P P* **:** External access enable pin, if ties low, indicates that the 8051 can address external program memory. In other words, the 8051 can execute a program in external memory, only if $\overline{EA}$ is tied low. For execution of program in internal memory, the $\overline{EA}$ must be tied high. This pin also receives 21 volts for programming of the on-chip EPROM.

P SEN **:** Program store enable is an active-low output signal that acts as a strobe to read the external program memory. This goes low during external program memory accesses.

Port 0 (P0.0 - P0.7): Port 0 is an 8-bit bidirectional bit addressable I/O port. This has been allotted an address in the SFR address range. Port 0 acts as multiplexed address/data lines during external memory access. In case of controllers with on-chip EPROM, port 0 receives code bytes during programming of the internal EPROM.

Port 1 (P1.0 - P1.7): Port 1 acts as an 8-bit bidirectional bit addressable port. This has been allotted an address in the SFR address range.

Port 2 (P2.0 - P2.7): Port 2 acts as 8-bit bidirectional bit addressable I/O port. It has been allotted an address in the SFR address range of 8051. During external memory access, port 2 emits higher eight bits of address (A_8 - A_{15}) which are valid, if ALE goes high and *EA* is low. Port 2 also receives higher order address bits during programming of the on-chip EPROM.

Port 3 (P3.0 - P3.7): Port 3 is an 8-bit bidirectional bit addressable I/O port which has been allotted an address in the SFR address range of 8051. The port 3 pins also serve the alternative functions as listed below:

P3.0 → Acts as serial input data pin (RXD) *P3.1* → Acts as serial output data pin (TXD) *P3.2* → Acts as external interrupt pin 0 (*IN T* 0) *P3.3* → Acts as external interrupt pin 1 (*IN T* 1) *P3.4* → Acts as external input to timer 0 (T0) *P3.5* → Acts as external input to timer 1 (T1)

P3.6 → Acts as write control signal for external data memory (*W R*)

P3.7 → Acts as read control signal for external data memory read operation (*RD*)

XTAL1 & XTAL2: There is an inbuilt oscillator which derives the necessary clock frequency for the operation of the controller. XTAL1 is the input of amplifier and XTAL2 is the output of the amplifier. A crystal is to be connected externally between these two pins to

complete the feedback path to start oscillations. The controller can be operated on an external clock. In this case the external clock is fed to the controller at pin XTAL2 and XTAL1 pin should be grounded. Commercially available version of 8051 run on 12MHz to 16MHzfrequency.

6.(b). Explain input output ports of 8051.

The I/O section of 8051 microcontroller includes four I/O ports. Each port has D-type O/P latch for each pin. The SFR for each port is made up of these 8 latches, which can be addressed at the SFR address, for that port. The port latches should not be confused with port pins; the data on the latches does not have to be the same as that onthe pins.

The diagrams (refer Figure 3.5: 8051 I/O Ports) are shown the in which two data paths by the circuits that read the latch or pin data using two entirely separate buffers. The upper buffer is enabled when latch data is read and lower buffer is enabled when pin state is read. The status of each latch may be read from a latch buffer while an input buffer is connected directly to each pin, so that pin status may be read independently of latch status.

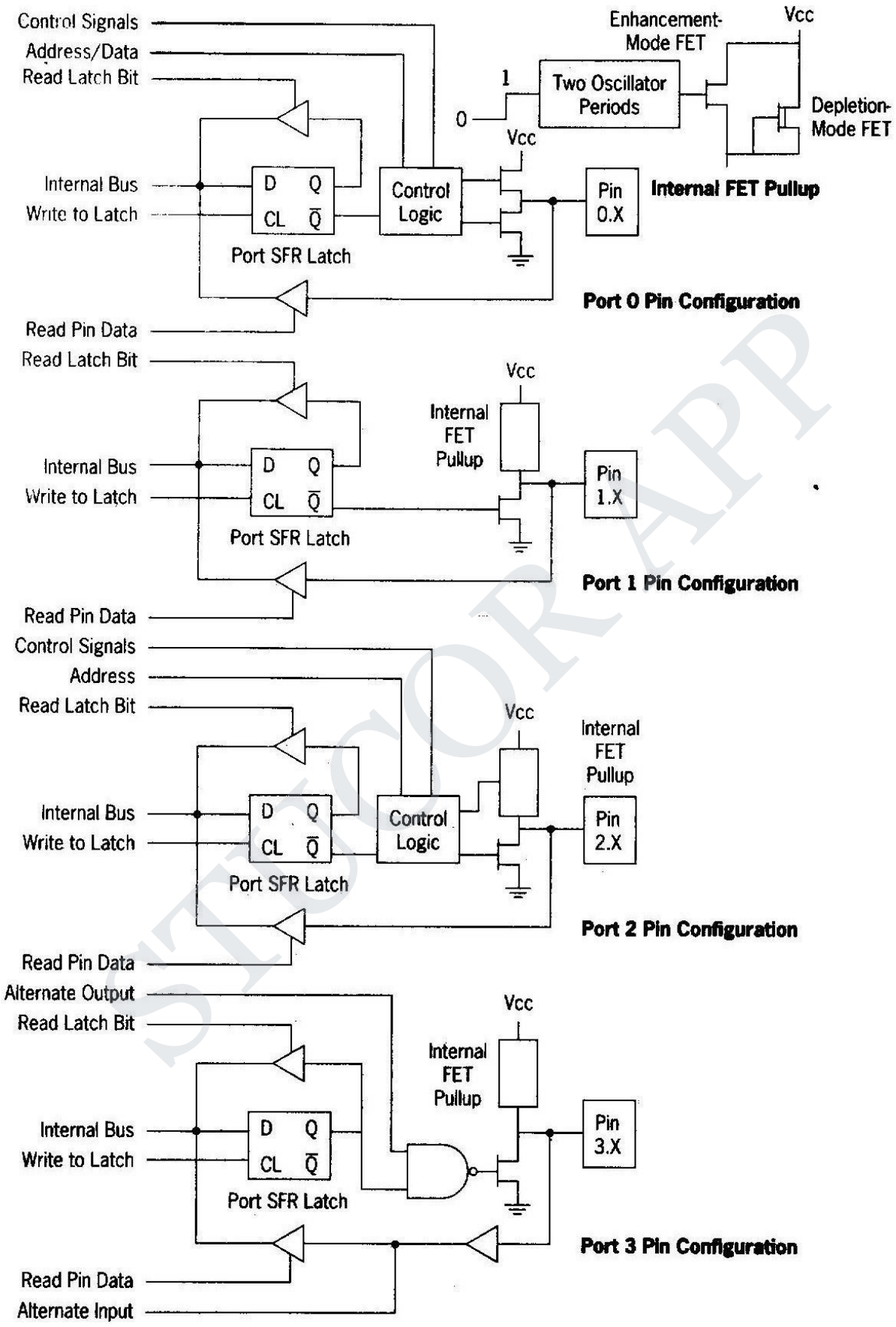
Port 0

It serves as inputs, outputs or when used together it provides low order multiplexed address and data bus for expanding 8051 with external memories and peripherals.For example, when a pin is used as input, 1 must be written to the corresponding port 0 latch by the program thus turning both of the output transmitters off, which in turn causes the pin to *float* in high impedance state and pin is essentially connected to the input buffer.

When port 0 is used as output, the pin latches that are programmed to 0 will turn the lower FET, grounding the pin. When port 0 is used as address bud to external memory, internal control signals switch the address lines to the gates of FET.

Port 1

Port 1 pins have no dual functions. Therefore output latch is connected directly to the gate of lower FET, which FET circuit had labeled internal FET pull-up as an active pull-up load.



8051 I/O Ports

Port 2

It may be used as input/output port similar in operation to port 1. It provides the high order address bus when expanding the 8051 with external program memory. The alternate use of port 2 is to supply a high order address byte in conjunction with port 0 low order byte address to external memory.

Port 2 pins are momentarily changed by the address control signals when supplying the high order byte of 16-bit address. Port 2 latches remain stable when external memory is addressed, as they do not have turned around for data input as the case for port 0.

Port 3

Port 3 is an I/O p o r t similar to port 1. The I/O f u n c t i o n s can be programmed under the control of Port 3 latches or under the control of various other special function registers which can be configured individually. To provide external interrupt it requires inputs, control inputs, serial ports, receiver input and transmitter output and to generate the control signals for reading and writing external data memory.

7.(a) Explain 8051 interrupt structure.

A computer program has only two ways to determine the conditions that exist in internal and external circuits. One method uses software instructions that jump to subroutines on the state of flags and port pins. The second method responds to hardware signals called interrupts that force the program to call a subroutine. Interrupts are often the only way in which the real time programming can be done successfully. Five interrupts are provided in 8051MC. Three internal interrupts and two external interrupts.

Two interrupts are triggered by external signals provided by the circuitry that is connected to pins *I N T* 0 and *I N T* 1 (port pins P3.2 & P3.3). All interrupt functions are under the control of program. The programmer is able to alter control bits in the interrupt Enable *Register* (IE), the *Interrupt Priority Register* (IP)and *Timer Control Register* (TCON). The program can stop all or any combination of the interrupts by suitably setting or cleaning bits in these registers.

- Internal Interrupt → Three are generated automatically by internal operations
  - Timer flag 0
  - Timer flag 1
  - Serial I/O port
- External Interrupt → these two interrupts are controlled by the SFR IE and IP

Internal Interrupts Timer Flag Interrupt

When a timer /counter overflows the corresponding TF0 or TF1 is set to 1. The flag is cleared to 0 when the resulting interrupt generates a program to the appropriate timer subroutine in memory.

Serial Port Interrupt

If a data byte is received as interrupt bit, R1 is set to 1 in the SCON register, When a data byte has been transmitted, an interrupt bit T1is set in SCON.These OR-ed together to provide a single interrupt called serial port intrrupt.These bits are not cleared when the interrupt generated program call is made by the processor. The program that handles serial data communication must read reset R1 or T1 to 0 to enable the next data communication operation.

External Interrupts

Pins $\overline{INT0}$ and $\overline{INT1}$ are used by circuitry. Inputs on these pins can set the interrupt flags IE0 and IE1 in the TCON register to 1 by two different methods. The IEX flags may be set when the INTX pin signal reaches a low level or the flags may be set when high to low transition takes place on the $\overline{INTX}$ pins. Bits IT0 and IT1 in TCON, program the $\overline{INTX}$ pins for low level interrupts when set to 0 and program the $\overline{INTX}$ pins for transition interrupt when set to 1.

Flags IEX will be reset when a transition generated interrupt is accepted by the processor and the interrupt subroutine is accessed. It is the responsibility of the system designer and programmer to reset any level-generated external interrupts when they are serviced by the program. The external circuit must remove the low level before an RET1 is executed.

EA	–	ET2	ES	ET1	EX1	ET0	EX0

EA	IE.7	Disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, interrupt source is individually enable or disabled by setting or clearing its enable bit.
-	IE.6	Not implemented, reserved for future use*.
ET2	IE.5	Reserved for future use*.
ES	IE.4	Enable or disable the Serial port interrupt.
ET1	IE.3	Enable or disable the Timer 1 overflow interrupt.
EX1	IE.2	Enable or disable External interrupt 1.
ET0	IE.1	Enable or disable the Timer 0 overflow interrupt.
EX0	IE.0	Enable or disable External Interrupt 0.

If the bit is 0, the corresponding interrupt is disabled. If the bit is 1, the corresponding interrupt is enabled.

8051 Interrupt Enable Register

Reset

A reset is said to be the ultimate interrupt because the program may not block the action of the voltage on the RST pin. It is often called non-maskable because no combination of bits in any register can stop or mask, the reset action. Unlike other interrupts the PC is not stored for later program resumption; a reset is an absolute command to jump to program address 0000H and commence running from there.

Internal RAM contents may change during Reset,also, the states of internal Ram bytes when power first applied to 8051 are random. Register bank 0 is selected on reset as all bits in PSW are 0.

–	–	PT2	PS	PT1	PX1	PT0	PX0

-	IP.7	Not implemented, reserved for future use*.
-	IP.6	Not implemented, reserved for future use*.
PT2	IP.5	Reserved for future use*.
PS	IP.4	Defines the Serial Port interrupt priority level.
PT1	IP.3	Defines the Timer 1 Interrupt priority level.
PX1	IP.2	Defines External Interrupt priority level.
PT0	IP.1	Defines the Timer 0 interrupt priority level.
PX0	IP.0	Defines the External Interrupt 0 priority level.

If the bit is 0, the corresponding interrupt has a lower priority and if the bit is 1, the corresponding interrupt has a higher priority.

Interrupt control

The program must be able at critical times so that crucial operations can be finished. The IE Register holds the programmable bits to enable or disable all the interrupts as a group.

Often it is desirable to be able set the priorities among competing interrupts that may occur simultaneously. The IP register bits may be set by the program to assign priorities among the various interrupts sources s o that more important interrupts can be serviced first in which two or more interrupts occur at same time.

Interrupt Enable/Disable

Bits in the IE register are set to 1 if the corresponding interrupts source is to be enabled and set to 0 to disable the interrupt source. Bit EA is a master or global bit that can enable or disable all interrupts.

Interrupt Priority

Register IP bits determine if any interrupt is to have a high or low priority. Bits set to 1 assigns high priority and a 0 assigns low priority. Interrupts with high priority can interrupt another interrupt with lower priority, the lower priority interrupt continues after the higher is finished.

If two interrupts with same priority occur at same time, then they have the following ranking

1. IE0
2. TF0
3. IE1
4. TF1
5. Serial = RI or TI

Interrupt destinations

Each interrupt source causes the program to do a hardware call to one of the dedicated addresses in program memory. It is the responsibility of the programmer to place a routine at the address that will service the interrupt.

The interrupt saves the PC of the program, which is running at the time the interrupt is serviced on the stack in internal RAM. A call is then done to the appropriate memory location. Refer Table
3.3 - 8051 Interrupt Priority. At the end of the routine RETI instruction restores the PC to place in The interrupted programs and resets the interrupt logic so that another interrupt can be serviced.

Interrupt	Address(HEX)
IE0	0003H
TF0	000BH
IE1	0013H
TF1	001BH
Serial	0023H

8051 Interrupt Priority

Software Generated Interrupts

When any interrupt flag is set to 1 by any means an interrupt is generated unless blocked. This means that the program itself can cause interrupts of any kind to be generated simply by setting the desired interrupt flag to 1 using a program instruction.

7.(b)Explain microcontroller counters and timers with neat diagram.

Many microcontroller applications require the counting of external events, such as the frequency of pulse train, or the generation of precise internal time delays between computer actions. Both of these tasks can be accomplished using software techniques, but software loops for counting or timing keep the processor occupied so that other, perhaps move important, functions are

not done.

GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
TIMER 1				TIMER 0			

GATE	When TRx (in TCON) is set and GATE = 1, TIMER/COUNTERx will run only while INTx pin is high (hardware control). When GATE = 0, TIMER/COUNTERx will run only while TRx = 1 (software control).
C/$\overline{T}$	Timer or Counter selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from Tx input pin).
M1	Mode selector bit (NOTE 1).
M0	Mode selector bit (NOTE 1).

Note 1 :

M1	M0	OPERATING MODE	
0	0	0	13–bit Timer
0	1	1	16–bit Timer/Counter
1	0	2	8–bit Auto–Reload Timer/Counter
1	1	3	(Timer 0) TL0 is an 8–bit Timer/Counter controlled by the standard Timer 0 control bits, TH0 is an 8–bit Timer and is controlled by Timer 1 control bits.
1	1	3	(Timer 1) Timer/Counter 1 stopped.

8051 Timer Mode Control Register

To relieve the processor of this burden, two 16 bit up counters, named T0 and T1, are provided for the general use of the programmer. Each counter may be programmed t o count internal clock pulses, acting as a timer, or programmed to count external pulses as a counter. The counters are divided into two 8-bit registers called the timer low(TL0,TL1) and high (TH0,TH1) bytes. All counter action is controlled by bit states in the timer mode control register (TMOD), the timer/counter control register (TCON), and certain program instructions.

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. TCON 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.

The counters have been included on the chip to relieve the processor of timing and counting chores. When the program wishes to count a certain number of internal pulses or external events, a number is placed in one of the counters. The number represents the maximum countless the desired count, pulse 1. The counter increments from the initial number to the maximum and then rolls over to 0 on the final pulse and also sets a timer flag. The flag conditions may be tested by an instruction to tell the program that the count has been accomplished, or the flag may be used to interrupt the program.

Timing

If a counter is programmed to be a timer, it will count the internal clock frequency of the 8051 oscillator divided by 12d. As an example, if the crystal frequency is 6.0 Megahertz, then

the timer clock will have a frequency of 500 Kilohertz.

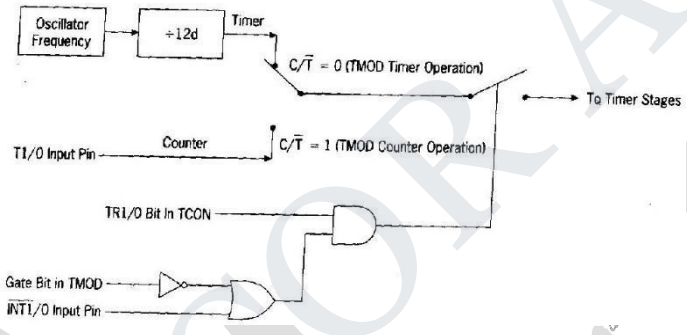
Timer M o d e s of Operation

The timers may operate in any one of four modes that are determined by the mode bits, M1 and M0, in the TMOD register.

Timer Mode 0

Setting timer X mode bits to 00b in the TMOD register results in using the THX registers as an 8-bit counter and TLX as a 5-bit counter; the pulse input is divided by 32d in TL so that TH counts

the original oscillator frequency reduced by a total 384d. As an example, the 6 Megahertz oscillator frequency would result in a final frequency to TH of 15625 hertz. The timer flag is set whenever THX goes from FFh to 00h, or in .0164 seconds for a 6 Megahertz crystal if THX starts at 00h.

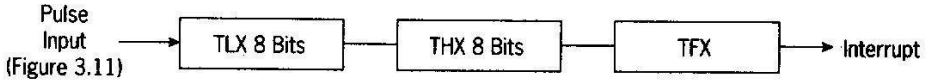


8051 Timer and Counter

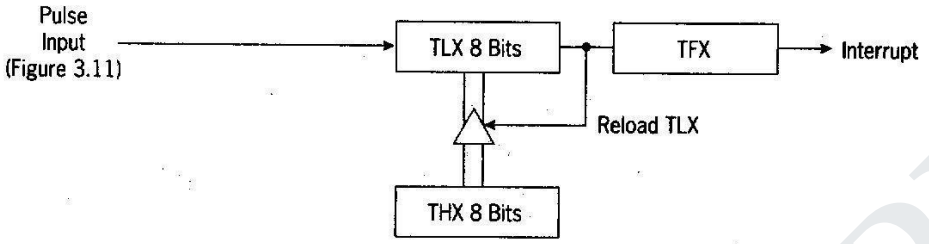
Timer Mode 1

Mode 1 is similar to mode 0 except TLX is configured as a full 8-bit counter when the mode bits are set to 01b in TMOD. The Timer flag would be set in 0.1311 seconds using a 6Megahertz crystal.

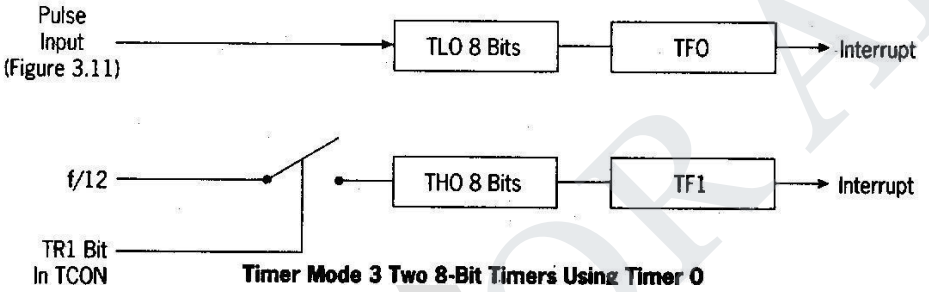
Timer Mode 0 13-Bit Timer/Counter



Timer Mode 1 16-Bit Timer/Counter



Timer Mode 2 Auto-Reload of TL from TH



Timer Mode 3 Two 8-Bit Timers Using Timer 0

8051 Timer Modes

Timer Mode 2

Setting the mode bits to 10b in TMOD configures the timer to use only the TLX counter as an 8-bit counter. THX is used to hold a value that is loaded into TLX ever y time TLX overflows from FFh to 00h. The Timer flag is also set when TLX overflows. This mode exhibits an auto-reload features: TLX will count up from the number in THX, overflow, and be initialized again with the contents of THX.

Timer Mode 3

Timers 0 and 1 may be programmed to be in mode 0, 1, or 2 independently of a similar mode for the other timer. This is not true for mode 3, the timers do not operate independently if mode3 is chosen for timer 0. Placing timer 1 in mode 3 causes it to stop counting; the control bit TR1 and the timer 1 flag TF1 are then used by timer 0.

Timer 0 in mode 3 becomes two completely separate 8-bit counters. TL0 is controlled by the gate arrangement and sets timer flag TF0 whenever it overflows from FFh to 00h. TH0 receives the timer clock (the oscillator divided by 12) under the control of TR1 only and set the TF1 flag when it overflows. Timer 1 may still be used in mode 0, 1, and 2, while timer 0 is in mode 3 with one important exception: No interrupts will be generated by timer 1 while timer 0 is using TF1 overflow flag.

Counting

The only different between counting and timing is the source of the clock pulses to the counters. When used as a timer, the clock pulses are from the oscillator through the divide by 12d circuit. When used as a counter, pin T0 supplies pulses to counter 0, and pin T1 to counter 1. The C/*T bit* in TMOD must be set to 1 to enable pulses from the TX pin to reach the control circuit.