SYLLABUS EE- 309 F MICROPROCESSING AND INTERFACING

Sessional : 50 Marks
Theory : 100 Marks
Total : 150 Marks
Duration of Exam. : 3 Hrs.

Section-A

THE 8085 PROCESSOR : Introduction to microprocessor, 8085 microprocessor, Architecture, instruction set, interrupt structure, and Assembly language programming.

Section-B

THE 8086 MICROPROCESSOR ARCHITECTURE : Architecture, block diagram of 8086, details of sub-blocks such as EU, BIU; memory segmentation and physical address computations, program relocation, addressing modes, instruction formats, pin diagram and description of various signals.

Section-C

INSTRUCTION SET OF 8086 : Instruction execution timing, assembler instruction format, data transfer instructions, arithmetic instructions, branch instructions, looping instructions, NOP and HLT instructions, flag manipulation instructions, logical instructions, shift and rotate instructions, directives and operators, programming examples.

Section-D

INTERFACING DEVICE : 8255 Programmable peripheral interface, interfacing keyboard and seven segment display, 8254 (8253) programmable interval timer, 8259A programmable interrupt controller, Direct Memory Access and 8237 DMA controller.

NOTE : Examiner will set 9 questions in total, with two questions from each section and one question covering all sections which will be Q.1. This Q.1 is compulsary and of short answer type. Each question carries equal mark (20marks). Students have to attempt 5 questions in total at least one question from each section.

MICROPROCESSING & INTERFACING

Dec 2014

Paper Code: EE-309-F

Figures in the bracket indicate full marks.

Note: Attempt five questions in total selecting one question from each of the four Units. Question No.1 is compulsory.

Q.1.(a) Differentiate vectored and non-vectored interrupt.

Ans. Vectored interrupt : In the vecotored interrupt, the branch address is supplied by the external device which interrupts the microprocessor.

When a vectored interrupt occurs, CPU calculates internally, the vector address by multiplying the vector number by 8. For example, if an interrupt in received through RST 5.5 pin of 8085, CPU internally finds the vector address by multiplying finds the vector address by multiplying $5.5 \times 8 = 002C$. The CPU transfer control of 002C (by loading 002C into PC) and then execute the program located at that address onwards.

Non Vectored interrupt : In the non vectored interrupt, the branch address is assigned to fixed location.

When CPU receive a non-vectored interrput, it expect some external device to supply the vector address. For example, if the 8085 CPU is interrupted through INTR pin, then CPU outputs one $\overline{\text{INTR}}$ for which an external circuit should supply an RST instructions opcode or call instructions opcode. If CALL instruction opcode is supplied, then two more $\overline{\text{INTR}}$ are generated during which the complete address is to be supplied.

CPU on receiving the address, branches to that address. If RST instruction is supplied, CPU branches to the RST instructions vector location.

Q.1.(b) What is pipelining ? How does this occur.

Ans. The process of fetching the next instruction, when the present instruction is being executed is called as pipelining.

Pipelining has become possible due to the use of queue. BIU fills in the queue, until the entire queue is full. BIU restarts filling in the queue when atleast two locations of queue are vaccant. The EU always reads the next instruction byte from the que in BIU. This is much faster than sending out an address to the memory and wainting for the next instruction byte to come. In short pipelining eliminates the waiting time of EU and speeds up the procesing .

Q.1.(c) What are the advantages/disadvantages of memory mapped I/O over I/O mapped I/O ? **(4)**

Ans. In Memory Mapped I/O Mode, the larger Instruction set of the 8085A pertaining to memory operations may be used to access I/O devices, in contrast to just two (IN and OUT) Instructions in the I/O Mapped I/O Mode. Also, in the former, data from any register can be moved to a part, or data from a can be moved to any register, while in the I/O Mapped I/O Mode, all data to and from ports have to be routed through the Accumulator. However in Memory Mapped I/O Mode, where I/O devices are addressed as memory, fewer memory locations are

available than in the case of I/O Mapped I/O Mode. Also, while accessing ports, IN and OUT instructions can be excauted with a lesser number of clock cycle (that is, they are faster) than LDA or STA instruction in the Memory Mapped I/O mode because the port address in the case of I/O Mapping is one byte while it is two bytes in Memory Mapping (more time is spent on getting the port address during the execution of the Instruction).

Q.1.(d) Explain the difference between segment and FROC directives.

Ans. SEGMENT directive : An assembly program in .EXE format consists of one or more segments. The art of these segment are defined by SEGMENT directive and the ENDS statement indicates the end of the segment.

Format :
```
name    SEGMENT [options]   ; Begin segment
  .
  .
name    ENDS ;              End segment
```

Example :
```
CODE    SEGMENT             End segement
  .
  .
CODE    ENDS
```

PROC directive : The procedures in the programs can be defined by PROC directive. The procedure name must be present, must be unique, and must follow naming conventions for the langauage. After the PROC directive the term, NEAR or FAR are issued to specify the type of the procedure.

Example : FACT PROC FAR ; Identifies the start of a procedure named FACT and tells the assmebler that the procedure is far (in a segement with a different name from that which contains the instruction which calls the procedure).

Q.1.(e) Explain hidden DMA. **(4)**

Ans. Hidden DMA or transparent DMA : The microprocessor executes some states during which it floats the address and data busses. *e.g.*, T4, T5 and T6 of 8085 opcode fetch cycle. During these states, the microprocessor is isolated form the system bus. The DMA controller transfers data between memory and input/output devices during these states. This operation is transparent to microprocessor. The 8085 microprocessor executes idle machine cycle for some instructions (DAD Rp). During this cycle, the DMA controller can transfer data. This is the slowest DMA transfer. In this mode, the instruction execution speed of microprocessor is not reduced. But, the transparent DMA requires logic the states when the microprocessor is floating the buses.

The DMA controller contains four types of registers viz. address. count. control and status register. The address register is used to hold the starting address of memory. The count register indictates number of bytes to be transferred. Control register is used to set operating modes of the DMA controller and status register provides states of DMA cycle.

Unit-I

Q.2.(a) Describe the Pin diagram of 8085 microprocessor. **(10)**

Ans. Fig. shows functional pin diagram of 8085 microprocessor. The signals of 8085 can be classified into seven groups according to their functions :

(i) Power supply and frequency signals.
(ii) Data bus and address bus
(iii) Control bus
(iv) Interrupt signals
(v) Serial I/O signals
(vi) DMA signals
(vii) Reset signals

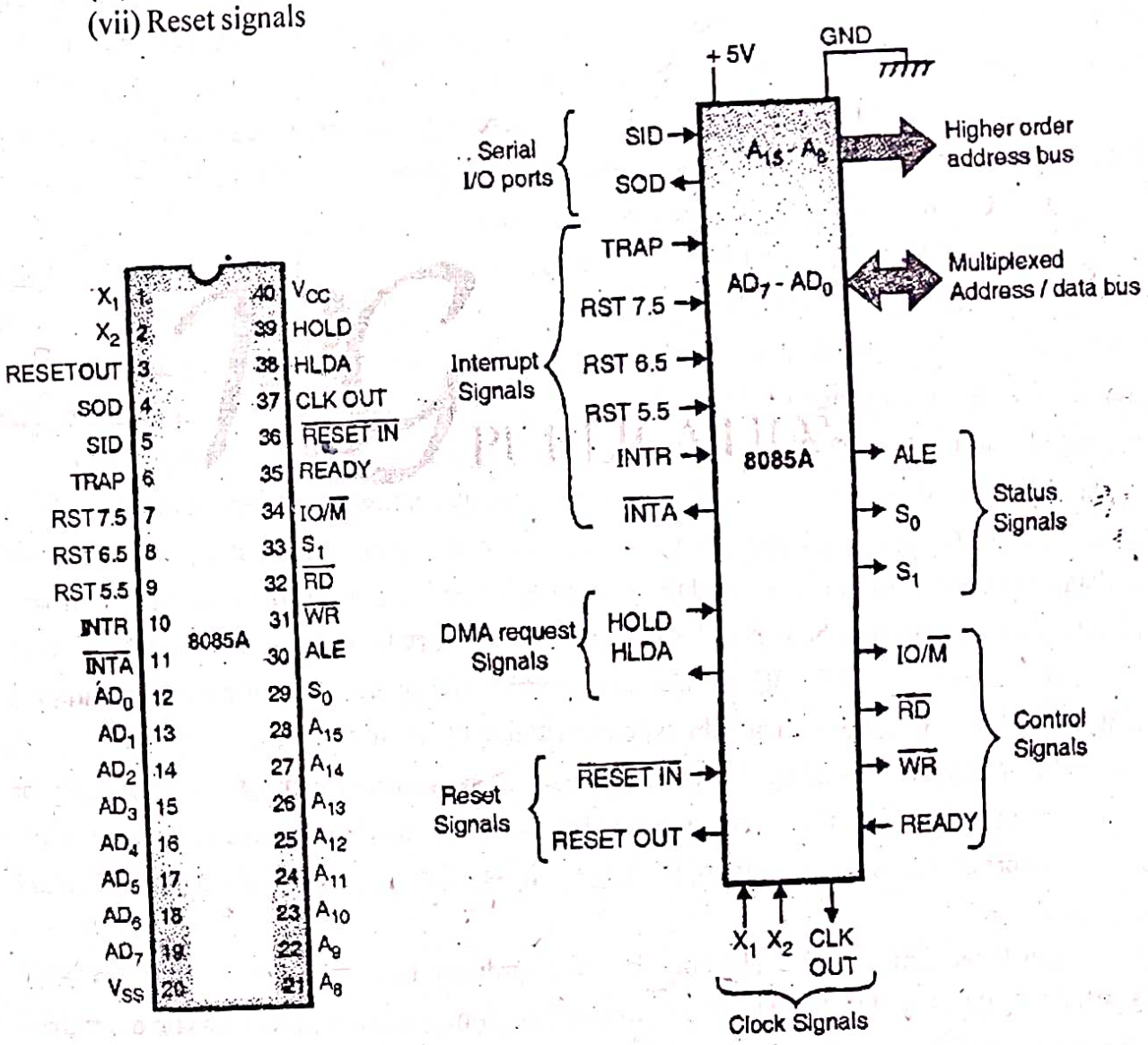


Fig. : Pin diagram of 8085 microprocessor

(i) Power Supply and Frequency Signals :

(a) V_{CC} : It requires a single +5V power supply.

(b) V_{SS} : Ground reference

(c) X_1 and X_2 : A tuned circuit like LC, RC or crystal is connected at these two pins. The internal clock generator divides oscillator frequency by 2, therefore, to operate a system at 3 MHz, the crystal of tuned circuit must have a frequency of 6 MHz.

(d) CLK OUT : This signal is used as a system clock for other devices. Its frequency is half of the oscillator frequency.

(ii) Data bus and address bus :

(a) AD_0 to AD_7 : The 8 bit data bus (D_0 - D_7) is multiplexed with the lower half (A_0 - A_7) of the 16 bit address bus. During first part of the machine cycle (T_1), lower 8 bits of memory address or I/O address appear on the bus. During remaining part of the machine cycle (T_2 T_3) these lines are used as a bi-directional data bus.

(b) A_8 to A_{15} : The upper half of the 16 bit address appears on the address lines A_8 to A_{15}. These lines are exclusively used for the most significant 8 bits of the 16 bit address.

(iii) Control and status signals :

(a) *ALE* (Address Latch Enable) : We know that AD_0 to AD_7 lines are multiplexed and the lower half of the address is also necesarry during T_2 and T_3 of machine cycle to access specific location in memory or I/O port. This means that the lower half of an address must be latched in T_1 of the machine cycle, so that it is available throughout the machine cycle. The latching of lower half of an address is done by external latch and ALE signal from 8085.

(b) $\overline{RD}$ and $\overline{WR}$: These signals are basically used to control the direction of the direction of the data flow between processor and memory or I/O device/port. A low on $\overline{RD}$ indicates that the data must be read from the selected memory location or I/O port via data bus. A low on $\overline{WR}$ indicates that the data must be written into the selected memory location or I/O port via data bus.

(c) IO/$\overline{M}$, S_0 and S_1 : IO/$\overline{M}$ indicates whether I/O operation or memory operation is being carried out. S_1 and S_0 indicate the type of machine cycle in progress.

(d) *Ready :* It is used by the microprocessor to sense whether a peripheral is ready or not for data transfer. If not, the processor waits. It is thus used to synchronize slower peripherals to the microprocessor. If peripherals are fast enough it is tied to V_{CC}. If it is left open, 8085 enters in the wait state.

(iv) Interrupt signals : The 8085 has five hardware interrupt signals : RST 5.5, RST 6.5, RST 7.5, TRAP and INTR. The microprocessor recognizes interrupt requests on these lines at the end of the current instruction execution.

The $\overline{INTA}$ (Interrupt Acknowledge) signal is used to indicate that the processor has acknowledged an INTR interrupt.

(v) Serial I/O signals :

(a) *SID* (Serial I/P data) : This input serial is used to accept serial data bit by bit from the external device.

(b) *SOD* (Serial O/P Data) : This is an output signal which enables the transmission of serial data bit by bit to the external device.

(vi) DMA Signals :

(a) *HOLD* : This signal indicates that another master is requesting for the use of address bus, data bus and control bus.

(b) *HLDA :* This active high signal is used to acknowledge HOLD request.

(vii) Reset signal :

(a) $\overline{RESET\ IN}$: A low on this pin

– Sets the program counter to zero (0000H).

– Resets the interrupt enable and HLDA flip-flops. Before entering any interrupts service routine we may have to initialize certain passing parameters. Thus it is not desired to activate interrupts at the RESET time.

– Tri-states the data bus, addres bus and control bus.

– Affects the contents of processor's internal registers randomly.

On reset, the PC sets to 0000H which causes the 8085 to execute the first instruction from address 0000H. For proper reset operation reset signal must be held low for at least 3 clock cycles. The power-on reset circuit can be used to ensure execution of first instruction from address 0000H.

(b) *RESET OUT* : This active high signal indicates that processor is being reset. This signal is synchronized to the processor clock and it can be used to reset other devices connected in the system.

Q.2.(b) Write an assembly language Program to find the number of negative number in the array. **(10)**

Ans. *Program statement :* Given an arrray of N numbers. Write a program in ALP of 8085 to find the number of negative numbers in the array. Assume the array at memory location D001 H and memory location D000 H consists of the size of the array. Store the result at memory location E000 H.

Explanation :

– We have an array of N numbers. For e.g. we intialize the count with 5.

– Also we initialize a pointer to point the elements in the array.

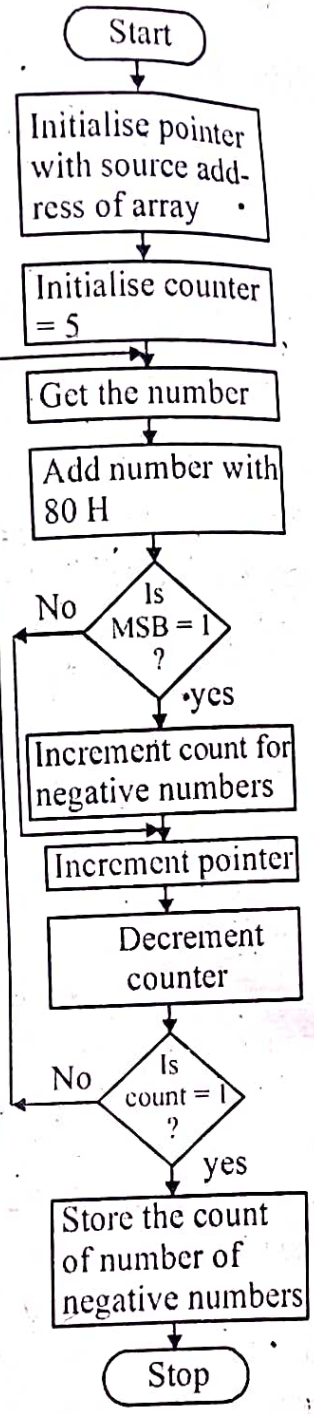
– We will check for the MSB. If the MSB of number is 1, the number is negative.

– Increment count for counting the negative numbers.

– Increment the pointer to point to the next elements. Check if MSB is 1. Repeat process till all the number are scanned.

– Store the count of negative numbers.

Label	Instruction	Comment	Operation
	LDA D000 H	Load count in accumulator	A = 05 H
	MOV C,A	Initialize count	C = 05 H
	MVI B, 00	Initialize the negative number count = 0	B = 00 H
	LXI H, D001H	Initialize pointer to array	H = D0H, L = 01 H
BACK:	MOV A,M	Get the number	A ← (HL)
	ANI 80 H	Check of MSB	And with 80 H
	JZ SKIP	Check if MSB = 1	If MSB = 1, number is negative
	INR B	Increment negative number count	B = B + 1
SKIP :	INX H	Increment memory pointer	HL = HL + 1
	DCR C	Decrement count	C = C – 1
	JNC BACK	If count 0 repeat	
	MOV A, B	Load negative count in A	A = B
	STA E000H	Store the result	E000H : No of negative number Result
	HLT	Terminate program execution	Stop



Flow chart

Q.3.(a) What is interrupt ? Explain interrupt structure of 8085 microprocessor with suitable diagram. **(10)**

Ans. Interrupt : It is a mechanism by which an I/O device (Hardware interrupt) or an instruction (Software interrupt) can suspend the normal execution of the processor and get iteself serviced.

8085 Interrupt structure : 8085 supports two types interrupts. They are :

(1) Hardware Interrputs : Peripheral device activates interrupt by activating the respective pin. In response to the interrupt request microprocessor completes the current instruction execution in main program and transfer program control to interrupt service routine.

In ISR routine, required task is complete. Task may be to read data, to write data to update the status, to update to counter etc. After completing the task, the program control is transferred back to the main program. These types of interrupts where the microprocessor pins are used to receive interrupt requests, are called hardware interrupts. The microprocessor 8085 has five hardware interrupts they are :

(i) TRAP
(ii) RST 7.5
(iii) RST 6.5
(iv) RST 5.5
(v) INTR

The interrupt structure is a 5 level structure. Fig. shows the interrupt structure.

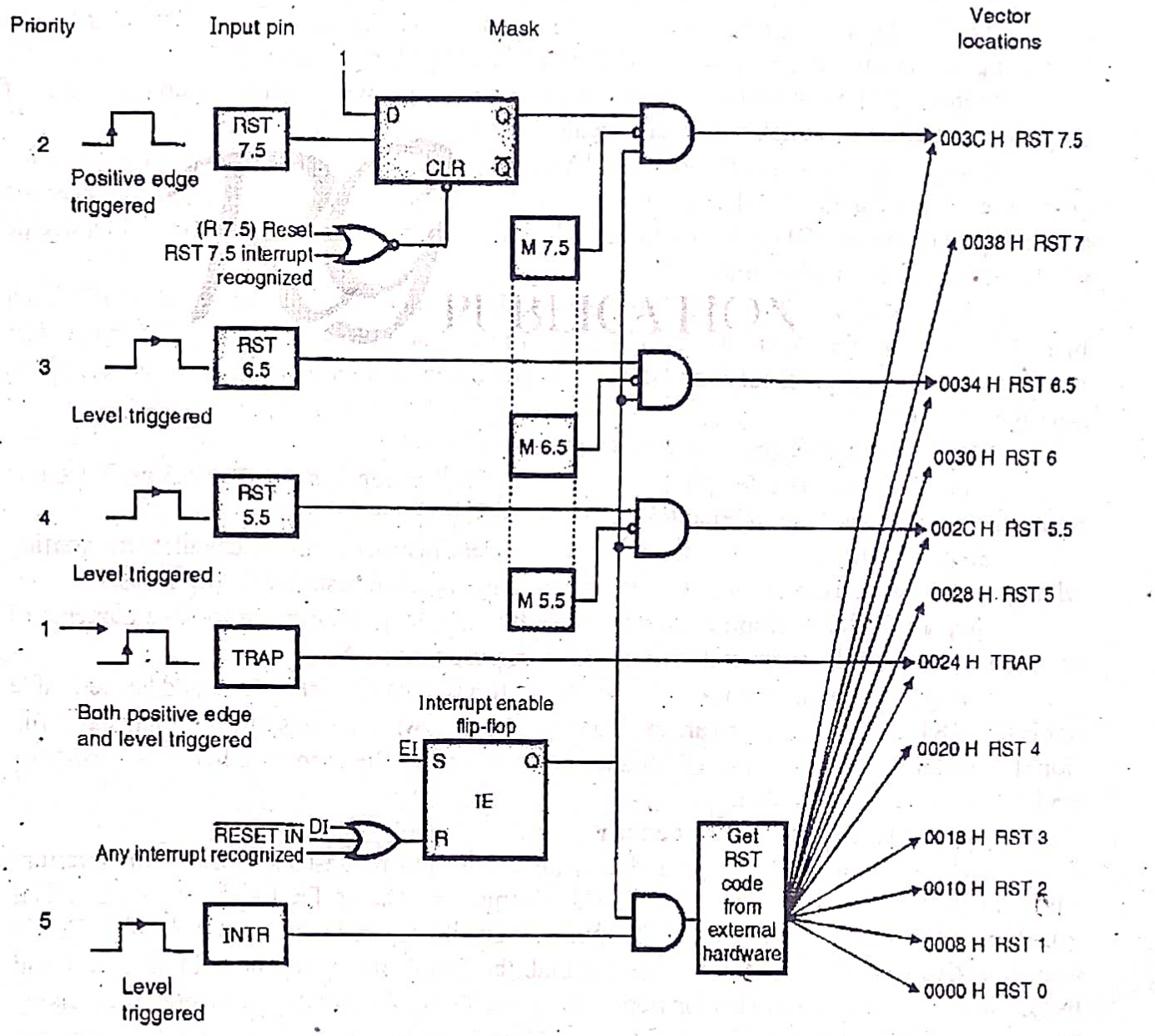


Fig. : 8085 interrupt structure

(i) *TRAP :* It is a non maskable edge and level triggered interrupt. It is unaffected by any mask or interrupt enable. The TRAP signal must a LOW to HIGH transition and remain HIGH until acknowledged. This avoids false triggering due to noise or glitches. It has the highest priority among all interrupts. This interrupts transfers the microprocessor's control to location 0024 H. It is used for emergency purpose like power failure, parity, error checker, smoke detector etc.

(ii) *RST* 7.5 : It is maskable, edge triggered, interrupt resquest input line. This interrupt is triggered at the rising edge of the signal. It has highest priority among all maskable interrupts and second priority among all interrupts. The interrupt vector location for this interrupt is 003C H.

Action taken by microprocessor, when it receives RST 7.5 :

Step I : The peripheral activates RST 7.5 signal to set R7.5 flip-flop.

Step II : When R7.5 flip flop is set and if INTE flip -flop is set and M7.5 mask latch is reset, the logic activates an internal RST 7.5 resquest signal.

Step III : In response to this signal, the microprocessor completes current instruction cycle and calculate starting address of ISR. It activates RST 7.5 acknowledge and any interrupt acknowledge signals to reset R 7.5 and INTE flip flop respectively.

Step IV : The microprocessor executes two memory write cylces to save contents of program counter on the stack and then executes corresponding ISR.

Using SIM instruction the M6.5 and M5.5 flip flop can be set or reset and hence enable or disable RST 6.5 and 5.5. DI clears the interrupt enable flip flop, while EI sets the interrupt enable flip flop. (EI and DI are instructions). M7.5 and interrupt enable flip flop together decide whether RST 7.5 is enabled or disabled.

(iii) *RST 6.5 and RST 5.5* : These are level triggered, maskable interrupt request input lines. RST 6.5 transfers microprocessor's control to location 0034 H while RST 5.5 transfers microprocessor's control to location 002C H. Action taken by microprocessor, when it receives RST 6.5 interrupt :

Step I : The peripheral activates RST 6.5 signal.

Step II : When RST 6.5 pin is at logic 1, INTE flip flop is set and M 6.5 mask latch is reset, the logic activates an internal RST 6.5 request signal.

Step III : The microprocessor completes current instruction cycle calculate the starting address of ISR. It activates any interrupt acknowledge cycle to reset INTE flip flops.

Step IV : The microprocessor executes two memory write cycle to save contents of program counter on the stack and then executes corresponding ISR.

Using SIM instruction the M6.5 and M5.5 flip flop can be set or reset and hence enable or disable RST 6.5 and 5.5. DI clears the interrupt flip flop, while EI sets the interrupt enable flip flop (EI and DI are instructions). M7.5 and interrupt enable flip flop together decide whether RST 7.5 is enabled or disabled.

The above steps are also true for RST 5.5.

(iv) *INTR :* It is level triggered, maskable interrput request input line. This interrupt works in conjunction with RST N or CALL instruction. The INTR logic consists of INTE flip-flop, OR gate and inverter. The INTR pin is logically ANDed with the output of INTE flip flop. To activate an internal INTR request signal, the INTR pin should be held at logic 1 and INTE flip flop should be set. This flip flop is set by executing EI instruction. It is reset by executing DI instruction. It is also reset by $\overline{\text{RESET IN}}$ or any interrrupt acknowledge signal. In response

to any interrupt the microproessor resets in INTE flip flop that means it disables all maskable interrupts.

Using SIM instruction the M6.5 and M5.5 flip flop can be set or reset and hence enable or disable RST 6.5 and RST 5.5. DI clears the interrupt enable flip flop, while EI sets the interrupt enable flip flop. (EI and DI are instructions). M6.5 and M5.5 and interrupt enable flip flop together decide whether RST 6.5 and RST 5.5 are enabled or disabled.

Action taken by the microprocessor when it receives an INTR :

Step I : The 8085 microprocessor checks for the status of the INTR signal while it executes each instruction.

Step II : If the INTR signal is high, then the microprocessor completes the execution of the current of the current instruction and in response to the INTR signal sends an active low $\overline{INTA}$ interrupt acknowledge singal, if the interrupt is enabled.

Step III : In response to the $\overline{INTA}$ signal, the external logic places on opcode on the data bus. In case of multibyte instruction, additional interrupt acknowledge cycles are generated by the microprocessor, in order to transfer additional bytes to the microprocessor.

Step IV : The 8085 then saves the address of the next instruction onto the stack and execute the received instruction.

Software interrupts : In case of software interrupts the cause of the interrupt is the execution of the instruction. The microprocessor 8085 has eight instructions. These eight instructions are RST 0 to RST 7. Such interrupts are called as software interrupts. They allow the microprocessor to transfer program control from the main program to the subroutine program (*i.e.* predefined service routine address).

After completing the subroutine program, the program control returns back to the main program.

Q.3.(b) Write an assembly language Program to transfer 150 bytes of data from memory location starting at 21000H to memory addressed by 31000H. (10)

Ans. Let take

⇒ Contents of *DS* = 2000 H, SI = 1000 H

ES = 2848 H, DI = 2520 H

Physical Address of *DS* : *SI* *Physical Address of ES* : *DI*

DS =	2	0	0	0	[0]	H			
SI =		1	0	0	0	H			
	2	1	0	0	0	H			

ES =	2	8	4	8	[0]	H			
DI =		2	5	2	0	H			
	3	1	0	0	0	H			

CLD	Clear Directions Flag, *DF* = 0
MOV AX, 2000 H	MOVE the starting address of DS Register into Accumulator.
MOV DS, AX	Starting address of DS Register loaded to DS Register.
MOV AX, 2848 H	Starting address of ES Register is moved to Accumulator.
MOV ES, AX	Loaded to ES Register.
MOV CX, 96 H	Hexadecimal equivalent of 150 (= 96 H) loaded to CX as count.
MOV SI, 1000 H	Offset address 1000 H moved to Source Index Register
MOV DI, 2520 H	Offset address 2520 H moved to Distination Index Register
REP MOV SB	Contents starting from 21000 H (150 bytes) transferred to 31000 H.

Unit-II

Q.4.(a) Explain BIU and EU of 8086 microprocessor. **(15)**

Ans. Bus Interface Unit (BIU) : The Bus Interface unit fetches instructions from memory, reads data from ports and memory and writes data to ports and memory. It handles all transfers of data and addresses on the buses for the execution unit. The Bus interface Unit consists of the following :

(a) Instruction Queue (b) Segment Registers (c) Instruction Pointer

(a) *Instruction Queue :* The instruction queue is a first-in-first-out group of registers. To speed up program execution, the Bus Interface Unit fetches as many as six instruction bytes are held for the Execution Unit in a Instruction Queue. The BIU can be fetching instruction bytes while the Execution Unit (EU) is decoding the instruction or executing an instruction which does not require use of the buses. When the Execution Unit is ready for its next instruction, it simply reads the instruction from the Instruction Queue in the BIU. This scheme is much faster than sending out an address to memory and then waiting for memory to send sack the next instruction byte. The prefetch instruction and queue scheme greatly speeds up processing. The arrangement of fetching the next instruction while the current instruction executes is called pipe lining.

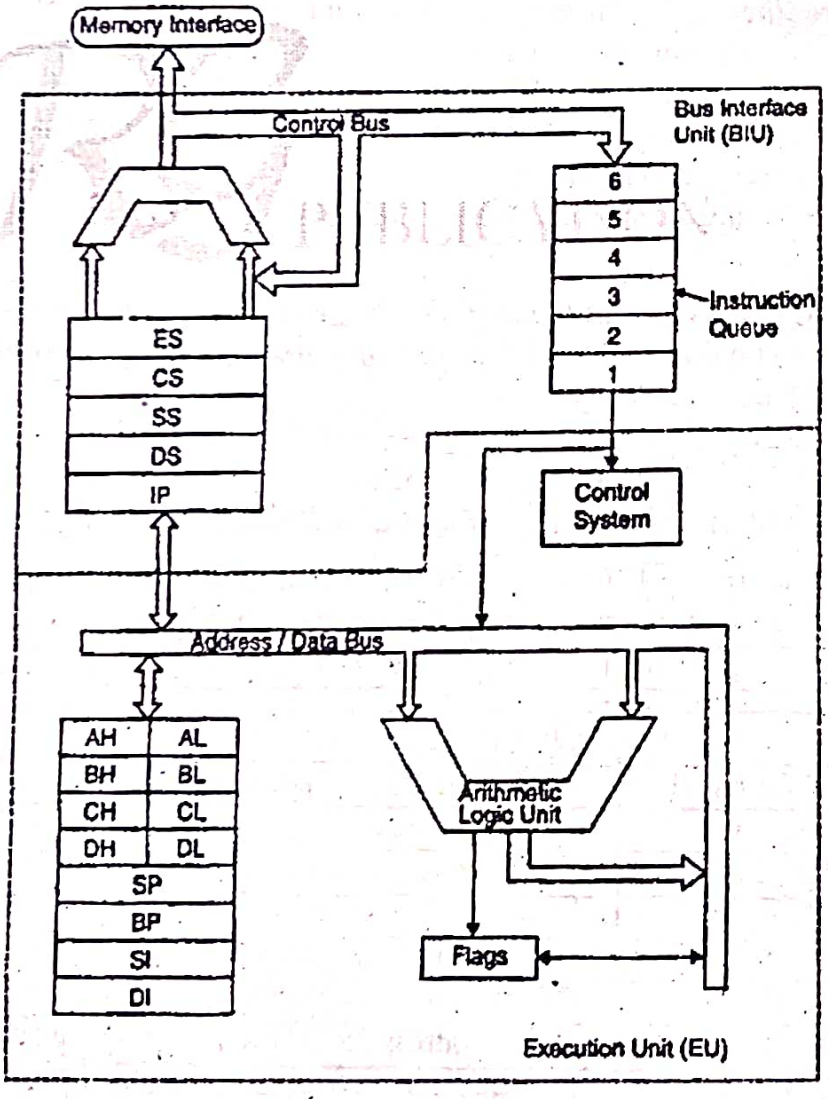


Fig. : Block diagram of 8086 microprocessor

(b) *Segment Registers :* The Big Interface Unit (BIU) contains four 16 bit segment registers. They are :

– Code Segment Register (CS)
– Stock Segment Register (SS)
– Extra Segment Register (ES)
– Data Segment Register (DS)

(c) *Instruction Pointer :* The instruction pointer register is a 16 bit register which holds the address of the next code byte that is to be fetched within the code segment. This register contains the address value which is an offset, because this value must be added to the segment base address contained in CS register to produce the required 20 bit physical address.

Execution Unit (EU) : The execution unit of 8086 performs the following major operations :

– It tells the BIU, from where to fetch instructions or data.
– It decodes and executes instructions.

To perform the above operations, the execution unit consists of the following sections :

(a) Instruction Decoder, ALU and control circuitry.
(b) Flag Register
(c) General Purpose Registers
(d) Stack Pointer Register
(e) Other Pointer and Index Registers

(a) *Instruction decoder, ALU and control circuitry :* The instruction decoder in the EU translate instructions fetched from memory into a series of actions which are further carried out. The Airthmetic and Logic Unit (ALU) of 8086 is of 16 bits which can add, subtract, AND, OR, XOR, increment, decrement, complement or shift the binary numbers. All internal operations of EU are controlled by control circuitry.

(b) *Flag register :* 8086 microprocessor contains one 16 bit flag register (status register). A flag register is a flip flop which indicates the status of some conditions produced by the execution of an instruction or controls certain operations of the Execution Unit. In a 16 bit flag register, there are nine active flags. Six of the nine flag bits are used to indicate some conditions produced by an instructions. These six flags are called status flag (conditional flags). The remaining three flag bits in the flag register are used to control certain operations of the processor and are called control flags.

(c) *General purpose registers :* The execution unit 8086 contains eight general purpose registers labelled as AH, AL, BH, BL, CH, CL, DH and DL in the fig. below.

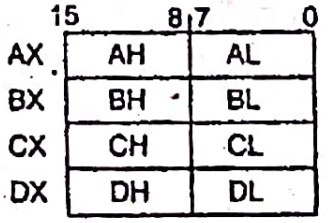


Fig. : 8086 general purpose register

(d) *Stack pointer register :* The stack pointer register SP is a 16 bit register which contains the 16 bit offset address from the start of the stack segment to the memory location where a word was most recently stored on the stack. The stack memory location where a word was mostrecently stored is called top of the stack.

(e) *Other pointer and index registers :* In addition to the stack pointer register SP, the EU of the 8086 also contains a 16 bit base pointer register BR The base pointer register BP contains the 16 bit offset address relative to the stack segment register SS but it is employed in the based addressing mode of 8086.

(5)

Q.4.(b) What is Pipelining ?

Ans. Pipelining : The process of fetching the next instruction, when the present instruction is being executed is called as pipelining. Pipelining has become possible due to the use of queue. BIU fills in the queue, until the entire queue is full. BIU restarts filling in the queue when atleast two locations of queue are vaccant.

Advantage of Pipelining :

(i) The EU always reads the next instruction byte from the queue in BIU. This is much faster than sending out an address to the memory and waiting for the next instruction byte to come.

(ii) In short pipelining eliminates the waiting time of EU and speeds up the processing.

(iii) The 8086 BIU will not initiate a fetch unless and until there are two empty bytes in its queue. 8086 BIU normally obtains two instruction bytes per fetch.

(10)

Q.5.(a) Calculate the Physical address represented by :

(a) 1234 H : 0002H

(b) F2 F2 H : 1234H

Ans. Physical address is obtained in 8086 µp by adding the constant of CS (Code Segement) Register which holds the upper 16 bits of starting address of code segment & the contents (i.e 16 bit address) of IP (Instruction Pointer) which actually holds the offset.

These contents of CS and IP are added Because 8086 up's Physical address is of 20 bits.

(a) Physical address of 1234 H : 0002H will be

Additional zero is added to compare 20 bit address.

	1	2	3	4	[0]	H
Physical address	0	0	0	0	2	H
	1	2	3	4	2	H

Physical address ⇒ 12342 H

(b) *CS* = F2 F2 H and 1234 *H* = *IP*

Additional zero

F	2	F	2	[0]	H.
	1	2	3	4	H
F	4	1	5	4	H

Physical Address ⇒ *F* 4 1 5 4 H

Q.5.(b) Describe the 8086 instruction queue. Explain the advantages of Program relocation in 8086 microprocessor. **(10)**

Ans. Instruction queue : The execution unit is supposed to decode or execute an instruction. Decoding does not require the use of buses. When EU is busy in decoding and

executing an instruction, the BIU fetches upto six instruction bytes for the next instructions. These bytes are called as the prefetched bytes and they are stored in a first in first out (FIFO) register set, which is called as a "queue".

Significance of Queue : To understand the significance of the queue, refer fig.

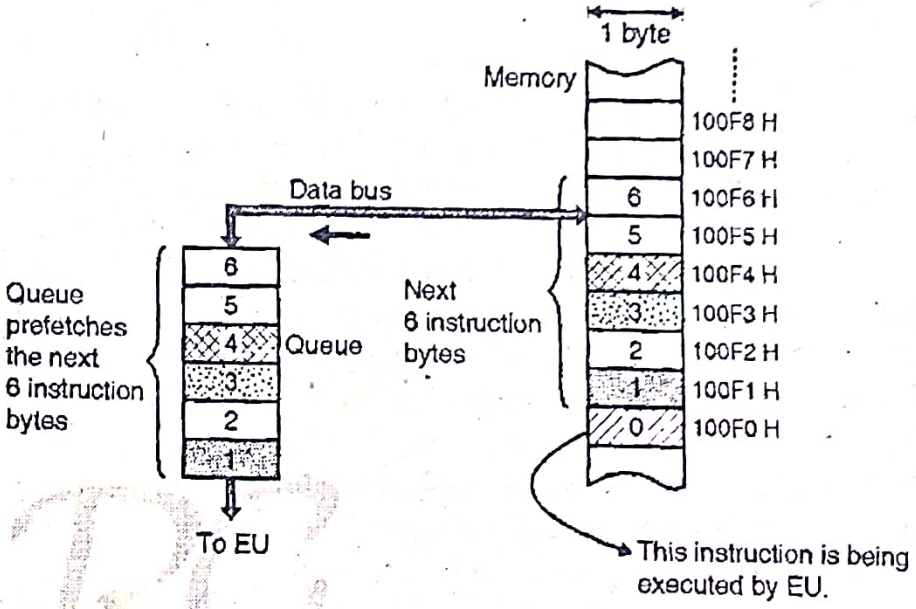


Fig. : Significance of queue

As shown in fig., while the EU is busy in decoding the instruction corresponding to memory location 100F0, the BIU fetches the next six instruction bytes for locations 100F1 to 100F6 numbered as 1 to 6. These instruction bytes are stored in the 6 bytes Queue on the first in first out (FIFO) basis. When EU completes the execution of the existing instruction, and becomes ready for the next instruction, it simply reads the instruction bytes in the sequence 1, 2,.... from the Queue. Thus the Queue will always hold the instruction bytes of the next instructions to be executed by the EU.

Advantages of Program relocation in 8086 microprocessor :

(*i*) We can address 1 MB memory spare with only 16 bits Registers.

(ii) It allows a program to be put in different areas of memory each time program in executed.

(iii) Relocatable program is executed without any changes.

(iv) Relocated program can work on seprate data sets.

(v) Can relocate a program by simply changing the contents of code segment.

(vi) Program Relocation is suitable for general purpose computers.

(vii) Without relocation, a program has to be rewritten to be altered if a memory segement has to be moved.

Unit-III

Q.6.(a) Explain directives and identifiers used in 8086 microprocessor. (10)

Ans. Directives used in assembly language program are as follows :

Directive	Action
ALIGN	aligns next variable or instruction to byte which is multiple of operand.
ASSUME	selects segment register(s) to be the default for all symbol in segment(s)
COMMENT	indicates a comment
DB	allocates and optionally intializes bytes of storage
DW	allocates and optionally intializes doublewords of storage
DD	allocates and optionally initializes doublewords of storage
DQ	allocates and optionally initializes quadwords of storage
DT	allocates and optionally initializes 10-byte long storage units
END	terminates assembly; optionally indicates program entry point
ENDM	terminates a macro defination
ENDP	marks end of procedure definition
ENDS	marks end of segment or structure
EQU	assigns expression to name
EVEN	aligns next variable or instruction to even byte
ESITM	terminates macro expansion
EXTRN	indicates externally defined symbols
LABEL	creates a new label with specified type and current location counter
LOCAL	declares local variables in macro definition
MACRO	starts macro definition
MODEL	specifies mode for asssembling the program
ORG	sets location counter to argument
PAGE	sets length and width of program listing; generates page break
PROG	starts procedure definition
PTR	assigns a specific type to a variable or to a label
PUBLIC	identifies symbols to be visible outside module
TITLE	defines the program listing title

Identifiers : An *identifier* is basically a name that is applied to an item in the program for reference purpose.

An identifier is basically a symbole cased for reference is a program.

Basically these are two types of identifiers in 8086 µp.

(i) NAME identifer : This type of identifiers identifies the item by name or can say refers to the address of data item.

e.g. COUNTER (Name identifier) in Counter DBO.

(ii) Label Identifier : Refer to the address of an instruction, procedure or segment

e.g. MAIN PROC FAR MAIN ⇒ MAIN is label identifier.

B 30 : ADD BL, 30 ⇒ B30 is label identifier

Maximum length of an identifier is 31 characters upto MASM 6.0 & 247.

Following characters can be used as Indentifiess

(i) Alphabetic characters : - A to Z and a -z

(ii) Digit : 0 – 9 (but not as 1st character)

(iii) Special charackes : ? _ $ @. (but not as 1st character)

Q.6.(b) Write a simple assembly language Program to add two memory locations where each memory location is one byte wide. (10)

Ans.

Program	Comments
LDA 2500 H	Load Accumulator directly from memory location 2500 *H*.
MOV B, A	MOVE contents of Accumulator (i.e from 2500 H) to Register *B*.
MVI C, 00 H	Initialize carry by putting value 0 in Resgister C.
LDA 2501 H	Load Accumulator directly from 2501 H (i.e. 2nd number.
ADD *B*	Add contents of Register *B* to Accumulator (i.e 1st number + 2nd number = stored in Accumulator)
JNC	Jump to LABEL if no carry generated
INR C	Otherwise increment Register C by 1 (i.e. carry generated)
LABEL : STA = 2502	Store Result of addition to memory location 2502 H.
MOV A, C	Move contents of Register C (i.e. = 01) to Accumulator.
STA 2503 H	Store content of Accumulator to memory location 2503 H.
HLT	Program Ended ⇒ Halt.

Q.7.(a) Explain NOP and HLT instructions with the help of suitable example. (10)

Ans. NOP instruction :

Mnemonic NOP **Flags :** It does not affect any flag.

NOP : No operation

Algorithm Do nothing

Addr. Mode Implied addressing mode

Operation The execution of this instruction causes the CPU to do nothing.

(i) This instruction causes the CPU to do nothing.This instruction uses three clock cycles and increments the instruction pointer to point to the next instruction.

(ii) It can be used to increase the delay of a delay loop.

Example :

```
MOV AL, 00011011b
NOT AL ; AL = 11100100b
RET
```

Flags : all unchanged.

HLT (Halt until interrupt or reset) Instruction :

Mnemonic Halt processing **Flags :** No flags are affected..

Operation (i) The HLT instruction will cause the 8086 to stop fetching and executing instructions. The 8086 enters into a half state. To come out of the halt state, there are 3 ways given below.

(a) Interrupt signal on INTR pin, (b) Interrupt signal on NMI pin

(c) Reset signal on reset pin.

(ii) It may be used as an alternative to an endless software loop in situations where a program must wait for an interrupt.

Q.7.(b) Explain instruction execution timing with example. **(10)**

Ans. Instruction Execution timing : The execution time for instruction varies and depends upon the type of instruction and addressing modes. Basic Instruction time can determined by multiplying the number of clock needed to execute the instruction by clock's period.

The total instruction execution time is the sum of Basic time plus time required to calculate the effective Address if a memory operand is accessed. The Basic instruction time assumes that the instruction to be executed has already been prefetched and stored in instruction queue. If not so, then additonal clock cycles must be added to find the total execution time.

Total instruction execution time depends upon :

(i) No. of memory seferences required by instruction : If operand is a single byte or word with an even address, then only are memory reference is needed, if with an odd address then two memory references is needed. Each memory reference require 4 clock cycle.

(ii) Addressing nodes : In case of 2 operand instruction, register to register addressing gives fasters operation than register to memory or vice-versa operation, and between memory to register and register to memory operation, the memory to register operation is faster.

(iii) Variable Basic Execution Times : Some Instruction are data dependent & Require Variable Basic Execution Times. These are multiply, divide, shift & Rotate.

(iv) Conditional Branch Instruction : For these, 2 different timings are specific smaller time should be considered when condition is not met and branch is not taken. Larger time is considered when condition met. Hence some extra cycle are required to fetch the next instruction.

Let's take on example of Ready & Wait state : The Ready input causes wait state for slower memory & I/O Devices to cope with up. A wait state is an extra clocking period inserted between T2 and T3. Due to this memory access time of 5 MHz with 460 ns is increased by 200 ns to make it (460 ns + 200 ns = 660 ns)

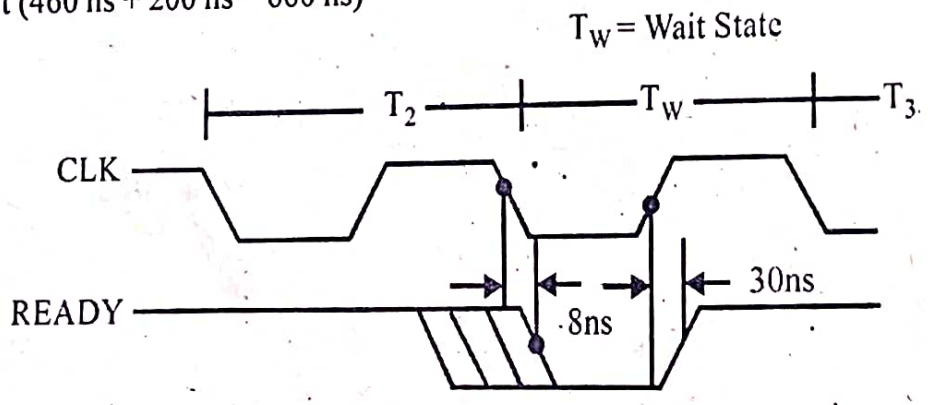


Fig. : READY input timing in 8086 µp.

If READY is at logic 0 after T_2, then T_3 is delayed and T_w is inserted between T_2 and T_3. Ready is next sampled at middle of T_w to determine if next state is T_w or T_3. It is tested for a logic 0 on 1 to 0. Transition of clock at the end of T_2 & for a 1 on 0 to 1 transmition of clock in middle of T_w. Also READY input 8086 µp has some stringent tining reguirements. Such timing requirements are fulfilled by the internal READY synchronization circuitry of 8284 A clock generator.

Unit-IV

Q.8.(a) Describe the major components of 8279 keyboard/display/CPU interface with the help of its block diagram. **(10)**

Ans. Fig. shows a block diagram of 8279 funticonally 8279 is divided into three sections:

(i) CPU Interface and control section.

(ii) Keyboard section.

(iii) Display section.

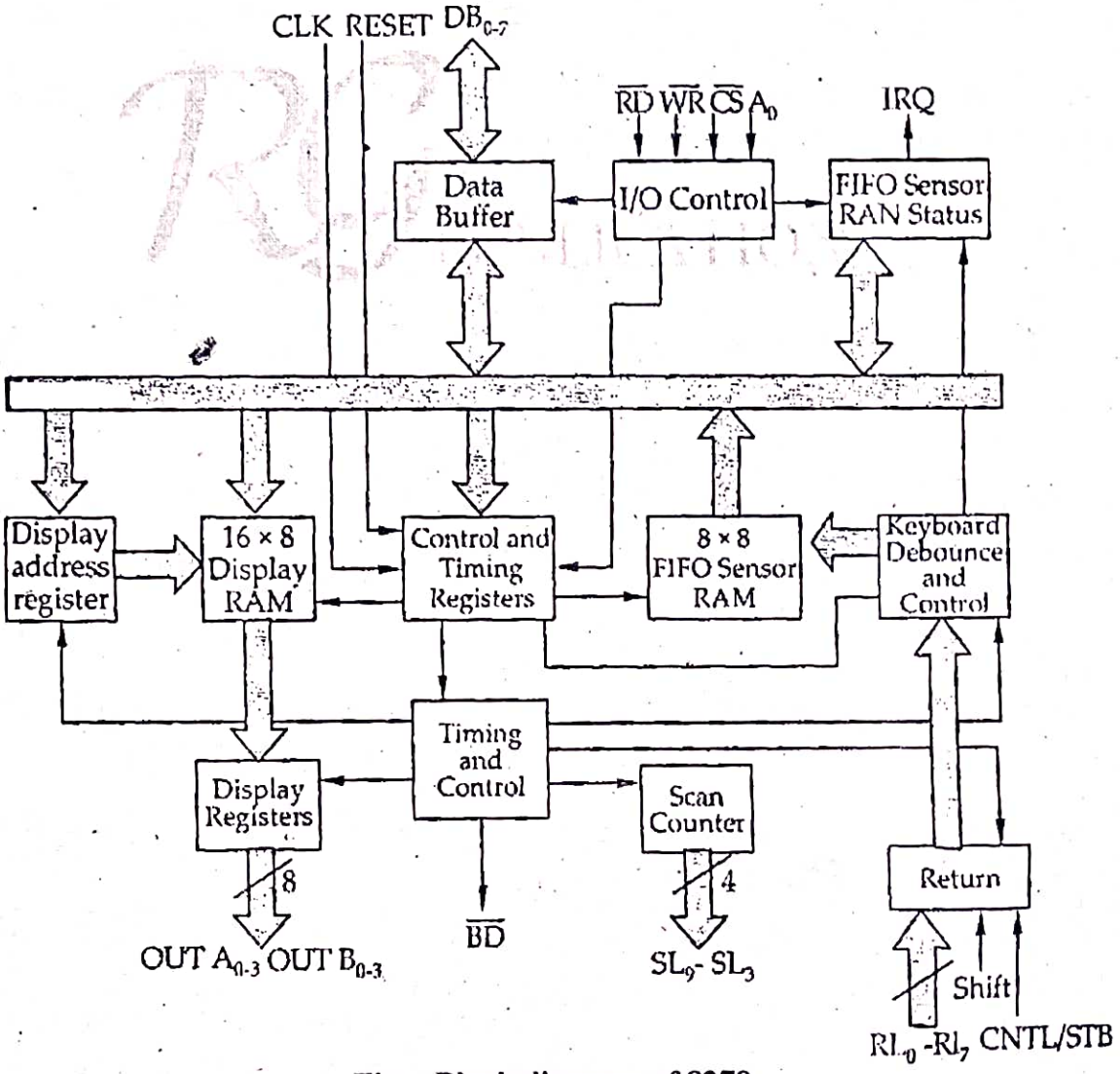


Fig. : Block diagram of 8279

(i) CPU interface and control section : It consists of data buffers, I/O control, control and timing registers and timing and control logic.

(*a*) *Data Buffers :* These are bidirectional, tristate buffers. It connects external data bus to the internal bus.

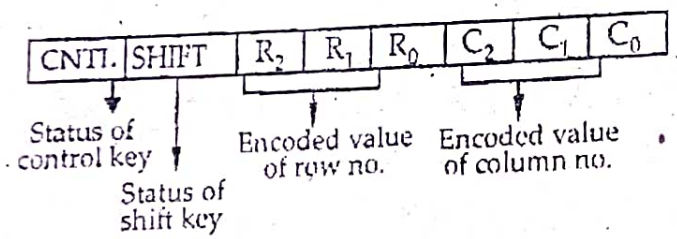
(*b*) *I/O Control :* It uses $\overline{CS}$, A_0, $\overline{RD}$ and $\overline{WR}$ lines to control data flow to and from the various internal registers and buffers.

(*c*) *Control and timing register :* This block consists of control and timing registers. 8279 provides 8 control and timing registers which are distinguished by higher three bits of command word. To access these registers, A_0 line should held at logic 1. These registers store the keyboard and display modes and other operating conditions of chip. These registers are affected by reset signal.

(*d*) *Timing and control logic :* It generates all internal keyboard and display control signal. It also generates $\overline{BD}$ signal, provides proper key scan, row scan, display scan and debounce timings.

(ii) Keyboard section :

(a) *FIFO/Sesnor RAM :* It is an 8 × 8 bits read only RAM. Its function depends upon operating modes. In scanned keyboard mode, it functions as a FIFO RAM. In this mode the position of depressed key (Row no. and function no) along with status control and shift keys is entered into FIFO RAM location. The data format of FIFO RAM location for scanned keyboard mode is given below.



The lower three bits (C_2, C_1, C_0) indicate to which return line the depressed key was connected. R_2, R_1, R_0 bits give the value of coresponding scan count in encoded mode. In decoded mode, R_2 is always zero. The inidicate to which row the depressed key was found. In this mode, the microprocessor cannot access other entries without accesing first one, in sensor matrix mode, this block functions as sensor RAM. In this mode, the data on the return sensor RAM is specified by scan counter (row no of sensor matrix). Therefore each sensor switch position can be scanned by the CPU. The shift and CNTI inputs are ignored.

The data format of a sensor RAM location for sensor matrix mode is given below.

RL_7	RL_6	RL_5	RL_4	RL_3	RL_2	RL_1	RL_0

(*b*) *Keyboard debounce and control :* In scanned keyboard mode, it debounces, encodes and stores encoded value into FIFO key closures. It also saves status of shift and control keys into FIFO. It updates information stored into FIFO/sensor RAM status register. In sensor matrix mode, the debounce logic is inhibited and the contents of the retrun lines is directly transferred to the corresponding location of sensor RAM.

(*c*) *Retrun Buffers :* The RL_0- RL_7 lines are buffered and lateched by this block. It scans RL_0 – RL_7 lines during each row scan in scanned keyboard or sensor matrix mode. $RL_0 - RL_7$ lines are not scanned in strobed input mode.

(iii) Display section : It consists of display RAM, display address registers and display registers.

(*a*) *Display Section :* It is 16 × 18 bits display RAM. During initialization, all locations are loaded with clear code or blank code. The microprocessor can read or write this RAM. The address of location to be accessed is specified by command word. During each digit time, the contents of selected location are transferred into display register. In the decoded mode, the 8279 used only first four locations. In the encoded mode, the 8279 uses first 8 locations (8 digits) or all locations (16 digits).

(*b*) *Display address registers :* It constains two types of address registers viz. microprocessor controlled address registers and timing controlled address registers.

(*c*) *Display registers :* These registers holds the codes of character to be displayed. They are divided into two nibbles viz. nibble *A* and nibble *B*.

Nibble A and nibble B can be blanked and inhibited individually.

During each digit scan, the contents of some registers hold blanking code of the display.

Q.8.(b) List the operating modes of 8255 PPI chip. **(10)**

Ans. Various **operating modes of 8255 PPI chip** are as follows :

1. BSR Mode : The BSR mode is a port C bit set/reset mode. The individual bit of port C can be set or reset by writing control word in the control register. The control word format of BSR mode is shown in fig.

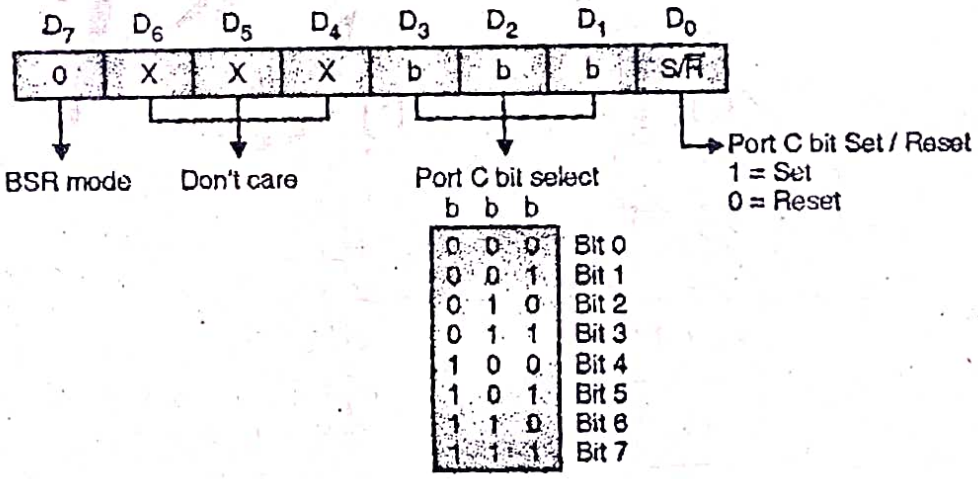


Fig. : BSR control word format

The pin of port C is selected using bit select bits [b b b] and set or reset is decided by bit S/R. The BSR mode affects only one bit of port C at a time. The bit set using BSR mode remains set unless and until you change the bit. So to set any bit of port C, bit pattern is loaded in control register. If a BSR mode is selected it will not affect I/O mode.

2. I/O modes of 8255 PPI chip : I/O modes of 8255 PPI chip are as follows :

(i) *Mode-0 operation :* In the mode-0, the 8255 can be programmed as simple input or output. Each of the four ports, port *A*, port *B*, port C_U (upper) and port C_L (Lower) of 8255 can be programmed to be either an input port or an output port. Data is written into or read from the specific port. The input/output features of 8255 in the mode-0 are :

– Two 8 bit ports and two bit ports

– Any port can be an input port or an output port.

– Outputs are latched where as inputs are not latched.

– Ports do not have handshake or interrupt capability.

(ii) *Mode-1 operation :* In the mode-1 operation of 8255, the handshake signals are exchanges betweeen the microprocessor and peripherals prior to data transfer. The features of this mode are :

– Two ports, port *A* and port *B* can function as 8 bit I/O ports. Port *A* and port *B* can be configured either as input port or output port.

– Each port (port *A* and port *B*) uses three lines from port *C* as handshake signals. The remaining two lines of port *C* can be used for simple I/O functions.

– Interrupt logic is supported.

– Both inputs and outputs are latched.

(iii) *Mode-2 operation :* In this mode, the port *A* can be used as a bidirectional port and port *B* can be used either in mode 0 and 1 operation. Port *A* uses five signals from port *C* as handshake signals for data transfer. The remaining three signals of port *C* can be used either as simple I/O function or as handshake signals for port *B*. The mode-2 operation is generally used for bidirectional data transfer. Data transfer between two computers can be accomplished using 8255 programmable peripheral interface in mode-2 operation.

Q.9.(a) Explain 8237 DMA controller with help of Pin diagram. **(10)**

Ans. The fig. shows pin configuration of 8237.

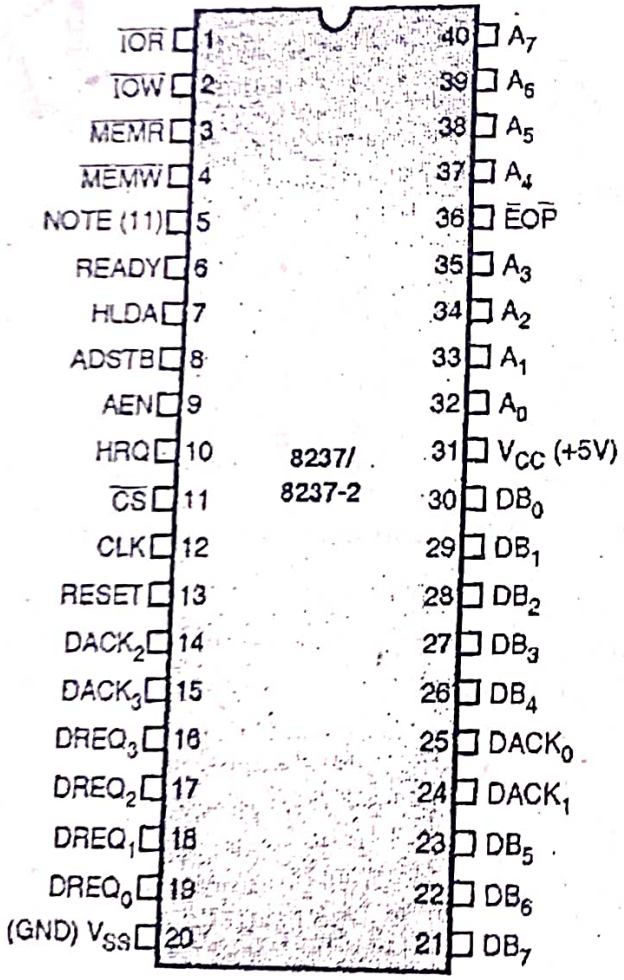


Fig. : Pin configuration of 8237

Symbol	Description
CLK	This is a clock input line ignored in slave mode. In master mode, this signal controls all internal and external DMA operations. The data transfer rate depends upon the frequency of this signal.
$\overline{CS}$	In slave mode, this signal is generated by address decoder to select 8237 chip for communication between CPU and 8237. In master mode, this signal is ignored.
Reset	It is an asynchronous input line. This signal clears the command, status, request and temporary register and forces 8237 into slave mode.
READY	In master mode, this signal is used to add wait states into a DMA cycle.
HRQ	It is a hold request output line. It is connected to hold input of the CPU. It is used to request control of the system bus.
HLDA	It is a hold acknowledge input line. This signal is generated by CPU. In response to this signal, the 8237 gains control of the system bus and enters into master mode.
$\overline{IOR}$	It is an active low bi-directional tristate line. In slave mode, it acts as an input line and used to read contents of 8237 registers. In master mode, it acts as an output line. This signal is generated during DMA cycle to read data from I/O device.
$\overline{IOW}$	It is an active low bi-directional line. In slave mode, it acts as an input line and used by CPU to write contents to 8237 registers. In master mode, it acts as an output line. This signal is generated during DMA read cycle to write data into I/O device.
A_0 - A_3	These are bi-directional, address lines. In slave mode, these lines act as input lines, used to select one of the registers of 8237. In master mode, the 8237 provides lower bits of memory address on these lines.
A_4 - A_7	These are tristate address output lines. These lines are tristated in slave mode. In master mode, the 8237 transfers bits of memory address on these lines.
$\overline{MEMR}$	It is an active low tristate control output line. It is tristated in slave mode. In master mode, this signal is generated during DMA read cycle or during memory to memory transfer cycle to read contents of source memory.

$\overline{MEMW}$	It is an active low tristate output line. It is tristated in slave mode. In master mode, this signal is activated during DMA write or during memory to memory transfer cycle to read contents of destination memory.
DB_0 - DB_7	These are bi-directional tristate buffered data lines. In slave mode, these lines are used to transfer data between CPU and 8237's registers. In master mode, these lines act as address output lines. The 8237 places higher byte of address on these lines during DMA cycles.
AEN	This active high output enables the 8 bit latch that drives the upper 8 bit address bus. The AEN pin is used to disable other bus drivers during DMA transfers.
ADSTB	This output line is used to strobe the upper address byte generated by 8237 in master mode into an external latch.
$DREQ_0$ - $DREQ_0$	These are asynchronous DMA channel request lines used by peripheral. The polarity of each signal is programmable i.e. these lines can be used as either active high or active low input. DREQ must be maintained until the corresponding DACK is activated.
$DACK_0$ - $DACK_3$	These are DMA acknowledge output lines. The polarity of each line is programmable. This signal indicates that the requesting peripheral has been granted fro DMA cycle.
$\overline{EOP}$	*End of Process :* It is an active low bi-directional signal. This line is also used to terminate DMA cycle. The DMA cycle can be terminated by pulling $\overline{EOP}$ input low. The 8237 also generates EOP pulse, when the terminal count for any channel is reached.

Q.9.(b) Explain :

(i) Mode-2 strobbed bidirection I/O of 8255

(ii) Priority Resolver. **(10)**

Ans. (i) Mode-2 strobbed bidirection I/O of 8255Working of 8255 : In this mode group A is used as input and output i.e. for transmitting and receiving data from peripheral through 8255 as shown in fig.(a).

The transfer of data is achieved by port C handshake signals. The group B can be in Mode 0 or Mode 1.

The bi-directional data is transferred through port A so it consists of input and output latch.

The Mode 2 is combination of Mode 1 input and output both at a time to port A.

The interrupt signals of input and output mode are combined to generate common interrupt signal to CPU. The internal organization of these signals is as shown in fig. (b).

The different handshake signals used are $\overline{OBF_A}$, $\overline{ACK_A}$, $\overline{STB_A}$, IBF_A and $INTER_A$ handshake signals are used for output operation, 2 are used for input operation and one is common to both.

Output operation :

– **$\overline{OBF}$** (*Output buffer full*) : This is an active low output signal generated by 8255. When CPU writes data to output port 8255 will enable OBF signal to indicate peripheral that data is available in output buffer.

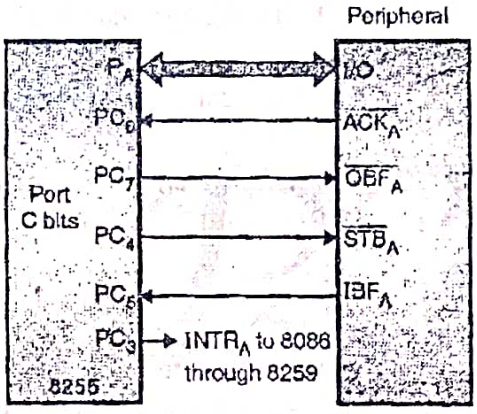


Fig. (a) : Mode 2 interfacing

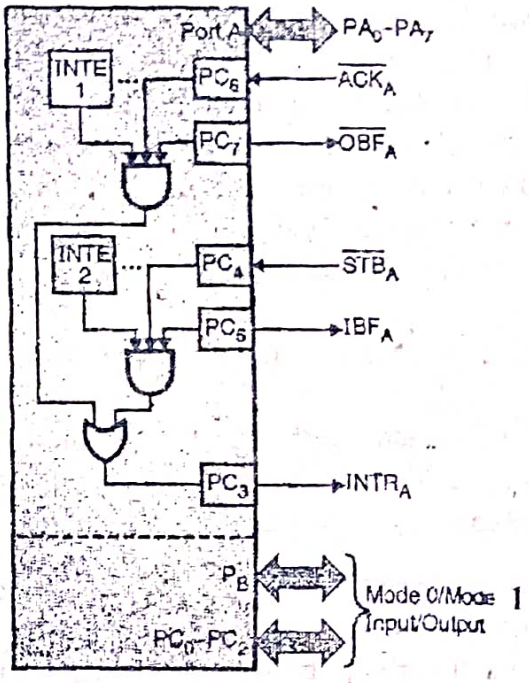


Fig. (b) : P_A, P_B and P_C in mode 2

– **$\overline{ACK}$** (*Acknowledge*) : This is an active low input signal for 8255. When the peripheral detects $\overline{OBF}$ signal, it reads data from 8255 port and makes $\overline{ACK}$ = 0 and the ACK signal is used to acknowledge 8255 that data is read from port so 8255 will remove $\overline{OBF}$ signal to indicate output buffer is empty.

Input operation :

– **$\overline{STB}$** (*strobe*) *:* This is an active low input signal. When the peripheral writes data to input buffer, it generates a signal STB to indicate 8255 that it has written data.

– *IBF* (*Input buffer full*) *:* When data is available in input buffer 8255 will enable IBF signal to indicate that data is available in input buffer.

Ans. (ii) Priority resolver : It determines the priorities of the bit set in the IRR. To make decision, the priority resolver looks at the ISR. If the higher priority bit in the InSR is set then it ignores the new request. If the priority resolvers finds that the new interrupt has a higher priority than the highest priority interrupt currently being serviced and the new interrupt is not in service, then it will set appropriate bit in the InSR and send the INT signal to the microprocessor for new interrupt request.

MICROPROCESSING & INTERFACING

Dec 2015

Paper Code: EE-309-F

Note: *Attempt five questions in all, selecting one question from each section. Q. No.* **1** *is compulsory.*

Q.1.(a) Define Memory Segmentation. (4)

Ans. Memory segmentation is the division of a computer's primary memory into segments or sections. In a computer system using segmentation, a reference to a memory location includes a value that identifies a segment and an offset within that segment. Segments or sections are also used in object files of compiled programs when they are linked together into a program image and when the image is loaded into memory.

Some of the advantages of memory segmentation in the 8086 are as follows:

(i) With the help of memory segmentation a user is able to work with registers having only 16-bits.

(ii) By memory segmentation the various portions of a program can be of more than 68kb.

(iii) The data and the users code can be stored separately allowing for more flexibility.

(iv) Also due to segmentation the logical address range is from 0000H to FFFFH the code can be loaded at any location in the memory.

Q.1.(b) Describe the purpose of program Counter in 8085. (4)

Ans. Program counter is a 16 bit resistor. This resistor is a memory pointer. The function of program counter is point to the memory location from which the next instruction is to be executed. When a program is to be executed, it is first entered into the memory locations. Now the program counter is loaded by the address of the first instruction to be executed. Once the first instruction is executed, the program counter is automatically incremented to point to the address of next instruction. The process is repeated till the end of the program.

Q.1.(c) Describe the purpose of Trap flag in 8086. (4)

Ans. The trap flag (TF) : Setting TF puts the processor into single step mode for debugging. In single stepping, microprocessor executes a instruction and enters into single step ISR.

After that user can check registers or memory contents, if found ok, he/she will proceed further, else necessary action will be taken. This utility is, to debug the program. If TF = 1, the CPU automatically generates an internal executes instruction by instruction, allowing a program to be inspected as it executes instruction by instruction.

Used by debuggers for single step operation.

TF = '1' – Trop on, TF = '0' – Trap off.

Q.1.(d) Explain Register Addressing Mode in 8086. (4)

Ans. Register addressing mode : In this mode of addressing, data is in the register, and instruction specifies the particular register as shown in fig.

This addressing mode is normally preferred because the instructions are compact and fastest executing of all instruction forms. The reason why it is fastest executing is just because, all the register reside on chip, therefore data transfer is within the chip and external bus is not at all required.

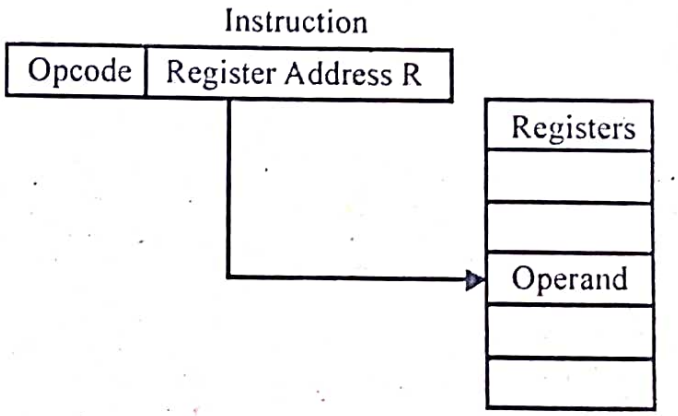


Fig. : Register addressing

Registers may be used as source operands, destination operands or both.
The register may be 8/16 bit.

e.g. MOV AX, BX

The instruction copies the contents of BX register to AX register.

Q.1.(e) What is programmable Interval Timer. **(4)**

Ans. There are two types of programmable interval timer are generally used. Intel 8253 is a programmable Interval Timer/Counter which can generate accurate time delays and waveforms ranging from 0 Hz to 2 MHz using software control. 8254 is its upgraded version which can operate with higher clock frequency range (DC – 8 MHz) and it is pin to pin compatible with 8253.

Unit – I

Q.2.(a) Explain the purpose of following instructions in 8085.

(i) RIM
(ii) SIM
(iii) RRC
(iv) XCHG
(v) DADRp **(10)**

Ans. (i) RIM : Read interrupt mask.

This is multipurpose instruction used to read the status of interrupts RST 7.5, RST 6.5, RST 5.5, and read serial data input pin. The instruction loads 8 bits in the accumulator in the following format.

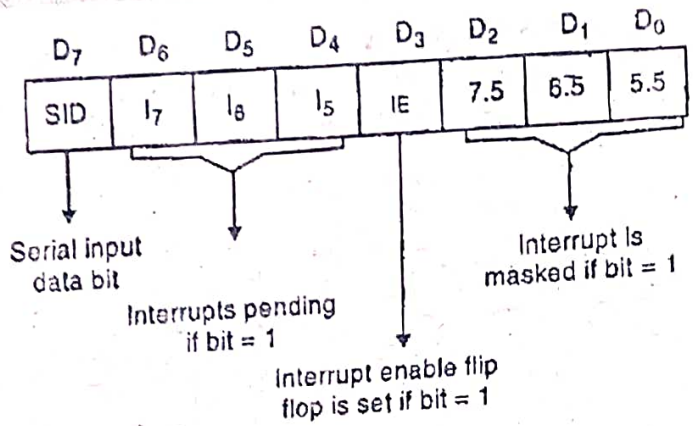


Fig. : RIM Instruction format

(ii) SIM : Set interrupt mask.

This is a multipurpose instruction used to control interrupts and serial data output pin. The instruction interprets the accumulator contents as follows :

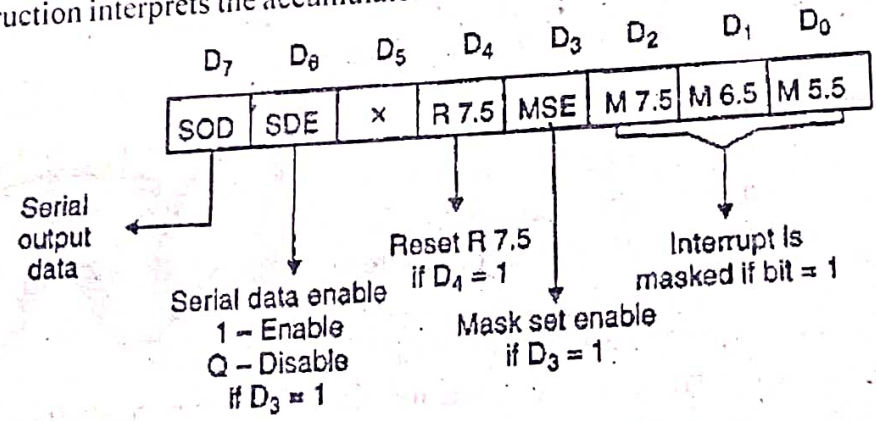


Fig. : SIM Instruction format

(iii) RRC : Rotate accumulator right.

This instruction rotates the contents of accumulator right by 1 bit. D_7 will be transferred to D_6, D_6, to D_5 and so on D_1 to D_0, D_0 to D_7 as well as to the carry flag.

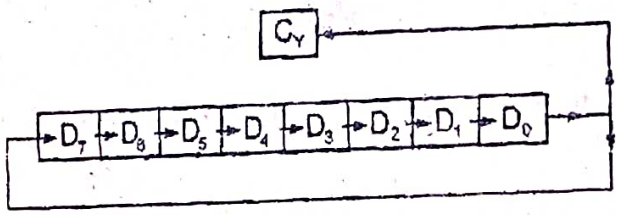


Fig. : RRC Instruction format

Operation : Accumulator : $D_{n+1} \rightarrow D_n (n = 0 \text{ to } 6)$

$D_0 \rightarrow D_7, D_0 \rightarrow Cy$

(iv) XCHG : Exchange the contents of HL, with DE pair.

This instruction exhanges contents of H reg. With D reg. and L reg. With E reg.

Operation : $H \leftrightarrow D, L \leftrightarrow E$

(v) DADRp : Add the specified register pair to HL pair.

This instruction adds the contents of specified register pair to HL pair and stores the result in HL pair. The example of R_p are SP, BC, DE and HL. Only carry flag is modified to reflect the result of operation.

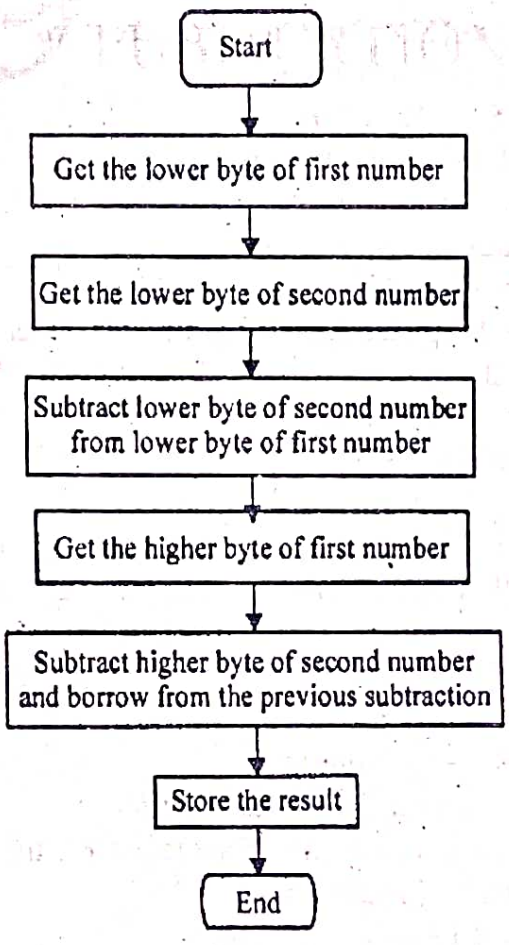
Operation : $R_P + HL \rightarrow HL$

Q.2.(b) Subtract the 16 bit number in memory locations 2002H and 2003H from the 16 bit number in memory locations 2000H and 2001H. The most significant eight bits of the two numbers are in memory locations 2001H and 2003H. Store the result in memory location 2004H and 2005H with the most significant byte in memory location 2005H. **(10)**

Ans. Sample problem

(2000H) = 19H
(2001H) = 6AH
(2002H) = 15H
(2003H) = 5cm

Result = 6A 19H–5C 15H= OE04H
(2004H) = 04H
(2005H) = OEH



Program

LHLD 2000H → Get first 16 bit number in HL.
XCHG → Save first 16 bit number in DE.
LHLD 2002H → Get second 16 bit number.
MOV A, E → Get lower byte of the first number.
SUB L → Subtract lower byte of the second number.
MOV L, A → Store result in L register.
MOV A, D → Get higher byte of the first number.
SUB H → Subtract higher byte of second number with borrow.
MOV H, A → Store 16 bit result in memory locations 2004H and 2005H.
SHLD 2004 → Store 16 bit result in memory locations 2004H and 2005H.
HLT → Terminate program execution.

Q.3.(a) Explain pin diagram of 8085 and describe various signals. **(10)**

Ans. Refer Q.3(a) of paper Dec. 2016.

Q.3.(b) Discuss in detail 8085 interrupts. **(10)**

Ans. Refer Q.2(a) of paper Dec. 2016.

Unit – II

Q.4.(a) Explain flag manipulation instruction of 8086 in detail. **(10)**

Ans. Flag Instructions (Flag Transfer) : These instructions are related to movement of flag register to/from a register and memory.

Flag Instructions (Flag Transfer) :

(a) LAHF (Load AH register from flags)
(b) SAHF (Store AH register in flags)
(c) PUSHF (Push flags onto stack)
(d) POPF (Pop flags off stack)

1. LAHF – Load AH register from flags : (Copy lower byte of flag register to AH)

Mnemonic LAHF *Flags* No flags are affected.
Algorithm AH = flag register's lower byte
Addr. Mode Implied Addressing mode
Operation AH ← Lower byte of flag register
The lower byte of 8086 flag register is copied to the AH register

2. SAHF – Store AH register in flags : (Copy contents of AH to lower byte of flag register)

Mnemonic SAHF *Flags* All the flags are changed.
Algorithm AH = flag register
Addr. Mode Implied Addressing mode
Operation AH → Lower byte of flag register

(i) This instruction copies the contents of AH register to the lower byte of flag register.
(ii) It is included for 8085 compatibility.
(iii) The OF, DF, IF and TF are not affected.

3. PUSHF – Push flags onto stack : (PUSH flag register on the stack)

Mnemonic PUSHF *Flags* No flags are changed.

Algorithm SP = SP – 2

SS : [SP] (top of stack) = operand

Addr. Mode Register Addressing mode

Operation SP → SP – 2 SS → data from flag register

(i) This instruction decrements the stack pointer by 2 and copies word in the flag register to the memory location pointed by stack pointer.

(iii) The stack segment register is not affected.

4. POPF – Pop flags off stack :

Mnemonic POPF *Flags* All flags are affected.

Algorithm SS = data to flag register SP = SP + 2

Addr. Mode Register Addressing mode

Operation SS : [SP] → Copy data to flag register SP = SP + 2

(i) This instruction copies a word from the two memory locations at the top of the stack to flag register and increments the stack pointer by 2.

(iii) The stack segment register and word on the stack are not affected.

Q.4.(b) Describe 8086 Microprocessor architecture with its block diagram.(10)

Ans. Refer Q.5(a) of paper Dec. 2016.

Q.5.(a) Explain instructions formats of 8086 microprocessor in detail. (10)

Ans. In 8086 all instruction will not be of same size. The instruction vary from 1 to 6 bytes in length. The length of instruction bytes is dependent upon addressing mode used by programmer i.e. immediate, register relative, based indexed, relative based indexed and so on.

Basically instruction bytes will contain information of :

(i) OPCODE

(ii) Addressing mode designations :

(a) 2 byte Effective Address.

(b) 1 or 2 byte displacement.

(c) 1 or 2 byte immediate operand.

From Fig.1.(a) normally first byte is OPCODE byte, second byte normally specifies addressing mode. Sometime it may also contain OPCODE part. After OPCODE and addressing mode bytes, we have following different cases :

(a) No additional bytes [fig.1(a), fig.1(b), fig.1(c), fig.1(d)]

(b) A 2 byte EA (for direct addressing mode [fig.1(e)].

(c) A 1 or 2 byte immediate operand [fig.1(f)]

(d) A 1 or 2 byte displacement followed by 1 or 2 byte immediate operand [fig.1.(g)]

REG - Register
MOD - Mode
R/M - Register or memory
DISP - Displacement
DATA - Immediate data

One - byte instruction - implied operand(s)

(a) | OPCODE |

One - byte instruction - register mode

(b) | OPCODE | REG |

Register to register

(c) | OPCODE | 11 | REG | R/M |

Register to/from memory with no displacement

(d) | OPCODE | MOD | REG | R/M |

Register to/from memory with displacement

(e) | OPCODE | MOD | REG | R/M | Low-order DISP | High-order DISP |
(If 16-bit displacement is used)

Immediate operand to register

(f) | OPCODE | 11 | OP CODE | R/M | Low-order DATA | High-order DATA |
(If 16-bit data are used)

Immediate operand to memory with 16-bit displacement

(g) | OPCODE | MOD | OP CODE | R/M | Low-order DISP | High-order DISP | Low-order DATA |
(If 16-bit data are used)

Fig.(1) : Summary of 8086 instruction format

If a displacement or immediate operand is 2 bytes long, the low order byte always appears first, this is Intel standard.

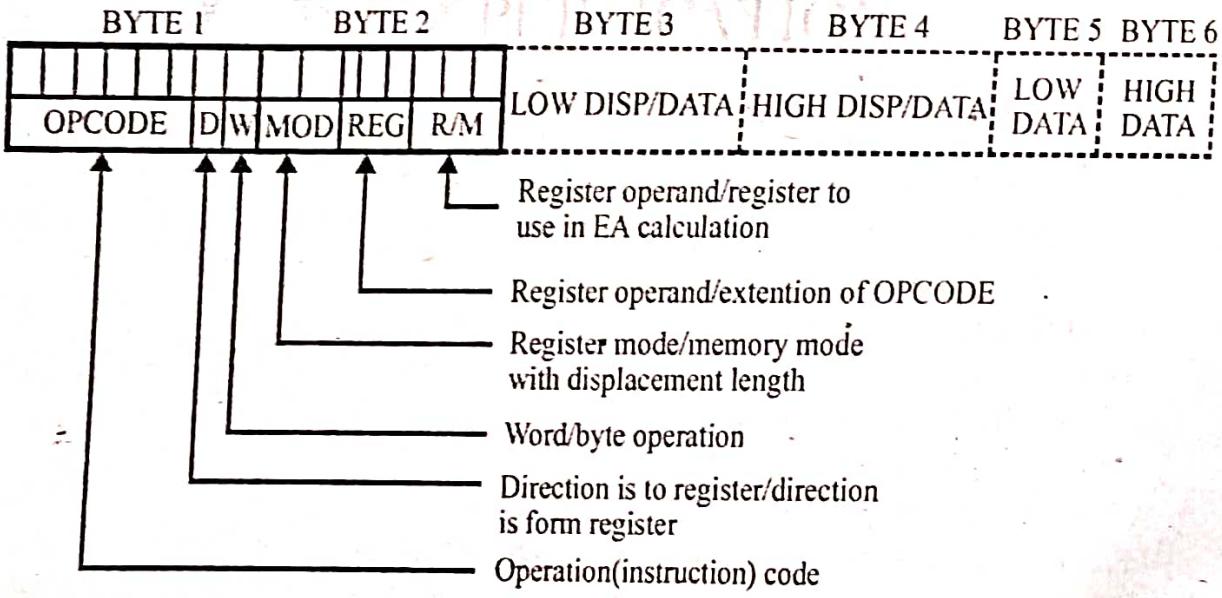


Fig.(2) : instruction format

As shown in fig.(2) the first six bits of multibyte instruction generally contains an opcode that identifies the basic instruction type i.e. ADD, XOR etc. The following bit, called the D field, generally specifies the direction of the operation.

D = 1 means instruction source is specified in REG field.

D = 0 means instruction destination is specified in REG field.

The next following bit is W. This bit identifies between byte and word operation.

W = 0 Instruction operates on byte data

= 1 Instruction operates on word data

In this we have three single bit fields S, V, Z.

S bit : S bit is used in conjunction with W to indicate sign extension of Immediate fields in arithmetic instructions.

S = 0 No sign extension

= 1 Sign extended 8 bit immediate data to 16 bits if W = 1.

Therefore for 8 bit operation : $S = W = 0$

16 bit operation with a 16 bit immediate operand : $S = 0$, $W = 1$

16 bit operation with a sign extended 8 bit immediate operand : $S = W = 1$

V bit : Used by shift and rotate, to determine single and variable – bit shifts and rotate.

V = 0 shift/rotate count is one

= 1 shift/rotate count is specified in CL register.

Z bit : This bit is used as a compare bit with zero flag in conditional repeat (REP) and loop instructions.

Z = 0 Repeat/loop while zero flag is clear

= 1 Repeat/loop while zero flag is set.

MOD : The mode (MOD) field indicates whether one of the operands is in memory or whether both operands are register.

REG : The Register (REG) field identifies a register that is one of the instruction operands REG field depends upon W bit.

R/M (Register or memory) : This field is of 3 bits. The meaning of R/M bits changes depending upon mode (MOD) field.

Q.5.(b) Explain the purpose of EU and BIU in 8086. **(10)**

Ans. BUS Interface Unit (BIU) : The BIU interface 8086 to the outside word. It provides a full 16 bit bidirectional data bus and 20bit address bus. The BIU is responsible for performing all external bus operations as given below :

(i) It sends address of the memory or I/O.

(ii) It fetches instruction from memory.

(iii) It reads data from port/memory.

(iv) It writes data into port/memory.

(v) It supports instruction queuing.

(vi) It provides the address location facility.

The BIU has a dedicated order. The main function of this order is to produce 20 bit physical address. The bus control logic of the BIU generates all bus control signals such as READ and WRITE for memory and I/O.

Instruction Queue : To speed up program execution, the BIU fetches six instruction bytes ahead of time from memory. These prefetched instruction bytes are held for the execution unit in a group of registers called queue. With the help of queue it is possible to fetch next instruction while current instruction is in execution. There are number of instructions in 8086 which need a quite large number of clock cycles for execution. During this execution time the

BIU fetches the next instruction or instructions from memory into the instruction queue instead of remaining idle. The BIU continues this process as long as the queue is not full. Due to this, execution unit gets, the ready instruction in the queue and instruction fetch time is eliminated (while decoding or executing an instruction EU does not require use of the buses).

This system has the advantage over the 8085 because, while the EU is executing an instruction, the BIU is fetching and storing in the queue the next instructions.

The BIU's instruction queue is based on first in first out (FIFO). So that the EU gets the instructions for execution in the order they are fatched. If the queue is full and EU does not request BIU for accessing memory, the BIU does not perform any bus cycle. On the other hand, if the queue is not full and even through the EU does not request BIU for accessing the memory the BIU can fill the queue on its own. If the EU interrupts the BIU, the BIU Ist completes the prefetching and then attains to the service of the EU.

In case of JUMP and CALL instruction, instruction already fetched in queue are of no use. Hence, in these cases queue is dumped and newly formed by loading instructions from new address specified by JUMP or CALL instruction.

Execution Unit (EU) : The EU of 8086 tells the BIU from where to fetch instructions or data, decodes instruction and executes instructions. It contains :

(i) Control Circuitry
(ii) Instruction Decoder
(iii) Arithmetic Logic Unit (ALU)
(iv) Flag register
(v) General purpose registers.
(vi) *Pointers* and Index registers.

The central circuitry in the EU directs the internal operation. A decoder in the EU translates the instructions *fetched* from memory into a series of actions which the EU performs. ALU is 6 bit. If can add, subtract, AND, OR, XOR, increment, decrements complement and shift binary numbers.

More about Queue : In the beginning, the CS : IP is loaded with the required address from which the execution is to be started. In the initial condition the queue will be empty and the microprocessor starts a fetch operation to bring one byte (the first byte) of instruction code, if the CS : IP address is odd, and two bytes at a time, if the CS : IP address is even. The first byte is a complete opcode in case of some instructions(one byte opcode instructions), the remaining part of opcode may lie in the second byte. But invariably the first byte of an instruction is an opcode. These opcodes along with data are fetched and arranged in the queue. When the first byte from the queue goes for decoding and interpretation, one byte in the queue becomes empty and subsequently the queue is up dated.

(i) The microprocessor does not perform the next fetched operation till at least two bytes of the instruction queue are emptied. The instruction execution cycle is never broken for fetch operation. After decoding the first byte the decoding circuit decides whether the instruction is of single opcode byte or double opcode byte. If the single opcode bytes, the next bytes are treated as data bytes depending upon the decoded instruction length. Otherwise, the next byte in the queue is treated as the second byte of the instruction opcode.

(ii) The queue is updated after every byte is read from the queue but the fetch cycle is entreated by BIU only if at least two bytes of the queue are empty and the EU may be concurrently executing the fetched instructions. Fig. shows the queue operation.

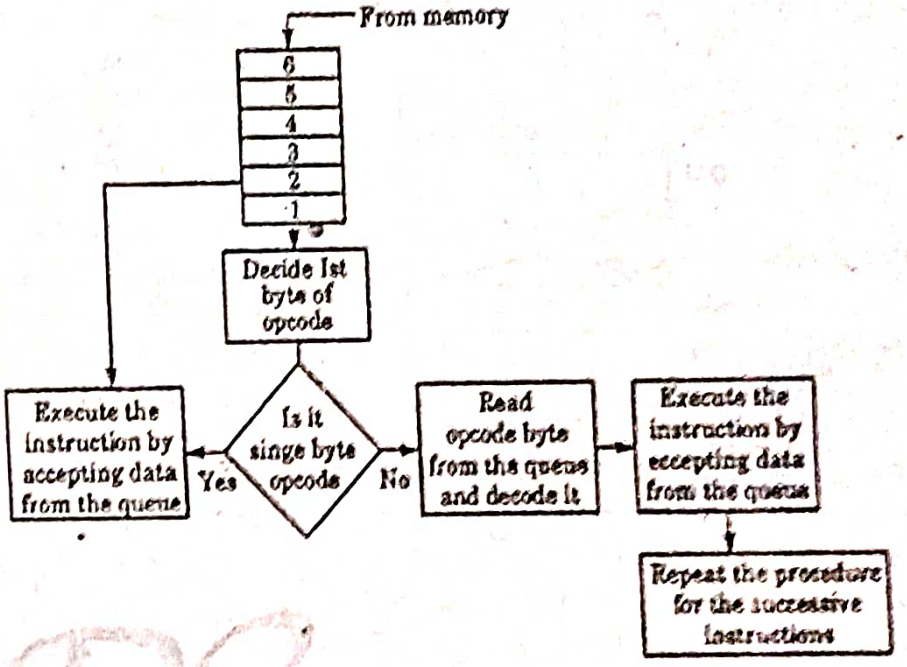


Fig : Queue operation

Unit – III

Q.6.(a) Write the purpose of following instruction in 8086 with example. (10)

(i) LOOP

(ii) RET

(iii) AAM

(iv) MOVSB

Ans. (i) LOOP : This instruction is used to repeat a series of instructions some number of times. The number of times the instruction sequence is to be repeated is loaded into CX. Each time loop executes CX is decremented by 1.

- If CX ≠ 0 execution will jump to destination specified by label.
- If CX = 0 execution will go to the next instruction after loop.

(ii) RET : It POPs a word (16–bit) from the top of the stack into the IP (near return) and places it in Ip and CS or 32–bit number (far return) and places it in IP and CS. The execution starts where from it left the main program. Its object code is C3. The stack has 0003H = IP. CS = 1000H. the new address = 10000H = 0003H in Fig.

(iii) AAM : Numerical data coming into a computer from a terminal through keyboard is usually in ASCH code. The numbers 0 to 9 are represented by ASCH codes 30 H to 39 H.

Before multiplying two ASCH digits, the upper nibble bits of each need to be masked. This leaves unpacked BCD in each byte. After the two unpacked BCD digits are multiplied, the AAM instruction is used to adjust the product of two unpacked BCD digits in AX.

It works only on registor AL.

It is used after multiplying the two unpacked BCD numbers.

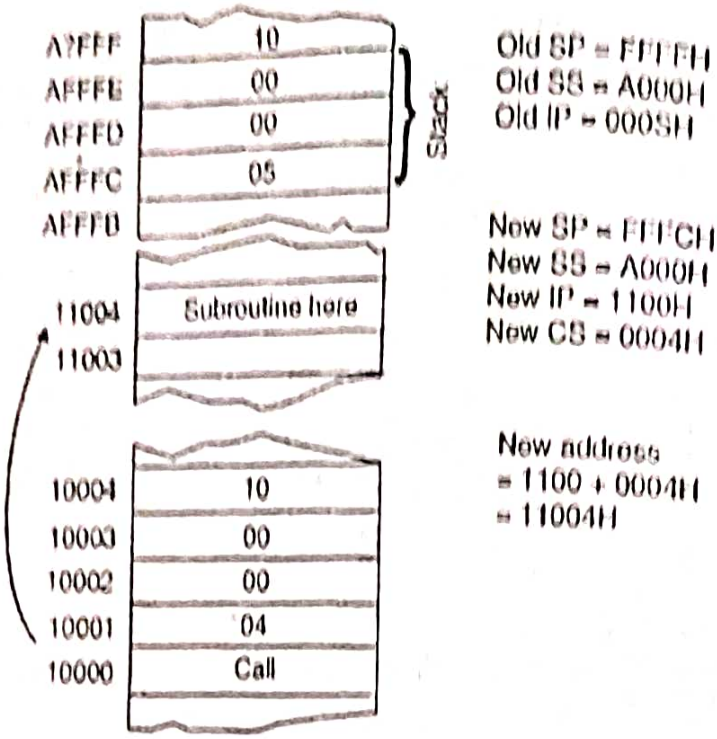


Fig. : Near RET

(iv) MOVSB (Move string) : This instruction copies a byte or a word from a location in the data segment to a location in the extra segment.

The offset of the source byte/word in the DS must be SI register.

The offset of the destination in ES must be in DI register.

After byte or word is moved, SI and DI are automatically adjusted to point to the next source and next destination.

Q.6.(b) Explain directives used in assembly language program ? (10)

Ans. Assembler directives : Assembler directives are statements which give direction to the assembler to perform the task of assembly process. These are not translated into machine code.

Commonly used assembler directive in 8086 assembly language programming are explained below :

(i) ASSUME : The assume directive is used to tell the assembler the name of the logical segment it should used 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 the 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.

(ii) ALIGN : The align directive is used to align the next segment at an address divisible by specified number. The general syntax for this directive is as shown below :

ALIGN n

where n can be 2, 4, 8 or 16

(iii) **CODE :** The code directive is used to provide shortcut in definition of the code segment. General syntax for this directive is as shown below :

.code [name]

The name is optional.

(iv) **DATA :** The data directive is used to provide shortcut in definition of the data segment.

(v) **GROUPS :** A program may contain several segments of the same type i.e. code, data, or stack. The purpose of the GROUP is to collect them all under one but, so that they reside within one segment, usually a data segment.

Format : Name GROUP Seg-name,....., Seg-name.

(vi) **LENGTH :** It is an operator which tells the assembler to determine the number of elements in some named data item such as a string or array.

(vii) **MACRO and ENDM :** The macros in the program can be defined by MACRO directive. ENDM directive is used along with the MACRO directive. ENDM defines the end of the macro.

(viii) **NAME :** The name directive is used at the start of a source program to give specific names, to each assembly module.

(ix) **OFFSET :** It is an operator which tells the assembler to determine the offset or displacement of a named data item (variable) from the start of the segment which contains.

(x) **ORG :** It is an assembler that uses a location counter to account for its relative position in a data or code segment.

Format : ORG expression

(xi) **PAGE :** The PAGE directive help to control the format of a listing of an assembled program. At the start of a program the PAGE directive specifies the maximum number of lines to list on a page and the maximum number of characters on a line.

Format : PAGE [length], [width]

(xii) **ENDP :** ENDP directive is used along with the PROC directive. ENDP defines the end of the procedure.

(xiii) **TYPE :** It is an operator which tells assembler to determine the type of specified variable. Assembler determines the type of specified variable in number of bytes. For byte type variable the assembler gives a value of 1. For word type variable the assembler gives a value of 2 and for double word type variable the assembler gives a value of 4.

Q.7. Write an ALP of 8086 to add series of N 16 bit numbers. (20)

Ans. Program statement : Write an ALP to add a block of N numbers. Assume the result to be 16 bit.

Explanation :

- Consider that a block of N bytes is present at source location.
- Let the number of bytes N = 10 for example.
- We have to add these N bytes.
- We will initialize this as count in the CX register.
- We know that source address is in the SI register. This SI register will act as pointer.
- Clear the direction flag.
- Using ADD instruction add the contents, byte by byte of the block.

- Increment SI to point to next element.
- Decrement the counter and add the contents till all the contents are added.
- Result is stored in AX.

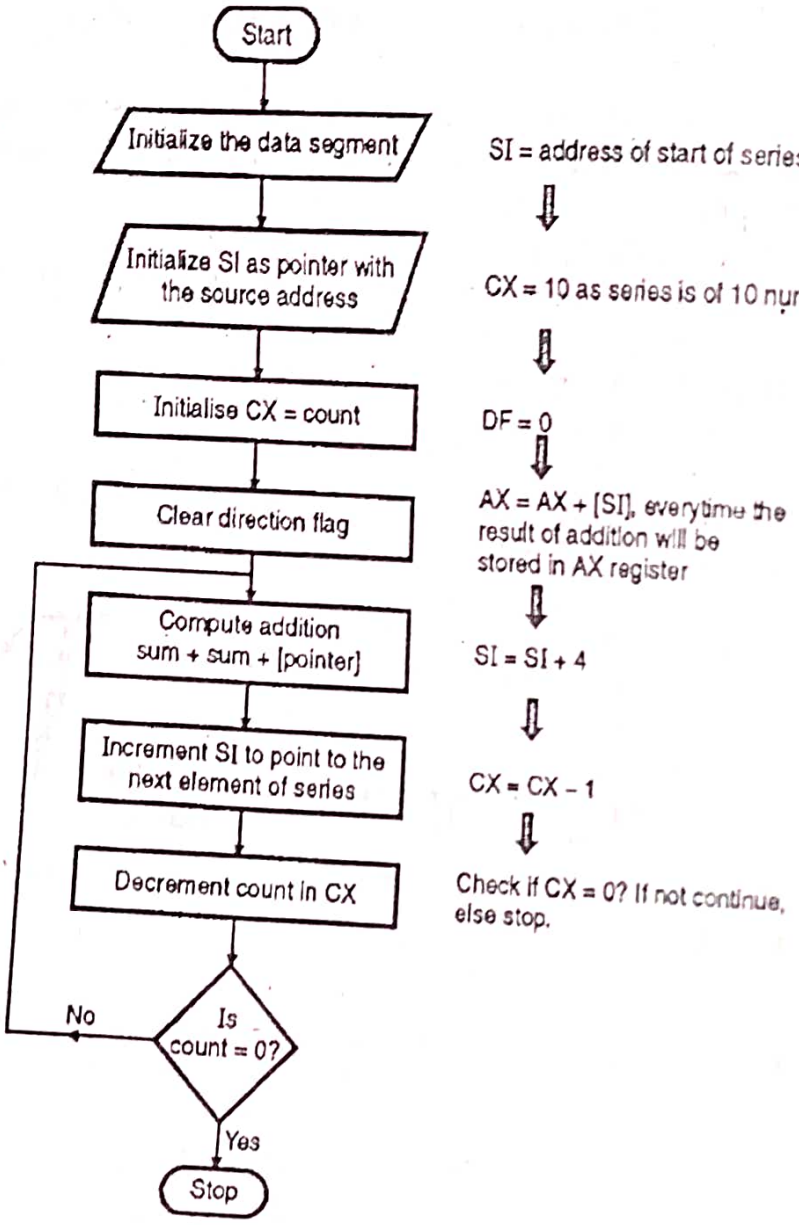


Fig. : Flowchart

Algorithm :

Step I	:	Initialise the data segment.
Step II	:	Initialise SI as pointer with source address.
Step III	:	Initialise CX register with count.
Step IV	:	Initialise direction flag to zero.
Step V	:	Add data, word by word.
Step VI	:	Increment pointer i.e. SI by 2 as 16 bit addition.
Step VII	:	Decrement counter CX.

Step VIII : Check for count in CX, if not zero goto step V else goto step IX.

Step IX : Store the result of addition.

Step X : Stop.

Flowchart : Refer flowchart.

Program :

Label	Instruction	Comment
	.model small	
	.data	
	series db 0111H, 0231H, 0341H, 0456H, 0578H,	
	06ABH, 0733H, 0845H, 0976, OA12H	
	count dw OAH	
	.code	
	mov ax, @data	initialise data segment
	mov ds, ax	
	mov ax, 0	
	mov si, offset blk 1	initialise pointer
	mov cx, count	initialise counter
	cld	df = 0
11:	add ax, [si]	add numbers
	inc si	increment pointer
	inc si	increment pointer
	dec count	decrement counter
	jnz11	check if all nos are added
	end	

Result : AX = 39FCH

Unit – IV

Q.8. Explain the working of 8255 in mode 2 and BSR Mode. Also explain how the contents of control registers are interpreted in BSR and I/O mode. (20)

Ans. Working of 8255 in mode 2 : In this mode group A is used as input and output i.e. for transmitting and receiving data from peripheral through 8255 as shown in fig.(a).

The transfer of data is achieved by port C handshake signals. The group B can be in Mode 0 or Mode 1.

The bi-directional data is transferred through port A so it consists of input and output latch.

The Mode 2 is combination of Mode 1 input and output both at a time to port A.

The interrupt signals of input and output mode are combined to generate common interrupt signal to CPU. The internal organization of these signals is as shown in fig. (b).

The different handshake signals used are $\overline{OBF_A}$, $\overline{ACK_A}$, $\overline{STB_A}$, IBF_A and $INTER_A$ handshake signals are used for output operation, 2 are used for input operation and one is common to both.

Output operation :

$\overline{OBF}$ (Output buffer full) : This is an active low output signal generated by 8255. When CPU writes data to output port 8255 will enable OBF signal to indicate peripheral that data is available in output buffer.

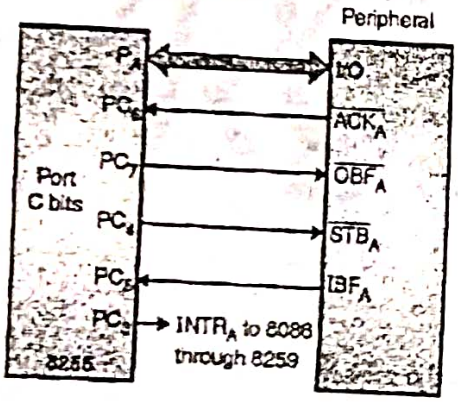


Fig. (a) : Mode 2 interfacing

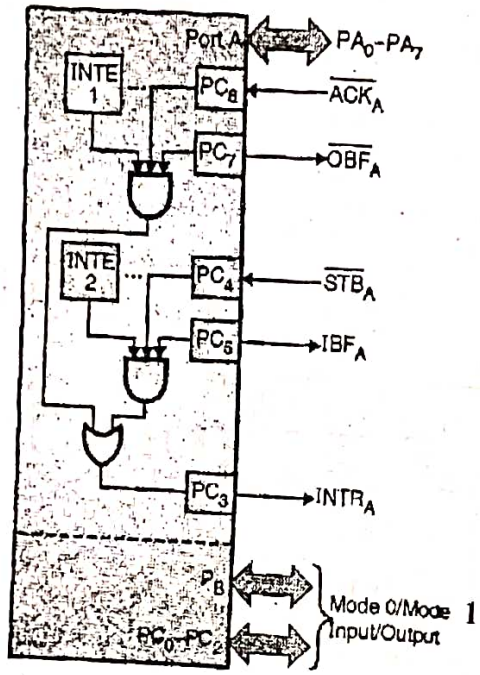


Fig. (b) : P_A, P_B and P_C in mode 2

$\overline{ACK}$ (Acknowledge) : This is an active low input signal for 8255. When the peripheral detects $\overline{OBF}$ signal, it reads data from 8255 port and makes $\overline{ACK}$ = 0 and the ACK signal is used to acknowledge 8255 that data is read from port so 8255 will remove $\overline{OBF}$ signal to indicate output buffer is empty.

Input operation :

$\overline{STB}$ (strobe) : This is an active low input signal. When the peripheral writes data to input buffer, it generates a signal STB to indicate 8255 that it has written data.

IBF (Input buffer full) : When data is available in input buffer 8255 will enable IBF signal to indicate that data is available in input buffer.

INTR (Interrupt request) : This is an output signal given by 8255 to request CPU service.

The INTR is generated in two different conditions input and output.

The interrupt is generated for input mode when IBF = 1, $\overline{STB}$ = 1 and $INTE_1$ = 1 and for output mode when $\overline{OBF}$ = 1, $\overline{ACK}$ = 1 and $INTE_2$ = 1. The $INTE_1$ and $INTE_2$ are set/reset using BSR mode, port C bits used are PC_6 and PC_4 respectively.

The logical equation will be,

$$INTR_A = INTE_1.\overline{ACK_A}.\overline{OBF_A} + INTE_2.\overline{STB_A}.IBF_A$$

The timing diagram of Mode 2 bi-directional data transfer for data transfer from peripheral to CPU and CPU to peripheral are as shown in Fig. (c).

The mode 2 also supports both modes of data transfer i.e. Interrupt drive I/O and status driven I/O. The port C is used as status word and its definitions are as follows :

$\overline{OBF_A}$ $INTE_1$ IBF_A $INTE_2$ $INTR_A$ x x x

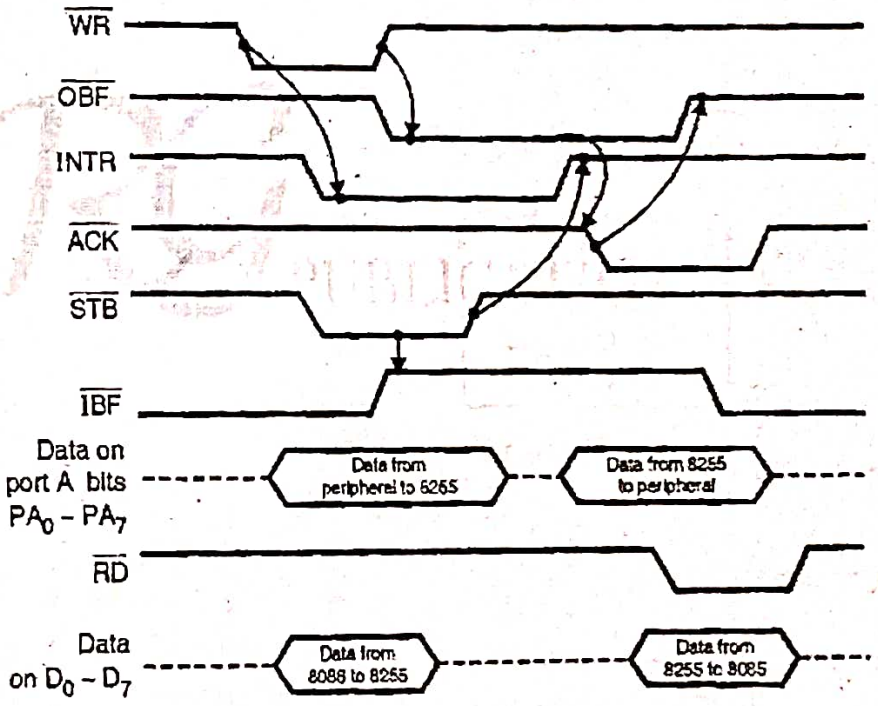


Fig. (c) : Timing diagram of mode 2

Working of 8255 in BSR : The BSR is a Port C bit set reset mode. The mode used eight bits of Port C only. The individual bit of Port C can be set or reset by writing control word in the control register.

The control word format of BSR mode is as shown in Fig.(d)(i) and mode selection formal is in fig.(d)(ii).

The pin of Port C, *i.e.*, Bit 0 to Bit 7 to set or rest is chosen using Bit select bits $b_2b_1b_0$ *i.e.*, D_3, D_2 and D_1, of control word register. The bit to set or reset is decided by bit S/R, i.e., D_0. The BSR mode affects only one bit of Port C at a time. The bit set using BSR mode remains set unless and until you change the bit. The bit to be set/reset is decided by control word. So to set any bit of Port C, bit pattern is loaded in control register. Even though a BSR mode is selected it will not affect I/O mode.

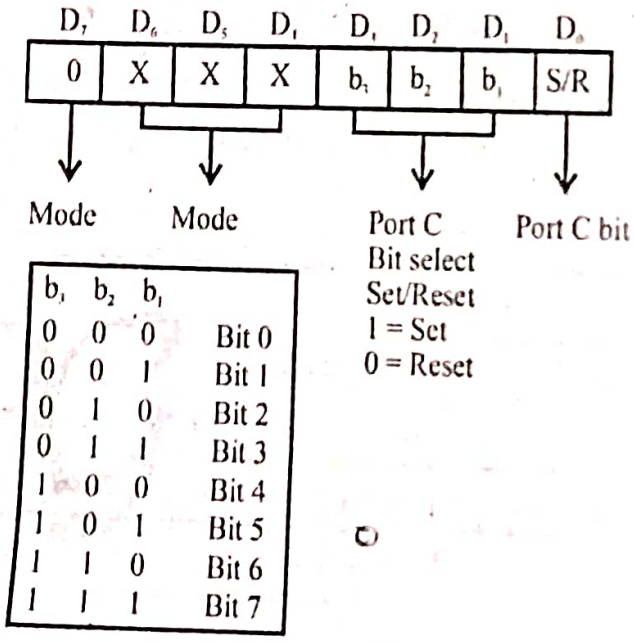


Fig : (d)(i)

Control Word :

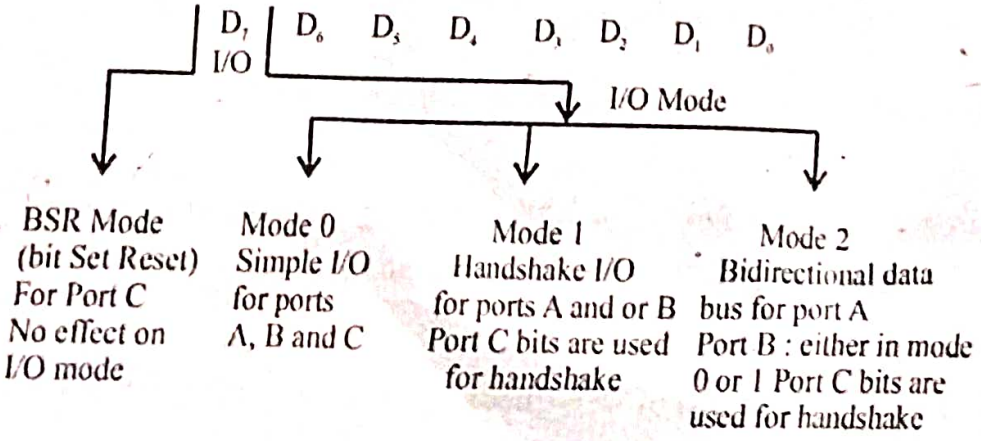


Fig : (d)(ii)

Q.9.(a) Explain pin diagram of 8237 DMA controller. **(10)**

Ans. The fig. shows pin configuration of 8237.

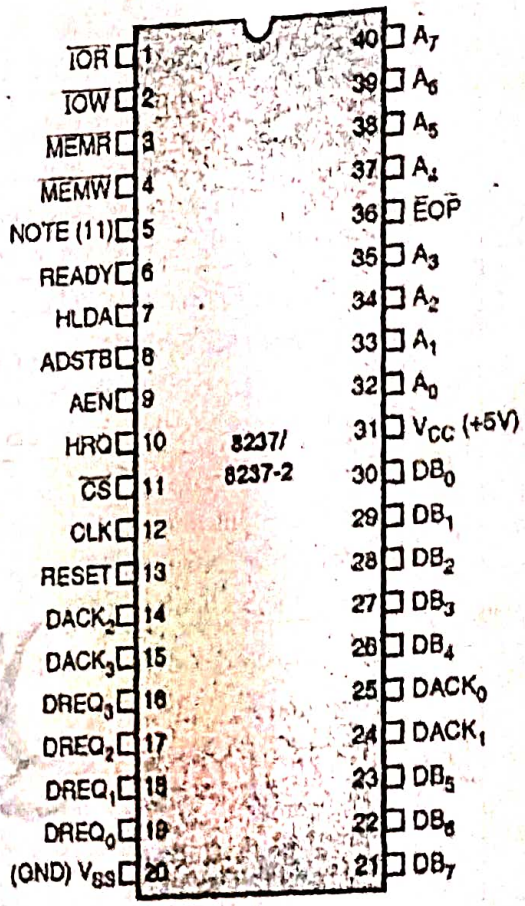


Fig. : Pin configuration of 8237

Symbol	Description
CLK	This is a clock input line ignored in slave mode. In master mode this signal controls all internal and external DMA operations. The data transfer rate depends upon the frequency of this signal.
$\overline{CS}$	In slave mode, this signal is generated by address decoder to select 8237 chip for communication between CPU and 8237. In master mode, this signal is ignored.
Reset	It is an asynchronous input line. This signal clears the command, status, request and temporary register and forces 8237 into slave mode.
READY	In master mode, this signal is used to add wait states into a DMA cycle.

HRQ	It is a hold request output line. It is connected to hold input of the CPU. It is used to request control of the system bus.
HLDA	It is a hold acknowledge input line. This signal is generated by CPU. In response to this signal, the 8237 gains control of the system bus and enters into master mode.
$\overline{IOR}$	It is an active low bi-directional tristate line. In slave mode, it acts as an input line and used to read contents of 8237 registers. In master mode, it acts as an output line. This signal is generated during DMA cycle to read data from I/O device.
$\overline{IOW}$	It is an active low bi-directional line. In slave mode, it acts as an input line and used by CPU to write contents to 8237 registers. In master mode, it acts as an output line. This signal is generated during DMA read cycle to write data into I/O device.
A_0 - A_3	These are bi-directional, address lines. In slave mode, these lines act as input lines, used to select one of the registers of 8237. In master mode, the 8237 provides lower bits of memory address on these lines.
A_4 - A_7	These are tristate address output lines. These lines are tristated in slave mode. In master mode, the 8237 transfers bits of memory address on these lines.
$\overline{MEMR}$	It is an active low tristate control output line. It is tristated in slave mode. In master mode, this signal is generated during DMA read cycle or during memory to memory transfer cycle to read contents of source memory.
$\overline{MEMW}$	It is an active low tristate output line. It is tristated in slave mode. In master mode, this signal is activated during DMA write or during memory to memory transfer cycle to read contents of destination memory.
DB_0 - DB_7	These are bi-directional tristate buffered data lines. In slave mode, these lines are used to transfer data between CPU and 8237's registers. In master mode, these lines act as address output lines. The 8237 places higher byte of address on these lines during DMA cycles.
AEN	This active high output enables the 8 bit latch that drives the upper 8 bit address bus. The AEN pin is used to disable other bus drivers during DMA transfers.
ADSTB	This output line is used to strobe the upper address byte generated by 8237 in master mode into an external latch.

$DREQ_0$ - $DREQ_0$	These are asynchronous DMA channel request lines used by peripheral. The polarity of each signal is programmable i.e. these lines can be used as either active high or active low input. DREQ must be maintained until the corresponding DACK is activated.
$DACK_0$ - $DACK_3$	These are DMA acknowledge output lines. The polarity of each line is programmable. This signal indicates that the requesting peripheral has been granted fro DMA cycle.
$\overline{EOP}$	*End of Process :* It is an active low bi-directional signal. This line is also used to terminate DMA cycle. The DMA cycle can be terminated by pulling $\overline{EOP}$ input low. The 8237 also generates EOP pulse, when the terminal count for any channel is reached.

Q.9.(b) Explain BSR and I/O modes of 8255 PPI chips. **(10)**

Ans. BSR mode : It is a bit Set/Reset mode. This mode is concerned only with 8 bits of port C. A control word with bit D_7= 0 is recognized as BSR control word. It does affect any previously transmitted control word with bit D_7= 1 and thus the I/O operations of port A and port B are not affected by a BSR control word. In the BSR mode, the individual 8 bits of port C can be used for applications such as on/off switch.

BSR Control Word : The BSR control word written in the control register, sets or resets one bit at a time as shown in fig.

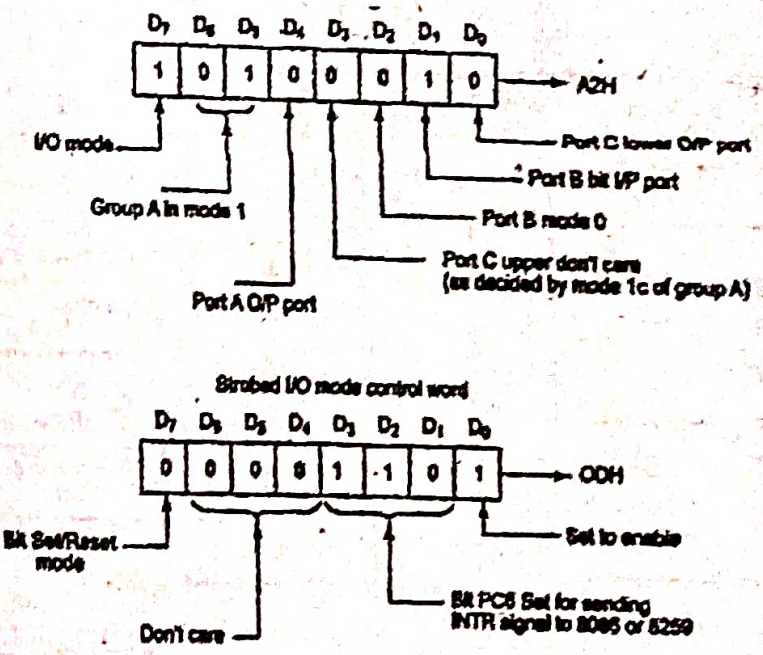


Fig. : Bit Set/Reset Control Word

I/O modes : There are three I/O modes of operation :

(1) mode 0 – Basic I/O

(2) Mode 1 – Strobed I/O

(3) Mode 2 – Bi-directional I/O

The I/O modes are programmed using control register. The control word format of I/O modes is as shown in fig.

Function of each bit is as follows :

(i) D_7 : When the bit $D_7 = 1$ then I/O mode is selected, if $D_7 = 0$ then BSR mode is selected . The function of bits D_0 to D_6 is dependent on mode (I/O mode or BSR mode).

(ii) D_6 and D_5 : In I/O mode the bits D_6 and D_5 specifies the different I/O modes for group A i.e. Mode 0, Mode 1 and Mode 2 for port A and port C upper.

(iii) D_4 and D_3 : In I/O mode the bits D_4 and D_3 selects the port function for group A. If these bits = 1 the respective port specified is used as input port. But if bit = 0, the port is used as output port.

(iv) D_2 : In I/O mode the bit D_2 specifies the different I/O modes for group B i.e. Mode 0 and Mode 1 for port B and port C lower.

(v) D_1 and D_0 : In I/O mode the bits D_1 and D_0 selects the port function for group B. If these bits = 1 respective port specified is used as input port. But if bit = 0, the port is used as output port.

From the above explanation you can observe that all the 3 modes i.e. Mode 0, Mode 1 and Mode 2 are only for group A ports, but for group B only 2 modes i.e. Mode 0 and Mode 1 are provided.

When 8255 is reset, it will clear control word register contents and all the ports are set to input mode. The ports of 8255 can be programmed for other modes by sending appropriate bit pattern to control register.

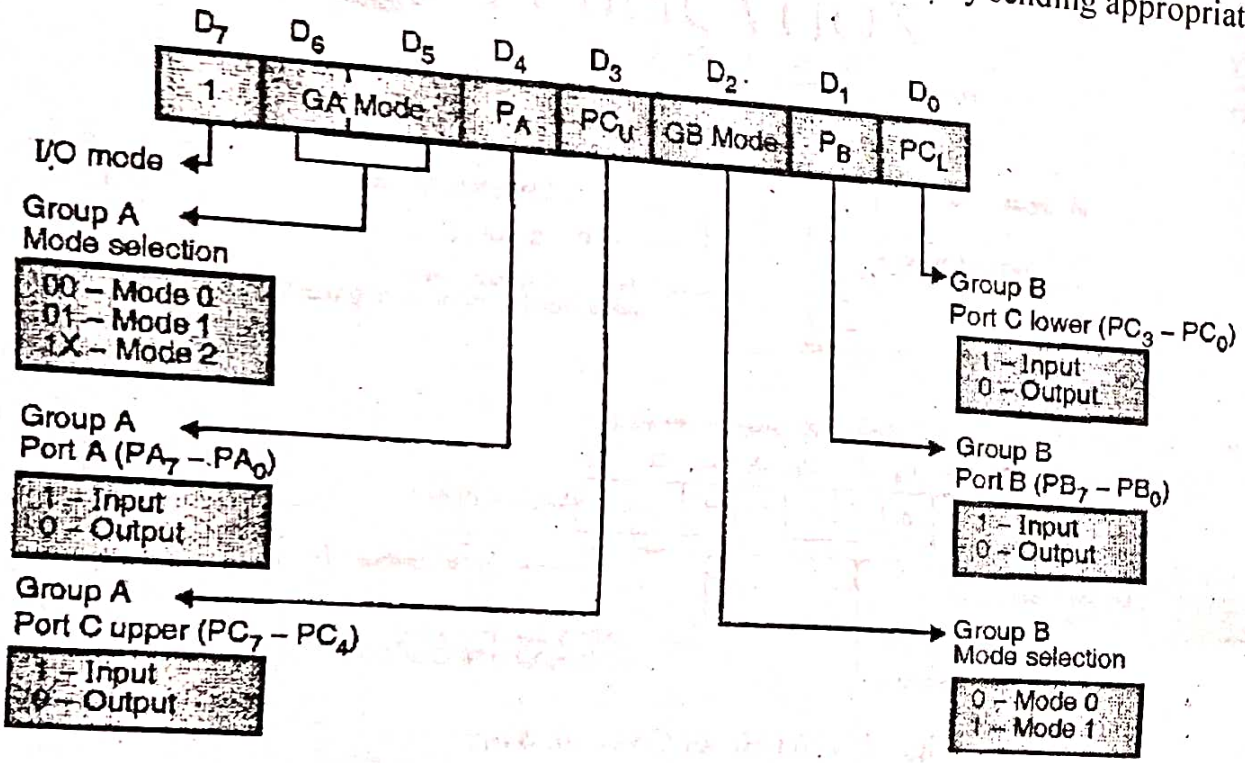


Fig. : I/O modes control word format

MICROPROCESSING & INTERFACING

Dec - 2016
Paper Code:-EE-309-F

Note : ***Attempt five questions in all, selecting one question from each Section. Question No. 1 is compulsory. All questions carry equal marks.***

Q.1.(a) Write a program of find 1's complement of the number. **(5)**

Ans. *Statement :* Find the 1's complement of the number stored at memory location 2200H and store the complemented number at memory location 2300H.

Sample Problem :

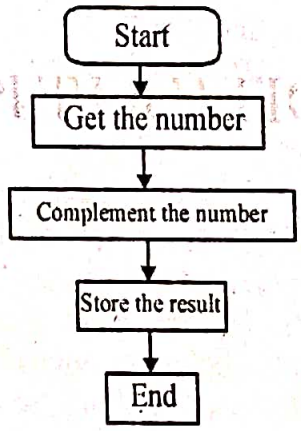
(2200H) = 55H

Result = (2300H) = A AH

Program

LDA 2200H → Get the number
CMA → Complement number
STA 2300H → Store the result
HLT → Terminate program execution.

Flowchart



Q.1.(b) What are NOP and HLT instructions ? **(5)**

Ans. NOP instruction :

Mnemonic NOP **Flags :** It does not affect any flag.
NOP : No operation
Algorithm Do nothing
Addr. Mode Implied addressing mode
Operation The execution of this instruction causes the CPU to do nothing.
(i) This instruction causes the CPU to do nothing.This instruction uses three clock cycles and increments the instruction pointer to point to the next instruction.
(ii) It can be used to increase the delay of a delay loop.

Example :

```
MOV AL, 00011011b
NOT AL ; AL = 11100100b
RET
```

Flags : all unchanged.

HLT (Halt until interrupt or reset) Instruction :

Mnemonic Halt processing **Flags :** No flags are affected.

Operation (i) The HLT instruction will cause the 8086 to stop fetching and executing instructions. The 8086 enters into a half state. To come out of the halt state, there are 3 ways given below.
(a) Interrupt signal on INTR pin, (b) Interrupt signal on NMI pin
(c) Reset signal on reset pin.
(ii) It may be used as an alternative to an endless software loop in situations where a program must wait for an interrupt.

Q.1.(c) What is programmable interval timer ? (5)

Ans. There are two types of programmable interval timer are generally used. Intel 8253 is a programmable Interval Timer/Counter which can generate accurate time delays and waveforms ranging from 0 Hz to 2 MHz using software control. 8254 is its upgraded version which can operate with higher clock frequency range (DC – 8 MHz) and it is pin to pin compatible with 8253.

Q.1.(d) Explain direct memory access. (5)

Ans. Direct memory access (DMA) is a feature of modern computers that allows certain hardware subsystems within the computer to access system memory independently of the central processing unit (CPU).

Without DMA, when the CPU is using programmed input/output, it is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU initiates the transfer, does other operations while the transfer is in progress, and receives an interrupt from the DMA controller when the operation is done. This feature is useful any time the CPU cannot keep up with the rate of data transfer, or where the CPU needs to perform useful work while waiting for a relatively slow I/O data transfer. Many hardware systems use DMA, including disk drive controllers, graphics cards, network cards and sound cards. DMA is also used for intra-chip data transfer in multi-core processors. Computers that have DMA channels can transfer data to and from devices with much less CPU overhead than computers without a DMA channel. Similarly, a processing element inside a multi-core processor can transfer data to and from its local memory without occupying its processor time, allowing computation and data transfer to proceed in parallel.

SECTION – A

Q.2.(a) Define interrupts ? Explain 8085 interrupts. (10)

Ans. Interrupts : *The interrupt driven I/O is one of the data transfer techniques used in the microprocessor systems. By using this techniques, the external device or a peripheral can inform the processor that it is ready for communication.*

8085 microprocessor provides hardware and software interrupts.

Hardware Interrupts : 8085 microprocessor provides five hardware interrupt viz. TRAP, RST 7.5, RST 6.5, RST5.5 and INTR. The interrupt structure is a five level structure.

(i) *TRAP :* It is non-maskable edge and level triggered interrupt, request input line. It is used for emergency purpose like power failure, parity error checker, smoke detector etc. The microprocessor does not execute any interrupt acknowledge cycle to read interrupt information from the interrupting device. The interrupt information is provided by control section of microprocessor internally. But microprocessor executes ideal machine cycle to acknowledge this interrupt. To generate starting address of TRAP interrupt service routine. The TRAP signal must make low to high transition and remain high until acknowledged that means this interrupt is triggered only at the rising edge of the signal. The avoids false trigerring due to noise or glitches. It is not affected by any instruction. It has the highest priority among all interrupt. It is always enabled. This interrupt transfers microprocessors control to location 0024 H. User cannot rest TRAP flip-flops that means we cannot cancel this interrupt.

(ii) *RST 7.5 :* It is maskable edge triggered interrupt request input line. The microprocessor does not execute any interrupt acknowledge cycle to read interrupt information from the interrupting device. The interrupt information is provided by control selection of microprocessor internally. Instead of interrupt acknowledge cycle, the microprocessor executes ideal machine cycle (6T) to acknowledge this interrupt. During this cycle it executes RST 7.5. Instruction to generate starting address of interrupt service routine. This interrupt is triggered at the rising edge of the signal. It priority among all maskable interrupt. This interrupt to location 003CH. User can reset R 7.5 flip-flop that means we can cancel this interrupt by SIM instruction.

(iii) *RST 6.5 and RST 5.5 :* These are level triggered maskable interrupt request input lines. The microprocessor does not execute any interrupt acknowledge cycle to read interrupt information from the interrupting device. The microprocessor executes idle machine cycle (6T) to acknowledge these interrupts. Durring this cycle it executes RST 6.5 and RST 5.5 instructions to generate address of ISR 6.5 and ISR 5.5 respectively. They can be disabled by executing SIM or EI instruction. RST 6.5 transfers microprocessor's control to location 0034 H while RST 5.5 transfers microprocessor's control to location 002 CH.

(iv) *INTR :* It is level triggered, maskable interrupt request input line. The microprocessor executes interrupt acknowledge cycle to read interrupt information from interrupting device. The microprocessor executes one interrupt acknowledge cycle (6T) and three interrupt acknowledge cycles (6T + 3T + 3T) for RST N and CALL instructions respectively.

The starting address of ISR depends upon interrupt information. This interrupt is not affected by SIM instruction. It is enabled by executing EI instruction while disabled by DI instruction.

Software interrupts : (i) In case of software interrupts the cause of the interrupt is the execution of the instruction.

(ii) The microprocessor 8085 has eight instructions. These eight instructions are RST 0 to RST 7. Such interrupts are called as software interrupts.

(iii) They allow the microprocessor to transfer program control from the main program to the subroutine program (i.e. predefined service routine addresses).

(iv) After completing the subroutine program, the program control returns back to the main program.

Q.2.(b) Describe various arithmetic instructions with examples in 8085. (10)

Ans. The arithmetic group of instructions include following instructions :

(i) ADD R	(ii) ADD M	(iii) ADC R	(iv) ADC M		
(v) ADI data	(vi) ACI data	(vii) DAD R_P	(viii) SUB R		
(ix) SUB M	(x) SBB R	(xi) SBB M	(xii) SUI data		
(xiii) SBI data	(xiv) INR R	(xv) INR M	(xvi) DCR R		
(xvii) DCR M	(xviii) INX R_P	(xix) DCX R_P	(xx) DAA		

Some of the describe as follows :

ADD R :

Description : Add register R contents to accumulator.
This instruction adds the contents of register R and accumulator and stores the result in accumulator. The example of R are all general purpose registers such as A, B, C, D, E, H and L. In addition to the result in accumulator all the flags are modified to reflect the result of operation.

Operation : $A + R \rightarrow A$

Example : ADD : $A + B \rightarrow A$.
Suppose A = 40 H and B = 65 H.

```
  A  0100  0000
+B   0110  0101
  A  1010  0101   = A5
```

ADC R :

Description : Add register R cut carry flag contents to accumulator.
This instruction adds the contents of Register R, Carry flag Cy and accumulator and stores the result in accumulator. The example of R are all general purpose registers such as A, B, C, D, E, H and L. In addition to the result in accumulator all the flags are modified to reflect the result of operation.

Operation : $A + R \rightarrow C_y \rightarrow A$

Example : ADC : $A + B + C_y \rightarrow A$.
Suppose B = 20, A = 3 F and C_y = Set and ADC B instruction is executed.

```
 A  = 0011   1111
 B  = 0011   0000
Cy  =                1
 A  = 0110   0000
```

ADI data :

Description : Add immediate data to accumulator.
This instruction adds the 8 bits of data specified along with the instruction to accumulator and result is stored in accumulator. All flags are also modified to reflect the result of operation. The storing format of this instruction will be 1st byte opcode and 2nd byte operand (data).

Operation : A + data → A

Example : ADI B7 H : Add B7 H data to accumulator and store result in accum-ulator.

Suppose A = 59 H and instruction ADI B7 is executed.

```
A  = 0 1 0 1     1 0 0 1
M  = 1 0 1 1     0 1 1 1
   1 0 1 1 0     0 0 0 0
```

DAD R_P :

Description : Add the specified register pair to HL pair.
This instruction adds the contents of specified register pair to HL pair and stores the result in HL pair. The example of R_p are SP, BC, DE and HL. Only carry flag is modified to reflect the result of operation.

Operation : $R_P + HL \rightarrow HL$

Example : (i)DAD B : BC + HL → HL
(ii)DAD SP : SP + HL → HL
DAD B
Suppose B = 20, C = 35, H = 80, L = 45, is executed.
The result of instruction will be

```
  2 0 3 5     B C
+ 8 0 4 5     H L
  A 0 7 A     H L
```

SUB M :

Description : Subtract memory location contents from accumulator.
This instruction subtracts memory location contents (whose address is given by HL) from accumulator and result is placed in accumulator. The subtraction is performed in the sameways SUB R instruction. All flags are modified to reflect the result of operation.

Operation : A – (HL) → A

Example : SUB M : A – (HL) → A.
Suppose A = 50H, H = C2, L = 00, at memory location C200 : 20 H is stored and instruction SUB M is excuted.

```
(C200)  = 0 0 1 0     0 0 0 0
2's com = 1 1 1 0     0 0 0 0
    A   = 0 1 0 1     0 0 0 0
2's comp. = 1 1 1 0   0 0 0 0
          [1] 0 0 1 1 0 0 0 0
```

SUI data :

Description : Subtract immediate data from accumulator.
This instruction subtracts the data specified along with instruction from accumulator. The subtraction is performed by using 2's complement method and operation is same as SUB R instruction.

Operation : A – data → A

Example : SUI 50 : A – 50 → A.
Suppose A = 20 and instruction SUI 50 is executed.

INR M :

Description : Increment memory contents by one.

This instruction increments the contents of memory location address by HL register pair by 1 and result is stored back at same memory location. Only carry flag is not modified, all other flags are modified.

Operation : (HL) + 1 → (HL) or M + 1 → M

Example : INR M : (HL) + 1 → (HL).

Suppose H = C2, L = 02, at memory location C202 : 04 is stored, flag reg = 10 × 1 × 0 × 1 and and instruction INR M is executed.

Q.3.(a) Explain pin diagram of 8085 microprocessor. **(10)**

Ans. Fig. shows functional pin diagram of 8085 microprocessor. The signals of 8085 can be classified into seven groups according to their functions :

(i) Power supply and frequency signals.
(ii) Data bus and address bus
(iii) Control bus
(iv) Interrupt signals
(v) Serial I/O signals
(vi) DMA signals
(vii) Reset signals

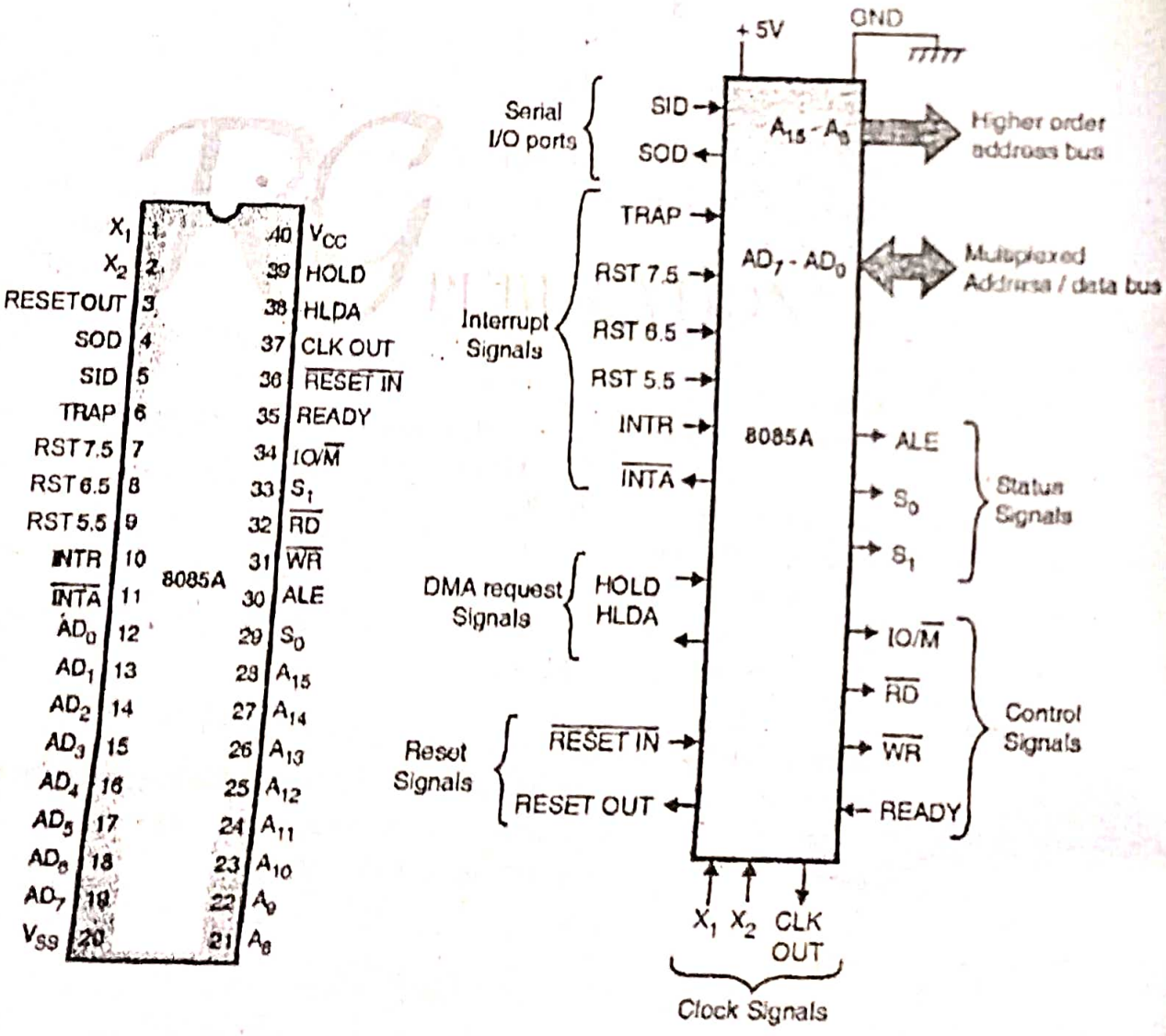


Fig. : Pin diagram of 8085 microprocessor

(i) Power Supply and Frequency Signals :

(a) V_{CC} : It requires a single +5V power supply.

(b) V_{SS} : Ground reference

(c) X_1 and X_2 : A tuned circuit like LC, RC or crystal is connected at these two pins. The internal clock generator divides oscillator frequency by 2, therefore, to operate a system at 3 MHz, the crystal of tuned circuit must have a frequency of 6 MHz.

(d) CLK OUT : This signal is used as a system clock for other devices. Its frequency is half of the oscillator frequency.

(ii) Data bus and address bus :

(a) AD_0 to AD_7 : The 8 bit data bus (D_0 - D_7) is multiplexed with the lower half (A_0 - A_7) of the 16 bit address bus. During first part of the machine cycle (T_1), lower 8 bits of memory address or I/O address appear on the bus. During remaining part of the machine cycle (T_2 - T_3) these lines are used as a bi-directional data bus.

(b) A_8 to A_{15} : The upper half of the 16 bit address appears on the address lines A_8 to A_{15}. These lines are exclusively used for the most significant 8 bits of the 16 bit address.

(iii) Control and status signals :

(a) *ALE* (Address Latch Enable) : We know that AD_0 to AD_7 lines are multiplexed and the lower half of the address is also necessary during T_2 and T_3 of machine cycle to access specific location in memory or I/O port. This means that the lower half of an address must be latched in T_1 of the machine cycle, so that it is available throughout the machine cycle. The latching of lower half of an address is done by external latch and ALE signal from 8085.

(b) $\overline{RD}$ and $\overline{WR}$: These signals are basically used to control the direction of the direction of the data flow between processor and memory or I/O device/port. A low on $\overline{RD}$ indicates that the data must be read from the selected memory location or I/O port via data bus. A low on $\overline{WR}$ indicates that the data must be written into the selected memory location or I/O port via data bus.

(c) IO/$\overline{M}$, S_0 and S_1 : IO/$\overline{M}$ indicates whether I/O operation or memory operation is being carried out. S_1 and S_0 indicate the type of machine cycle in progress.

(d) *Ready* : It is used by the microprocessor to sense whether a peripheral is ready or not for data transfer. If not, the processor waits. It is thus used to synchronize slower peripherals to the microprocessor. If peripherals are fast enough it is tied to V_{CC}. If it is left open, 8085 enters in the wait state.

(iv) Interrupt signals : The 8085 has five hardware interrupt signals : RST 5.5, RST 6.5, RST 7.5, TRAP and INTR. The microprocessor recognizes interrupt requests on these lines at the end of the current instruction execution.

The $\overline{INTA}$ (Interrupt Acknowledge) signal is used to indicate that the processor has acknowledged an INTR interrupt.

(v) Serial I/O signals :

(a) *SID* (Serial I/P data) : This input serial is used to accept serial data bit by bit from the external device.

(b) *SOD* (Serial O/P Data) : This is an output signal which enables the transmission of serial data bit by bit to the external device.

(vi) DMA Signals :

(a) *HOLD* : This signal indicates that another master is requesting for the use of address bus, data bus and control bus.

(b) *HLDA* : This active high signal is used to acknowledge HOLD request.

(vii) Reset signal :

(a) $\overline{RESET\ IN}$: A low on this pin

– Sets the program counter to zero (0000H).

– Resets the interrupt enable and HLDA flip-flops. Before entering any interrupts service routine we may have to initialize certain passing parameters. Thus it is not desired to activate interrupts at the RESET time.

– Tri-states the data bus, addres bus and control bus.

– Affects the contents of processor's internal registers randomly.

On reset, the PC sets to 0000H which causes the 8085 to execute the first instruction from address 0000H. For proper reset operation reset signal must be held low for at least 3 clock cycles. The power-on reset circuit can be used to ensure execution of first instruction from address 0000H.

(b) *RESET OUT* : This active high signal indicates that processor is being reset. This signal is synchronized to the processor clock and it can be used to reset other devices connected in the system.

Q.3.(b) What is the functioning of timing and control unit in 8085 microprocessor? Discuss all its signals in details. **(10)**

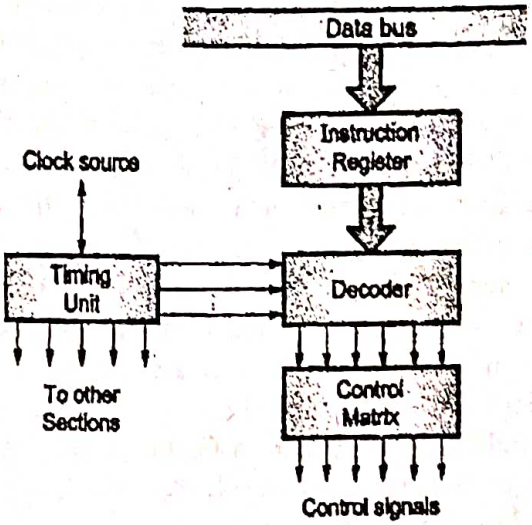
Ans. Timing and Control : This is a control section of 8085 made up of synchronous sequential logic circuit. It controls all internal and external circuits in the microprocessor system. It operates with reference to clock signal. It accepts information from instruction decoder and generates microsteps to perform it. In addition to this, the block accepts clock inputs, performs sequencing and synchronising operations. The synchronization is required for communication between microprocessor and peripheral devices. Fig. shows the control section of microprocessor.

The contents of the instruction register are in the form of 0's and 1's. They are converted to meaningful form by the decoding network called matrices. The control matrix provides internal signals for controlling operation and data between registers. The control unit also generates timing signals essential for microprocessor to operate. The microprocessor uses a quartz crystal (LC or RC circuit) to determine the clock frequency, so that other timing and control signals are developed. The speed of microprocessor is directly proportional to the speed of the crystal. The clock speed and access time must be compatible for maximum performance. To implement this it uses different status and control signals.

Control and status signals :

(a) *ALE* (Address Latch Enable) : We know that AD_0 to AD_7 lines are multiplexed and the lower half of the address is also necessary during T_2 and T_3 of machine cycle to access specific location in memory or I/O port. This means that the lower half of an address must be latched in T_1 of the machine cycle, so that it is available throughout the machine cycle. The latching of lower half of an address is done by external latch and ALE signal from 8085.

(b) $\overline{RD}$ and $\overline{WR}$: These signals are basically used to control the direction of the data flow between processor and memory or I/O device/port. A low on $\overline{RD}$ indicates that the data must be read from the selected memory location or I/O port via data bus. A low on $\overline{WR}$ indicates that the data must be written into the selected memory location or I/O port via data bus.



(c) IO/$\overline{M}$, S_0 and S_1 : IO/$\overline{M}$ indicates whether I/O operation or memory operation is being carried out. S_1 and S_0 indicate the type of machine cycle in progress.

(d) *Ready :* It is used by the microprocessor to sense whether a peripheral is ready or not for data transfer. If not, the processor waits. It is thus used to synchronize slower peripherals to the microprocessor. If peripherals are fast enough it is tied to V_{CC}. If it is left open, 8085 enters in the wait state.

SECTION – B

Q.4.(a) Explain BIU and EU of 8086 microprocessor. **(15)**

Ans. BUS Interface Unit (BIU) : The BIU interface 8086 to the outside word. It provides a full 16 bit bidirectional data bus and 20bit address bus. The BIU is responsible for performing all external bus operations as given below :

(i) It sends address of the memory or I/O.

(ii) It fetches instruction from memory.

(iii) It reads data from port/memory.

(iv) It writes data into port/memory.

(v) It supports instruction queuing.

(vi) It provides the address location facility.

The BIU has a dedicated order. The main function of this order is to produce 20 bit physical address. The bus control logic of the BIU generates all bus control signals such as READ and WRITE for memory and I/O.

Instruction Queue : To speed up program execution, the BIU fetches six instruction bytes ahead of time from memory. These prefetched instruction bytes are held for the execution unit in a group of registers called queue. With the help of queue it is possible to fetch next instruction while current instruction is in execution. There are number of instructions in 8086 which need a quite large number of clock cycles for execution. During this execution time the BIU fetches the next instruction or instructions from memory into the instruction queue instead of remaining idle. The BIU continues this process as long as the queue is not full. Due to this,

execution unit gets, the ready instruction in the queue and instruction fetch time is eliminated (while decoding or executing an instruction EU does not require use of the buses).

This system has the advantage over the 8085 because, while the EU is executing an instruction, the BIU is fetching and storing in the queue the next instructions.

The BIU's instruction queue is based on first in first out (FIFO). So that the EU gets the instructions for execution in the order they are fetched. If the queue is full and EU does not request BIU for accessing memory, the BIU does not perform any bus cycle. On the other hand, if the queue is not full and even through the EU does not request BIU for accessing the memory the BIU can fill the queue on its own. If the EU interrupts the BIU, the BIU Ist completes the prefetching and then attains to the service of the EU.

In case of JUMP and CALL instruction, instruction already fetched in queue are of no use. Hence, in these cases queue is dumped and newly formed by loading instructions from new address specified by JUMP or CALL instruction.

Execution Unit (EU) : The EU of 8086 tells the BIU from where to fetch instructions or data, decodes instruction and executes instructions. It contains :

(i) Control Circuitry
(ii) Instruction Decoder
(iii) Arithmetic Logic Unit (ALU)
(iv) Flag register
(v) General purpose registers
(vi) *Pointers* and Index registers

The central circuitry in the EU directs the internal operation. A decoder in the EU translates the instructions *fetched* from memory into a series of actions which the EU performs. ALU is 6 bit. If can add, subtract, AND, OR, XOR, increment, decrements complement and shift binary numbers.

More about Queue : In the beginning, the CS : IP is loaded with the required address from which the execution is to be started. In the initial condition the queue will be empty and the microprocessor starts a fetch operation to bring one byte (the first byte) of instruction code, if the CS : IP address is odd, and two bytes at a time, if the CS : IP address is even. The first byte is a complete opcode in case of some instructions(one byte opcode instructions), the remaining part of opcode may lie in the second byte. But invariably the first byte of an instruction is an opcode. These opcodes along with data are fetched and arranged in the queue. When the first byte from the queue goes for decoding and interpretation, one byte in the queue becomes empty and subsequently the queue is updated.

(i) The microprocessor does not perform the next fetched operation till at least two bytes of the instruction queue are emptied. The instruction execution cycle is never broken for fetch operation. After decoding the first byte the decoding circuit decides whether the instruction is of single opcode byte or double opcode byte. If the single opcode bytes, the next bytes are treated as data bytes depending upon the decoded instruction length. Otherwise, the next byte in the queue is treated as the second byte of the instruction opcode.

(ii) The queue is updated after every byte is read from the queue but the fetch cycle is entreated by BIU only if at least two bytes of the queue are empty and the EU may be concurrently executing the fetched instructions. Fig. shows the queue operation.

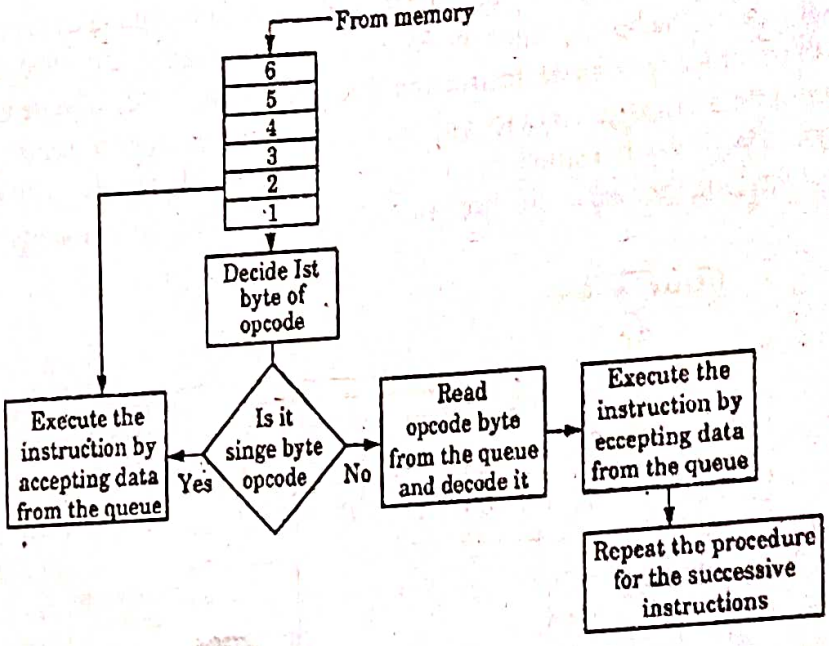


Fig : Queue operation

Q.4.(b) Discuss Pipelining. **(5)**

Ans. Pipelining : The process of fetching the next instruction, when the present instruction is being executed is called as pipelining. Pipelining has become possible due to the use of queue. BIU fills in the queue, until the entire queue is full. BIU restarts filling in the queue when atleast two locations of queue are vacant.

Advantage of Pipelining :

(i) The EU always reads the next instruction byte from the queue in BIU. This is much faster than sending out an address to the memory and waiting for the next instruction byte to come.

(ii) In short pipelining eliminates the waiting time of EU and speeds up the processing.

(iii) The 8086 BIU will not initiate a fetch unless and until there are two empty bytes in its queue. 8086 BIU normally obtains two instruction bytes per fetch.

Q.5.(a) Explain the block diagram of 8086 microprocessor. **(10)**

Ans. Block Diagram of 8086 : The block diagram of 8086 microprocessor is shown in figure. As shown in block diagram, the 8086 is divided into two independent functional units. The Bus interface Unit (BIU) and the Execution.

(i) The Bus Interface Unit : The Bus Interface unit fetches instructions from memory, reads data from ports and memory and writes data to ports and memory. It handles all transfers of data and addresses on the buses for the execution unit. The Bus interface Unit consists of the following :

(a) Instruction Queue (b) Segment Registers (c) Instruction Pointer

(a) *Instruction Queue :* The instruction queue is a first-in-first-out group of registers To speed up program execution, the Bus Interface Unit fetches as many as six instruction byte are held for the Execution Unit in a Instruction Queue. The BIU can be fetching instructio bytes while the Execution Unit (EU) is decoding the instruction or executing an instruction whic does not require use of the buses. When the Execution Unit is ready for its next instruction, i simply reads the instruction from the Instruction Queue in the BIU. This scheme is much faste than sending out an address to memory and then waiting for memory to send sack the nex instruction byte. The prefetch instruction and queue scheme greatly speeds up processing. The arrangement of fetching the next instruction while the current instruction executes is called pipe lining.

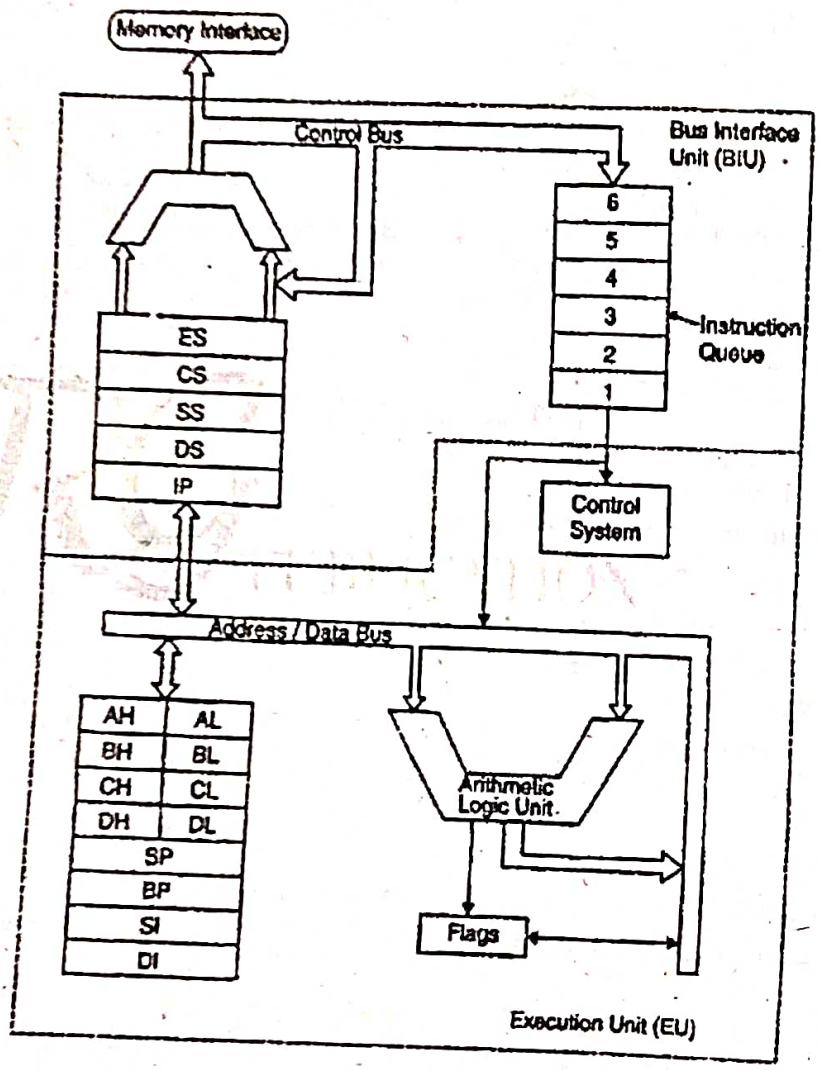


Fig. : Block diagram of 8086 microprocessor

(b) *Segment Registers :* The Big Interface Unit (BIU) contains four 16 bit segment registers. They are :

- Code Segment Register (CS)
- Stock Segment Register (SS)
- Extra Segment Register (ES)
- Data Segment Register (DS)

(c) *Instruction Pointer :* The instruction pointer register is a 16 bit register which holds the address of the next code byte that is to be fetched within the code segment. This register contains the address value which is an offset, because this value must be added to the segment base address contained in CS register to produce the required 20 bit physical address.

(ii) The Execution Unit : The execution unit of 8086 performs the following major operations :

– It tells the BIU, from where to fetch instructions or data.

– It decodes and executes instructions.

To perform the above operations, the execution unit consists of the following sections :

(a) Instruction Decoder, ALU and control circuitry.

(b) Flag Register

(c) General Purpose Registers

(d) Stack Pointer Register

(e) Other Pointer and Index Registers

(a) *Instruction decoder, ALU and control circuitry :* The instruction decoder in the EU translate instructions fetched from memory into a series of actions which are further carried out. The Arithmetic and Logic Unit (ALU) of 8086 is of 16 bits which can add, subtract, AND, OR, XOR, increment, decrement, complement or shift the binary numbers. All internal operations of EU are controlled by control circuitry.

(b) *Flag register :* 8086 microprocessor contains one 16 bit flag register (status register). A flag register is a flip flop which indicates the status of some conditions produced by the execution of an instruction or controls certain operations of the Execution Unit. In a 16 bit flag register, there are nine active flags. Six of the nine flag bits are used to indicate some conditions produced by an instructions. These six flags are called status flag (conditional flags). The remaining three flag bits in the flag register are used to control certain operations of the processor and are called control flags.

(c) *General purpose registers :* The execution unit 8086 contains eight general purpose registers labelled as AH, AL, BH, BL, CH, CL, DH and DL in the fig. below.

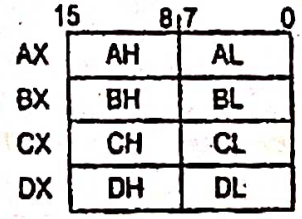


Fig. : 8086 general purpose register

(d) *Stack pointer register :* The stack pointer register SP is a 16 bit register which contains the 16 bit offset address from the start of the stack segment to the memory location where a word was most recently stored on the stack. The stack memory location where a word was most recently stored is called top of the stack.

(e) *Other pointer and index registers :* In addition to the stack pointer register SP, the EU of the 8086 also contains a 16 bit base pointer register BR The base pointer register BP contains the 16 bit offset address relative to the stack segment register SS but it is employed in the based addressing mode of 8086.

Q.5.(b) How physical address is computated in 8086 microprocessor ? (1

Ans. The generation of 20 bit physical address of the location in the code segme which contains the next code byte.

(i) The sequence of operation is as follows.

Physical Address Generation :

Step 1: The CS register contains the upper 16 bits of the starting address of the co segment.

∴ CS Register : | 3 | 4 | 8 | A | Segment base

⇓

Step 2 : The BIU will automatically insert zeros for the lowest four bits of the segme base address to get the 20 bit physical address for the starting of code segment.

∴ Starting address of code segment : | 3 | 4 | 8 | A | 0 |

⇓ BIU adds this zero

Step 3 : The I.P. register contains the offset or distance from this address. The offse here is 4214H.

∴ I.P. Register : | 4 | 2 | 1 | 4 |

⇓

Step 4 : Add the starting address of code segment (20 bit) to the offset to get the physical address of the location containing the next code byte as follows :

Starting address of code segment	→	3	4	8	A	0 ← Hard wired 0
Offset in the I.P. Register	→ +		4	2	1	4
Physical address of the location containing the next code byte	→	3	8	A	B	4

(ii) This 20 bit physical address is then sent out by the BIU to fetch the next code byte stored at this location.

Alternative way to represent the physical address :

(i) An alternative way of representing a 20 bit physical address is as follows :

Segment base : Offset

(ii) For example the 20 bit physical address in the above example is 348 A : 4214.

SECTION – C

Q.6.(a) Explain directives and identifiers used in 8086 microprocessor. **(10)**

Ans. Directives used in assembly language program are as follows :

Directive	Action
ALIGN	aligns next variable or instruction to byte which is multiple of operand.
ASSUME	selects segment register(s) to be the default for all symbol in segment(s)
COMMENT	indicates a comment
DB	allocates and optionally initializes bytes of storage
DW	allocates and optionally initializes doublewords of storage
DD	allocates and optionally initializes doublewords of storage
DQ	allocates and optionally initializes quadwords of storage
DT	allocates and optionally initializes 10-byte long storage units

END	terminates assembly; optionally indicates program entry point
ENDM	terminates a macro definition
ENDP	marks end of procedure definition
ENDS	marks end of segment or structure
EQU	assigns expression to name
EVEN	aligns next variable or instruction to even byte
ESITM	terminates macro expansion
EXTRN	indicates externally defined symbols
LABEL	creates a new label with specified type and current location counter
LOCAL	declares local variables in macro definition
MACRO	starts macro definition
MODEL	specifies mode for asssembling the program
ORG	sets location counter to argument
PAGE	sets length and width of program listing; generates page break
PROG	starts procedure definition
PTR	assigns a specific type to a variable or to a label
PUBLIC	identifies symbols to be visible outside module
TITLE	defines the program listing title

Identifiers : An *identifier* is basically a name that is applied to an item in the program for reference purpose.

An identifier is basically a symbol cased for reference is a program.

Basically these are two types of identifiers in 8086 µp.

(i) NAME identifier : This type of identifiers identifies the item by name or can say refers to the address of data item.

e.g. COUNTER (Name identifier) in Counter DBO.

(ii) Label Identifier : Refer to the address of an instruction, procedure or segment

e.g. MAIN PROC FAR MAIN ⇒ MAIN is label identifier.

B 30 : ADD BL, 30 ⇒ B30 is label identifier

Maximum length of an identifier is 31 characters upto MASM 6.0 & 247.

Following characters can be used as Indentifies

(i) Alphabetic characters : - A to Z and a -z

(ii) Digit : 0 – 9 (but not as 1st character)

(iii) Special characters : ? _ $ @. (but not as 1st character)

Q.6.(a) Write a simple assembly program to subtract two memory location, where each memory location is one byte wide. **(10)**

Ans. Assembly program :

Instruction		Comments
LDA 4201 H	;	Content of 4201 H loaded into Accumulator
MOV B, A	;	Content of A copied to B register
LDA 4200 H	;	Get the minuend in A register
MVI C, 00 H	;	Clear C register to account for sign
SUB B	;	Get the difference in A register

```
        JNC AHEAD  ;    if CF = 0 then go the AHEAD
        CMA        ;    Complement Accumulator
        ADI 01 H   ;    2's complement of difference in A
AHEAD : STA 4202 H ;    Store the result in memory
        MOV A, C   ;
        STA 4203 H ;    Store the sign bit in memory
        HLT        ;    Halt program
        End
```

Sample data :

Input data : Minuend = 5 EH
Subtrahend = 34H
Output data : Difference = 2AH
Sign Bit = 00H

Memory Address	Content
4200	5EH
4201	34H
4202	2AH
4203	00H

Q.7.(a) Write short notes on directives and operators. **(10)**

Ans. Assembler directives : These are the statements that direct the assembler to do something. As the name says, it direct the assembler to do a task. The speciality of these statements is that they are effective only during the assembly of a program but they do not generate any code that is machine executable. We can divide the assembler directives into two categories namely the general purpose directives and the special directives. They are classified into the following categories based on the functions performed by them.

(i) Simplified segment directives
(ii) Data allocation directives
(iii) Segment directives
(iv) Macros related directives
(v) Code label directives
(vi) Scope directives
(vii) Listing control directives
(viii) Miscellaneous directives

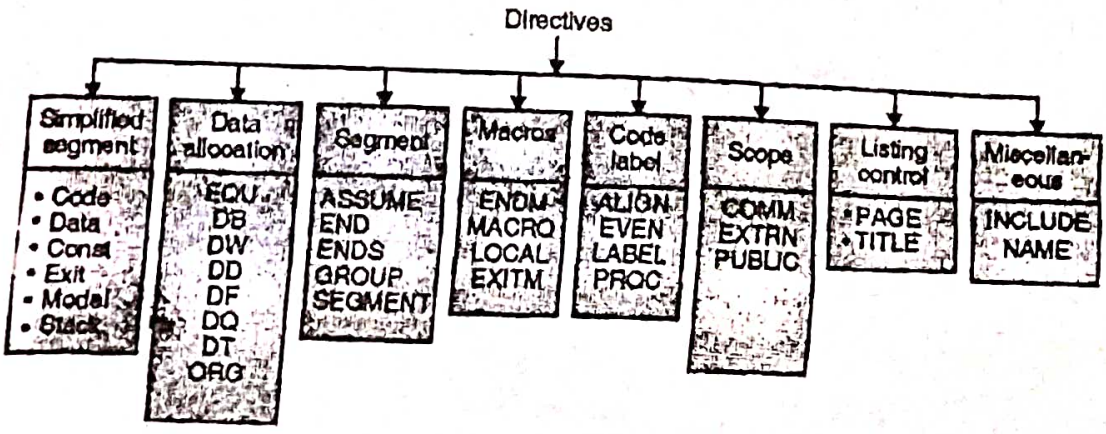


Fig : Assembly directives

Operators : Various operators are as follows :

(i) Length : It is an operator which tells the assembler to determine the number of elements in some named data item such as a string or array.

Example : MOV BX, LENGTH STRING 1; Loads the Length of string in BX.

(ii) OFFSET : It is an operator which tells the assembler to determine the offset or displacement of a named data item (variable) from the start of the segment which contains.

Example : MOV AX, OFFSET MES1, : Loads the offset of variable, MES1 in AX register.

(iii) Short : A short is an operator. It tells the assembler that only 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. Using the short operator saves 1-byte of memory by telling the assembler that it only needs to reserve 1-byte for this particular jump. The short operator should be used only when the destination is fit the range of -128 bytes of +127 bytes from the address of the instruction after the jump.

Example : IMP SHORT NEAR–LABEL

(iv) Type : It is an operator which tells assembler to determine the type of specified variable. Assembler determines the type of specified variable in number of bytes. For byte types variable the assembler gives a value of 1. For word type variable the assembler gives a value of 2 and for doubles word type variable the assembler gives a value of 4.

Q.7.(b) Write a 8086 assembly language program to find largest number in data array. **(10)**

Ans. Program :

```
0201  BE, 00, 03        MOV SI, 0300H          MEMORY ADDRESS IN SI
0204  8B, 0C            MOV CX, [SI] COUNT IN CX
0206  B8, 00, 00        MOV AX, 0000 INITIAL VALUE 0000 FOR COMPARISON
0209  46        BCK     INC SI                 INCREMENT SI
020A  46                INC SI                 INCREMENT SI
020B  3B, 04            CMP, AX, [SI] COMPARE PREVIOUS MAX, WITH
                                               NEXT NUMBER
020D  73, 02            JAE GO                 JUMP IF NUMBER IN AX IS
                                               GREATER
020F  8B, 04            MOV AX, [SI] SAVE NEW LARGER NUMBER IN AX
0211  E2, F6    GO      LOOP BCK               JUMP UNTIL CX = 0
0213  A3, 51, 03        MOV [0351], AX         STORE LARGEST NUMBER IN
                                               MEMORY
0216 CC                 INT 3                  BREAKPOINT
```

SECTION – D

Q.8. Explain 8259 interrupt controller with the help of block diagram. **(20)**

Ans. The block diagram of 8259 is shown in fig.(a). It contains following blocks :

(i) Data bus buffer (ii) Read/write logic

(iii) Cascade buffer and comparator (iv) Control logic

(v) IRR (Interrupt Request Register) (vi) InSR (In-Service Register)
(vii) Priority resolver (viii) IMR (Interrupt Mask Register)

(i) Data bus buffer : It is used to transfer data between microprocessor and internal bus.

(ii) Read/Write control logic : It sets the direction of data bus buffer. It controls all internal read/write operations. It contains initialization and operation command registers.

(iii) Cascaded buffer and comparator : In master mode, it functions as a cascaded buffer. The cascaded buffers outputs slave identification number on cascade lines. In slave mode, it functions as a comparator. The comparator reads slave identification number from cascade lines and compares this number with its internal identification number. In buffered mode it generates an $\overline{EN}$ signal.

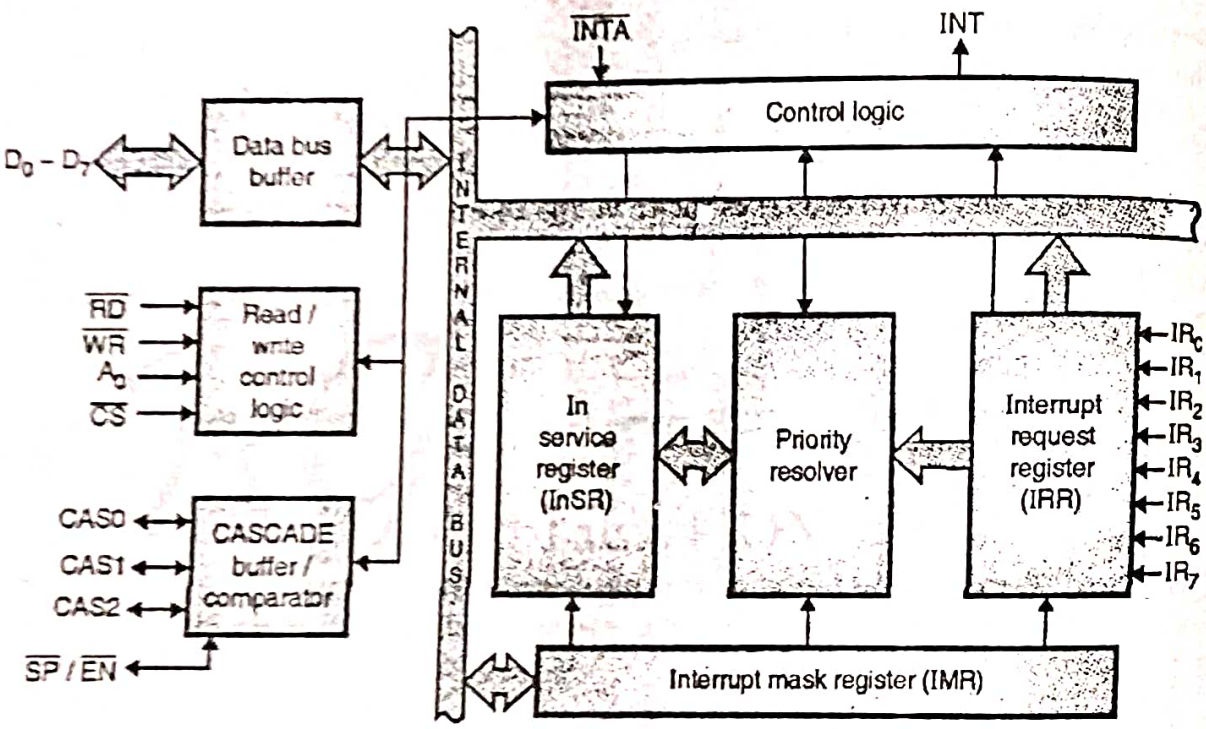


Fig. (a) : Functional block diagram of 8259

(iv) Control logic : It generates an INT signal. In response to an $\overline{INTA}$ signal, it releases three byte CALL address or one byte Vector number. It controls read/write control logic, cascade buffer/comparator, in service register, priority resolver and IRR.

(v) Interrupt request register (IRR) : It is used to store all pending interrupt requests . Each bit of this register is set at the rising edge or at the high level of the corresponding interrupt request line. The microprocessor can read contents of this register by issuing appropriate, command word.

(vi) In service register (InSR) : It is used to store all interrupt levels currently being serviced. Each bit of this register is set by priority resolver and reset by End of interrupt command word. The microprocessor can read contents of this register by issuing appropriate command word.

(vii) Priority resolver : It determines the priorities of the bit set in the IRR. To make decision, the priority resolver looks at the ISR. If the higher priority bit in the InSR is set then it

ignores the new request. If the priority resolvers finds that the new interrupt has a higher priority than the highest priority interrupt currently being serviced and the new interrupt is not in service, then it will set appropriate bit in the InSR and send the INT signal to the microprocessor for new interrupt request.

(viii) Interrupt mask register (IMR) : It is a programmable register. It is used to Mask unwanted interrupt request, by writing appropriate command word. The microprocessor can read contents of this register without issuing any command word.

Q.9. Explain the following :
(a) 8253/8254 programmable interval timer. (20)
(b) Instruction register and priority resolver.

Ans.(a) 8253/8254 programmable Interval timer : The 8253/8354 is an intelligent Programmable Interval Timer/Counter (PIT). It is available in 24-pin DIP package using nMOS technology and operates with +5V power supply. It has three 16-bit presettable down counters. Each of the 3-counters can operate either in binary or BCD mode. Each counter has 2-inputs : Clock (CLK) and Gate (GATE) and one Output (OUT). Various functions of the 8253/8254 are as follows :

(i) As event counter,
(ii) As programmable rate generator,
(iii) As binary rate multiplier,
(iv) As digital mono shot,
(v) As complex motor controller.

Pinout and Block Diagram : The pin outs and its internal details in the form of block diagram is shown in fig.(a) and fig.(b) respectively. Each one of the 3-counters of the 8253 is capable of counting from DC to 2 MHz. The input clock is fed to its CLK terminal with the frequency < 2 MHz. The functions of the GATE and OUT terminals depends on the mode and setting of the particular counter.

In	*Name*
$D_7 \Leftrightarrow D_0$	Data bus (8-bit)
CLK_N	Counter clock pulses
$GATE_N$	Counter gate inputs
OUT_N	Counter outputs
$\overline{RD}$	Read counter
$\overline{WR}$	Write counter
$\overline{CS}$	Chip select
A_1A_0	Counter select bus
V_{CC}	+5 V
GND	Ground

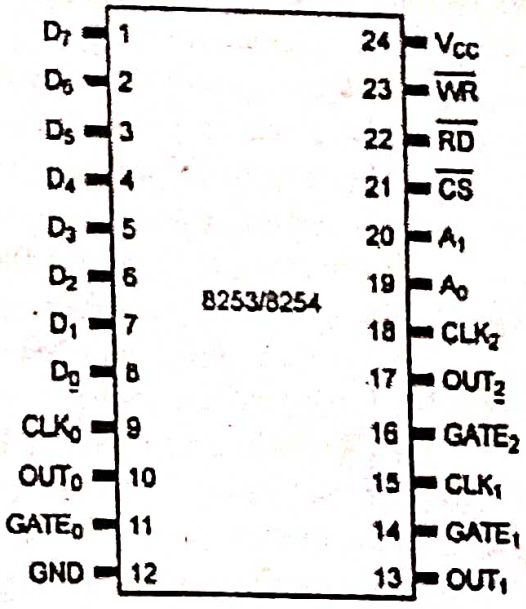


Fig.(a) : Pinouts of 8253

The 8254 has exactly the same pinout as that of the 8253. But the 8254 differs from the of the 8253 in 2 ways, namely,

(i) no read back → Once the particular counter of the 8253 is programmed, its status can not be read back.

(ii) The maximum clock frequency it can count is limited to 2 MHz. The upper limit of clock frequency of 8254 is 8 MHz and that of the 8254-2 is 10 MHz.

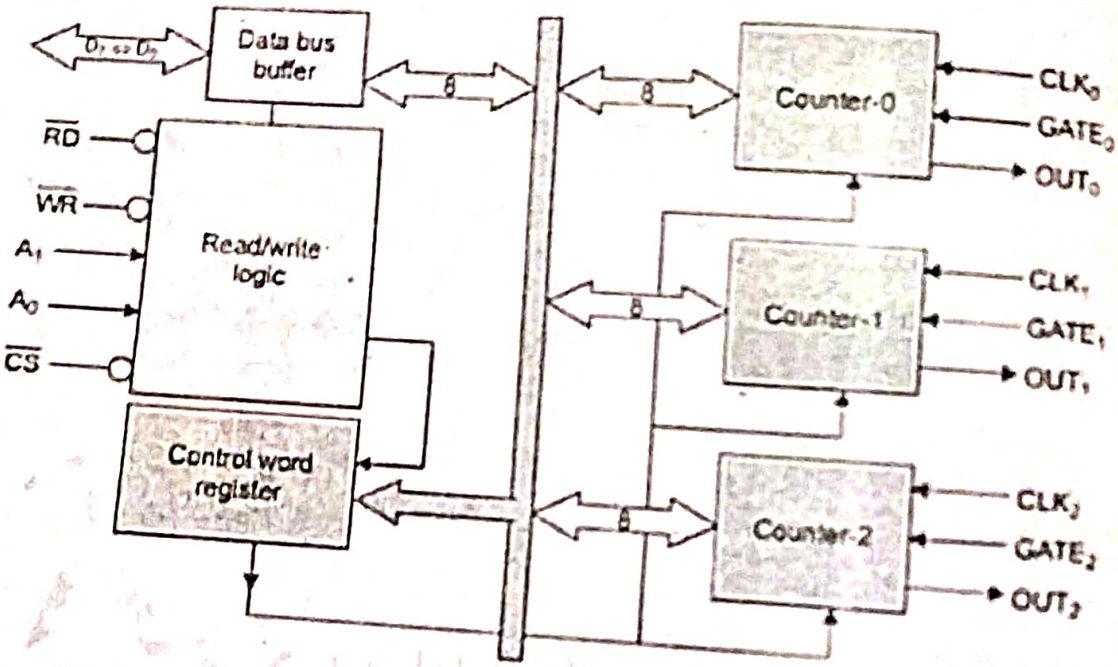


Fig.(b) : Block diagram of 8253/8254 programmable interval timer

– *Electrical characteristics :* The access time of the 8254 is 200ns where as the access time of the 8254-2 is 175ns. The GATE and CLK inputs present 10 µA load and OUT output can source 400 µA in the high state and sink 2 mA in the low state.

– *Data bus buffer :* The 8-bit tri-state data buffer has 3-basic functions :

(a) Programming different modes of the 8253/8254,

(b) Loading the count register, and

(c) Reading the count value.

– *Read/write control logic :* The 8253/8254 has 5-signals, namely, $\overline{CS}$, A_0, A_1, $\overline{RD}$, $\overline{WR}$. The functions of these control signals are :

(a) $\overline{RD}$ *(Read Bar)* : A low on this pin informs the 8253/8254 that the µP is ready to accept data in the form count value.

(b) $\overline{WR}$ *(Write Bar)* : A low on this pin informs the 8253/8254 that the µP is ready to output data in the form of mode informations or data.

Hence, in the isolated I/O mode, $\overline{RD}$ and $\overline{WR}$ are directly connected to $\overline{IOR}$ and $\overline{IOW}$. On the other hand, in the memory mapped I/O, $\overline{RD}$ and $\overline{WR}$ are connected to $\overline{MEMR}$ and $\overline{MEMW}$ respectively.

(c) $\overline{CS}$ (*Chip select Bar*) : The 8253/8254 is selected only when a low (0) is applied at this terminal. No reading or writing operations will be performed by or on the 8253/8254 unless this chip is selected. The $\overline{CS}$ = 1 does effect the operation of the counters.

(d) A_1A_0 (*Address Pin*) : These lines (A_1 A_0) are connected to the address bus. The function of A_1 and A_0 are to select any one of the three counters and the control word register of 8253/8254 is indicated in below table :

Table : Selection of counter and CWR

A_1	A_0	Selection of counter
0	0	Counter-0 Selected
0	1	Counter-1 Selected
1	0	Counter-2 Selected
1	1	Control word register selected (CWR)

- *Control word Register* : As indicated in above table, the control word register is selected only when A_1A_0 = 11. It is loaded with the formation of the control word to indicate which particular counter is to be used in which mode and whether READ and WRITE operation is to be performed.

Ans.(b) Instruction Register : This register is not accessible to the user. The instruction register holds the opcode of the instruction that is decoded and executed.

The opcode is further sent to the instruction decoder to select one of the 256 alternatives (operations). The contents of the instruction decoder are in the form of 0's and 1's.

Priority resolver : It determines the priorities of the bit set in the IRR. To make decision, the priority resolver looks at the ISR. If the higher priority bit in the InSR is set then it ignores the new request. If the priority resolvers finds that the new interrupt has a higher priority than the highest priority interrupt currently being serviced and the new interrupt is not in service, then it will set appropriate bit in the InSR and send the INT signal to the microprocessor for new interrupt request.

MICROPROCESSING AND INTERFACING

Dec - 2017

Paper Code:-EE-309-F

Note : *Attempt five questions in all, selecting one question from each Section. Question No. 1 is compulsory. All questions carry equal marks.*

Q.1.(a) What is the use of AD0-AD7 lines in 8085 ? (2)

Ans. $AD_0 - AD_7$: Multiplexed address/data Bus : These are bidirectional *i.e.*, input, output, Tristate signals having two set of signals they are address and data. The lower order 8 bits of 16 bit address (*i.e.*, $A_0 - A_7$) is multiplexed or time shared with data bus ($D_0 - D_7$). From same 8 bit line two types of signals are transmitted.

All the operations of the microprocessor are performed sequentially with reference to the clock. Microprocessor performs an operation in a specific period, that is known as operation cycles.

In an operation cycle during earlier part it is used as lower order address and in later part it is used as data bus. But for peripheral devices we want separate address and data signals so these signals are demultiplexed by using latch and ALE signal. These lines are tristated by 8085 for conditions same as $A_8 - A_{15}$.

Q.1.(b) Explain the execution of the instruction CMA instruction in 8085. (2)

Ans. Description : Complement accumulator.

This instruction complements the contents of accumulator and result is placed in accumulator. The complement is performing a NOT operation with each bit *i.e.*, all 0's will be replaced by 1 and all 1's with 0's.

Operation : $\overline{A} \rightarrow A$ (Bar placed above A is NOT operation)

Example : CMA : Suppose A = AA H, flag reg. = 10 × 1 × 1 × 1 and CMA instruction is executed.

$$A = 1010 \quad 1010$$
$$\overline{A} = 0101 \quad 0101$$

When CMA instruction is executed A will contain 55 as a result and there will be no change in flag status.

Q.1.(c) What is the need for ALE signal in 8085 ? (2)

Ans. ALE (Address Latch Enable) : The ALE signal is available during T_1 state of each machine cycle. The ALE is pulse signal used to latch the address from $AD_0 - AD_7$ signal. The ALE with respect to clock signal is as shown in figure.

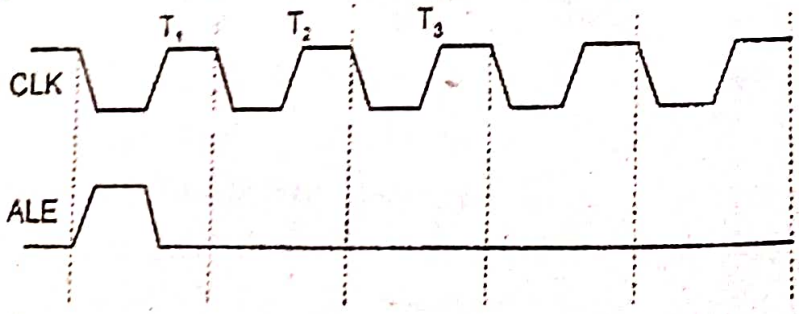


Fig. : ALE signal

Q.1.(d) Describe the difference between the instructions MOV AX, 2437H and MOV AX, [2437H]. (2)

Ans. Difference between the instructions MOV AX, 2437H and MOVAX,[2437H] are former instruction takes 2437 as 16-bit data and latter instruction takes 2437 as 16-bit address.

Q.1.(e) State the function of Direction flag in 8086. (2)

Ans. Direction flag is used by the string manipulation instruction.

0 String is processed from lower address to higher address, called auto-increasing mode.

1 String bytes are accessed from high memory address to low memory address, called auto-decrementing mode. (SI and DI are decremented).

Q.1.(f) Explain ALIGN directive. (2)

Ans. ALIGN : The align directive forces the assembler to align the next segment at an address divisible by specified divisor. The general format for this directive is as shown below.

ALIGN number

where number can be 2, 4, 8 or 16.

```
Example : ALIGN 8   ; This forces the assembler to align the next segment.
                    ; at an address that is divisible by 8. The assembler fills
                    ; the unused bytes with 0 for data and NOP instructions
                    ; for code.
```

Usually ALIGN 2 directive is used to start the data segment on a word boundary and ALIGN 4 directive is used to start the data segment on a double word boundary.

Q.1.(g) What is the size of 8086 instructions ? (2)

Ans. The size of 8086 Instructions is one to six bytes. The first byte consists of opeode and special bit indicators. The second byte will specify the addressing mode of the operands. The subsequent bytes will specify immediate data or address.

Q.1.(h) What is meant by effective address ? (2)

Ans. The offset for a memory operand is called the operand's effective address or EA. It is an unassigned 16 bit number that expresses the operand's distance in bytes from the beginning of the segment in which it resides.

In 8086 we have base registers and index registers. So EU calculates EA by summing a displacement, the content of base register and the content of index register.

EA = {Base Register} + {Index Register} + {8 0r 16 bit displacement}

Q.1.(i) What is the modes of operations used in 8253 ? (2)

Ans. 8253 can operate in six different modes.

(i) Mode 0	Interrupt on terminal count
(ii) Mode 1	Programmable one shot
(iii) Mode 2	Rate Generator
(iv) Mode 3	Square Wave Generator
(v) Mode 4	Software triggered strobe
(vi) Mode 5	Hardware triggered strobe

Q.1.(j) Give the different types of command words used in 8259 ?(2)

Ans. Two type of command word are required to program 8259 are :

- Initialization Command Words (ICWs)
- Operational Command Words (OCWs)

8259 can be initialized with four Initialization Command Words from these four first two are compulsory & other two are optional depending upon the mode that is used. These Initialization Command Words must be issued in a given sequence. After initialization operational command words (OCWs) are used to put 8259 PIC in various operational mode.

Section – A

Q.2.(a) Write an 8085 assembly language program to find out the largest number from a given unordered array of 16 bit numbers, stored in the locations starting from a known address. (10)

Ans.

```
        LXI H, 9200H
        MVI C, 0F
        MVI A, 00
UPP1 :  MVI B, M
        CMP B
        JNC BELOW
        MOV A, B
BELOW : INX H
        DCR C
        JNZ : UPP1
        STA 9210
        HLT.
```

Q.2.(b) Explain the architecture of microprocessors 8085. (10)

Ans. Architecture of 8085 microprocessor: Fig. shows the architecture of 8085. We divide the architecture in different groups as follows :

(i) Arithmetic and Logical Group : This group consists of ALU, accumulator, temporary register and flag register.

ALU : The ALU performs arithmetic and logical operations such as addition, subtraction ANDing, ORing, EXORing, etc.

Accumulator : The accumulator is a 8 bit general purpose register connected to internal data bus and to ALU.

Temporary Register : The other input to ALU is given by temporary register. This register is not available for user. It is only used internally by microprocessor, so the name given temporary register.

Flag Register : The flag is nothing but a group of flip-flops used to give status of different operations result.

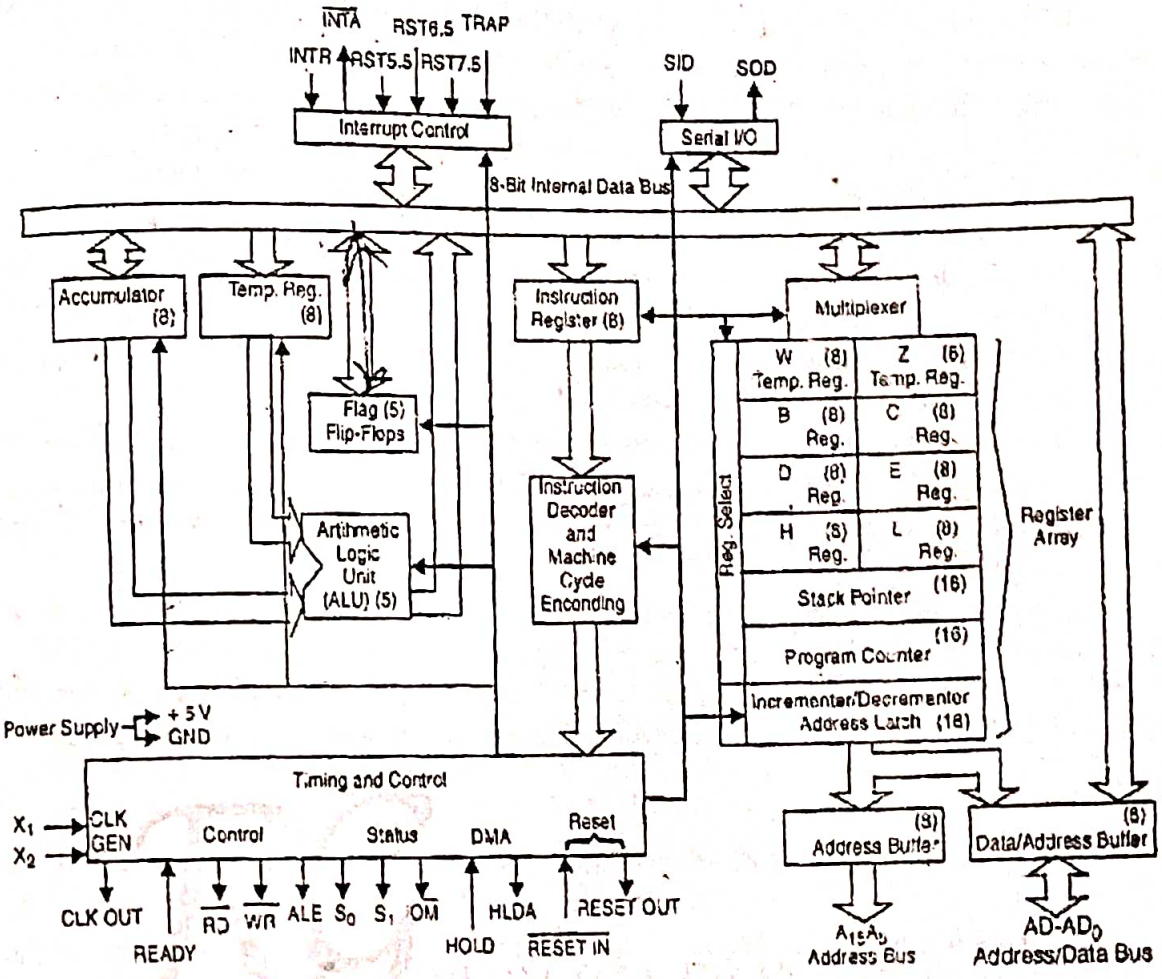


Fig : The 8085A Microprcessor : Functional Block Diagram.

(ii) Register Group : This group consists of 3 types of registers :

(a) *Temporary registers* (*W* and *Z*) : These are not available for user and are used only for internal operations such as to store operand immediately operand or address of memory. These are used internally by microprocessor only to store 8 bit data/information required for execution of certain instructions.

(b) *General Purpose Registers :* The 8085 contains 6 general purpose registers of 8 bits each named as *B*, *C*, *D*, *E*, *H* and *L*. These can be used to store 8 bits or can be used to form a register pair to store 16 bits. The register pairs available are *BC*, *DE* and *HL*. These register are programmable by user. User can store any data in these registers and use it to perform different operations.

(c) *Special purpose registers :* The 8085 contains 3 special purpose registers such as program counter incrementer/decrementer latch and stack pointer.

(iii) Interrupt Control : This block accepts different interrupt request inputs such as TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR and informs control logic to take action in response to each signal. The response for TRAP, RST 7.5, RST 6.5 and RST 5.5 is CALL at restart address. But for INTR it generates a signal INTA and excepts external device should insert a RST code or CALL instruction.

(iv) Serial I/O Control Group : The data transferred on D_0 to D_7 lines is parallel data, but under certain condition it is advantageous to use serial data transfer. 8085 implements this by using SID and SOD signals and the data on these lines is accepted or transferred under software control by serial I/O control block, by using special instructions RIM and SIM.

(v) Instruction Register, Decoder and Control Group :

(a) *Instruction Register :* When an instruction is fetched from memory it is loaded in instruction register from there it is provided to decoder for decoding. This register is only activated when a instruction code or OPcode is available on internal data bus. It is non-programmable register, i.e., not available for programmers use. Remember it accepts only OPcode of instruction, operands are not accepted by this instead they are stored in registers.

(b) *Instruction Decoder :* This accepts a bit pattern from instruction register decodes it and gives the decoded information to control logic. The information includes what operation is to be performed, who is going to perform it, how many operand bytes the instruction contains etc.

(c) *Timing and Control :* This is a control section of 8085. This accept information from instruction decoder and generates microsteps to perform it, so 8085 is called as microprogrammed. In addition to this the block accepts clock inputs and performs sequencing and synchronising operations required for communication between microprocessor and peripheral devices.

Q.3.(a) Explain the direct addressing modes and indirect addressing modes of 8085 with example. (10)

Ans. Direct Addressing Mode : In direct addressing mode the 16 bit address of the operand is given within the instruction itself. The instrucctions in the direct addressing mode are 3 byte instruction. First byte is a OPCODE, second is lower order address byte and third is higher order address byte. For I/O instruction that use direct addressing mode are 2 byte as the address of I/O is one byte. Fig. (1) shows the location of operand.

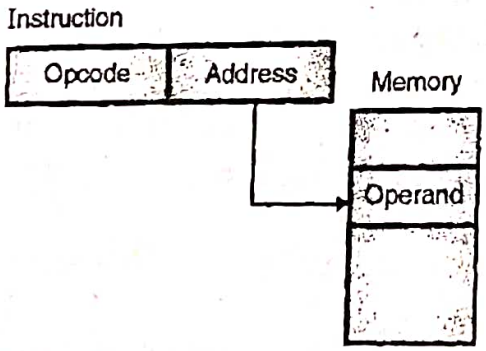


Fig.(1) : Direct addressing mode instruction format

Examples :

(i) LDA C200H : Load accumulator directly from memory location. In this instruction the contents of C200 memory location are transferred to accumulator.

(ii) STA C200 H : Store accumulator directly to memory location. In this instruction the contents of accumulator are stored at memory location C200 H.

Indirect Addressing Mode : In indirect addressing mode the instructions reference the memory through a register pair *i.e.* the memory address where the operand is located is specified by the contents of a register pair. Fig. (2) shows the location of operand.

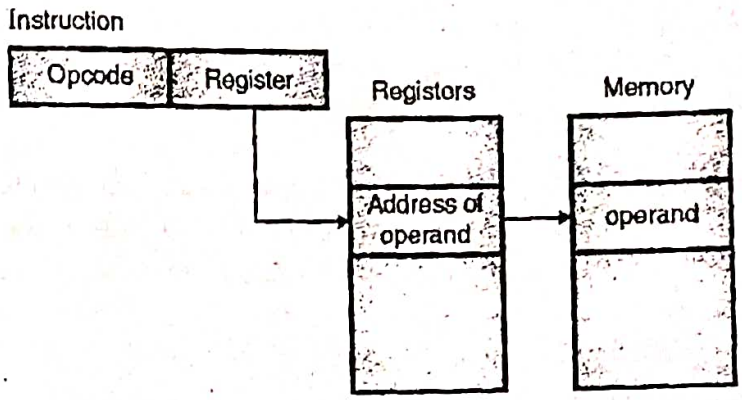


Fig.(2) Indirect addressing mode instruction format

Examples

(i) MOV A,M : In this case M is a memory pointer specifying HL register pair where the address is stored. The contents of HL pair are used as address and the contents of that memory location are transferred to accumulator.

(ii) LDAX B : In this case BC register pair is used as address and the contents of the memory location specified by BC register pair are copied to accumulator.

Q.3.(b) With suitable examples explain 8085 instruction set in detail. (10)

Ans. The instructions are generally classified into the functional categories as follows :

- Data transfer group
- Arithmetic group
- Logical group
- Branching group
- Stack and machine control group.

(i) Data transfer group : Data transfer instructions are used to move data between processor registers, momory and I/O devices.

The data transfer group of instructions include following instructions :

(1) MOV R_d, R_s

(2) MOV R, M and MOV M, R

(3) MVI R, data

(4) MVI M, data

(5) LXI R_p, 16 bit data

(ii) Arithmetic group : Arithmetic instructions are used to perform Arithmetic operations like additions, subtraction, increment / decrement etc.

The arithmetic group of instructions include following instructions :

(1) ADD R

(2) ADD M

(3) ADC R

(4) ADC M

(5) ADI data

(iii) Logical group : Logical instructions are used to perform logical operations like Rotate, Compare, AND, OR, complement etc.

The logical group of instructions include following instructions :

(1) ANA R
(2) ANA M
(3) ANI data
(4) ORA R
(5) ORA M

(iv) Branching group : Branching instructions are used to transfer the execution control. They resemble GOTO instruction of 'C' language of Jump, Call, Return, Restart.

The branch group of instructions include following instructions :

Jump Instruction
- (a) JMP address
- (b) Conditional jump instructions
- (c) PCHL

Call and return instructions
- (a) CALL address
- (b) Conditional call instructions
- (c) RET
- (d) Conditional Ret instructions

Restart instructions
- (a) RST N

(v) Stack and machine control group : Stack instructions are used to read/write stack memory. Machine control instructions are used to control the system like enabling/disabling interrupts, halt etc.

The stack and machine control group of instructions include following instructions :

(1) PUSH R_p
(2) POP R_p
(3) SPHL
(4) XTHL
(5) NOP

Section – B

Q.4.(a) Draw the architecture of 8086 microprocessor and explain in detail.(10)

Ans. Block Diagram of 8086 : The block diagram of 8086 microprocessor is shown in figure. As shown in block diagram, the 8086 is divided into two independent functional units. The Bus interface Unit (BIU) and the Execution.

(i) The Bus Interface Unit : The Bus Interface unit fetches instructions from memory, reads data from ports and memory and writes data to ports and memory. It handles all transfers of data and addresses on the buses for the execution unit. The Bus interface Unit consists of the following:

(a) Instruction Queue
(b) Segment Registers
(c) Instruction Pointer

(a) *Instruction Queue :* The instruction queue is a first-in-first-out group of registers. To speed up program execution, the Bus Interface Unit fetches as many as six instruction bytes are held for the Execution Unit in a Instruction Queue. The BIU can be fetching instruction bytes while the Execution Unit (EU) is decoding the instruction or executing an instruction which does not require use of the buses. When the Execution Unit is ready for its next instruction, it simply reads the instruction from the Instruction Queue in the BIU. This scheme is much faster than sending out an address to memory and then waiting for memory to send sack the next instruction byte. The prefetch instruction and queue scheme greatly speeds up processing. The arrangement of fetching the next instruction while the current instruction executes is called pipe lining.

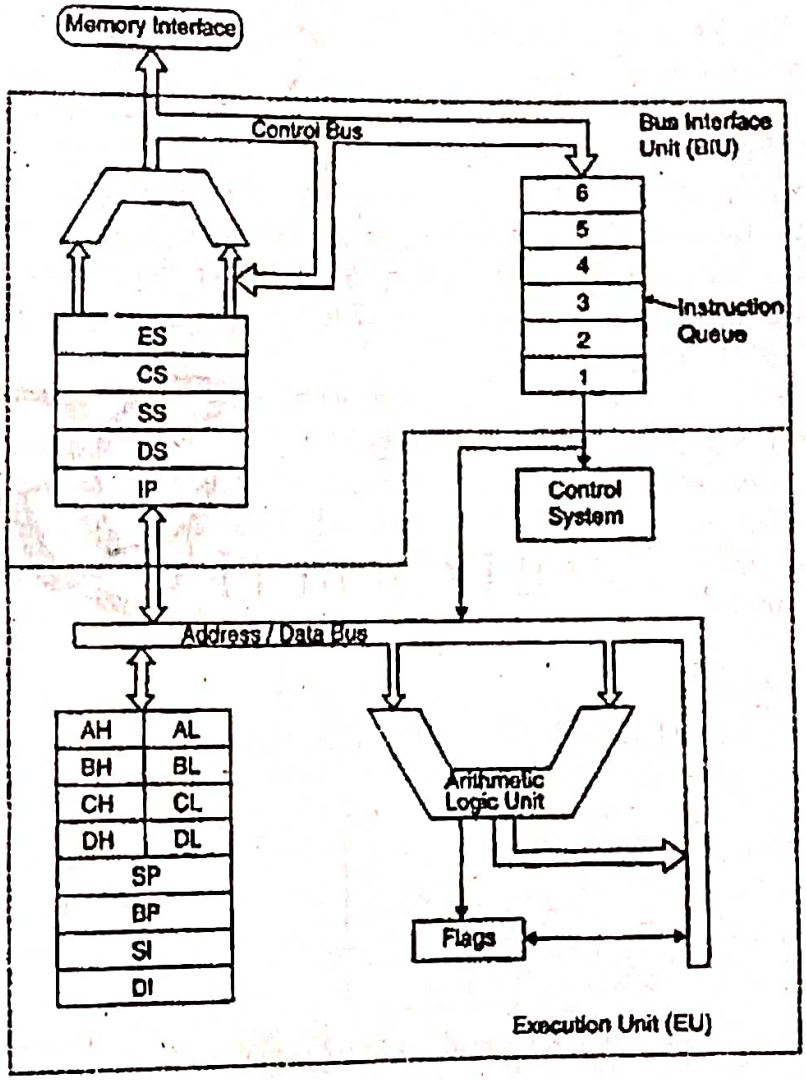


Fig. : Block diagram of 8086 microprocessor

(b) *Segment Registers :* The Big Interface Unit (BIU) contains four 16 bit segment registers. They are :

- Code Segment Register (CS)
- Stock Segment Register (SS)
- Extra Segment Register (ES)
- Data Segment Register (DS)

(c) *Instruction Pointer :* The instruction pointer register is a 16 bit register which holds the address of the next code byte that is to be fetched within the code segment. This register contains the address value which is an offset, because this value must be added to the segment base address contained in CS register to produce the required 20 bit physical address.

(ii) The Execution Unit : The execution unit of 8086 performs the following major operations :

– It tells the BIU, from where to fetch instructions or data.

– It decodes and executes instructions.

To perform the above operations, the execution unit consists of the following sections :

(a) Instruction Decoder, ALU and control circuitry.

(b) Flag Register

(c) General Purpose Registers

(d) Stack Pointer Register

(e) Other Pointer and Index Registers

(a) *Instruction decoder, ALU and control circuitry :* The instruction decoder in the EU translate instructions fetched from memory into a series of actions which are further carried out. The Arithmetic and Logic Unit (ALU) of 8086 is of 16 bits which can add, subtract, AND, OR, XOR, increment, decrement, complement or shift the binary numbers. All internal operations of EU are controlled by control circuitry.

(b) *Flag register :* 8086 microprocessor contains one 16 bit flag register (status register). A flag register is a flip flop which indicates the status of some conditions produced by the execution of an instruction or controls certain operations of the Execution Unit. In a 16 bit flag register, there are nine active flags. Six of the nine flag bits are used to indicate some conditions produced by an instructions. These six flags are called status flag (conditional flags). The remaining three flag bits in the flag register are used to control certain operations of the processor and are called control flags.

(c) *General purpose registers :* The execution unit 8086 contains eight general purpose registers labelled as AH, AL, BH, BL, CH, CL, DH and DL in the fig. below.

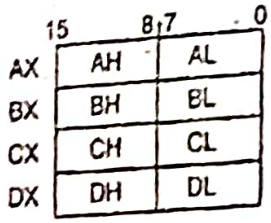


Fig. : 8086 general purpose register

(d) *Stack pointer register :* The stack pointer register SP is a 16 bit register which contains the 16 bit offset address from the start of the stack segment to the memory location where a word was most recently stored on the stack. The stack memory location where a word was most recently stored is called top of the stack.

(e) *Other pointer and index registers :* In addition to the stack pointer register SP, the EU of the 8086 also contains a 16 bit base pointer register BR The base pointer register BP contains the 16 bit offset address relative to the stack segment register SS but it is employed in the based addressing mode of 8086.

Q.4.(b) With suitable examples explain 8086 addressing modes in detail. (10)

Ans. The 8086 microprocessor is provided with various addressing techniques called addressing modes of 8086 are :

(i) Register Addressing
(ii) Immediate Addressing
(iii) Direct Addressing
(iv) Resister Indirect Addressing
(v) Direct Relative Addressing
(vi) Based Index Address
(vii) Relative Based Index Addressing.

The various modes have been discussed below alongwith examples.

(i) Register Addressing : In this type of addressing, the operand to be accessed is specified as residing on one of the internal register of 8086.

The contents of CX (16 bit word), the source operand is moved to BX, the destination of operand. In this instruction, both the source and destination operand have been specified as the consents of external registers of 8086, i.e., CX and BX respective.

The datum is in the register that is specified by the instruction for a 16 bit operand, a register may be AX, BX, CX, DX, SI, DI, SP or BP and for an 8 bit operand a register may be AL, AH, BL, BH,CL,CH,DL, or DH.

Instruction Register → Register Operand

(ii) Immediate Addressing : In this type of addressing the source operand is the part of the instruction itself instead of the contents of any internal register or memory location. The immediate operand can be either a byte or word of data.

Example : MOV BL, 10 H

The source operand is 10H and is an example of byte wide immediate source operand.

MOV BX, 1234 H

The source is 1234H and is an example of word wide immediate source operand.

Instruction
Operand

(iii) Direct Addressing : In this type of addressing the bit effective memory address of the operand follows the instruction opcode. This effective address (EA) is actually an offset of the storage location of the operand from the location specified by the current value of the DS register. The EA is combined with the current content of DS in the 8086 to produce the actual physical address (PA) of the operand. As an example, in the following instruction. The source operand follows the direct addressing mode.

MOV CX, [1234]

It means "Move the contents of the memory location which is at an offset 1234 from the current data segment, into the internal register CX". When the instruction is executed, the 8086 combines 1234 H with the current contents of DATA segment to get the physical address of the source operand. Let DS = 0400H. Then, PA = 04000 + 1234 = 05234 H. So the contents of the physical address 05234 H its copied into the internal register CX of 8086.

(iv) Register Indirect Addressing : In this mode, the effective address of the operand is not the direct part of the instruction, but resides in either a base register or an index register within the 8086. The base register can either be BX or base pointer register BP, and the enclose register can either be SI or DI, i.e., source index register a destination index register.

Example : MOV. CX [SI] The EA is in the source index register.

"Move the contents of the memory location when is at on offset equal to the contents of source index register from the current data segment, into the internal response CX".

Let, DS = 0400H, SI = 1002H

PA = 0400 + 1002 = 05002 H

and then the contents of 05002H memory location is stored into internal register CX of 8086.

(v) Direct- Relative Addressing : In this type of addressing, the EA of the operand in the sum of an 8 bit displacement and the contents of either a base register or an index register.

The base register can be either BX or BP and index register can be either SI or DI. As an example, in the following instruction the EA is the sum of source indirect displacement.

MOV AH, [SI] + DISP

i.e., EA = DI + EA + DISP

PA = DS + EA

Let, [DS] = 0400H

[SI] = 2000H and

DISP = 1234 H

PA = 04000 + 2000 + 1234 = 007234H

$$\begin{array}{r} 04000 \\ 2000 \\ +\ 1234 \\ \hline 07234 \end{array}$$

Then as a result of the indirect on given above the contents of 07234H is copied in the AH register, i.e., higher-byte part of register AX of 8086.

$$EA = \begin{bmatrix} (BX) \\ (BP) \\ (SI) \\ (DI) \end{bmatrix} + \{8 \text{ bit or } 16 \text{ - bit displacement}\}$$

Based-Index Addressing : The EA in this type of addressing mode is calculated as the sum of the contents of any base register and the contents of an array index register. The base register can be either BX or BP and the index register can be either SI or DI. Both of the base register and index register contents are specified in the instruction itself. As an example, the EA on the following instruction is the sum of contents of BX and is register.

MOV AH, [BX] [SI]

Then, $EA = [BX] + [SI]$

Let, $[DS] = 0400H, [BX] = 2000H, [SI] = 1000\ H$

$PA = 0400 + 2000 + 1000 = 07000H$

and as a result of above mentioned instruction, the consents of 07000H is opend on the AH register of register.

$$EA = \begin{bmatrix}(BX)\\(BP)\end{bmatrix} + \begin{bmatrix}(SI)\\(DI)\end{bmatrix}$$

Relative Based Indexed : In this more powerful mode of addressing, the EA is given as the sum of an 8 bit or 16 bit displacement and a based indirect address, i.e.,

$$EA = \begin{bmatrix}(BX)\\or\\(BP)\end{bmatrix} + \begin{bmatrix}(SI)\\or\\(DI)\end{bmatrix} + 8 \text{ bit displacement or 16-bit displacement}$$

As an example, the EA in the following instruction to the sum of contents 01 BX, SI and actual given direct displacement "DISP".

MOV [BX] [SI] + DISP, AH

$ED = [BX] + [SI] + DISP$

Let, $[DS] = 0400H, [BX] = 2000H, [SI] = 1000H$ AND $DISP = 1234H$

$PA = 04000 + 2000 + 1000 + 1234H = 08234H.$

and as a result of the above proportional instruction, the contents of register 'AH' to copied onto the memory location having address = 08234H.

Q.5.(a) Explain various instruction formats used in 8086. **(15)**

Ans. In 8086 all instruction will not be of same size. The instruction vary from 1 to 6 bytes in length. The length of instruction bytes is dependent upon addressing mode used by programmer i.e. immediate, register register relative, based indexed, relative based indexed and so on.

Basically instruction bytes will contain information of :

(i) OPCODE

(ii) Addressing mode designations :

(a) 2 byte Effective Address.

(b) 1 or 2 byte displacement.

(c) 1 or 2 byte immediate operand.

From Fig.1.(a) normally first byte is OPCODE byte, second byte normally specifies addrssing mode. Sometime it may also contain OPCODE part. After OPCODE and addressing mode bytes, we have following different cases :

(a) No additional bytes [fig.1(a), fig.1(b), fig.1(c), fig.1(d)]

(b) A 2 byte EA (for direct addressing mode [fig.1(e)].

(c) A 1 or 2 byte immediate operand [fig.1(f)]

(d) A 1 or 2 byte displacement followed by 1 or 2 byte immediate operand [fig.1.(g)]

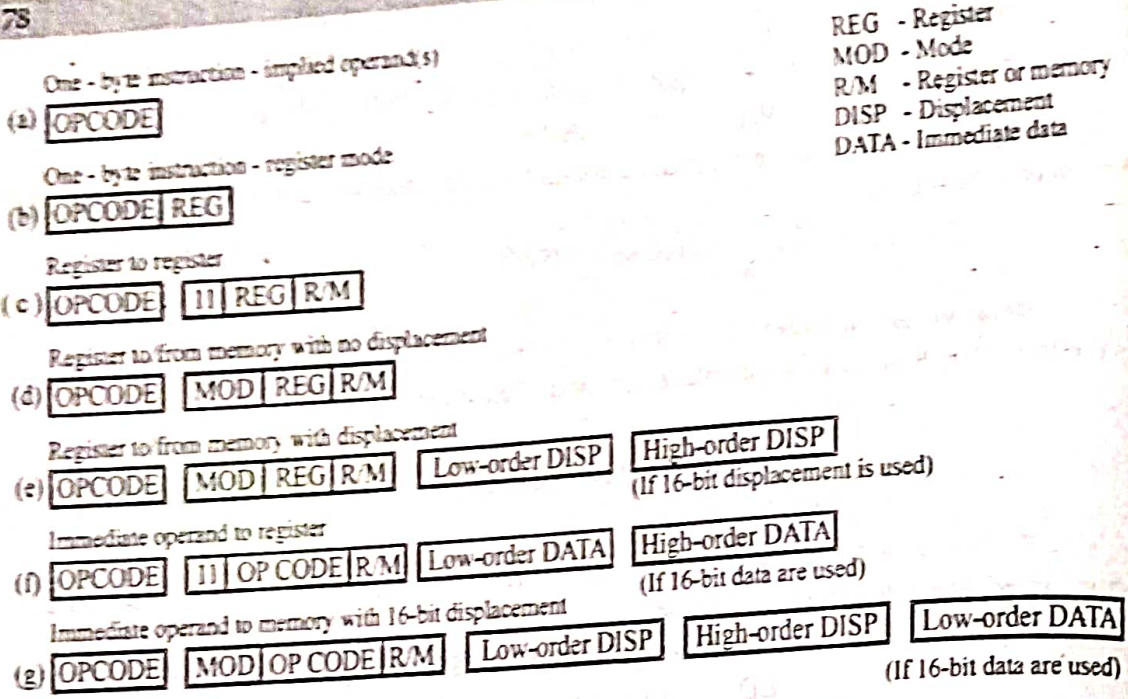


Fig.(1) : Summary of 8086 instruction format

If a displacement or immediate operand is 2 bytes long, the low order byte always appears first, this is Intel standard.

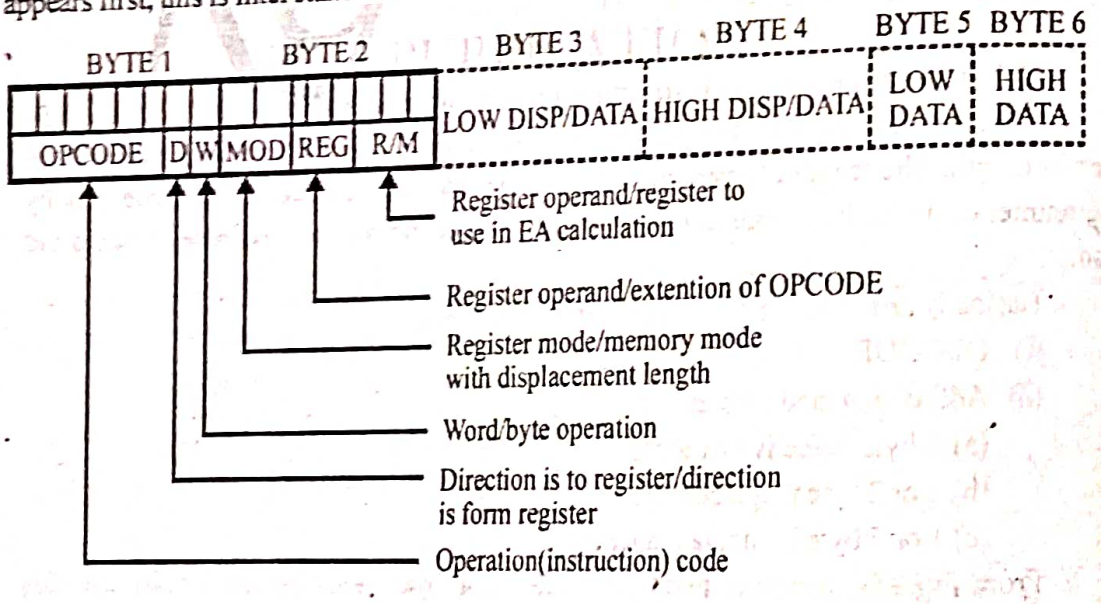


Fig.(2) : instruction format

As shown in fig.(2) the first six bits of multibyte instruction generally contains an opcode that identifies the basic instruction type i.e. ADD, XOR etc. The following bit, called the D field, generally specifies the direction of the operation.

D = 1 means instruction source is specified in REG field.

D = 0 means instruction destination is specified in REG field.

The next following bit is W. This bit identifies between byte and word operation.

W = 0 Instruction operates on byte data

= 1 Instruction operates on word data

In this we have three single bit fields S, V, Z.

S bit : S bit is used in conjunction with W to indicate sign extension of Immediate fields in arithmetic instructions.

S = 0 No sign extension

= 1 Sign extended 8 bit immediate data to 16 bits if W = 1.

Therefore for 8 bit operation : $S = W = 0$

16 bit operation with a 16 bit immediate operand : $S = 0$, $W = 1$

16 bit operation with a sign extemded 8 bit immediate operand : $S = W = 1$

V bit : Used by shift and rotate, to determine single and variable – bit shifts and rotate.

V = 0 shift/rotate count is one

= 1 shift/rotate count is specified in CL register.

Z bit : This bit is used as a compare bit with zero flag in conditional repeat (REP) and loop instructions.

Z = 0 Repeat/loop while zero flag is clear

= 1 Repeat/loop while zero flag is set.

MOD : The mode (MOD) field indicates whether one of the operands is in memory or whether both operands are register.

REG : The Register (REG) field identifies a register that is one of the instruction operands REG field depends upon W bit.

R/M (Register or memory) : This field is of 3 bits. The meaning of R/M bits changes depending upon mode (MOD) field.

Q.5.(b) What is pipelining ? **(5)**

Ans. Pipelining : The process of fetching the next instruction, when the present instruction is being executed is called as pipelining. Pipelining has become possible due to the use of queue. BIU fills in the queue, until the entire queue is full. BIU restarts filling in the queue when atleast two locations of queue are vacant.

Advantage of Pipelining :

(i) The EU always reads the next instruction byte from the queue in BIU. This is much faster than sending out an address to the memory and waiting for the next instruction byte to come.

(ii) In short pipelining eliminates the waiting time of EU and speeds up the processing.

(iii) The 8086 BIU will not initiate a fetch unless and until there are two empty bytes in its queue. 8086 BIU normally obtains two instruction bytes per fetch.

Section – C

Q.6.(a) Draw and explain the timing diagram of memory read cycle. (10)

Ans. Timing diagram of memory read or I/O read machine cycle : Memory read or I/O read machine cycle requires 3T status.

Description :

(1) T_1 Clock Cycle : Microprocessor will transfer 16 bit memory location address on $A_8 - A_{15}$ and $AD_0 - AD_7$ pins. If microprocessor is executing IN instructions, then microprocessor will transfer 8 bit input port address $A_8 - A_{15}$ address as well as duplicated on $AD_0 - AD_7$ pins.

Timing diagram of memory read or I/O read machine cycle is shown in Fig.(1).

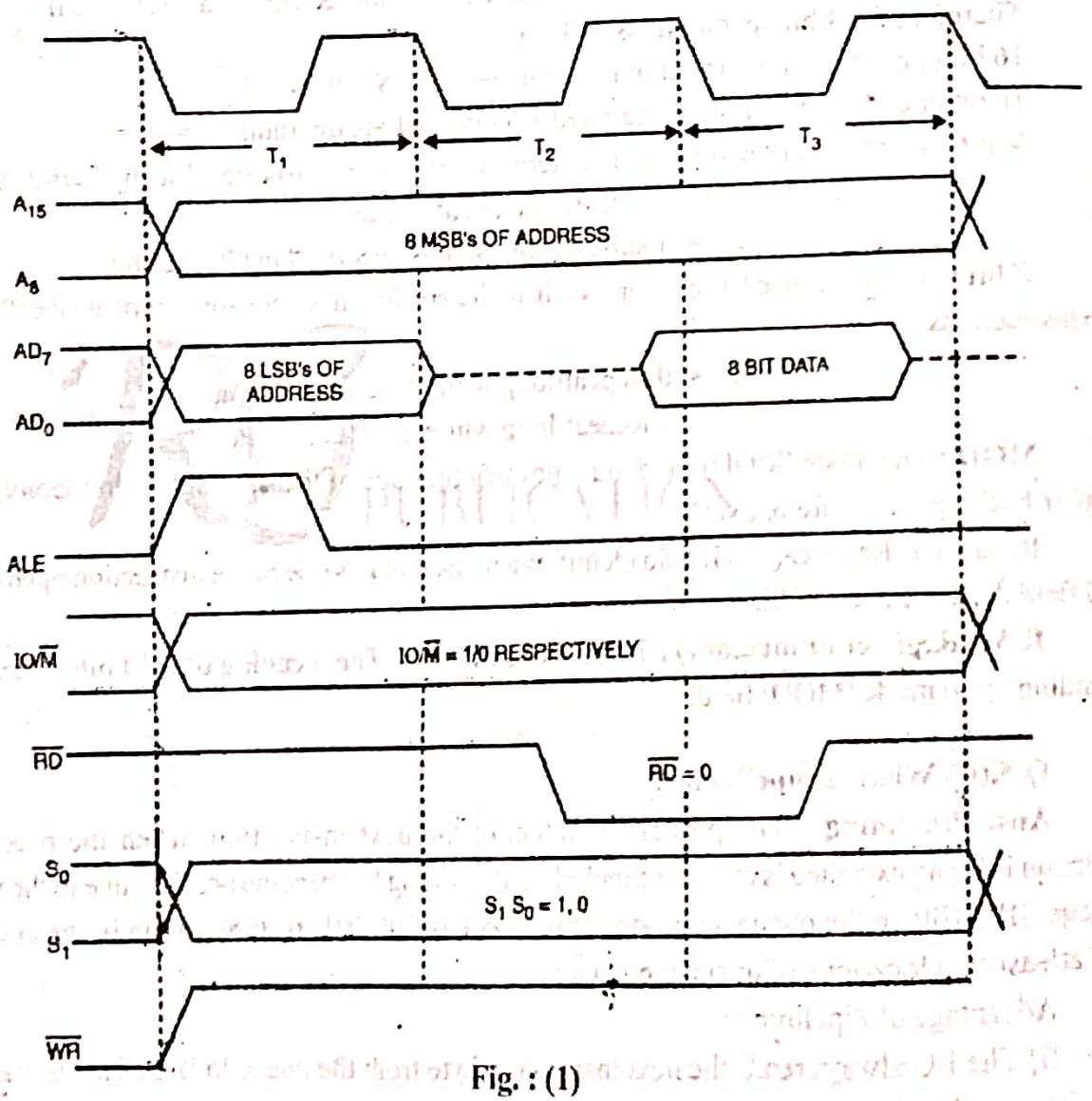


Fig. : (1)

To latch 8 LSB's of address on $AD_7 - AD_0$ pins, microprocessor gives logic '1' on ALE pin. Microprocessor also gives $IO/\overline{M}$ = 1/0 respectively and S_1, S_0 = 1, 0.

(2) T_2 Clock Cycle : The eight data pins $AD_0 - AD_7$ are tristated and then microprocessor gives $\overline{RD}$ = 0. As $IO/\overline{M}$ = 1/0, $\overline{RD}$ = 0, $\overline{WR}$ = 1, $\overline{IORD}/\overline{MRD}$ = 0

respectively. So 8 bit data of selected input port or memory location is transferred to data pins $AD_7 - AD_{10}$.

(3) T_3 Clock Cycle : Microprocessor will transfer 8 bit data from $AD_1 - AD_4$ pins to any of its corresponding internal register.

Q.6.(b) What are assembler directives ? What are their advantages and disadvantages ? **(10)**

Ans. Assembler directives : Assembler Directives are defined in form of words and these are directions to the assembler. These are not the instruction for 8086. These directives are used to wright and execute programes either turbo-assembler (TASM) 'or' macro-assembler (MASM). In addition to instructions, programmes contains assembler directives which are direction to assembler, called assembler directives for pesudo-instructions. These are various directives used for 8086 asembly language programming.

ORG, LTORG, EQU, RESW, RESB etc., are the different examples of assembler directives.

Advantages :

(1) Reduced errors.

(2) Faster translation time.

(3) Changes are made easier and faster.

Disadvantages :

(1) Many instructions are required to achieve small tasks.

(2) Source programs tend to be large and difficult to follow.

(3) Programmer requires knowledge on processor architecture and instruction set.

(4) Programs are machine-dependent, and needs to be completely re-written when hardware is changed.

Q.7.(a) Explain instruction execution timing with example. **(10)**

Ans. Instruction Execution timing : The execution time for instruction varies and depends upon the type of instruction and addressing modes. Basic Instruction time can determined by multiplying the number of clock needed to execute the instruction by clock's period.

The total instruction execution time is the sum of Basic time plus time required to calculate the effective Address if a memory operand is accessed. The Basic instruction time assumes that the instruction to be executed has already been prefetched and stored in instruction queue. If not so, then additonal clock cycles must be added to find the total execution time.

Total instruction execution time depends upon :

(i) No. of memory seferences required by instruction : If operand is a single byte or word with an even address, then only are memory reference is needed, if with an odd address then two memory references is needed. Each memory reference require 4 clock cycle.

(ii) Addressing nodes : In case of 2 operand instruction, register to register addressing gives fasters operation than register to memory or vice-versa operation, and between memory to register and register to memory operation, the memory to register operation is faster.

(iii) Variable Basic Execution Times : Some Instruction are data dependent & Require Variable Basic Execution Times. These are multiply, divide, shift & Rotate.

(iv) Conditional Branch Instruction : For these, 2 different timings are specific smaller time should be considered when condition is not met and branch is not taken. Larger time is considered when condition met. Hence some extra cycle are required to fetch the next instruction.

Let's take on example of Ready & Wait state : The Ready input causes wait state for slower memory & I/O Devices to cope with up. A wait state is an extra clocking period inserted between T2 and T3. Due to this memory access time of 5 MHz with 460 ns is increased by 200 ns to make it (460 ns + 200 ns = 660 ns)

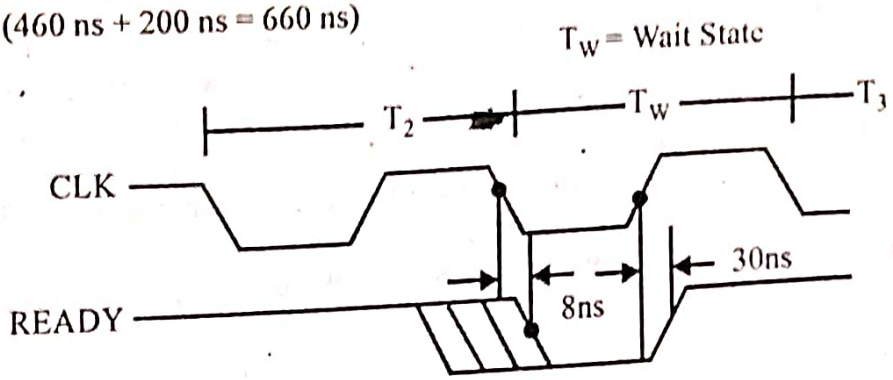


Fig. : READY input timing in 8086 µp.

If READY is at logic 0 after T_2, then T_3 is delayed and T_w is inserted between T_2 and T_3. Ready is next sampled at middle of T_w to determine if next state is T_w or T_3. It is tested for a logic 0 on 1 to 0. Transition of clock at the end of T_2 & for a 1 on 0 to 1 transmition of clock in middle of T_w. Also READY input 8086 µp has some stringent tining reguirements. Such timing requirements are fulfilled by the internal READY synchronization circuitry of 8284 A clock generator.

Q.7.(b) Write a program in 8086 to find the sum of numbers in the array of 10 elements. **(10)**

Ans.

```
        PAGE        52, 80
        TITLE       8086 ALP to find sum of numbers in the array.
• model small
• data
ARRAY   DB      12, 24, 26, 63, 25, 86, 20, 33, 10, 35
SUM     DW      0
MES     DB      10, 13, 'Sum of array elements is : $'
• code
START:  MOV AX, @ data      ; [ Initialise
        MOV DS, AX          ;  data segment ]
        MOV CL, 10          ; Initialise counter
        XOR DI, DI          ; Initialise pointer
        LEA BX, ARRAY       ; Initialise array base pointer
BAC:    MOV AL, [BX+DI]     ; Get the number
```

```
MOV AH, 00H      ; Make higher byte 00h
ADD SUM, AX      ; SUM = SUM + number
INC DI           ; Increment pointer
DEC CL           ; Decrement counter
JNZ BAC          ; if not 0 go to back
MOV AX, SUM      ; Get the result
CALL ATB4D       ; Display sum of array
MOV AH, 4CH
INT 21 H
```

Section – D

Q.8.(a) Discuss various operating modes of 8253 timer with necessary control words. **(10)**

Ans. The programmable time IC provides following modes of operations.

- Mode 0 : Interrupt on Terminal Count
- Mode 1 : Programmable One Shot / Hardware Triggerable One Shot
- Mode 2 : Rate Generator / Pulse Generator
- Mode 3 : Square Wave Generator
- Mode 4 : Software Triggered Strobe
- Mode 5 : Hardware Triggered Strobe

Mode 0 : Interrupt on terminal count : Control word format required for mode 0, counter 0, Read / Load LSB data byte only and BCD counter will be.

0 0 0 1 0 0 0 1 = 11 H

Mode 1 : Programmable One Shot / Hardware Triggerable One Shot : The control word format required for Mode 1, counter 0, Read / Load LSB data byte and BCD counter will be

0 0 0 1 0 0 1 1 = 13 H

Mode 2 : Rate Generator / Pulse Generator : The control word format required for Mode 2, counter 0, Read / Load LSB data byte and BCD counter will be

0 0 0 1 × 1 0 1 = 15 H

Mode 3 : Square wave generator : The control word format required for Mode 3, counter 0, Read / Load LSB data byte and BCD counter will be,

0 0 0 1 × 1 1 1 = 17 H

Mode 4 : Software Triggered Strobe : The control word format required for mode 4, counter 0, Read/Load LSB data and BCD counter will be,

0 0 0 1 1 0 0 1 = 19 H

Mode 5 : Hardware Triggered Strobe : The control word format required for mode 5, counter 0, Read / Load LSB and BCD counter will be,

0 0 0 1 1 0 1 1 = 1 BH

Q.8.(b) Explain the architecture and various modes of 8255. **(10)**

Ans. Block diagram of 8255 :

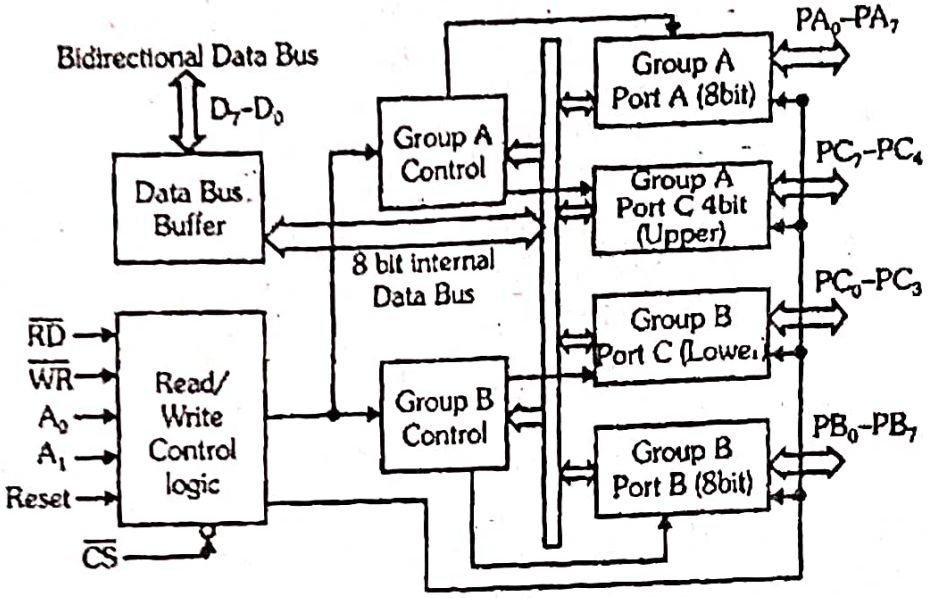


Fig.

Data bus buffer is a bidirectional buffer. It connects the system data bus to the 8 bi internal data bus of 8255. It transfers or receives data or instructions to or from the microprocessor Read/Write control logic block controls the internal and external transfer of data. The singals $\overline{RD}$, $\overline{WR}$, A_0, A_1, $\overline{CS}$ and reset associated with this block. Group A control block receives commands from Read/Write control logic unit depending upon the control word. It controls the operation of Port A (8 bit) and Port C (upper) 4 bit. Group B control unit receives the commands from read/write control logic and control the operations of the 8 bit port B and 4 bit Port C (lower). Port A and Port B are of 8 bit port whereas Port C (upper) and port C (lower) are of 4 bits ports.

Operating modes of 8255 : The 8255 is programmable peripheral interface which can be programmed to operate in one of the following four modes by writing an appropriate control word in the control register.

(*i*) Mode 0	Simple input/output
(*ii*) Mode 1	Strobed input/output
(*iii*) Mode 2	Bidirectional port
(*iv*) BSR mode	Bit Set/Reset mode.

(*i*) Mode–0 Operation : In the mode–0, the 8255 can be programmed as simple input or output. Each of the four ports, port A, port B, port C_U (Upper) and port C_L (Lower) of 8255 can be programmed to the either an input port or an output port. Data is written into or read from the specific port. The input/output features of 8255 in the mode–0 are :

(*i*) Two 8 bit ports and two 4 bit ports
(*ii*) Any port can be an input port or an output port.
(*iii*) Outputs are latched where as inputs are not latched.
(*iv*) Ports do not have handshake or interrupt capability.

(ii) Mode–1 Operation : In the mode–1 operation of 8255, the handshake signals are exchanged between the microprocessor and peripherals prior to data transfer. The features of this mode are :

(*i*) Two ports, port A and port B can function as 8 bit I/O ports. Port A and port B can be configured either as input port or output port.

(*ii*) Each port (port A and port B) uses three lines from port C as handshake signals. The remaining two lines of port C can be used for simple I/O functions.

(*iii*) Interrupt logic is supported.

(*iv*) Both inputs and outputs are latched.

(iii) Mode–2 Operation : In this mode, the port A can be used as a bidirectional port and port B can be used either in mode 0 or mode 1 operation. Port A uses five signals from port C as handshake signals for data transfer. The remaining three signals of port C can be used either as simple I/O function or as handshake signals for port B. The mode-2 operation is generally used for bidirectional data transfer. Data transfer between two computers can be accomplished using 8255 programmable peripheral interface in mode-2 operation.

(iv) BSR mode : It is a bit Set/Reset mode. This mode is concerned only with 8 bits of port C. A control word with bit $D_7 = 0$ is recognized as BSR control word. It does affect any previously transmitted control word with bit $D_7 = 1$ and thus the I/O operations of port A and port B are not affected by a BSR control word. In the BSR mode, the individual 8 bits of port C can be used for applications such as on/off switch.

Q.9.(a) What is DMA ? Explain DMA controller in detail with the help of diagram. **(10)**

Ans. Direct memory access (DMA) is a feature of modern computers that allows certain hardware subsystems within the computer to access system memory independently of the central processing unit (CPU).

Without DMA, when the CPU is using programmed input/output, it is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU initiates the transfer, does other operations while the transfer is in progress, and receives an interrupt from the DMA controller when the operation is done. This feature is useful any time the CPU cannot keep up with the rate of data transfer, or where the CPU needs to perform useful work while waiting for a relatively slow I/O data transfer. Many hardware systems use DMA, including disk drive controllers, graphics cards, network cards and sound cards. DMA is also used for intra-chip data transfer in multi-core processors. Computers that have DMA channels can transfer data to and from devices with much less CPU overhead than computers without a DMA channel. Similarly, a processing element inside a multi-core processor can transfer data to and from its local memory without occupying its processor time, allowing computation and data transfer to proceed in parallel.

The device which supervises, data transfer is named as DMA controller. Now let's have diagrammatic representation of the scheme, which depicts microprocessor, DMA controller, memory and I/O device.

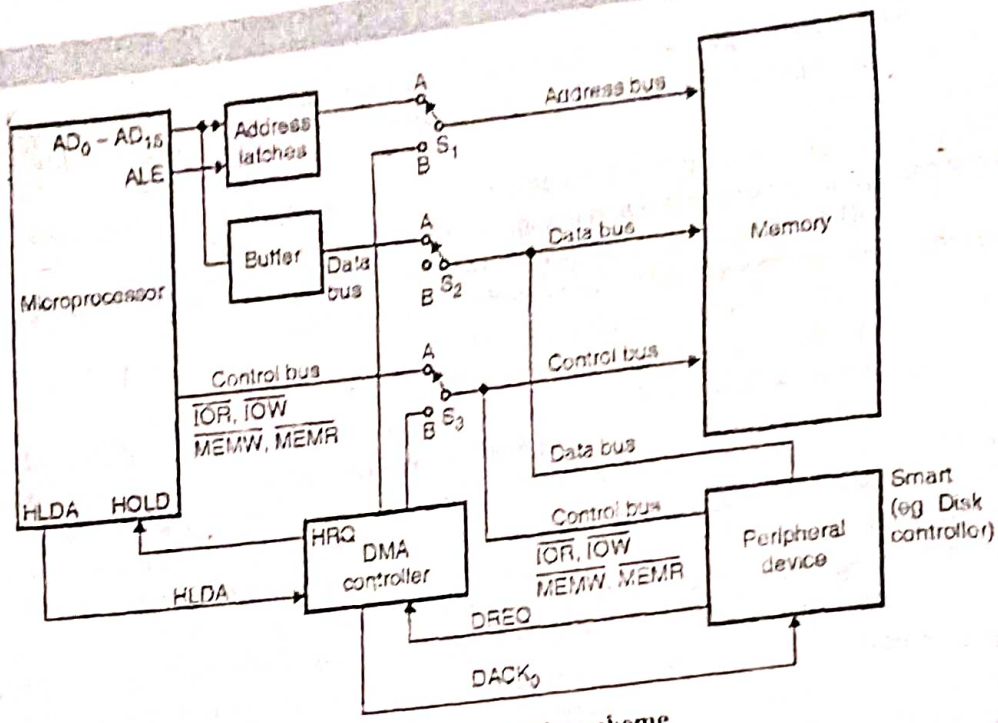


Fig. : DMA controller scheme

Let's understand the concept clearly.

(1) Initially, switches S_1, S_2 and S_3 are at position A.

(2) No direct access to memory by I/O.

(3) Microprocessor is MASTER of all three buses; address, data and control.

(4) Microprocessor treats DMA controller, as I/O device ONLY.

(5) Using IN/OUT instruction, you can program DMA controller chip, for various modes.

(6) Whenever, peripheral device is ready to transfer data, DIRECTLY to memory, it will generate REQUEST (DRQ → DMA REQUEST), to DMA controller, asking for direct access.

(7) In response to DRQ, DMA controller will activate, HRQ (HOLD request); connected to HOLD pin of microprocessor. By activating HOLD line, DMA controller request microprocessor, to HOLD for sometime and allow him to become a master of all three buses.

(8) Moment HOLD pin is HIGH, microprocessor will complete the present job and also activate HLDA (HOLD acknowledge) signal; informing, DMA controller to become master.

(9) Microprocessor tristates, all its buses, so total cutoff from memory and I/O device. Thus microprocessor relinquishes the buses and provides control to DMA controller.

(10) Now DMA controller is master. It will position all three switches to position B.

(11) DMA controller will also generate $\overline{DACK}$ (DMA acknowledge) signal to peripheral device; informing that, direct access is allowed.

(12) Now it will generate address and control signal. Data will flow from memory to I/O or vice-versa.

(13) After completing data transfer, DMA controller will deactivate HOLD line. It also positions, switches back to position A.

(14) Now microprocessor will regain the control over the three buses.

(15) Microprocessor will start executing instructions from main program. Till DMA is inactive or not master of the bus, is referred as DMA IDLE Cycle. When DMA controller gains the control, it is referred as DMA Active Cycle.

Q9.(b) Explain architecture and functioning of 8259 interrupt controller. (10)

Ans. The block diagram of 8259 is as shown in fig. It contains following blocks :

(i) Data bus buffer, (ii) Read/write logic,

(iii) Cascade buffer and comparator, (iv) Control logic,

(v) IRR (vi) ISR,

(vii) Priority resolver, (viii) IMR.

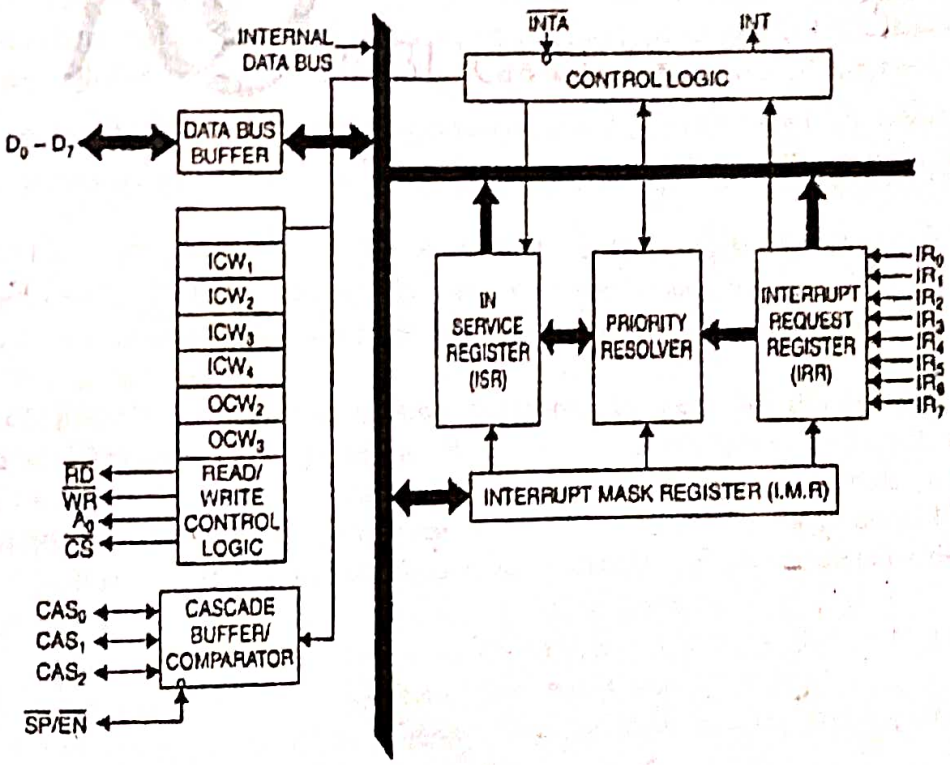


Fig : Functional block diagram of 8259.

(i) Data bus buffer : This 3 state bidirectional 8 bit buffer is used to interface the 8259 to the system data bus control words and status information are transferred through the data bus buffer. It is used to transfer data between microprocessor and internal bus.

(ii) Read/write logic : It sets the direction of data bus buffer. It control all internal read/write operations. It contains *initialization* and operation, command registers. Which store the various control format for device operation. Also allows and the status of the 8259 to be transferred onto the data bus.

(iii) Cascade buffer and comparator : In master mode, it functions as a cascaded buffer. The cascaded buffers output slave identification number of cascade lines. In slave mode, it function as a comparator. The comparator reads slave identification number from cascade lines and compares this number with its internal identification number. In buffered mode it generates an $\overline{EN}$ signal.

(iv) Control logic : In generates an INT signal. In response to an $\overline{INTA}$ signal, it release three byte CALL address or one byte Vector number. It controls read/write control logic, cascade buffer comparator, in service register, priority resolver and IRR.

(v) Interrupt Request Register (IRR) : It is used to store all pending interrupt requests. Each bit of this register is set at the rising edge or at the high level of the corresponding interrupt request line. The microprocessor can read contents of this register by issuing appropriate, command word. The IRR issued to store all the interrupt levels which one require services.

Buffered Mode : This type of operation must be used in large interrupt systems. In large interrupt system the PIC cannot derive the microprocessor during read/cycle, hence tristate bidirectional data buffers are connected between system data bus and PIC. The microprocessor reads contents of PIC registers during the I/O read operation or interrupt acknowledge operation. In this mode, $\overline{EN}$ output line of PIC used to enable tristate buffers, The $\overline{SP}/\overline{EN}$ is generally used to specify master or slave in non-buffered mode. But in buffered mode it is used to enable buffer.

(vi) In Service Register (ISR) : It is used to store all interrupt levels currently being serviced. Each bit of this register is set by priority resolver and reset by End of interrupt command word. The microprocessor can read contents of this register by issuing appropriate command word.

(vii) Priority Resolve : It determines the priorities of the bit set in the IRR. To make decision, the priority resolver looks at the ISR. If the highest priority bit in the ISR is set then it ignores the new request. If the priority resolvers finds that the new interrupt has a highest priority than the highest priority interrupt currently being serviced, then it well set appropriate bit in the ISR and send the INT signal to the microprocessor for new interrupt request.