Subject- Microprocessors
Sem - B.Tech 5th sem
Subject code -(AS- 4150)
Section A

Solution
Model Answer

Quo1 (i) 64 (c)

(ii) d

(iii) d

(iv) OF , set, reset

(v) SIM, RIM

(vi) Universal syncronous asyncronous Receiver Transmitter

(vii) (c) Intel 8086

(viii) level

(ix) (c) CS, DS, SS, ES

(x)(a) IF

Section B

Ans. 2 (a) Every instruction has opcode and operand. When any instruction is executed then first machine cycle is always opcode fetch cycle. Microprocessor interprets the first byte it fetches as an opcode. When it decodes the code it ~~knows~~ knows that ~~it is~~ what operation should be performed. So microprocessor decodes the ~~co~~ instruction code in first machine ~~cycle~~ cycle and in next machine cyle it performs the operation.

(2) (b) ~~b~~ In 8085 microprocessor lower order address bus and data bus are multiplexed. Common lines are used for identifying the address or transferring the datas. These lines are used on time sharing basis. Multiplexing makes the system compact. Latch is connected with the bus AD7 - AD0. When ALE signal goes low, latch holds the lower order address and lower order bus contains the data. So with the ~~held~~ help of latch address ~~a~~ remain same it does not change when lower order address bus contains the data. Latch is active low when ~~it~~ ALE signal is low, it is

~~(2) (C)~~ activated. It hold the address when ALE signal is low. So when bus carries the data, then address will not be changed.

Ans. 2(C)

```
LXI H D000H
LXI B D100H
LDAX B
MOV D, A
MOV A, M
STAX B
MOV A, D
STAX H
HLT
```

~~LXI H, D000H~~

~~MOV A, M~~

~~LXI D, D100H~~

~~STA, D100H~~

M ~~LDAX D~~

~~STA D000H~~

~~HLT~~

Ans. 3 (a) In 8085 microprocessor there are 5 types of interrupts TRAP, RST 7.5, RST 6.5, RST 5.5, INTR.

These are divided in to two types maskable and non maskable interrupt.

The microprocessor can ignore or delay a maskable interrupt request if it is performing some critical task, however it has to respond to a non maskable request immediately.

Maskable interrupts are RST 7.5, RST 6.5 and RST 5.5 and INTR

Non maskable interrupt is TRAP

TRAP → ① It is independent of EI and DI

② No external hardware ~~a~~ is required

③ level and Edge sensitive

④ It is vectored interrupt.

RST 7.5 → ① ~~It is~~. It is controlled by EI and DI

② No external hardware

③ edge sensitive

④ vectored interrupt

RST 6.5 → (1) It is maskable interrupt.
(2) It is controlled by EI and DI.
(3) No external hardware
(4) level sensitive
(5) vectored interrupt.

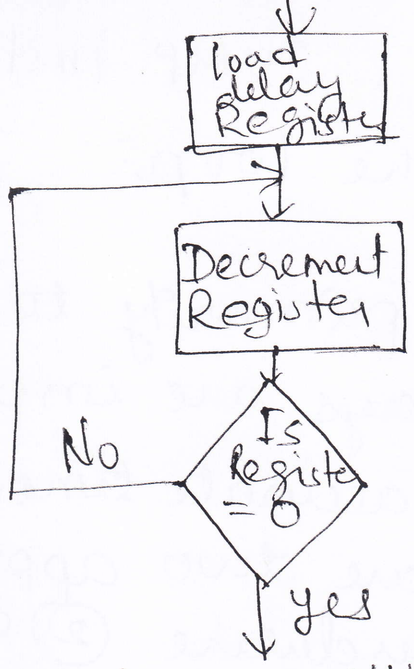
RST 5.5 → (1) It is maskable interrupt.
(2) It is controlled by EI and DI
(3) No external hardware
(4) level sensitive
(5) vectored interrupt.

INTR → (1) It is maskable interrupt.
(2) It is controlled by EI and DI.
(3) It is level sensitive non vectored interrupt
(4) ~~RS~~
(5) ~~It~~

Ans. 3(b) → CALL instructions are used when program sequence is transferred from main program to subroutine. for implementing the CALL instruction it need 18 T states & JUMP instructions are used for executing the loop. It needs 10 T states. CALL instructions are

written in main program. It is used (2)
for calling the subroutine. JUMP instructions
are used for executing the loop.

Ans. 3 (c) Counters are used primarily to keep
track of events. time delays are important
in setting up reasonably accurate timeing
between two events. There are two approaches
for using counters (1) Hardware (2) software
In hardware approach timer is used for
this purpose like 8253/8254.
In software approach by using instructions,
time delay or tracking the event are
introduced in the program.
In hardware ~~of~~ approach proper control word
is Used, which is stored in the control
word register of 8254 timer. In software
approach register is used. A register is loaded
with appropriate number, depending on
the time delay required and then register
is decremented Untill it reaches Zero by
~~sett~~ setting up a loop with conditional
JUMP instructions. The loop causes the
delay, depending up on the clock period
of the system.



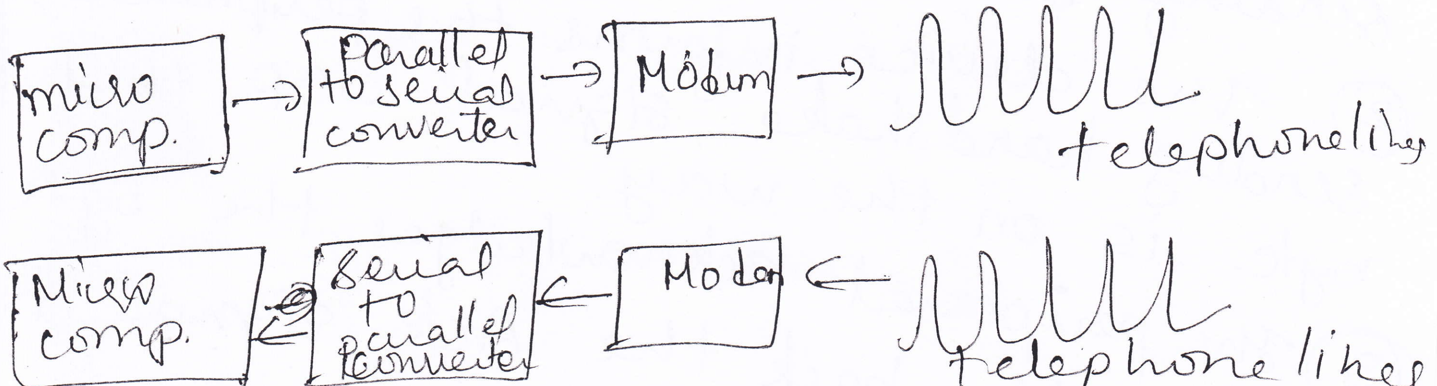
```
example         MVI  B, 0A H   → load register B
        LOOP:   DCR  C         → Decrement C
                JNZ  loop      → Jump back to
                                 decrement C
```

Unit-III

Qu. 4(a) Microprocessor and peripherals operates at different speeds therefore signals are exchanged prior to data transfer between the fast responding MPU and slow responding peripherals such as printer and data converters. These signals are called handshake signal. The exchange of handshake signals prevent the MPU from writing over the previous data before a peripheral has had a chance to accept it or reading the same data before a peripheral has had a time to send the next data byte.

Ans. 4(b) The serial I/O technique can be used to send data over long distance through telephone lines. the bandwidth of telephone lines ranges from 300 Hz o 3300 Hz. The digital signal with rise me in nsecond requires a bandwidth of everal megahertz. Therefore data bits should e converted into audio tones.

A Modem is circuit that translates igital data into audio tone frequencies r transmission over the telephone lines nd converts audio frequencies into digital ata for reception. computer can exchange formation over telephone lines by using wo modems.



calling computer (or a terminal) also ~~[illegible] as~~ and receiving computer through telephono lines.

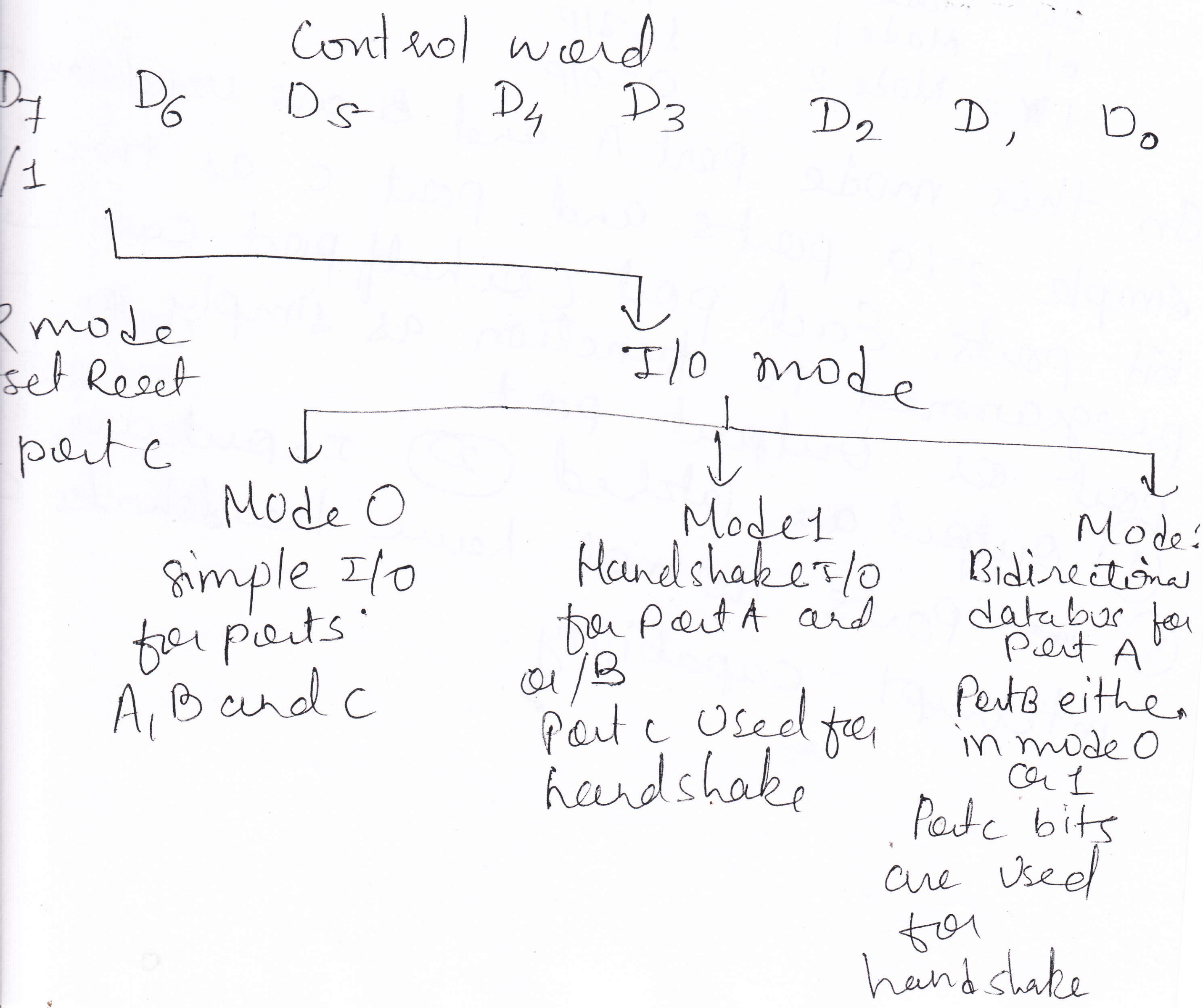
In input mode two handshake signal STB
and IBF and one ~~I~~ INTR signal is
used.
Data input with handshake → ① A
peripheal strobes a data byte in the input
port and informs the interfacing device
by sending ~~han~~ STB signal.
② If ~~IBF~~ input port is full, this
message is conveyed to the peripheral
by sending hand shake signal IBF input
buffer full.
③ Data output with handshake
① The MPU writes a byte into the output
port of the programmable device by
sending control signal $\overline{WR}$.
② The device informs the peripheral, by
sending handshake signal OBF, that a
byte is on the way.
③ The peripheral acknowledges the byte
by sending back the ACK signal to
the device.
④ The device interrupts the MPU to
ask for the next byte, or the MPU
finds out that the byte ~~has~~ has been
acknowledged by the status check

Ans. 4 (c) 8255 is a widely used programmable, parallel I/O. Functions of 8255A classified according to two modes :-

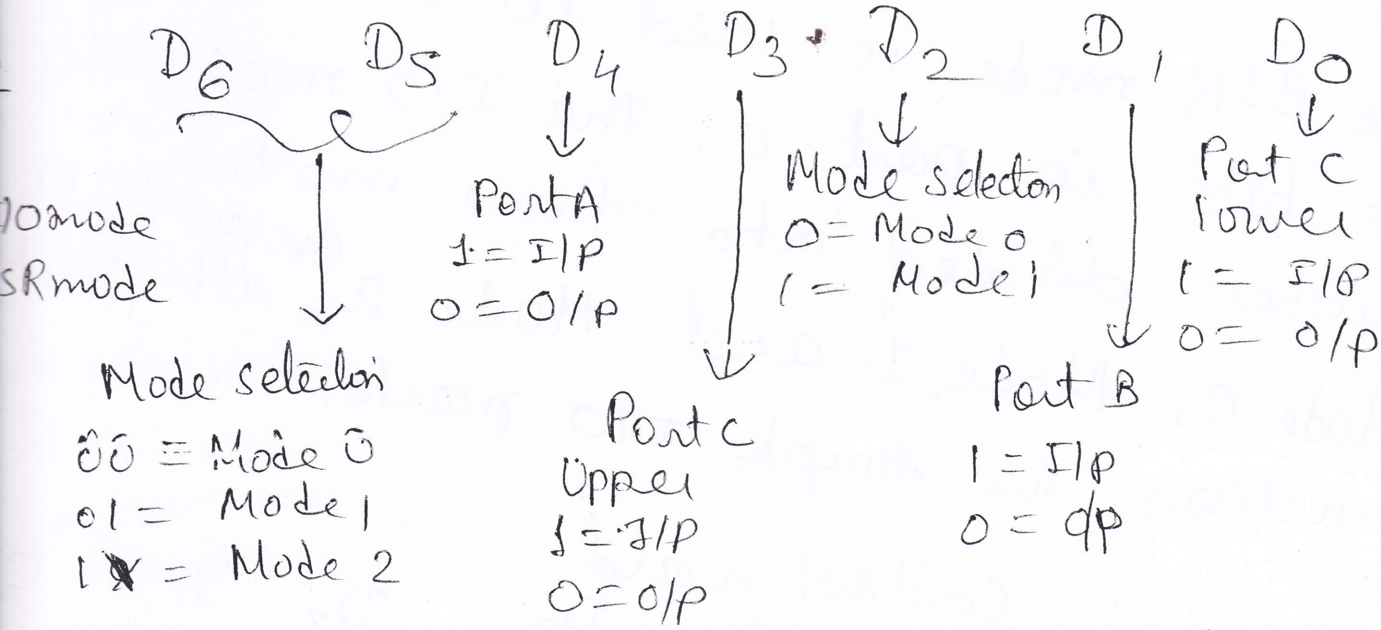
(i) BSR (Bit set reset mode)

(ii) I/O mode

he BSR mode is used to set or reset he bits in port c. The I/O mode is urther divided into three modes lode 0, Mode 1 and Mode 2. In mode 0 all ports unction as simple I/O ports.



In control word format on the position
D7 if bit 0 is present BSR mode
is selected if bit 1 is present then
IO ~~is~~ mode is selected. In I/O mode
mode 0 control word format is



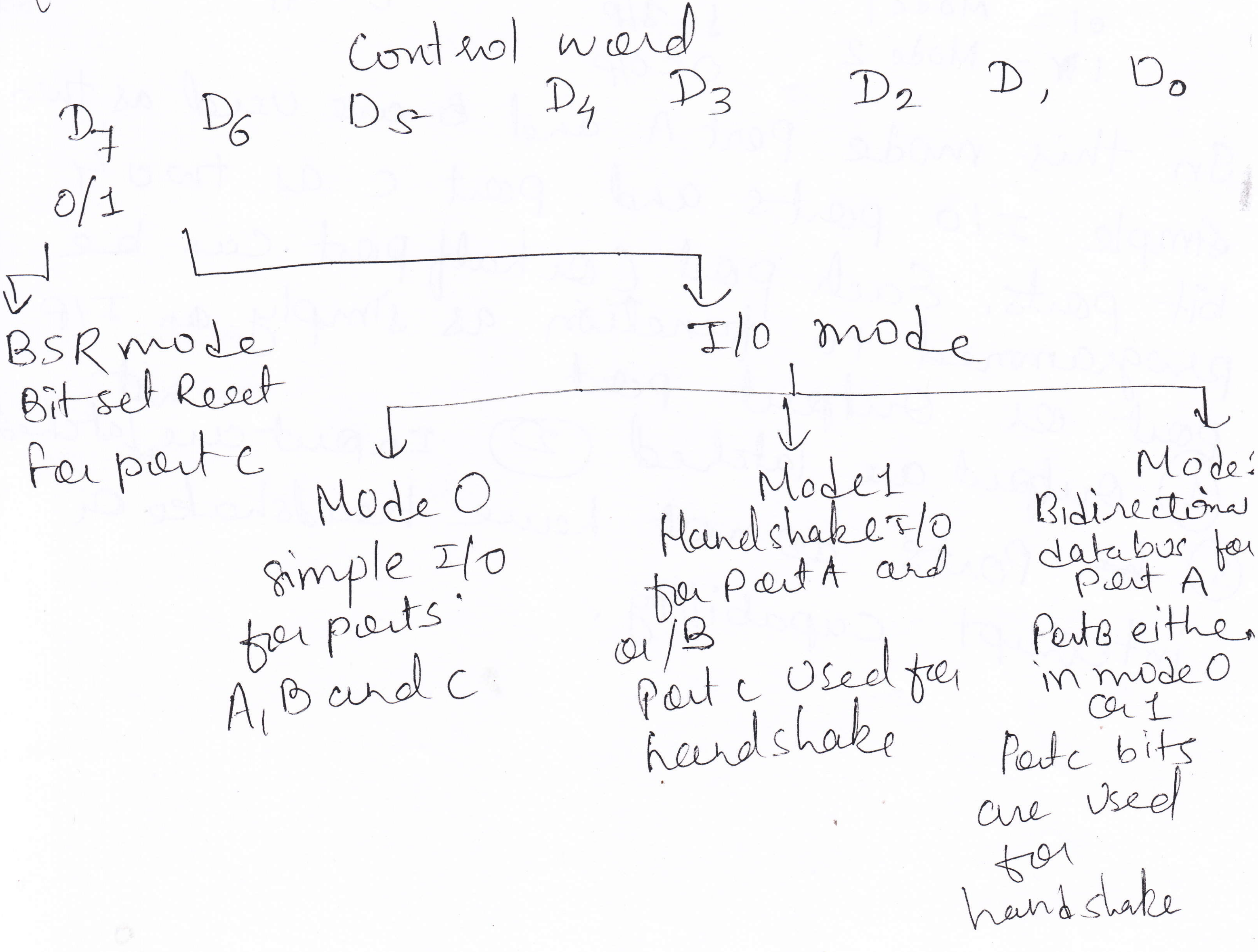
this mode Port A and B are used as two
ple I/O ports and port C as two 4
ports. Each port (or half port) can be
grammed to function as simply an I/P
t or output port.
Output are latched $\textcircled{2}$ Input are not latched
~~-~~ Ports do not have handshake or
terrupt capability.

Ans. 4 (c) 8255 is a widely used programm-able, parallel I/O. Functions of 8255A classified according to two modes.

① BSR (Bit set reset mode)

② I/O mode

The BSR mode is used to set or reset the bits in port C. The I/O mode is further divided into three modes Mode 0, Mode 1 and Mode 2. In mode 0 all ports function as simple I/O ports.


Control word
D7 D6 D5 D4 D3 D2 D1 D0
0/1
BSR mode Bit set Reset for port C
I/O mode
Mode 0 simple I/O for ports A, B and C
Mode 1 Handshake I/O for Port A and or B Port C used for handshake
Mode 2 Bidirectional databus for Port A Port B either in mode 0 or 1 Port C bits are used for handshake

Unit - IV

5 (a) (i) ALE - Address latch enable -
This output signal indicates the availability of the valid address on the address/data lanes and is connected to latch enable input of latch. This signal is active high.

(ii) READY - This is acknowledgement from the slow device or memory that they have completed the data transfer. The signal made available by the device is syncronize by the 8254 clock generator to provide ~~the~~ ready input to the 8086. This signal is active high.

iii) INTR - This is a level triggered input. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle. This can be internally masked by resetting the interrupt enable flag.

(iv) NMI - Non maskable interrupt - This is an edge triggered input which causes a Typ 2 interrupt. The NMI is not masked internally by software. A transition from low to high initiates the

interrupt response at the end of the
current instruction. This input is
nternally syncronized.

Ans: 6 (b)

logical instruction set are AND, OR, XOR and NOT

AND: logical AND - This instruction bit by bit ANDs the source operand that may be an immediate, a register or a memory location to the destination operand that may be registe or a memory location.

1. AND AX, 0008H
2. AND AX, BX
3. AND AX, [5000H]

```
                                 0000      1111   = 3F0FH
 0011         1111               0000      1000   = 0008H   AND
 0000         0000               ----      ----
 ----         ----
 0000         0000               0000      1000   = 0008H
```

OR: logic OR The OR instruction carries out the OR operation in the same way.

1. OR AX, 0098H
2. OR AX, BX
3. OR AX, [5000H]

```
                                                  = 0F0FH
                            0000      1111          OR
 0000         1111          1001      1000        = 0098H
 0000         0000          ----      ----
 ----         ----          1001      1111        = 0F9FH
 0000         ~~0000~~
              1111
```

XOR - The XOR operation is again carried out in similar way to the AND and OR operation.

1. XOR AX, 0098H
2. XOR AX, BX

```
                                                  = 0F0FH
                            0000      1111          XOR
 0000         1111          1001      1000        = 0098H
 0000         0000
 ---------------------------------------------
 0000         1111          1001      0111        = 0F97H
```

Ans. 6(c) Push → Push to stack: This instruction pushes the content of the specified register/memory location on to the stack. The stack pointer is decremented by 2, after execution of the instruction. Hence the push operation decremented SP by two and then stores the two bytes contents of the operand on to the stack. The higher byte is pushed first and then the lower byte. Example

Push AX, ② PUSH DS ③ PUSH [5000H]

POP → This instruction when executed, loads the specified ~~[illegible]~~ register/memory location with the content of the memory location of which ~~old~~ address is formed using the current stack segment. and stack pointer as usual. The stack pointer is incremented by 2.

Example:

POP AX,

POP DS

POP [5000H]

Ans. 5 (b)

AD_{15}-AD_0 These are the time multiplexed memory I/O address and data lines. Address remains on the lines during T_1 state, while the data is available on the data bus during T_2,T_3,T_w and T_4. Here T_1,T_2,T_3,T_4 and T_w are the clock states of a machine cycle. T_w is a wait state. These lines are active high and float to a tristate during interrupt acknowledge and local bus hold acknowledge cycles.

A_{19}/S_6,A_{18}/S_5,A_{17}/S_4, A_{16}/S_3 These are the time multiplexed address and status lines. During T_1, these are the most significant address lines for memory operations. During I/O operations, these lines are low. During memory or I/O operations, status information is available on those lines for T_2,T_3,T_w and T_4. The status of the interrupt enable flag bit(displayed on S_5) is updated at the beginning of each clock cycle. The S_4 and S_3 combinedly indicate which segment register is presently being used for memory accesses as shown in Table 1.1. These lines float to tri-state off (tristated) during the local bus hold acknowledge. The status line S_6 is always low (logical). The address bits are separated from the status bits using latches controlled by the ALE signal.

Table 1.1 *Bus High Enable/status*

S_4	S_3	*Indications*
0	0	Alternate Data
0	1	Stack
1	0	Code or none
1	1	Data

$\overline{BHE}/S_7$-Bus High Enable/Status The bus high enable signal is used to indicate the transfer of data over the higher order (D_{15}–D_8) data bus as shown in Table 1.2. It goes low for the data transfers over D_{15}–D_8 and is used to derive chip selects of odd address memory bank or peripherals. $\overline{BHE}$ is low during T_1 for read, write and interrupt acknowledge cycles, whenever a byte is to be transferred on the higher byte of the data bus. The status information is available during T_2, T_3 and T_4. The signal is active-low and is tristated during 'hold'. It is low during T_1 for the first pulse of the interrupt acknowledge cycle.

Table 1.2

$\overline{BHE}$	A_0	*Indication*
0	0	whole word
0	1	Upper byte from or to odd address.
1	0	Lower byte from or to even address
1	1	None

$\overline{RD}$-Read Read signal, when low, indicates the peripherals that the processor is performing a memory or I/O read operation. $\overline{RD}$ is active low and shows the state for T_2, T_3, T_w of any read cycle. The signal remains tristated during the 'hold acknowledge'.

READY This is the acknowledgement from the slow devices or memory that they have completed the data transfer. The signal made available by the devices is synchronized by the 8284A clock generator to provide ready input to the 8086. The signal is active high.

INTR-Interrupt Request This is a level triggered input. This is sampled during the last clock cycle of each instruction to determine the availability of the request. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle. This can be internally masked by resetting the interrupt enable flag. This signal is active high and internally synchronized.

$\overline{TEST}$ This input is examined by a 'WAIT' instruction. If the $\overline{TEST}$ input goes low, execution will continue, else, the processor remains in an idle state. The input is synchronized internally during each clock cycle on leading edge of clock.

NMI-Non-maskable Interrupt This is an edge-triggered input which causes a Type2 interrupt. The NMI is not maskable internally by software. A transition from low to high initiates the interrupt response at the end of the current instruction. This input is internally synchronized.

RESET This input causes the processor to terminate the current activity and start execution from FFFF0H. The signal is active high and must be active for at least four clock cycles. It restarts execution when the RESET returns low. RESET is also internally synchronised.

CLK-Clock Input The clock input provides the basic timing for processor operation and bus control activity. Its an asymmetric square wave with 33% duty cycle. The range of frequency for different 8086 versions is from 5MHz to 10MHz.

Vcc +5V power supply for the operation of the internal circuit.

GND ground for the internal circuit.

MN/$\overline{MX}$ The logic level at this pin decides whether the processor is to operate in either minimum (single processor) or maximum (multiprocessor) mode.

The following pin functions are for the minimum mode operation of 8086.

M/$\overline{IO}$-Memory/IO This is a status line logically equivalent to $\overline{S}_2$ in maximum mode. When it is low, it indicates the CPU is having an I/O operation, and when it is high, it indicates that the CPU is having a memory operation. This line becomes active in the previous T_4 and remains active till final T_4 of the current cycle. It is tristated during local bus "hold acknowledge".

$\overline{INTA}$-Interrupt Acknowledge This signal is used as a read strobe for interrupt acknowledge cycles. In other words, when it goes low, it means that the processor has accepted the interrupt. It is active low during T_2, T_3 and T_w of each interrupt acknowledge cycle.

ALE-Address Latch Enable This output signal indicates the availability of the valid address on the address/data lines, and is connected to latch enable input of latches. This signal is active high and is never tristated.

DT/$\overline{R}$-Data Transmit/Receive This output is used to decide the direction of data flow through the transreceivers (bidirectional buffers). When the processor sends out data, this signal is high and when the processor is receiving data, this signal is low. Logically, this is equivalent to $\overline{S}_1$ in maximum mode. Its timing is the same as M/$\overline{IO}$. This is tristated during 'hold acknowledge'.

$\overline{DEN}$-Data Enable This signal indicates the availability of valid data over the address/data lines. It is used to enable the transreceivers (bidirectional buffers) to separate the data from the multiplexed address/data signal. It is active from the middle of T_2 until the middle of T_4. $\overline{DEN}$ is tristated during 'hold acknowledge' cycle.

HOLD, HLDA-Hold/Hold Acknowledge When the HOLD line goes high, it indicates to the processor that another master is requesting the bus access. The processor, after receiving the HOLD request, issues the hold acknowledge signal on HLDA pin, in the middle of the next clock cycle after completing the current bus (instruction) cycle. At the same time, the processor floats the local bus and control lines. When the processor detects the HOLD line low, it lowers the HLDA signal . HOLD is an asynchronous input, and it should be externally synchronized.

If the DMA request is made while the CPU is performing a memory or I/O cycle, it will release the local bus during T_4 provided :

1. The request occurs on or before T_2 state of the current cycle.
2. The current cycle is not operating over the lower byte of a word (or operating on an odd address).
3. The current cycle is not the first acknowledge of an interrupt acknowledge sequence.
4. A Lock instruction is not being executed.

So far we have presented the pin descriptions of 8086 in minimum mode.

The following pin functions are applicable for maximum mode operation of 8086.

[illegible]

$\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$-Status Lines These are the status lines which reflect the type of operation, being carried out by the processor. These become active during T_4 of the previous cycle and remain active during T_1 and T_2 of the current bus cycle. The status lines return to passive state during T_3 of the current bus cycle so that they may again become active for the next bus cycle during T_4. Any change in these lines during T_3 indicates the starting of a new cycle, and return to passive state indicates end of the bus cycle. These status lines are encoded in Table 1.3.

Table 1.3

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	*Indication*
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O Port
0	1	1	Halt
1	0	0	Code Access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

$\overline{LOCK}$ This output pin indicates that other system bus masters will be prevented from gaining the system bus, while the $\overline{LOCK}$ signal is low. The $\overline{LOCK}$ signal is activated by the 'LOCK' prefix instruction and remains active until the completion of the next instruction. This floats to tri-state off during "hold acknowledge". When the CPU is executing a critical instruction which requires the system bus, the LOCK prefix instruction ensures that other processors connected in the system will not gain the control of the bus. The 8086, while executing the prefixed instruction, asserts the bus lock signal output, which may be connected to an external bus controller.

QS_1, QS_0-Queue Status These lines give information about the status of the code-prefetch queue. These are active during the CLK cycle after which the queue operation is performed. These are encoded as shown in Table 1.4.

Table 1.4

QS_1	QS_0	*Indication*
0	0	No operation
0	1	First byte of opcode from the queue
1	0	Empty queue
1	1	Subsequent byte from the queue

This modification in a simple fetch and execute architecture of a conventional microprocessor offers an added advantage of *pipelined processing* of the instructions. The 8086 architecture has a 6-byte instruction prefetch queue. Thus even the largest (6-bytes) instruction can be prefetched from the memory and stored in the prefetch queue. This results in a faster execution of the instructions. In 8085, an instruction (opcode and operand) is fetched, decoded and executed and only after the execution of this instruction, the next one is fetched. By prefetching the instruction, there is a considerable speeding up in instruction execution in 8086. This scheme is known as *instruction pipelining*.

At the starting the CS:IP is loaded with the required address from which the execution is to be started. Initially, 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 or 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 instruction) and it is a part of opcode, in case of other instructions (two byte long 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. The microprocessor does not perform the next fetch 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 it is single opcode byte , 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. The second byte is then decoded in continuation with the first byte to decide the instruction length and the number of subsequent bytes to be treated as instruction data. The queue is updated after every byte is read from the queue but the fetch cycle is initiated by BIU only if at least two bytes of the queue are empty and the EU may be concurrently executing the fetched instructions.

The next byte after the instruction is completed is again the first opcode byte of the next instruction. A similar procedure is repeated till the complete execution of the program. The main point to be noted here is, that the fetch operation of the next instruction is overlapped with the execution of the current instruction. As shown in the architecture, there are two separate units, namely, execution unit and bus interface unit. while the execution unit is busy in executing an instruction, after it is completely decoded, the bus interface unit may be fetching the bytes of the next instruction from memory, depending upon the queue status. Figure 1.6 explains the queue operation.

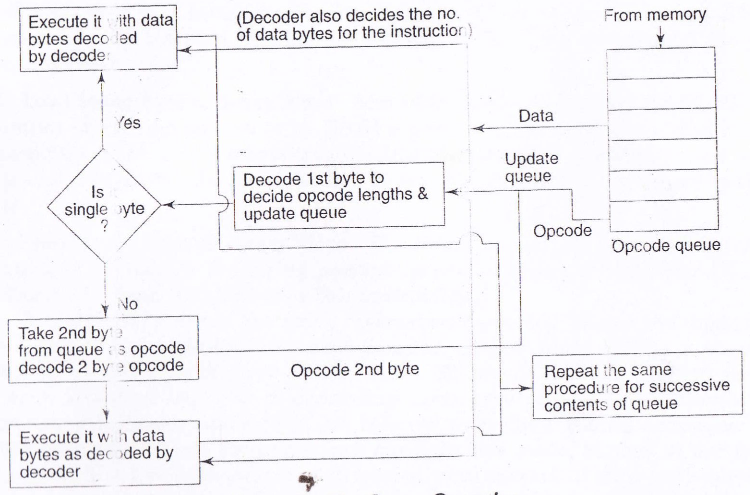

Execute it with data bytes decoded by decoder
(Decoder also decides the no. of data bytes for the instruction)
From memory
Yes
Data
Is single byte ?
Decode 1st byte to decide opcode lengths & update queue
Update queue
Opcode
Opcode queue
No
Take 2nd byte from queue as opcode decode 2 byte opcode
Opcode 2nd byte
Repeat the same procedure for successive contents of queue
Execute it with data bytes as decoded by decoder

Fig. 1.6 *The Queue Operation.*

$\overline{RQ}/\overline{GT}_0$, $\overline{RQ}/\overline{GT}_1$ Request/Grant These pins are used by other local bus masters, in maximum mode, to force the processor to release the local bus at the end of the processor's current bus cycle. Each of the pins is bidirectional with $\overline{RQ}/\overline{GT}_0$ having higher priority than $\overline{RQ}/\overline{GT}_1$. $\overline{RQ}/\overline{GT}$ pins have internal pull-up resistors and may be left unconnected. The request/grant sequence is as follows:

1. A pulse one clock wide from another bus master requests the bus access to 8086.
2. During T_4 (current) or T_1 (next) clock cycle, a pulse one clock wide from 8086 to the requesting master, indicates that the 8086 has allowed the local bus to float and that it will enter the "hold acknowledge" state at next clock cycle. The CPU's bus interface unit is likely to be disconnected from the local bus of the system.
3. A one clock wide pulse from the another master indicates to 8086 that the 'hold' request is about to end and the 8086 may regain control of the local bus at the next clock cycle.

Thus each master to master exchange of the local bus is a sequence of 3 pulses. There must be at least one dead clock cycle after each bus exchange. The request and grant pulses are active low. For the bus requests those are received while 8086 is performing memory or I/O cycle, the granting of the bus is governed by the rules as discussed in case of HOLD, and HLDA in minimum mode.

Until now, we have described the architecture and pin configuration of 8086. In the next section, we will study some operational features of 8086 based systems.

In other words, using this type of instruction the control will be transferred to a particular specified location, if a particular flag satisfies the condition.

2.3.6 *Unconditional Branch Instructions* Ans. 6 (a) ✓

CALL: Unconditional Call This instruction is used to call a subroutine from a main program. In case of assembly language programming, the term procedure is used interchangeably with subroutine. The address of the procedure may be specified directly or indirectly depending upon the addressing mode. There are again two types of procedures depending upon whether it is available in the same segment (Near CALL, i.e ±32K displacement) or in another segment (Far CALL, i.e anywhere outside the segment). The modes for them are respectively called as intrasegment and intersegment addressing modes. This instruction comes under unconditional branch instructions and can be described as shown with the coding formats. On execution, this instruction stores the incremented IP (i.e. address of the next instruction) and CS onto the stack along with the flags and loads the CS and IP registers, respectively, with the segment and offset addresses of the procedure to be called.

Direct Near	D_7 OPCODE D_0	D_7 DISP.LB D_0	D_7 DISP.HB D_0
Indirect Near	D_7 OPCODE D_0	D_7 OPCODE D_0	
Direct Far	D_7 OPCODE D_0	D_7 LB D_0 D_7 HB D_0 OFFSET	D_7 LB D_0 D_7 HB D_0 SEGMENT
Indirect Far	D_7 OPCODE D_0	D_7 OPCODE D_0	

RET: Return from the Procedure At each CALL instruction, the IP and CS of the next instruction is pushed onto stack, before the control is transferred to the procedure. At the end of the procedure, the RET instruction must be executed. When it is executed, the previously stored content of IP and CS along with flags are retrieved into the CS, IP and flag registers from the stack and the execution of the main program continues further. The procedure may be a near or a far procedure . In case of a FAR procedure, the current contents of SP points to IP and CS at the time of return. While in case of a NEAR procedure, it points to only IP. Depending upon the type of procedure and the SP contents, the RET instruction is of four types.

1. Return within segment
2. Return within segment adding 16-bit immediate displacement to the SP contents.
3. Return intersegment
4. Return intersegment adding 16-bit immediate displacement to the SP contents.

INT N: Interrupt Type N In the interrupt structure of 8086/8088, 256 interrupts are defined corresponding to the types from 00H to FFH. When an INT N instruction is executed , the TYPE byte N is multiplied by 4 and the contents of IP and CS of the interrupt service routine will be taken from the hexadecimal multiplication (N×4) as offset address and 0000 as segment address. In other words, the multiplication of type N by 4 (offset) points to a memory block in 0000 segment, which contains the IP and CS values of the interrupt service routine. For the execution of this instruction, the IF must be enabled.

Example 2.40

Thus the instruction INT 20H will find out the address of the interrupt service routine as follows:

```
        INT        20H
        Type* 4 = 20 * 4 = 80H
```

Pointer to IP and CS of the ISR is 0000 : 0080 H

Figure 2.12 shows the arrangement of CS and IP addresses of the ISR in the interrupt vector table.

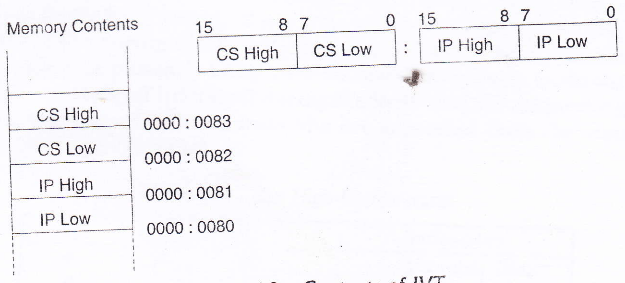


Fig. 2.12 *Contents of IVT*

INTO: Interrupt on Overflow This is executed, when the overflow flag OF is set. The new contents of IP and CS are taken from the address 0000:0000 as explained in INT type instruction. This is equivalent to a Type 4 interrupt instruction.

JMP: Unconditional Jump This instruction unconditionally transfers the control of execution to the specified address using an 8-bit or 16-bit displacement (intrasegment relative, short or long) or CS : IP (intersegment direct far). No flags are affected by this instruction. Corresponding to the three methods of specifying jump addresses, the JUMP instruction has the following three formats.

JUMP	DISP 8-bit				Intrasegment, relative, near jump
JUMP	DISP.16-bit (LB)	DISP.16-bit (UB)			Intrasegment, relative, Far jump
JUMP	IP(LB)	IP(UB)	CS(LB)	CS(UB)	Intersegment, direct, jump

IRET: Return from ISR When an interrupt service routine is to be called, before transferring control to it, the IP, CS and flag register are stored on to the stack to indicate the location from where the execution is to be continued, after the ISR is executed. So, at the end of each ISR, when IRET is executed, the values of IP, CS and flags are retrieved from the stack to continue the execution of the main program. The stack is modified accordingly.

LOOP: Loop Unconditionally This instruction executes the part of the program from the label or address specified in the instruction up to the loop instruction, CX number of times. The following sequence explains the execution. At each iteration, CX is decremented auto-