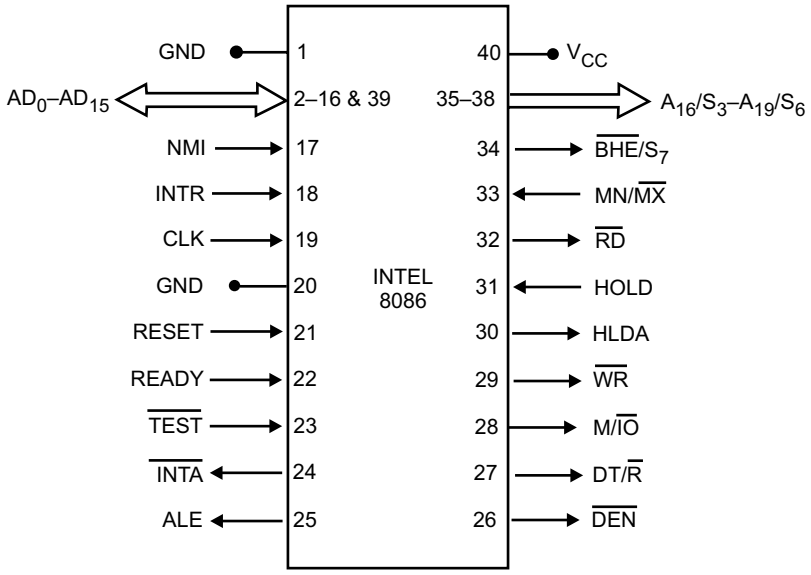
The 8086 Microprocessor

1. Draw the pin diagram of 8086.

Ans. There would be two pin diagrams—one for MIN mode and the other for MAX mode of 8086, shown in Figs. 11.1 and 11.2 respectively. The pins that differ with each other in the two modes are from pin-24 to pin-31 (total 8 pins).



Signals of intel 8086 for minimum mode of operation

2. What is the technology used in 8086 μP?

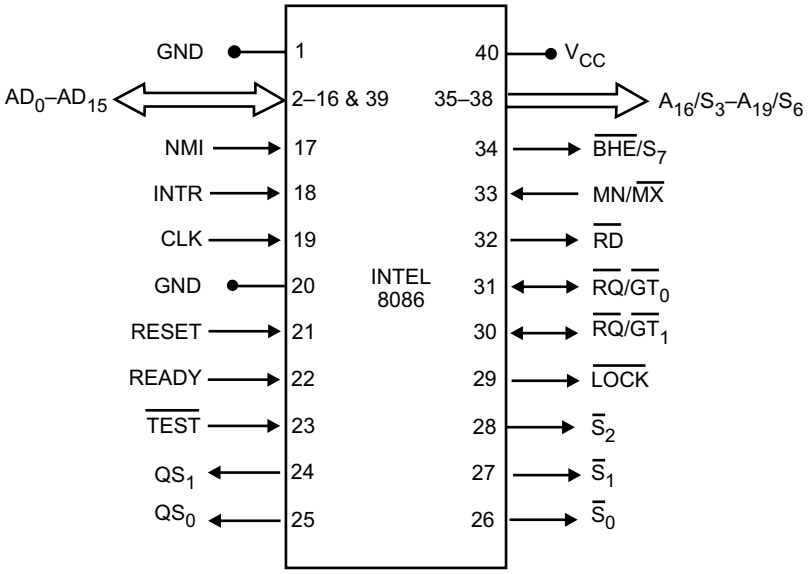
Ans. It is manufactured using high performance metal-oxide semiconductor (HMOS) technology. It has approximately 29,000 transistors and housed in a 40-pin DIP package.

3. Mention and explain the modes in which 8086 can operate.

Ans. 8086 μP can operate in two modes—MIN mode and MAX mode.

When MN/$\overline{MX}$ pin is high, it operates in MIN mode and when low, 8086 operates in MAX mode.

For a small system in which only one 8086 microprocessor is employed as a CPU, the system operates in MIN mode (Uniprocessor). While if more than one 8086 operate in a system then it is said to operate in MAX mode (Multiprocessor).



Signals of intel 8086 for maximum mode of operation

The bus controller IC (8288) generates the control signals in case of MAX mode, while in MIN mode CPU issues the control signals required by memory and I/O devices.

4. Distinguish between the lower sixteen address lines from the upper four.

Ans. Both the lower sixteen address lines $(AD_0 - AD_{15})$ and the upper four address lines $(A_{16}/S_3 - A_{19}/S_6)$ are multiplexed.

During T_1, the lower sixteen lines carry address $(A_0 - A_{15})$, while during T_2, T_3 and T_4 they carry data.

Similarly during T_1, the upper four lines carry address (A_{16}– A_{19}), while during T_2, T_3 and T_4, they carry status signals.

5. In how many modes the minimum-mode signal can be divided?

Ans. In the MIN mode, the signals can be divided into the following basic groups: address/data bus, status, control, interrupt and DMA.

6. Tabulate the common signals, Minimum mode signals and Maximum mode signals. Also mention their functions and types.

Ans. Table 11.1 shows the common signals, Minimum mode signals and the Maximum mode signals, along with the functions of each and their types.

Table 11.1 : (a) Signals common to both minimum and maximum mode, (b) Unique minimum-mode signals, (c) Unique maximum-mode signals for 8086.

Common signals		
Name	**Function**	**Type**
AD15-AD0	Address/data bus	Bidirectional, 3–state
A19/S6-A16/S3	Address/status	Output, 3-state
MN/$\overline{MX}$	Minimum/maximum Mode control	Input
$\overline{RD}$	Read control	Output, 3-state
$\overline{TEST}$	Wait on test control	Input
READY	Wait state control	Input
RESET	System reset	Input
NMI	Nonmasakable Interrupt request	Input
INTR	Interrupt request	Input
CLK	System clock	Input
V_{cc}	+5V	Input
GND	Ground	

(a)

Minimum mode signals (MN/$\overline{MX}$ =V_{cc})		
Name	**Function**	**Type**
HOLD	Hold request	Input
HLDA	Hold acknowledge	Output
$\overline{WR}$	Write control	Output, 3-state
M/$\overline{IO}$	IO/memory control	Output, 3-state
DT/$\overline{R}$	Data transmit/receive	Output, 3-state
$\overline{DEN}$	Data enable	Output, 3-state
ALE	Address latch enable	Output
$\overline{INTA}$	Interrupt acknowledge	Output

(b)

Maximum mode signals (MN/$\overline{MX}$ =GND)		
Name	**Function**	**Type**
$\overline{RQ}$/$\overline{GT1,0}$	Request/grant bus access control	Bidirectional
$\overline{LOCK}$	Bus priority lock control	Output, 3-state
$\overline{S2}$ – $\overline{S0}$	Bus cycle status	Output, 3-state
QS1, QS0	Instruction queue status	Output

(c)

7. Mention the different varieties of 8086 and their corresponding speeds.

Ans. The following shows the different varieties of 8086 available and their corresponding speeds.

Types	Speeds
8086	5 MHz
8086–1	10 MHz
8086–2	8 MHz

8. Mention (a) the address capability of 8086 and (b) how many I/O lines can be accessed by 8086.

Ans. 8086 addresses via its A_0–A_{19} address lines. Hence it can address 2^{20} = 1MB memory.

Address lines A_0 to A_{15} are used for accessing I/O's. Thus, 8086 can access 2^{16} = 64 KB of I/O's.

9. What is meant by microarchitecture of 8086?

Ans. The individual building blocks of 8086 that, as a whole, implement the software and hardware architecture of 8086. Because of incorporation of additional features being necessitated by higher performance, the microarchitecture of 8086 or for that matter any microprocessor family, evolves over time.

10. Draw and discuss the architecture of 8086. Mention the jobs performed by BIU and EU.

Ans. The architecture of 8086 is shown below in Fig. 11.3. It has got two separate functional units—Bus Interface Unit (BIU) and Execution Unit (EU).

8086 architecture employs parallel processing—i.e., both the units (BIU and EU) work at the same time. This is *Unlike* 8085 in which *Sequential* fetch and execute operations take place. Thus in case of 8086, efficient use of system bus takes place and higher performance (because of reduced instruction time) is ensured.

- BIU has segment registers, instruction pointer, address generation and bus control logic block, instruction queue while the EU has general purpose registers, ALU, control unit, instruction register, flag (or status) register.

The main jobs performed by BIU are:

- BIU is the 8086's interface to the outside world, i.e., all *External* bus operations are done by BIU.
- It does the job of instruction fetching, reading/writing of data/operands for memory and also the inputting/outputting of data for peripheral devices.
- It does the job of filling the instruction queue.
- Does the job of address generation.

The main jobs performed by the execution unit are:

- Decoding/execution of instructions.
- It accepts instructions from the output end of instruction queue (residing in BIU) and data from the general purpose registers or memory.
- It generates operand addresses when necessary, hands them over to BIU requesting it (BIU) to perform read or write cycle to memory or I/O devices.
- EU tests the status of flags in the control register and updates them when executing instructions.
- EU waits for instructions from the instruction queue, when it is empty.
- EU has no connection to the system buses.

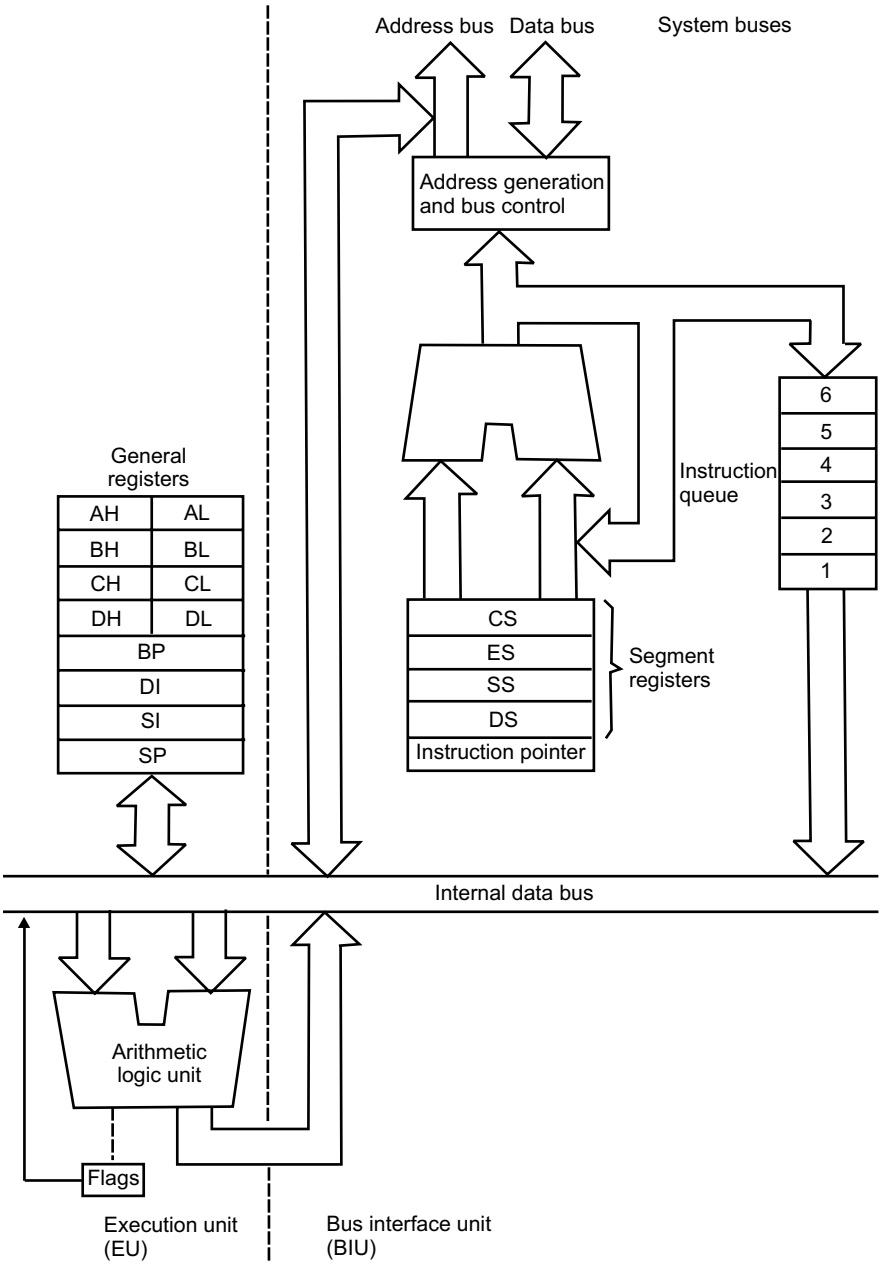


Fig. 11.3: CPU model for the 8086 microprocessor. A separate execution unit (EU) and bus interface unit (BIU) are provided.

11. Explain the operations of instructions queue residing in BIU.

Ans. The instruction queue is 6-bytes in length, operates on FIFO basis, and receives the instruction codes from memory. BIU fetches the instructions meant for the queue ahead of time from memory. In case of JUMP and CALL instructions, the queue is dumped and newly formed from the new address.

Because of the instruction queue, there is an overlap between the instruction execution and instruction fetching. This feature of fetching the next instruction when the current instruction is being executed, is called *Pipelining*.

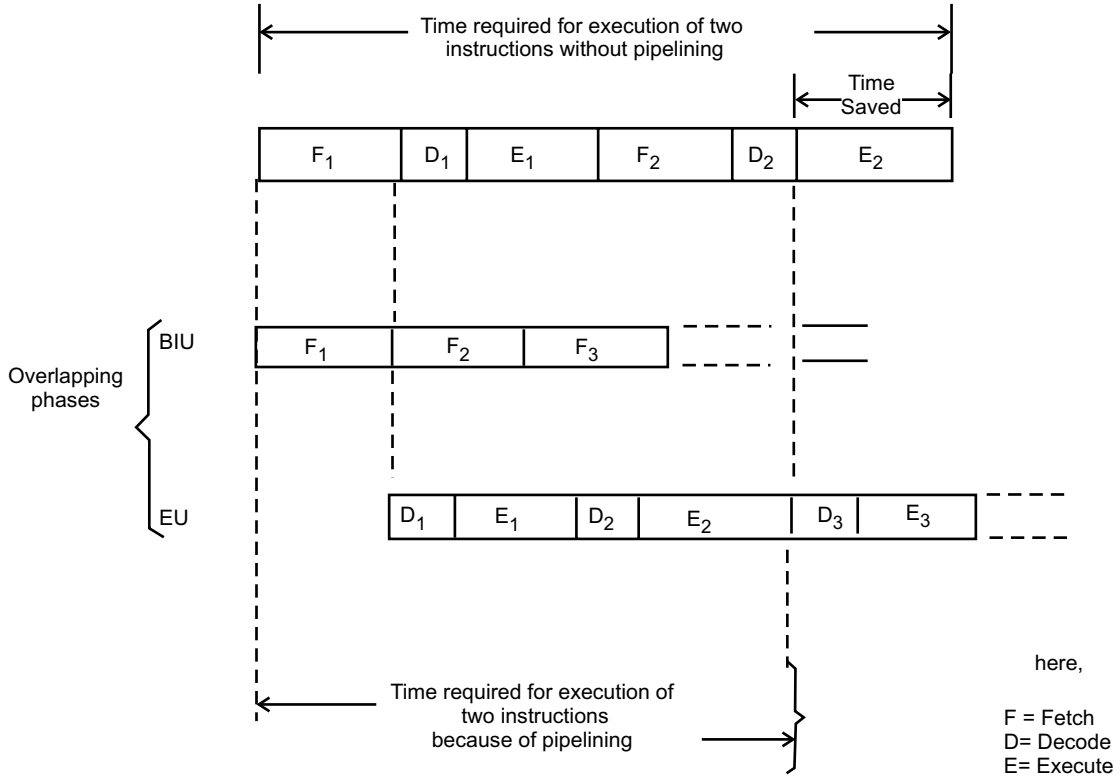


Fig.11.4: Pipelining procedure saves time

Fig. 11.4, which is self-explanatory, shows that there is definitely a time saved in case of overlapping phases (as in the case of 8086) compared to sequential phases (as in the case of 8085).

Initially, the queue is empty and CS : IP is loaded with the required address (from which the execution is to be started). Microprocessor 8086 starts operation by fetching 1 (or 2) byte(s) of instruction code(s) if CS : IP address is odd (even).

The 1st byte is always an opcode, which when decoded, one byte in the queue becomes empty and the queue is updated. The filling in operation of the queue is not started until two bytes of the instruction queue is empty. The instruction execution cycle is *never* broken for fetch operation.

After decoding of the 1st byte, the decoder circuit gets to know whether the instruction is of single or double opcode byte.

For a single opcode byte, the next bytes are treated as data bytes depending upon the decoded instruction length, otherwise the next byte is treated as the second byte of the instruction opcode.

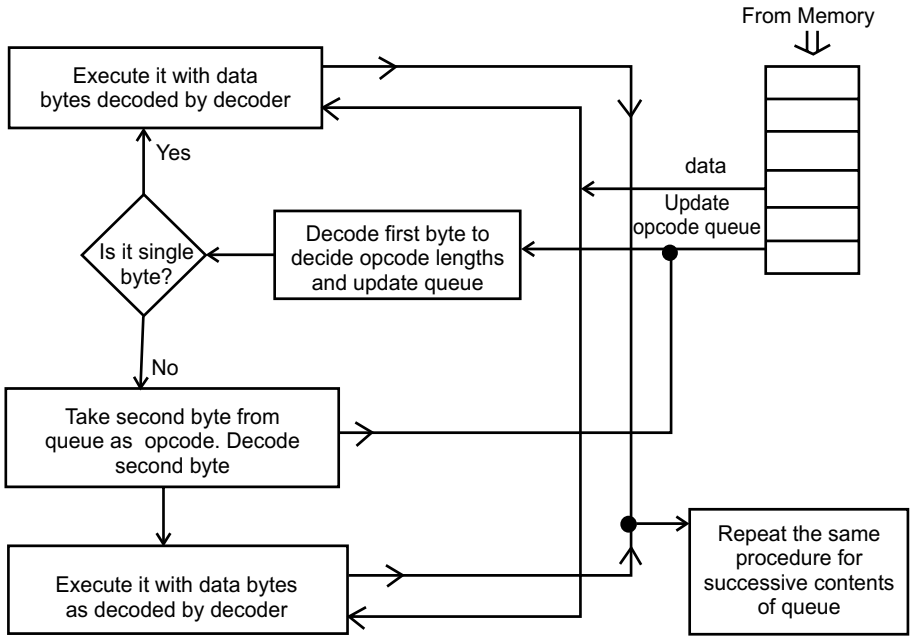


Fig.11.5 : The queue operation

For a 2-byte instruction code, the decoding process takes place taking both the bytes into consideration which then decides on the decoded instruction length and the number of subsequent bytes which will be treated as instruction data. Updation of the queue takes place once a byte is read from the queue.

The queue operation is shown in Fig. 11.5 in block schematic form.

12. Mention the conditions for which EU enters into WAIT mode.

Ans. There are three conditions that cause the EU to enter into WAIT state. These are:

- When an instruction requires the access to a memory location not in the queue.
- When a JUMP instruction is executed. In this case the current queue contents are aborted and the EU waits until the instructions at the jump address is fetched from memory.
- During execution of instruction which are very slow to execute. The instruction AAM (ASCII adjust for multiplication) requires 83 clock cycles for execution. For such a case, the BIU is made to wait till EU pulls one or two bytes from the queue before resuming the fetch cycle.

13. Mention the kind of operations possible with 8086.

Ans. It can perform bit, byte, word and block operations. Also multiplication and division operations can be performed by 8086.

14. Mention the total number of registers of 8086 and show the manner in which they are grouped.

Ans. There are in all fourteen numbers of 16-bit registers. The different groups are made as hereunder:

- Data group, pointers and index group, status and control flag group and segment group.
- The data group consists of AX (accumulator), BX (base), CX (count) and DX (data).
- Pointer and Index group consist of SP (Stack pointer), BP (Base pointer), SI (Source Index), DI (Destination index) and IP (Instruction pointer).
- Segment group consists of ES (Extra Segment), CS (Code Segment), DS (Data Segment) and SS (Stack Segment).
- Control flag group consists of a single 16-bit flag register.

Fig. 11.6 shows the registers placed in the different groups to form a programming model.

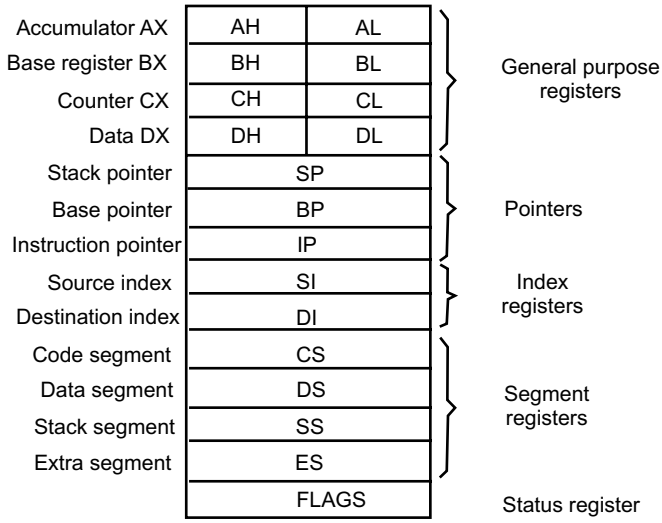
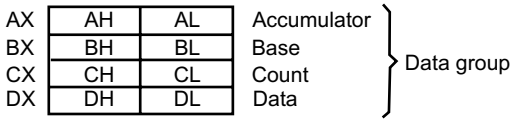


Fig.11.6: Schematic diagram of intel 8086 registers

15. Describe, in detail, the general purpose of data registers.

Ans. Fig. 11.7 shows the four data registers along with their dedicated functions also.



Register	Operations
AX	Word multiply, Word divide, Word I/O
AL	Byte multiply, byte divide, byte I/O, translate, decimal arithmetic
AH	Byte multiply, byte divide
BX	Translate
CX	String operations, Loops
CL	Variable shift and rotate
DX	Word multiply, Word divide, indirect I/O

Fig. 11.7: Data group registers and their functions

All the four registers can be used bytewise or wordwise. The alphabets X, H and L respectively refer to word, higher byte or lower byte respectively of any register.

All the four registers can be used as the source or destination of an operand during an arithmetic operation such as ADD or logical operation such as AND, although particular registers are earmarked for specific operations. Register C is used as a count register in string operations and as such is called a 'count' register. Register C is also used for multibit shift or rotate instructions.

Register D is used to hold the address of I/O port while register A is used for all I/O operations that require data to be inputted or outputted.

16. Describe the status register of 8086.

Ans. It is a 16-bit register, also called flag register or Program Status Word (PSW). Seven bits remain unused while the rest nine are used to indicate the conditions of flags. The status flags of the register are shown below in Fig. 11.8.

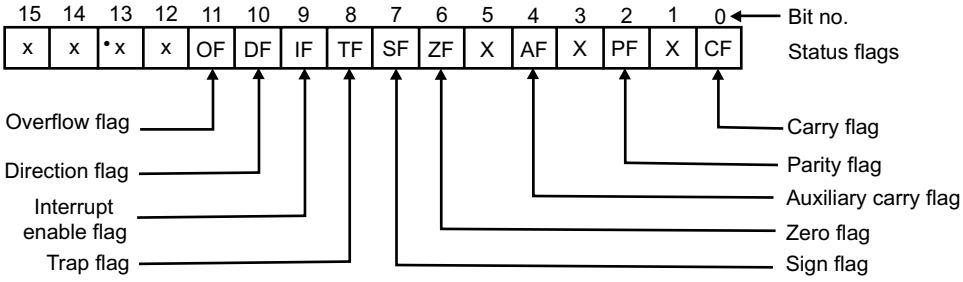


Fig.11.8: Status flags of intel 8086

Out of nine flags, six are condition flags and three are control flags. The control flags are TF (Trap), IF (Interrupt) and DF (Direction) flags, which can be set/reset by the programmer, while the condition flags [OF (Overflow), SF (Sign), ZF (Zero), AF (Auxiliary Carry), PF (Parity) and CF (Carry)] are set/reset depending on the results of some arithmetic or logical operations during program execution.

CF is set if there is a carry out of the MSB position resulting from an addition operation or if a borrow is needed out of the MSB position during subtraction.

PF is set if the lower 8-bits of the result of an operation contains an even number of 1's. AF is set if there is a carry out of bit 3 resulting from an addition operation or a borrow required from bit 4 into bit 3 during subtraction operation.

ZF is set if the result of an arithmetic or logical operation is zero.

SF is set if the MSB of the result of an operation is 1. SF is used with unsigned numbers.

OF is used only for signed arithmetic operation and is set if the result is too large to be fitted in the number of bits available to accommodate it.

The functions of the flags along with their bit positions are shown in Fig. 11.9 below.

Bit position	Name	Function
0	CF	Carry flag: Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity flag: Set if low-order 8-bit of result contain an even number of 1-bit; cleared otherwise
4	AF	Set on carry from or borrow to the low-order 4-bits of AL; cleared otherwise

6	ZF	Zero flag: Set if result is zero; cleared otherwise
7	SF	Sign Flag: Set equal to high-order bit of result (0 is positive, 1 if negative)
8	TF	Signal step flag: Once set, a single-step interrupt occurs after the next instruction executes; TF is cleared by the single-step interrupt
9	IF	Interrupt-enable flag: When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction flag: Causes string instructions to auto decrement the appropriate index register when set; clearing DF causes auto increment.
11	OF	Overflow flag: Set if the signed result cannot be expressed within the number of bits in the destination operand; cleared otherwise.

Fig.11.9: 8086 flags: DF, IF and TF can be set or reset to control the operations of the processor. The remaining flags are status indicators.

17. Discuss the three control flags of 8086.

Ans. The three control flags of 8086 are TF, IF and DF. These three flags are programmable, i.e., can be set/reset by the programmer so as to control the operation of the processor.

When TF (trap flag) is set (=1), the processor operates in single stepping mode—i.e., pausing after each instruction is executed. This mode is very useful during program development or program debugging.

When an interrupt is recognised, TF flag is cleared. When the CPU returns to the main program from ISS (interrupt service subroutine), by execution of IRET in the last line of ISS, TF flag is restored to its value that it had before interruption.

TF cannot be directly set or reset. So indirectly it is done by pushing the flag register on the stack, changing TF as desired and then popping the flag register from the stack.

When IF (interrupt flag) is set, the maskable interrupt INTR is enabled otherwise disabled (i.e., when IF = 0).

IF can be set by executing STI instruction and cleared by CLI instruction. Like TF flag, when an interrupt is recognised, IF flag is cleared, so that INTR is disabled. In the last line of ISS when IRET is encountered, IF is restored to its original value.

When 8086 is reset, IF is cleared, i.e., resetted.

DF (direction flag) is used in string (also known as block move) operations. It can be set by STD instruction and cleared by CLD. If DF is set to 1 and MOVS instruction is executed, the contents of the index registers DI and SI are automatically decremented to access the string from the highest memory location down to the lowest memory location.

18. Discuss the Pointers and Index group of registers.

Ans. The pointer registers are SP and BP while the index registers are SI and DI.

All the four are 16-bit registers and are used to store offset addresses of memory locations relative to segment registers. They act as memory pointers. As an example, MOV AH, [SI] implies, "Move the byte whose address is contained in SI into AH". If now, SI = 2000 H, then execution of above instruction will put the value FF H in register AH, shown in Fig. 11.10, [SI+1 : SI] = ABFF H, where obviously SI+1 points to memory location 2001 H and [SI+1] = AB H.

SI and DI are also used as general purpose registers. Again in certain string (block move) instructions, SI and DI are used as source and destination index registers

respectively. For such cases, contents of SI are added to contents of DS register to get the actual source address of data, while the contents of DI are added to the contents of ES to get the actual destination address of data.

SP and BP stand for stack pointer and base pointer with SP containing the offset address or the stack top address. The actual stack address is computed by adding the contents of SP and SS.

Data area(s) may exist in stack. To access such data area in stack segment, BP register is used which contains the offset address. BP register is also used as a general purpose register.

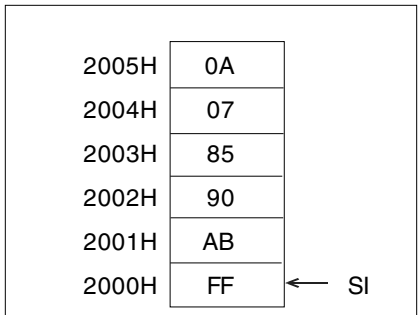


Fig. 11.10: SI is pointing to memory locations 2000 H.

Instruction pointer (IP) is also included in the index and pointers group. IP points to the offset instead of the actual address of the next instruction to be fetched (from the current code segment) in BIU. IP resides in BIU but cannot be programmed by the programmer.

19. Describe in brief the four segment registers.

Ans. The four segment registers are CS, DS, ES and SS—standing for code segment register, data segment register, extra segment register and stack segment register respectively. When a particular memory is being read or written into, the corresponding memory address is determined by the content of one of these four segment registers in conjunction with their offset addresses.

The contents of these registers can be changed so that the program may jump from one active code segment to another one.

The use of these segment registers will be more apparent in memory segmentation schemes.

20. Discuss A_{16}/S_3—A_{19}/S_6 Signals of 8086.

Ans. These are time multiplexed signals. During T_1, they represent $A_{19} - A_{16}$ address lines. During I/O operations, these lines remain low. During T_2–T_4, they carry status signals.

S_4 and S_3 (during T_2 to T_4) identify the segment register employed for 20-bit physical address generation.

Status signal S_5 (during T_2 to T_4) represents interrupt enable status. This is updated at the beginning of each clock cycle.

Status signal S_6 remains low during T_2 to T_4.

21. Discuss $\overline{BHE}/S_7$ signal.

Ans. During T_1, this becomes bus high enable signal and remains low while during T_2 to T_4 it acts as a status signal S_7 and remains high during this time.

During T_1, when $\overline{BHE}$ signal is active, i.e., remains low, it is used as a chip select signal on the higher byte of data bus—i.e., D_{15}–D_8.

Table 11.2 shows $\overline{BHE}$ and A_0 signals determine one of the three possible references to memory.

Table 11.2: Status of $\overline{BHE}$ and A_0 identify memory references

$\overline{BHE}$	A_0	Word/byte access
0	0	Both banks active, 16-bit word transfer on $AD_{15} - AD_0$
0	1	Only high bank active, upper byte from/to odd address on $AD_{15} - AD_8$
1	0	Only low bank active, lower byte from/to even address AD_7–AD_0
1	1	No bank active

22. Discuss the Reset pin of 8086.

Ans. Reset is an active high input signal and must be active for at least 4 CLK cycles to be accepted by 8086. This signal is internally synchronised and execution starts only after Reset returns to low value.

For proper initialisation, Reset pulse must *not* be applied before 50μS of 'power on' of the circuit. During Reset state, all three buses are tristated and ALE and HLDA are driven low.

During resetting, all internal register contents are set to 0000 H, but CS is set to F000 H and IP to FFF0 H. Thus execution starts from physical address FFFF0 H. Thus EPROM in 8086 is interfaced so as to have the physical memory location forms FFFF0 H to FFFFF H, i.e., at the end of the map.

23. Discuss the two pins (a) DT/$\overline{R}$ and (b) $\overline{DEN}$.

Ans. (a) DT/$\overline{R}$ is an output pin which decides the directions of data flow through the transreceivers (bidirectional buffers).

When the processor sends out data, this signal is 1 while when it receives data, the signal status is 0.

(b) $\overline{DEN}$ stands for data enable. It is an active low signal and indicates the availability of data over the address/data lines. This signal enables the transreceivers to separate data from the multiplexed address/data signal. It is active from the middle of T_2 until the middle of T_4.

Both DT/$\overline{R}$ and $\overline{DEN}$ are tristated during 'hold acknowledge'.

24. Elaborate the functions of the pins $\overline{S_2}$, $\overline{S_1}$ and $\overline{S_0}$.

Ans. These three are output status signals in the MAX mode, indicating the type of operation carried out by the processor.

The signals become active during T_4 of the previous cycle and remain active during T_1 and T_2 of the current cycle. They return to the 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. Table 11.3 shows the different bus cycles of 8086 for different combinations of these three signals.

Table 11.3: Bus status codes

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	CPU cycles
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

25. Explain the $\overline{\text{LOCK}}$ signal.

Ans. It is an active low output signal and is activated by LOCK prefix instruction and remains active until the completion of the next instruction. It floats to tri-state during hold acknowledge when $\overline{\text{LOCK}}$ signal is low, all interrupts get masked and HOLD request is not granted. $\overline{\text{LOCK}}$ signal is used by the processor to prevent other devices from accessing the system control bus. This symbol is used when CPU is executing some critical instructions and through this signal other devices are informed that they should not issue HOLD signal to 8086.

26. Explain the $\overline{\text{TEST}}$ signal.

Ans. It is an active low input signal. Normally the BUSY pin (output) of 8087 NDP is connected to the $\overline{\text{TEST}}$ input pin of 8086. When maths co-processor 8087 is busy executing some instructions, it pulls its BUSY signal high. Thus the $\overline{\text{TEST}}$ signal of 8086 is consequently high, and it (8086) is made to WAIT until the BUSY signal goes low. When 8087 completes its instruction executions, BUSY signal becomes low. Thus the $\overline{\text{TEST}}$ input of 8086 becomes low also and then only 8086 goes in for execution of its program.

27. Show how demultiplexing of address/data bus is done and also show the availability of address/data during read/write cycles.

Ans. The demultiplexing of lower 2-bytes of address/data bus (AD_0–AD_{15}) is done by 8282/8283 octal latch with 8282 providing non-inverting outputs while 8283 gives out inverted outputs. The chip outputs are also buffered so that more drive is available at their outputs.

A D latch is central to the demultiplexing operations of these latches. During T_1 when ALE is high, the latch is transparent and the output of latch is 'A' (address) only. At the end of T_1, ALE has a high to low transition which latches the address available at the D input of the latch, so that address is continued to be available from the Q output of the latch (i.e., whole of T_1 to T_4 states).

It is to be noted that memory and I/O devices do not access the data bus until the beginning of T_2, thus the 'data' is a 'don't care' till the end of T_1. This is shown in Fig. 11.11 and the timing diagram shows the availability of data for read and write cycles.

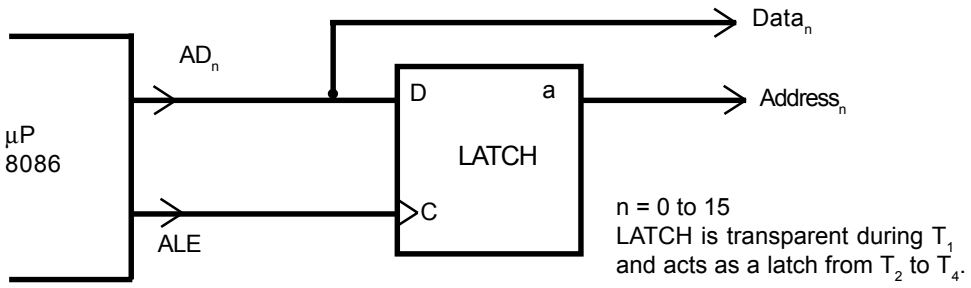


Fig.11.11: Demultiplexing the 8086 address/data bus

28. Discuss the Instruction Pointer (IP) of 8086.

Ans. Functionally, IP plays the part of Program Counter (PC) in 8085. But the difference is that IP holds the offset of the next word of the instruction code instead of the actual address (as in PC).

IP along with CS (code segment) register content provide the 20-bit physical (or real) address needed to access the memory. Thus CS:IP denotes the value of the memory address of the next code (to be fetched from memory).

Content of IP gets incremented by 2 because each time a word of code is fetched from memory.

29. Indicate the data types that can be handled by 8086 µP.

Ans. The types of data formats that can be handled by 8086 fall under the following categories:

- Unsigned or signed integer numbers—both byte-wide and word-wide.
- BCD numbers—both in packed or unpacked form.
- ASCII coded data. ASCII numbers are stored one number per byte.

30. Compare 8086 and 8088 microprocessors.

Ans. The Comparison between the two is tabulated below in Table 11.4.

Table 11.4: Comparison of 8086 and 8088

8086	8088
1. 2-byte data width, obtained by demultiplexing AD_0–AD_{15}.	1. 1-byte data width, obtained by demultiplexing AD_0–AD_7.
2. In MIN mode, pin-28 is assigned the signal M/$\overline{IO}$.	2. In MIN mode, pin-28 is assigned the signal IO/$\overline{M}$.
3. A 6-byte instruction queue.	3. A 4-byte instruction queue.
4. To access higher byte, $\overline{BHE}$ signal is used.	4. No such signal required, since data width is 1-byte only.
5. BIU dissimilar, but EU similar to 8088. Program instructions identical to 8088.	5. BIU dissimilar, but EU similar to 8086. Program instructions identical to 8086.
6. Program fetching from memory done only when 2-bytes are empty in queue.	6. Program fetching from memory done as soon as a byte is free in queue.
7. Pin-34 is $\overline{BHE}$/S_7. During T_1, $\overline{BHE}$ is used to enable data on D_8–D_{15}. During T_2–T_4, status of this pin is 0. In MAX mode, 8087 monitors this pin to identify the CPU—8086 or 8088? Accordingly it sets its queue length to 6 or 4 respectively.	7. Pin-34 is $\overline{SS_0}$. It acts as $\overline{S_0}$ in the MIN mode. In MAX mode $\overline{SS_0}$ = 1 always.

31. Comment on the instruction size of 8086.

Ans. It varies from 1 to 6 bytes.

32. Discuss the instruction format of 8086.

Ans. The instruction format of 8086 is shown in Fig. 11.12. It is extendable up to 6-bytes. The first byte contains D and W—Direction Register Bit and Data Size Bit respectively. Both D and W are 1-bit in nature.

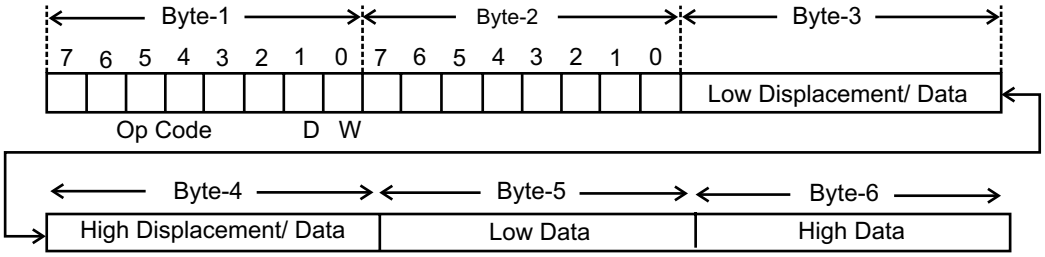


Fig. 11.12: 8086 Instruction format

- If D = 1, then register operand existing in byte-2 is the destination operand, otherwise (i.e., if D = 0) it is a source operand.
- W indicates whether the operation is an 8-bit or a 16-bit data. If W = 0 then it is an 8-bit operation, else (i.e., W = 1) it is 16-bit one.
- The 2nd byte (byte-2) indicates whether one of the operands is in memory or both are in registers. This byte contains three fields:

Field	Abbreviation	Length (no. of bits)
Mode field	MOD	2
Register field	REG	3
Register/Memory field	r/m	3

33. Discuss the MOD and r/m and REG fields.

Ans. MOD field is a 2-bit field. It addresses memory in the following manner.

MOD field values

0 0 ⟶ Memory addressing without displacement
0 1 ⟶ Memory addressing with 8-bit displacement
1 0 ⟶ Memory addressing with 16-bit displacement
1 1 ⟶ Register addressing with
W = 0 for 8-bit data
and W = 1 for 16-bit data

The r/m field which is a 3-bit field, along with MOD field defines the 2nd operand. If MOD = 11, then it is a register to register mode. Again if MOD = 00,01 or 10 then it is a memory mode. Table 11.5 shows how effective address of memory operand gets selected for MOD = 00,01 and 10 values.

Table 11.5: For effective address calculation, values of MOD and r/m

r/m	MOD 00	MOD 01	MOD 10	MOD 11	
				W = 0	W = 1
000	[BX] + [SI]	[BX]+[SI]+D8	[BX]+[SI]+D16	AL	AX
001	[BX] + [DI]	[BX]+[DI]+D8	[BX]+[DI]+D16	CL	CX
010	[BP] + [SI]	[BP]+[SI]+D8	[BP]+[SI]+D16	DL	DX
011	[BP] + [DI]	[BP]+[DI]+D8	[BP]+[DI]+D16	BL	BX
100	[SI]	[SI]+D8	[SI]+D16	AH	SP
101	[DI]	[DI]+D8	[DI]+D16	CH	BP
110	Direct Addressing	[BP]+D8	[BP]+D16	DH	SI
111	[BX]	[BX]+D8	[BX]+D16	BH	DI

Again, for MOD values 00,01 and 10, the default segment registers selected are shown in Table 11.6.

Table 11.6: Segment register for various memory addressing modes

r/m	MOD 00	MOD 01	MOD 10	Segment Register used
000	[BX] + [SI]	[BX]+[SI]+DS	[BX]+[SI]+D16	DS
001	[BX] + [DI]	[BX]+[DI]+DS	[BX]+[DI]+D16	DS
010	[BP] + [SI]	[BP]+[SI]+DS	[BP]+[SI]+D16	SS
011	[BP] + [DI]	[BP]+[DI]+DS	[BP]+[DI]+D16	SS
100	[SI]	[SI]+DS	[SI]+D16	DS
101	[DI]	[DI]+DS	[DI]+D16	DS
110	D16 Direct Addressing [DS]	[BP]+DS Stack pointer register [SS]	[BP]+D16 Stack segment register [SS]	DS or SS as in MOD column
111	[BX]	[BX]+DS	[BX]+D16	DS

The REG field is a 3-bit field and indicates the register for the first operand which can be source/destination operand, depending on D = 0/1.

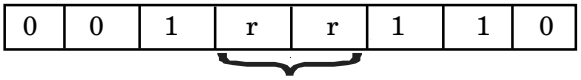
How REG field along with the status of W(0 or 1) select the different registers, is shown in Table 11.7.

Table 11.7: Definition of registers with 'W'

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

34. Discuss the instruction format for segment override prefix.

Ans. Default segment selection can be overriden by the override prefix byte, as shown in below.



Depending on the 2-bit rr values, the segments selected are shown in Table 11.8.

Table 11.8 : Segment selection by override prefix technique

rr values	Segments selected
00	ES
01	CS
10	SS
11	DS

The override prefix byte follows the opcode byte of the instruction, whenever used.

35. Is direct memory to memory data transfer possible in 8086?

Ans. No, 8086 does not have provision for direct memory to memory data transfer.

For this to be implemented, AX is used as an intermediate stage of data. The source byte (from the memory) is moved into AX register with one instruction. The second instruction moves the content of AX into destination location (into another memory location). As example,

```
MOV AH, [SI]
MOV [DI], AH
```

Here, the first instruction moves the content of memory location, whose offset address remains in SI, into AH. The second instruction ensures that the content of AH is moved into another memory location whose offset address is in DI.

36. Can the data segment (DS) register be loaded directly by its address?

Ans. No, it cannot be done directly. Instead, AX is loaded with the initial address of the DS register and then it is transferred to DS register, as shown below:

```
MOV AX, DS ADDR:   AX is loaded with initial address of DS register
MOV DS, AX:        DS is loaded with AX, i.e., ultimately with DS ADDR
```

37. Show, in tabular form, the default and alternate segment registers for different types of memory references.

Ans. Table 11.9 shows the default and alternate register segments which can be used for different types of memory references.

Table 11.9: Default and alternate register assignments

Type of memory reference	Default segment	Alternative segment	Offset (Logical address)
Instruction fetch	CS	None	IP
Stack operation	SS	None	SP, BP
General data	DS	CS, ES, SS	Effective address
String source	DS	CS, ES, SS	SI
String destination	ES	None	DI
BX used as pointer	DS	CS, ES, SS	Effective Address
BP used as pointer	SS	CS, ES, DS	Effective Address

12

Memory Organisation

1. Mention the address capability of 8086 and also show its memory map.

Ans. 8086, via its 20-bit address bus, can address 2^{20} = 1,048,576 or 1 MB of different memory locations. Thus the memory space of 8086 can be thought of as consisting of 1,048,576 bytes or 524,288 words.

The memory map of 8086 is shown in Fig. 12.1, where the whole memory space starting from 00000 H to FFFFF H is divided into 16 blocks—each one consisting of 64 KB. This division is arbitrary but at the same time a convenient one—because the most significant hex digit increases by 1 with each additional block. Thus, 30000 H memory location is 65,536 bytes higher in memory than the memory location 20000 H.

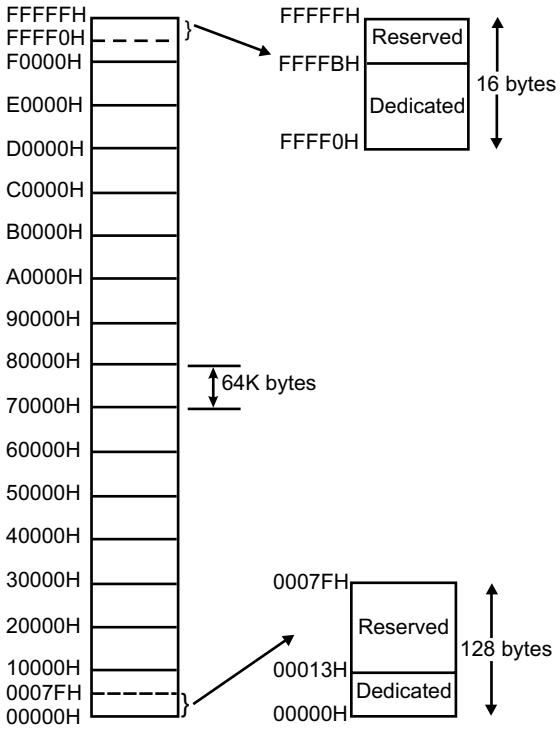


Fig. 12.1: Memory map for the 8086 microprocessor. Some memory locations are dedicated or reserved.

The lower and upper ends of the memory map are shown separately—earmarking some spaces as reserved and some as 'dedicated'.

The reserved locations are meant for future hardware and software needs while the dedicated locations are used for processing of specific system interrupts and reset functions.

2. Mention the different types of memory segmentations of 8086.

Ans. The different memory segmentations done in case of 8086 are

- Continuous
- partially overlapped
- fully overlapped and
- disjointed

This is shown in Fig.12.2.

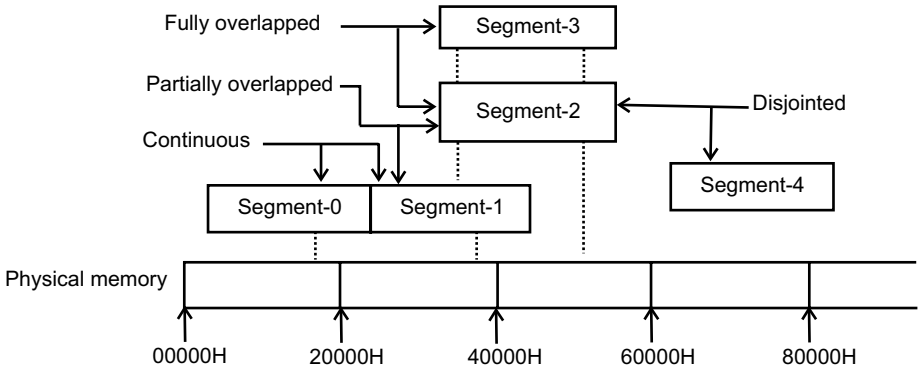


Fig. 12.2: Depiction of different types of segments

In the figure,

Segments-0 and 1 ⟶ Continuous
Segments-1 and 2 ⟶ Partially overlapped
Segments-2 and 3 ⟶ Fully overlapped
and Segments-2 and 4 ⟶ Disjointed

3. Describe memory segmentation scheme of 8086. What is meant by currently active segments?

Ans. 1 MB memory of 8086 is partitioned into 16 segments—each segment is of 64 KB length. Out of these 16 segments, only 4 segments can be active at any given instant of time—these are code segment, stack segment, data segment and extra segment. The four memory segments that the CPU works with at any time are called currently active segments. Corresponding to these four segments, the registers used are Code Segment Register (CS), Data Segment Register (DS), Stack Segment Register (SS) and Extra Segment Register (ES) respectively.

Each of these four registers is 16-bits wide and user accessible—i.e., their contents can be changed by software.

The code segment contains the instruction codes of a program, while data, variables and constants are held in data segment. The stack segment is used to store interrupt and subroutine return addresses.

The extra segment contains the destination of data for certain string instructions. Thus 64 KB are available for program storage (in CS) as well as for stack (in SS) while 128 KB of space can be utilised for data storage (in DS and ES).

One restriction on the base address (starting address) of a segment is that it must reside on a 16-byte address memory—examples being 00000 H, 00010 H or 00020 H, etc.

4. Mention the maximum size of memory that can be active for 8086.

Ans. The maximum size of active memory for 8086 is 256 KB. The break-up being
64 KB for program
64 KB for stack and
128 MB for data.

5. Why memory segmentation is done for 8086?

Ans. Memory segmentation, as implemented for 8086, gives rise to the following advantages:

- Although the address bus is 20-bits in width, memory segmentation allows one to work with registers having width 16-bits only.
- It allows instruction code, data, stack and portion of program to be more than 64 KB long by using more than one code, data, extra segment and stack segment.
- In a time-shared multitasking environment when the program moves over from one user's program to another, the CPU will simply have to reload the four segment registers with the segment starting addresses assigned to the current user's program.
- User's program (code) and data can be stored separately.
- Because the logical address range is from 0000 H to FFFF H, the same can be loaded at any place in the memory.

6. Discuss logical address, base segment address and physical address.

Ans. The logical address, also goes by the name of effective address or offset address (also known as offset), is contained in the 16-bit IP, BP, SP, BX, SI or DI.

The 16-bit content of one of the four segment registers (CS, DS, ES, SS) is known as the base segment address.

Offset and base segment addresses are combined to form a 20-bit physical address (also called real address) that is used to access the memory. This 20-bit physical address is put on the address bus ($AD_{19} - AD_0$) by the BIU.

7. Describe how the 20-bit physical address is generated.

Ans. The 20-bit physical (real) address is generated by combining the offset (residing in IP, BP, SP, BX, SI or DI) and the content of one of the segment registers CS, DS, ES or SS. The process of combination is as follows:

The content of the segment register is internally appended with 0 H (0000 H) on its right most end to form a 20-bit memory address—this 20-bit address points to the start of the segment. The offset is then added to the above to get the physical address.

Fig. 12.3 shows pictorially the actual process of generating a 20-bit physical address.

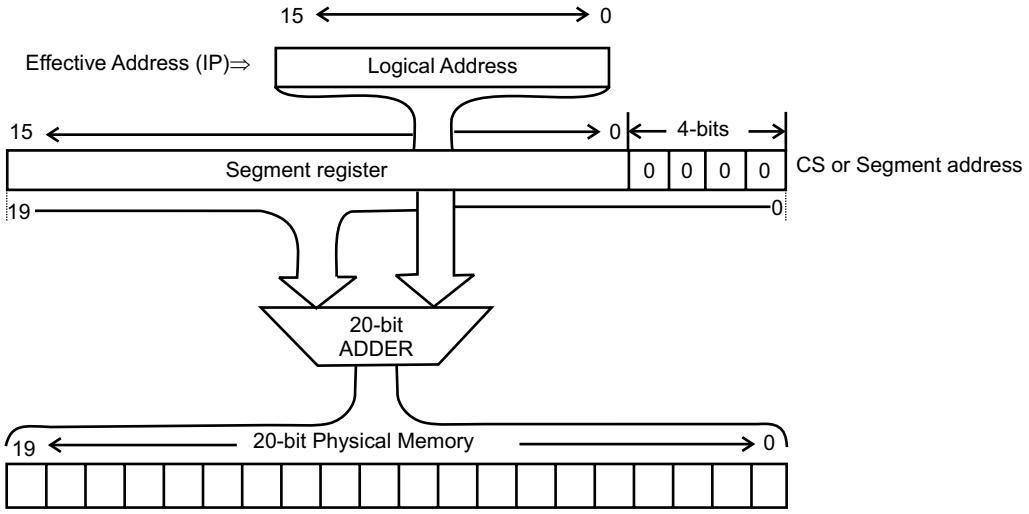


Fig. 12.3: Physical address generation

Thus, Physical Address = Segment Register content 16 D + Offset.

8. Although 8086 is a 16-bit μP, it deals with 8-bit memory. Why?

Ans. This is so for the following two reasons:

- It enables the microprocessor to work with both on bytes and words. This is very important because many I/O devices such as printers, terminals, modems etc, transfer ASCII coded data (7 or 8 bits).
- Quite a few of the operation codes of 8086 are single bytes while so many other instructions are there which vary from 2 to 7 bytes. By working with byte-width memory, these varied opcodes can easily be handled.

9. Is the flat scheme of memory applied for 8086 μP?

Ans. No, the flat (or unsegmented) scheme of memory is not applied for 8086 μP, because the memory of the same is a segmented one. In flat scheme, the entire memory space is thought of as a single addressable memory unit.

The flat scheme can be applied for 8086 by initialising all the segment registers with identical (or same) base address. Then all memory operations will refer to the same memory space.

10. Describe how memory is organised for 8086 μP?

Ans. The total address space 1 MB of 8086 is divided into 2 banks of memory—each bank of maximum size 512 KB. One is called the high order memory bank (or high bank) and the other low order memory bank (or low bank).

Low bank, high bank or both banks can be accessed by utilizing two signals $\overline{\text{BHE}}$ and A_0. Table 12.1 shows the three possible references to memory.

Table 12.1: Memory references

$\overline{BHE}$	A_0	Processing
0	0	Both Banks Active 16-bit world transfer on $AD_{15} \Leftrightarrow AD_0$
0	1	Only High bank Active (One byte from/to odd address on $AD_{15} \Leftrightarrow AD_8$)
1	0	Only Low bank Active (One byte from/to even address on $AD_7 \Leftrightarrow AD_0$)
1	1	No Bank Active

The high bank is selected for A_0=1 and $\overline{BHE}$ =0 and is connected to D_{15}–D_8 while the low bank is selected for A_0=0 and $\overline{BHE}$ =1. Neither low bank nor high bank would selected for A_0=1 and $\overline{BHE}$ =1.

Fig. 12.4 shows how the total address space (1MB) of 8086 is physically implemented by segregating it into low and high banks. It also shows that $\overline{CS}$ signal of the high bank is connected to $\overline{BHE}$ while the $\overline{CS}$ signal of the low bank is connected to A_0.

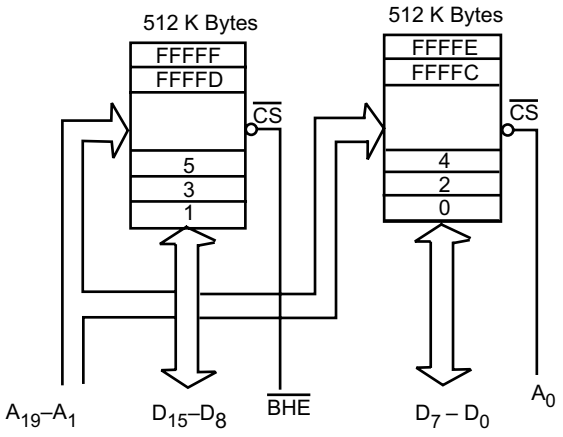


Fig.12.4: Selection of high and low banks of 8086

11. Show the profiles of low and high order memory banks.

Ans. The low and high order memory banks correspond to even and odd banks respectively.

The $\overline{CS}$ signal of low order memory bank is selected when $\overline{CS}$ = 0. Since A_0 (lowest address bus line) is connected to $\overline{CS}$, hence A_0 must have be to low for the low order bank to be selected. That is why the low order bank corresponds to even bank. Similarly the high order bank is selected when A_0 = 1. Hence, the higher order bank is called odd bank.

The profile of the low and high order banks are shown below in Fig. 12.5

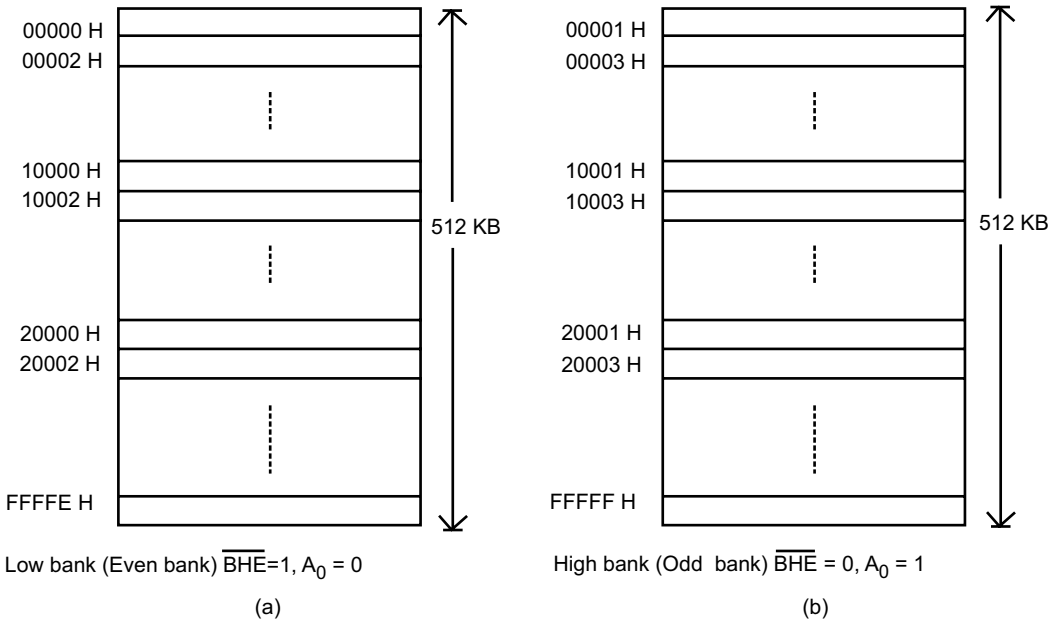


Fig. 12.5: (a) Low or even bank (b) High or odd bank

12. Draw the diagrams of (a) even-addressed byte transfer (b) odd-addressed byte transfer (c) even-addressed word transfer and (d) odd-addressed word transfer.

Ans. Fig. 12.6 shows the above four cases. A, B are representing the addresses while (A), (B) represent the content of address locations A and B respectively.

Figures (a) and (b) correspond to byte transfers for even and odd-addressed memory locations respectively. The shaded memory location indicates that the content of that particular memory location comes out either via higher byte data bus (D_{15}–D_8) or lower byte data bus (D_7–D_0) respectively.

Figures (a), (b) and (c) complete the data transfer in one bus cycle only.

$$\text{For Figure (a), } \overline{\text{BHE}} = 1, A_0 = 0$$

$$\text{For Figure (b), } \overline{\text{BHE}} = 0, A_0 = 1$$

$$\text{For Figure (c), } \overline{\text{BHE}} = 0, A_0 = 0$$

Figure (d) corresponds to an odd-addressed word transfer and it takes two bus cycles to complete this transfer.

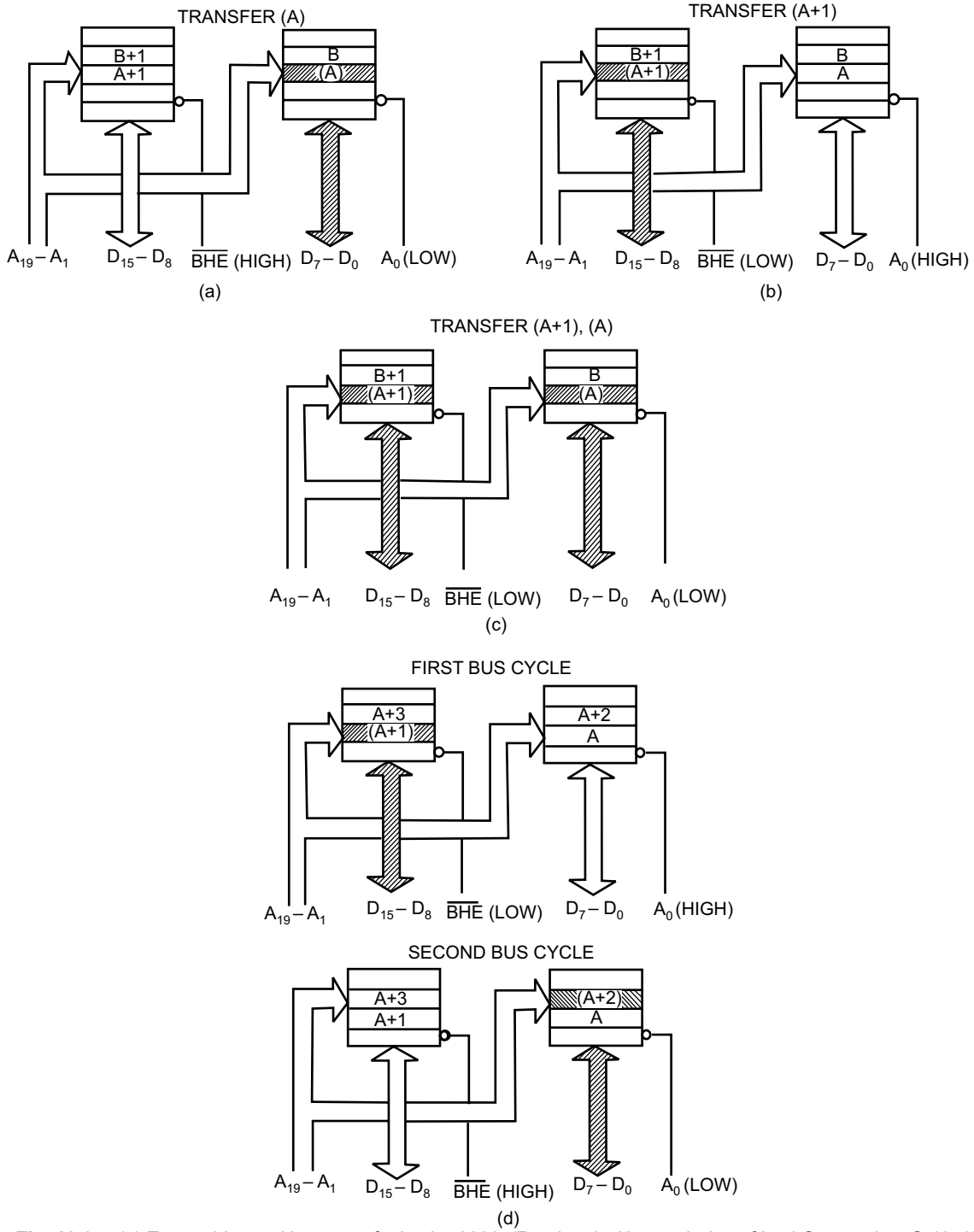


Fig. 12.6: (a) Even-addressed byte transfer by the 8086. (Reprinted with permission of Intel Corporation, © 1979)
(b) Odd-addressed byte transfer by the 8086. (Reprinted with permission of Intel Corporation, © 1979)
(c) Even-addressed word transfer by the 8086. (Reprinted with permission of Intel Corporation, © 1979)
(d) Odd-addressed word transfer by the 8086. (Reprinted with permission of Intel Corporation, © 1979)

This odd-addressed word is an unaligned one and the LSB of the address is in the high memory bank.

The odd byte of the word is at address location A + 1 and is selected by making $\overline{BHE}$ = 0 and A_0. Thus in the first bus cycle, data to transferred on $D_{15} - D_8$.

In the second bus cycle, 8086 automatically increments the address. Hence A_0 becomes 0, representing even address A + 2. This is in the low bank and is accessed by making $\overline{BHE}$ = 1 and A_0 = 0.

13. Which pins identify the segment registers used for 20-bit physical address generation?

Ans. Pins A_{16} and A_{17} become S_3 and S_4 from the second bus cycle. This 2-bit combination of S_3 and S_4 indicate the segment register used for physical address generation and is shown in Table 12.2

Table 12.2: Identifying the segment register used for 20-bit physical address generation

S_4	S_3	Segment Register
0	0	Extra
0	1	Stack
1	0	Code/none
1	1	Data

The two status codes are output both in the maximum and minimum mode.

14. What is the maximum size of the memory that can be accessed by 8086?

Ans. The two status codes S_4 and S_3 together point to the segment register used for 20-bit physical address generation and can be examined by external circuitry to enable separate 1 MB address space for each of CS, ES, DS, and SS. This would enable memory address to be expanded to a maximum of 4MB for 8086 μP.

15. Draw the Read and Write bus cycles for 8086 μP in Minimum mode.

Ans. Fig. 12.7 shows the Read and Write bus cycles for 8086 μP in the Minimum mode.

The bus cycle consists of 4T states. ALE signal stays high for T_1 state at the end of which it goes low which is utilised by latches to latch the address. Hence, during $T_2 - T_4$ states, $AD_{15} - AD_0$ lines act as data lines. The M/$\overline{IO}$, $\overline{RD}$ and $\overline{WR}$ signals can be combined to generate individual $\overline{IOR}$, $\overline{IOW}$ and $\overline{MEMR}$, $\overline{MEMW}$ signals.

The Read and Write cycles show that data are made available during T_3 and T_2 states respectively.

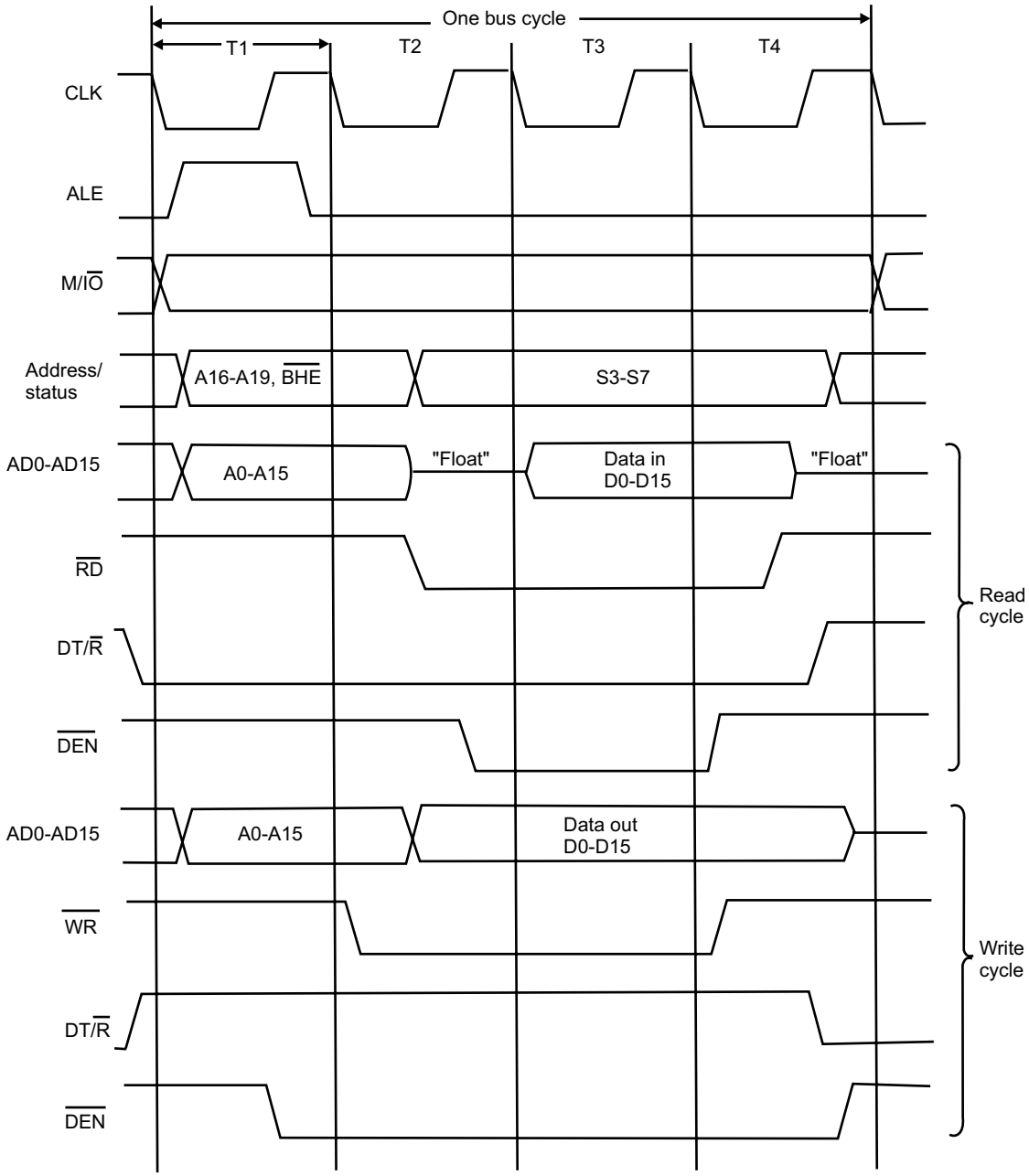


Fig.12.7: 8086 microprocessor read and write bus cycles.The address lines are valid during the T1 state but become the data lines and status indicators during T2–T4

13

Addressing Modes of 8086

1. What is meant by addressing mode?

Ans. An instruction consists of an opcode and an operand. The operand may reside in the accumulator, or in a general purpose register or in a memory location.

The manner in which an operand is specified (or referred to) in an instruction is called addressing mode.

2. Name the different addressing modes of 8086.

Ans. The following are the different addressing modes of 8086:

- Register operand addressing.
- Immediate operand addressing.
- Memory operand addressing.

3. Mention the different memory addressing modes.

Ans. The different memory addressing modes are:

- Direct Addressing
- Register Indirect Addressing
- Based Addressing
- Indexed Addressing
- Based Indexed Addressing and
- Based Indexed with displacement.

4. How the physical address is generated for the different memory addressing modes?

Ans. Physical address (for the operand) is the address from which either a read or write operation is initiated. The following shows the manner of generation of physical address.

Physical address = Segment base : Effective address

= Segment base : Base + Index + Displacement

$$= \begin{Bmatrix} CS \\ SS \\ DS \\ ES \end{Bmatrix} : \begin{Bmatrix} BX \\ BP \end{Bmatrix} + \begin{Bmatrix} SI \\ DI \end{Bmatrix} + \begin{Bmatrix} \text{8-bit Displacement} \\ \text{Or} \\ \text{16-bit Displacement} \end{Bmatrix}$$

i.e., effective address, for its generation, can have as many as three elements: Base, Index and Displacement. Thus the effective address is generated from the following:
Effective address = Base + Index + Displacement

The segment registers can be CS, SS, DS or ES. Base can be BX or PB. Index can be SI or DI and the Displacement can either be 8-bit or 16-bit.

It should be noted that not all the three elements viz., base, index or displacement are always used for effective address calculations.

5. Give examples each of (a) Register Addressing mode (b) Immediate Addressing mode.

Ans. (a) Register Addressing Mode.

In this mode, either an 8-bit or a 16-bit general purpose register contains the operand. Some examples are:

```
MOV AX, BX
MOV CX, DX
ADD AL, DH
ADD DX, CX
```

The content of BX register is moved to AX register in the first example, while in the third example, content of DH is added to the content of AL.

Here, source and destination of data are CPU registers.

(b) Immediate Addressing Mode.

In this mode, the operand is contained in the instruction itself, i.e., the operand forms a part of the instruction itself. The operand can be either 8-bit or 16-bit in length. Some examples are:

```
MOV AL, 83 H
ADD AX, 1284 H
```

In the first example, 83 H is moved to AL register while in the second example, 1284 H is added to the contents of AX register. Here, source of data is within the instruction.

6. Discuss the Direct Addressing Mode.

Ans. In a way it is similar to 'Immediate Addressing Mode'. In Immediate Addressing Mode, data follows the instruction opcode, while in this case an effective address follows the same. Thus in this case:

PA = Segment base : Direct address.

By default, the segment base register is DS. Thus,

PA = DS : EA

But if a segment override prefix (SEG) is used in the instruction, then any one of the four segment registers can be referenced. Hence, in general,

$$PA = \begin{Bmatrix} CS \\ DS \\ SS \\ ES \end{Bmatrix} : \text{Direct Address}$$

As an example, MOV CX, [ALPHA]

It means, "move the contents of the memory location, which is labelled as ALPHA in the current data segment, into register CX".

Thus, if DS = 0300 H, and value assigned to ALPHA is 3216 H, then

$$PA = 03000\ H + 3216\ H$$
$$= 06216\ H$$

Thus, data contained in address locations 06217 H and 06216 H will be stored in CH and CL registers respectively.

MOV [0404 H], CX would move the contents of CL to offset address 0404 H (relative to data segment register DS) and CH to 0405 H. Here, memory address is supplied within the instruction.

7. Discuss Register Indirect Addressing Mode.

Ans. In a way, this mode of addressing is similar to direct addressing mode in the sense that content of DS is combined with the effective address to get the physical address.

But the difference lies in the manner in which the offset is specified. In direct addressing mode EA is constant while in this mode EA is a variable.

EA can reside in either base register (BX or BP) or index register (SI or DI). The default segment register is DS, but again by using a segment override prefix (SEG), any of the four segment registers can be referenced.

Thus, PA can be computed as:

$$PA = \begin{Bmatrix} CS \\ DS \\ SS \\ ES \end{Bmatrix} : \begin{Bmatrix} BX \\ BP \\ SI \\ DI \end{Bmatrix}$$

An instruction of this mode of addressing is:

MOV CX, [SI]

Execution of this instruction entails moving the content of the memory location having its offset value in SI from the beginning of the current data segment to the CX register.

If DS = 0300 H and SI = 3216 H, then PA becomes

PA = 03000 H + 3216 H = 06216 H.

Thus data contained in 06217 H and 06216 H will be placed in CH and CL registers respectively. Here, memory address is supplied in an index or pointer register.

8. Discuss Based Addressing Mode.

Ans. The physical address in this case is generated as follows:

PA = Segment base: Base + Displacement

$$PA = \begin{Bmatrix} CS \\ DS \\ SS \\ ES \end{Bmatrix} : \begin{Bmatrix} BX \\ BP \end{Bmatrix} + \begin{Bmatrix} 8\text{-bit Displacement} \\ \text{or} \\ 16\text{-bit Displacement} \end{Bmatrix}$$

i.e., the physical address is generated by adding either an 8-bit or 16-bit displacement to the contents of either base register BX or base pointer register BP and the current value in DS or SS respectively.

Fig. 13.1 shows the utility of using either BX/BP and displacement. The figure shows a data structure starting from 'Element 0' to 'Element n'. Inserting a zero value for displacement would ensure accessing 'Element 0' of the structure. For accessing different elements within the same data structure, all that is to be done is to change the value of displacement.

Whereas, to access the same element is another data structure the value of the base register has to be changed, keeping the value of displacement same as before.

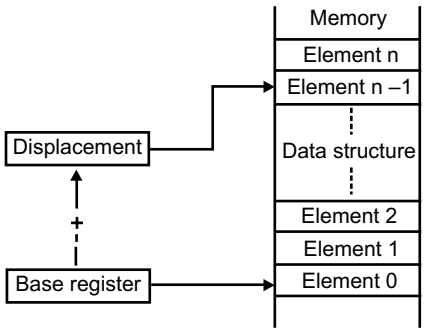


Fig.13.1: Based addressing mode of a structure of data

An example of this mode of addressing is MOV [BX] + ALPHA, AH

Here, ALPHA denotes displacement (which can be 8 or 16-bits) and BX the base register. Together, they give EA of the destination operand.

If DS = 3000 H, BX = 1234 H and displacement (16-bit) = 0012 H,

Then, PA = 03000 H + 1234 H + 0012 H = 04246 H

Thus the content of AH (source operand) is placed in the physical address 04246 H (destination operand memory location).

In this mode also, the default register i.e., DS can be changed by a segment override prefix (SEG). Also for accessing data from the stack segment of the memory, BP is to be used instead of BX.

Here, the memory address is the sum of BX or BP base registers plus an 8 or 16-bit displacement specified in the instruction.

9. Discuss Indexed Addressing Mode.

Ans. In a way, this mode of addressing is similar to the based addressing mode but the jobs carried out by base register and displacement in the based addressing mode are done by displacement and index register respectively in indexed addressing mode and shown in Fig. 13.2.

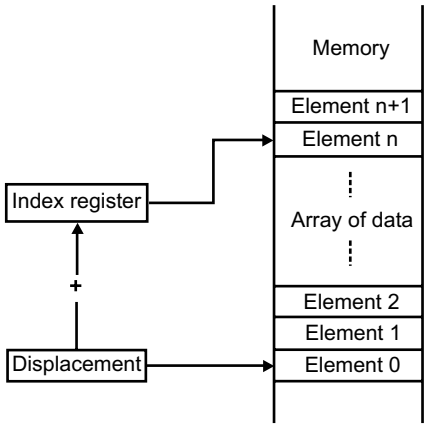


Fig.13.2: Indexed addressing mode of an array of data elements

The physical address in this case is generated as follows:
PA = Segment Base : Index + Displacement

$$PA = \begin{Bmatrix} CS \\ DS \\ SS \\ ES \end{Bmatrix} : + \begin{Bmatrix} SI \\ DI \end{Bmatrix} + \begin{Bmatrix} 8\text{-bit Displacement} \\ \text{Or} \\ 16\text{-bit Displacement} \end{Bmatrix}$$

An example of this mode of addressing is:
MOV [SI] + ALPHA, AH
where, ALPHA represents displacement.
Assuming, DS = 3000 H, SI = 1234 H and ALPHA (displacement) = 0012 H.
Thus, PA = 03000 H + 1234 H + 0012 H = 04246 H.
Thus, the data value residing in source operand AH will be moved to the physical address location 04246 H. Here, memory address is the sum of the index register plus an 8 or 16-bit displacement specified in the instruction.

10. Discuss Based Indexed Addressing mode.

Ans. It is a combination of based and indexed addressing modes. The physical address is generated in this case in the following manner:

PA = Segment Base : Base + Index

$$= \begin{Bmatrix} CS \\ DS \\ ES \\ SS \end{Bmatrix} : \begin{Bmatrix} BX \\ BP \end{Bmatrix} + \begin{Bmatrix} SI \\ DI \end{Bmatrix}$$

Here, BX and BP registers are used for data and stack segments respectively.
An example of this mode of addressing is as follows:

MOV AL, [BX] [SI]

If DS = 3000 H, BX = 1000 H and SI = 1234 H, then

PA = 03000 H + 1000 H + 1234 H = 05234 H

On executing this instruction, the value stored in memory location 05234 H will be stored in AL.

Here, memory address is the sum of an index register and a base register.

11. Discuss Based Indexed with displacement Addressing Mode.

Ans. It is a combination of based addressing mode and indexed addressing mode along with an 8 or 16-bit displacement. The physical address is generated in the following manner:

PA = Segment base : Base + Index + Displacement

$$PA = \begin{Bmatrix} CS \\ DS \\ ES \\ SS \end{Bmatrix} : \begin{Bmatrix} BX \\ BP \end{Bmatrix} + \begin{Bmatrix} SI \\ DI \end{Bmatrix} + \begin{Bmatrix} \text{8-bit displacement} \\ \text{or} \\ \text{16-bit displacement} \end{Bmatrix}$$

This addressing mode is used to access a two dimensional (m × n) array, as shown in Fig. 13.3. The displacement, having a fixed value, locates the starting position of the array in the memory while the base register specifies one coordinate (say m) and index register the other coordinate (say n). Any position in the array can be located simply by changing the values in the base and index registers.

An example of this mode of addressing is as follows:

MOV AL, [BX] [SI] + ALPHA

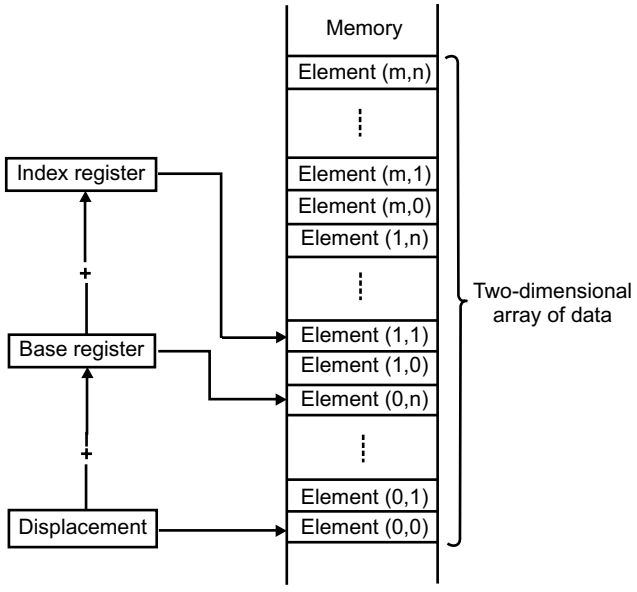


Fig.13.3: Based indexed with displacement addressing mode of a two-dimensional array of data

Thus the offset or effective address can be calculated from the contents of base and index registers and the fixed displacement as represented by ALPHA.

Assuming: DS = 3000 H, BX = 1000 H, SI = 1234 H and ALPHA (displacement) = 0012 H.

Thus, PA = 03000 H + 1000 H + 1234 H + 0012 H = 05246 H.

On executing this instruction, the value stored in memory location 05246 H (source operand) will be stored in AL. Here, memory address is the sum of an index register, a base register and an 8 or 16-bit displacement within the instruction.

14

The Instruction Set of 8086

1. How many instructions are there in the instruction set of 8086?

Ans. There are 117 basic instructions in the instruction set of 8086.

2. Do 8086 and 8088 have the same instruction set?

Ans. Yes, both 8086 and 8088 have the same instruction set.

3. Mention the groups in which the instruction set of 8086 can be categorised.

Ans. The instruction set of 8086 can be divided into the following number of groups, namely:

- Data transfer instructions
- Arithmetic instructions
- Logic instructions
- Shift instructions
- Rotate instructions
- Flag control instructions
- Compare instructions
- Jump instructions
- Subroutines and subroutine handling instructions
- Loop and loop handling instructions
- Strings and string handling instructions.

4. Mention the different types of data transfer instructions.

Ans. The different types include:

- Move byte or word instructions (MOV)
- Exchange byte or word instruction (XCHG)
- Translate byte instructions (XLAT)
- Load effective address instruction (LDA)
- Load data segment instruction (LDS)
- Load extra segment instruction (LES)

5. Can the MOV instruction transfer data directly between a source and destination that both reside in external memory?

Ans. No, it cannot. With the first MOV instruction, data from the source memory is to be moved into an internal register-normally accumulator. The second MOV instruction places the accumulator content into the destination memory.

6. Explain the following two examples:
(a) MOV CX, CS
(b) MOV AX, [ALPHA]

Ans. **(a) MOV CX, CS:** It stands for, "move the contents of CS into CX". If CS contains 1234 H, then on execution of this instruction, content of CX would become 1234 H i.e., content of CH = 12 H and Content CL = 34 H.

(b) MOV AX, [ALPHA]: Let, data segment register DS contains 0300 H and ALPHA corresponds to a displacement of 1234 H. Then the instruction stands for, "move the content of the memory location offset by 1234 H from the starting location 0300 H of the current data segment into accumulator AX". The physical address is

PA = 03000 H + 1234 H = 04234 H

Thus execution of the instruction results in content of memory location 04234 H is moved to AL and content of memory location 04235 H is moved to AH.

7. Show the forms of XCHG instruction and its allowed operands.

Ans. These are shown below in Fig. 14.1

Mnemonic	Meaning	Format	Operation	Flags affected
XCHG	Exchange	XCHG D,S	(D)↔(S)	None

(a)

Destination	Source
Accumulator Memory Register	Reg16 Register Register

(b)

Fig.14.1: (a) Exchange data transfer instruction (b) Allowed operands

8. Explain the instruction XCHG BX, CX.

Ans. The execution of this instruction interchanges the contents of BX and CX, i.e., original content of CX moves over to BX and original content of BX moves over to CX.

9. Explain XLAT instruction.

Ans. The translate (XLAT) data transfer instruction is shown in Fig.14.2. It can be used for say an ASCII to EBCDIC code conversion.

Mnemonic	Meaning	Format	Operation	Flags affected
XLAT	Translate	XLAT	((AL)+(BX)+(DS)O)→(AL)	None

Fig.14.2: Translate data transfer instruction

The content of BX represents the offset of the starting address of the look up table from the beginning of the current data segment while the content of AL represents the offset of the element which is to be accessed from the beginning of the look up table.

As an example, let DS = 0300 H, BX = 1234 H and AL = 05 H.

Hence, PA = 03000 H + 1234 H + 05 H = 04239 H

Thus, execution of XLAT would put the content of 04239 H into AL register.

Conceptually, the content of 04239 H in EBCDIC should be the same as the ASCII character equivalent of 05 H.

10. Explain the instruction LEA, LDS and LES.

Ans. These three instructions are explained in Fig.14.3. These instructions stand for load register with effective address (LEA), load register and data segment register (LDS) and load register and extra segment register (LES) respectively.

Mnemonic	Meaning	Format	Operation	Flags affected
LEA	Load effective address	LEA Reg 16, EA	(EA)→(Reg16)	None
LDS	Load register and DS	LDS Reg 16, Mem 32	(Mem 32)→(Reg16)	
			(Mem 32)→(Reg16)	
			(Mem 32+2)→(ES)	None
			(Mem 32+2)→(Reg 16)	
LES	Load register and ES	LES Reg 16, Mem 32	(Mem 32+2)→(ES)	None

Fig.14.3: LEA, LDS and LES data transfer instructions

LEA instruction loads a specified register with a 16-bit offset value. LDS and LES instructions load the specified register as well as DS or ES segment register respectively. Thus a new data segment will be activated by a single execution.

11. Indicate the different types of arithmetic instructions possible with 8086.

Ans. The different arithmetic instructions are addition, subtraction, multiplication and division and are shown in Fig.14.4.

	Addition
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
	Subtraction
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
	Multiplication
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to double word

Fig.14.4: Arithmetic instructions

12. Show the allowed operands for the instruction ADD, ADC and INC.

Ans. The allowed operands for ADD and ADC are shown in Fig.14.5 (a) and for INC it is shown in Fig.14.5 (b).

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

(a)

Destination
Reg 16
Reg 8
Memory

(b)

Fig.14.5: (a) Allowed operands for ADD and ADC
(b) Allowed operands for INC

13. Show the different subtraction arithmetic instructions. Also show the allowed operands for (a) SUB and SBB (b) DEC and (c) NEG instructions.

Ans. The different subtraction arithmetic instructions and the allowed operands, for the different instructions are shown in Fig. 14.6 (a), (b), (c) and (d) respectively.

Mnemonic	Meaning	Format	Operation	Flags affected
SUB	Subtract	SUB D,S	(D)–(S)→(D) Borrow→(CF)	OF, SF, ZF, AF, PF, CF
SBB	Subtract with borrow	SBB D,S	(D)–(S)–(CF)→(D)	OF, SF, ZF, AF, PF, CF
DEC	Decrement by 1	DEC D	(D)–1→(D)	OF, SF, ZF, AF, PF
NEG	Negate	NEG D	0–(D)→(D) 1→(CF)	OF, SF, ZF, AF, PF, CF
DAS	Decimal adjust for subtraction	DAS		SF, ZF, AF, PF, CF OF undefined
AAS	AASCII adjust for subtraction	AAS		AF, CF, OF, SF, ZF, PF undefined

(a)

Destination	Source
Register	Register
Register	Memory
Memory	Register
Accumulator	Immediate
Register	Immediate
Memory	Immediate

(b)

Destination
Reg16
Reg 8
Memory

(c)

Destination
Register
Memory

(d)

Fig.14.6: (a) Subtraction arithmetic operations
(b) Allowed operands for SUB and SBB instructions
(c) Allowed operands for DEC instruction
(d) Allowed operands for NEG instruction

14. Show the different multiplication and division instructions and also the allowed operands.

Ans. The different multiplication and division instructions and also the allowed operands are shown in Fig.14.7 (a) and (b) respectively.

Mnemonic	Meaning	Format	Operation	Flags affected
MUL	Multiply (unsigned)	MUL S	(AL).(S8)→(AX) (AX)·(S16)→(DX).(AX)	OF, CF, SF, ZF, AF, PF, undefined
DIV	Division (unsigned)	DIV S	(1) Q((AX)/(S8))→(AL) R((AX)/(S8))→(AH) (2) Q((DX, AX)/(S16))→(AX) R((DX,AX)/(S16))→(DX) If Q is FF_{16} in case (1) or $FFFF_{16}$ in case (2), then type 0 interrupt occurs.	OF, SF, ZF, PF, CF undefined
IMUL	Integer multiply (signed)	IMUL S	(AL)·(S8)→(AX) (AX)·(S16)→(DX),(AX)	OF, CF, SF, ZF, AF, PF undefined
IDIV	Integer divide (signed)	IDIV S	(1) Q((AX)/(S8))→(AX) R((AX)/(S8))→(AH) (2) Q((DX,AX)/(S16))→(AX) R((DX,AX)(S16))→(DX) If Q is positive and exceeds $7FFF_{16}$, or if Q is negative and becomes less than 8001_{16}, then type 0 interrupt occurs	OF, SF, ZF, AF, PF, CF undefined
AAM	Adjust AL for multiplication	AAM	Q((AL)/10))→AH R((AL)/10))→AL	SF, ZF, PF, OF, AF, CF undefined
AAD	Adjust AX for division	AAD	(AH) · 10 + AL → AL 00 → AH	SF, ZF, PF, OF, AF, CF undefined
CBW	Convert byte to word	CWD	(MSB of AL)→(All bits of AH)	None
CWD	Convert word to double word	CWD	(MSB of AX)→(All bits of DX)	None

(a)

Source
Reg 8
Reg 16
Mem 8
Mem 16

(b)

Fig.14.7: (a) Multiplication and division instructions (b) Allowed operands.

15. Show the different logic instructions and also the allowed operands for (a) AND, OR and XOR (b) NOT instructions.

Ans. The different logic instructions as also the allowed operands for different instructions are shown in Fig.14.8 (a), (b) and (c) respectively.

Mnemonic	Meaning	Format	Operation	Flags affected
AND	Logical AND	AND D, S	(S) . (D) → (D)	OF, SF, ZF, PF, CF, AF undefined
OR	Logical Inclusive —OR	OR D, S	(S) + (D) → (D)	OF, SF, ZF, PF, CF, AF undefined
XOR	Logical Exclusive —OR	XOR D, S	(S) ⊕ (D) → (D)	OF, SF, ZF, PF, CF, AF undefined
NOT	Logical NOT	NOT D	$(\overline{D})$ → (D)	None

(a)

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

(b)

Destination
Register
Memory

(c)

Fig. 14.8: (a) Logic instructions
(b) Allowed operands for the AND, OR and XOR instructions
(c) Allowed operands for NOT instruction.

16. What are the two basic shift operations?

Ans. The two basic shift operations are logical shift and arithmetic shift.

The two logical shifts are shift logical left (SHL) and shift logical right (SHR), while the two arithmetic shifts are shift arithmetic left (SAL) and shift arithmetic right (SAR).

17. Show the different shift instructions and the allowed operands.

Ans. The various shift instructions and the allowed operands are shown in Fig. 14.9 (a) and (b) respectively.

Mnemonic	Meaning	Format	Operation	Flags affected
SAL/SHL	Shift arithmetic left/shift logic left	SAL/SHL D, Count	Shift the (D) left by the number of bit positions equal to Count and fill the vacated bit positions on the right with zeros.	CF, PF, SF, ZF, OF AF undefined OF undefined if count ≠ 1
SHR	Shift logical right	SHR D, Count	Shift the (D) right by the number of bit positions equal to Count and fill the vacated bits positions on the left with zeros.	CF, PF, SF, ZF, OF AF undefined OF undefined if count ≠ 1
SAR	Shift arithmetic right	SAR D, Count	Shift the (D) right by the number of bit positions equal to Count and fill the vacated bits positions on the left with original most significant bit.	OF, SF, ZF, CF, PF AF, undefined

(a)

Destination	Count
Register	1
Register	CL
Memory	1
Memory	CL

(b)

Fig.14.9: (a) Shift instructions, (b) Allowed operands

18. Show the different Rotate instructions and the allowed operands.

Ans. The different Rotate instructions and the allowed operands are shown in Fig. 14.10 (a) and (b) respectively.

Mnemonic	Meaning	Format	Operation	Flags affected
ROL	Rotate left	ROL D, Count	Rotate the (D) left by the number of bit positions equal to Count. Each bit shifted out from the left most bit goes back into the rightmost bit position.	CF OF underfined if count ≠ 1
ROR	Rotate right	ROR D, Count	Rotate the (D) right by the number of bit positions equal to Count. Each bit shifted out from the rightmost bit goes into the leftmost bit position.	CF OF undefined if count ≠ 1
RCL	Rotate left through carry	RCL D, Count	Same as ROL except carry is attached to (D) for rotation.	CF OF undefined if count ≠ 1
RCR	Rotate right through carry	RCR D, Count	Same as ROR except carry is attached to (D) for rotation.	CF OF undefined if count ≠ 1

(a)

Destination	Count
Register	1
Register	CL
Memory	1
Memory	CL

(b)

Fig. 14.10: (a) Rotate instructions (b) Allowed operands

19. Name the different flags control instructions, the operations performed by them and also the flags affected.

Ans. Fig. 14.11 shows the different flags control instructions, their meaning and the flags affected by respective instructions.

Mnemonic	Meaning	Operation	Flags affected
LAHF	Load AH from flags	(AH) ← (Flags)	None
SAHF	Store AH into flags	(Flags) ← (AH)	SF, ZF, AF, PF, CF
CLC	Clear carry flag	(CF) ← 0	CF
STC	Set carry flag	(CF) ← 1	CF
CMC	Complement carry flag	(CF) ← $\overline{(CF)}$	CF
CLI	Clear interrupt flag	(IF) ← 0	IF
STI	Set interrupt flag	(IF) ← 1	IF

Fig. 14.11: Flag control instructions

20. Which register plays an important part in flag control instructions?

Ans. It is the accumulator register AH which plays an important part in flag control instructions.

For instance, if the present values in the flags are to be saved in some memory location then they are first to be loaded in the AH register and transfered to memory location, say M1, i.e.,

LAHF → Load AH from flags

MOV M1, AH → Move contents of AH into memory location M1.

As a second example, if the content of memory location, Say M2, is to be placed into flags, then

MOV AH, M2 → Move contents of memory location M2 into AH register.

SAHF → Store AH into flags.

21. List the characteristics of CMP instructions.

Ans. The following are the characteristics of CMP instruction:

- Can compare two 8-bit or two 16-bit numbers.
- Operands many reside in memory, a register in the CPU or be a part of an instruction.
- Results of comparison is reflected in the status of the six status flags—CF, AF, OF, PF, SF and ZF.
- CMP is a subtraction method—it uses 2's complement for this.
- Result of CMP is not saved—but based on CMP result, appropriate flags are either set/reset.

22. Explain CMP instruction.

Ans. The compare instruction, different operand combinations and the flags affected are shown in Fig. 14.12.

Mnemonic	Meaning	Format	Operation	Flags affected
CMP	Compare	CMP D, S	(D) – (S) is used in setting or resetting the flags	CF, AF, OF, PF, SF, ZF

(a)

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

(b)

Fig.14.12: (a) Compare instruction (b) Operand combination

23. What are the two basic types of unconditional jumps? Explain.

Ans. The two basic types of unconditional jumps are intrasegment jump and intersegment jump.

The intrasegment jump is a jump for which the addresses must lie within the current code segment. It is achieved by only modifying the value of IP.

The intersegment jump is a jump from one code segment to another. For this jump to be effective, both CS and IP values are to be modified.

24. Show the unconditional jump instructions and the allowed operands.

Ans. The unconditional jump instruction, along with the allowed operands are shown in Fig. 14.13 (a) and (b) respectively.

Mnemonic	Meaning	Format	Operation	Affected Flags
JMP	Unconditional Jump	JMP Operand	Jump is initiated to the address specified by the operand	None

(a)

Operands
Short-label
Near-label
Far-label
Memptr 16
Regptr 16
Memptr 32

(b)

Fig.14.13: (a) Unconditional jump instruction (b) Allowed operands

Jump instructions carried out with a Short-label, Near-label, Memptr 16 or Regptr 16 type of operands represent intrasegment jumps, while Far-label and Memptr 32 represent intersegment jumps.

25. Distinguish between Short-label and Near-label jump instructions.

Ans. The distinction between the two is shown in a tabular form.

Short-label	Near-label
1. It specifies the jump relative to the address of the jump instruction itself. 2. Specifies a new value of IP with an 8-bit immediate operand. 3. Can be used only in the range of –126 to + 129 bytes from the location of the jump instruction.	1. It specifies the jump relative to the address of the jump instruction itself. 2. Specifies a new value of IP with a 16-bit immediate operand. 3. Can be used to cover the complete range of current code segment.

26. Describe the Memptr 16 and Regptr 16 jump instructions.

Ans. Both these types permit a jump to any location (address) in the current code segment. Again the contents of a memory or register indirectly specifies the address of the jump, as the case may be.

27. Show the conditional jump instruction and their different types.

Ans. The conditional jump instruction and their different types are shown in Fig. 14.14 (a) and (b) respectively.

Mnemonic	Meaning	Format	Operation	Flags affected
JCC	Conditional Jump	JCC Operand	If the specified condition CC is true the jump to the address specified by the operand is initiated; otherwise the next instruction is executed	None

(a)

Mnemonic	*Meaning*	*Condition*	*Mnemonic*	*Meaning*	*Condition*
JA	above	CF = 0 and ZF = 0	JAE	above or equal	CF = 0
JB	below	CF = 1	JBE	below or equal	CF = 1 or ZF = 1
JC	carry	CF = 1	JCXZ	CX register is zero	(CF or ZF) = 0
JE	equal	ZF = 1	JG	greater	ZF = 0 and SF = OF
JGE	greater or equal	SF = OF	JL	less	(SF xor OF) = 1
JLE	less or equal	((SF xor OF) or ZF) = 1	JNA	not above	CF = 1 or ZF = 1
JNAE	not above nor equal	CF = 1	JNB	not below	CF = 0
JNBE	not below nor equal	CF = 0 and ZF = 0	JNC	not carry	CF = 0
JNE	not equal	ZF = 0	JNG	not greater	((SF xor OF) or ZF) = 1
JNGE	not greater nor equal	(SF xor OF) = 1	JNL	not less	SF = OF
JNLE	not less nor equal	ZF = 0 and SF = OF	JNO	not overflow	OF = 0
JNP	not parity	PF = 0	JNS	not sign	SF = 0
JNZ	not zero	ZF = 0	JO	overflow	OF = 1
JP	parity	PF = 1	JPE	parity even	PF = 1
JPO	parity odd	PF = 0	JS	sign	SF = 1
JZ	zero	ZF = 1			

(b)

Fig. 14.14: (a) Conditional jump instruction (b) Types of conditional jump instructions

28. Show the subroutine CALL instruction and the allowed operands.

Ans. The CALL instruction and the allowed operands are shown in Fig.14.15(a) and (b) respectively.

Mnemonic	Meaning	Format	Operation	Flags affected
CALL	Subroutine call	CALL operand	Execution continues from the address of the subroutine specified by the operand. Information required to return back to the main program such as IP and CS are saved on the stack.	None

(a)

Operands
Near-proc
Far-proc
Memptr 16
Regptr 16
Memptr 32

(b)

Fig.14.15: (a) Subroutine call instruction (b) Allowed operand

29. What are the two types of CALL instructions? Discuss.

Ans. The two types are: intrasegment CALL and intersegment CALL.
If the operands are Near-proc, Memptr16 and Regptr16, then they specify intrasegment CALL while Far-proc and Memptr32 represent intersegment CALL.

30. Show the PUSH and POP instructions, as also the allowed operands.

Ans. The PUSH and POP instructions, as also the allowed operands are shown in Fig.14.16 (a) and (b) respectively.

Mnemonic	Meaning	Format	Operation	Flags affected
PUSH	Push word onto stack	PUSH S	((SP))←(S)	None
POP	Pop word off stack	POP D	(D)←((SP))	None

(a)

Operand (S or D)
Register
Seg-reg (CS illegal)
Memory

(b)

Fig.14.16: (a) PUSH and POP instructions (b) Allowed operands

31. How many loop instructions are there and state their use.

Ans. There are in all three loop instructions. They can be used in place of certains conditional jump instructions. These instructions give the programmer a certain amount of flexibility in writing programs in a simpler manner.

32. List the different instructions and also the operations they perform.

Ans. The different loop instructions and the operations they perform are shown in Fig. 14.17.

33. What is meant by a 'string' and what are the characteristics of a string instruction?

Ans. A string is a series of data words (or bytes) that reside in successive memory locations. The characteristics of a string instruction are:

- Can move data from one block of memory locations to another one.
- A string of data elements stored in memory can be scanned for a specific data value. Successive elements of two strings can be compared to determine whether the two strings are same/different.

34. List the basic string instructions and the operations they perform.

Ans. The basic string instructions and the operations they perform are shown in Fig. 14.18.

Mnemonic	Meaning	Format	Operation
LOOP	Loop	LOOP Short-label	(CX)←(CX) – 1 Jump is initiated to location defined by short-label if (CX) ≠0; otherwise, execute next sequential instruction.
LOOPE/LOOPZ	Loop while equal/loop while zero	LOOPE/LOOPZ Short-label	(CX)←(CX) – 1 Jump to the location by short-label if (CX)≠0 and (ZF) = 1; otherwise execute next sequential instruction.
LOOPNE/ LOOPNZ	Loop while not equal/ loop while not zero	LOOPNE/LOOPNZ Short-label	(CX)←(CX) – 1 Jump to the location defined by short label if (CX)≠0 and (ZF) = 0; otherwise execute next sequential instruction

Fig. 14.17: LOOP instructions

Mnemonic	Meaning	Format	Operation	Flags Affected
MOVS	Move string	MOVS Operand	((ES)0+(DI))←((DS)0+(SI)) (SI)←(SI) ± 1 or 2 (DI)←(DI) ± 1 or 2	None
MOVSB	Move string byte	MOVSB	((ES)0+(DI))←((DS)0+(SI)) (SI)←(SI) ± 1 (DI)←(DI) ± 1	None

(*Contd...*)

MOVSW	Move string word	MOVSW	((ES)0+(DI))←((DS)0+(SI)) ((ES)0+(DI)+1)←((DS)0+(SI)+1) (SI)←(SI) ± 2 (DI)←(DI) ± 2	None
CMPS	Compare string	CMPS Operand	Set flags as per ((DS)0+(SI))–((ES)0+(DI)) (SI)←(SI) ± 1 or 2 (DI)←(DI) ± 1 or 2	CF, PF, AF, ZF, SF, OF
SCAS	Scan string	SCAS Operand	Set flags as per (AL or AX) – ((ES)0+(DI)) (DI)←(DI) ± 1 or 2	CF, PF, AF, ZF, SF, OF
LODS	Load string	LODS Operand	(AL or AX)←((DS)0+((SI)) (SI)←(SI) ± 1 or 2	None
STOS	Store string	STOS Operand	((ES)0+(DI))←(AL or AX)± 1 or 2 (DI)←(DI) ± 1 or 2	None

Fig.14.18: Basic string instructions

35. What is a 'REP' instruction? Discuss.

Ans. 'REP' stands for repeat and is used for repeating basic string operations—required for processing arrays of data.

There are a number of repeat instructions available and are used as a prefix in string instructions. The prefixes for use with the basic string instructions are shown in Fig. 14.19.

Prefix	Used with	Meaning
REP	MOVS STOS	Repeat while not end of string CX ≠ 0
REPE/REPZ	CMPS SCAS	Repeat while not end of string and strings are equal CX ≠ 0 and ZF = 1
REPNE/REPNZ	CMPS SCAS	Repeat while not end of string and strings are not equal CX ≠ 0 and ZF = 0

Fig.14.19: Prefixes for use with the basic string operations

36. Discuss the instructions for Autoindexing of string instructions.

Ans. When the system executes some string instruction, the addresses residing in DI and SI are incremented/decremented automatically. The content of direction flag (DF) decides the above. Two instructions—CLD (clear DF flag) or STD (Set DF flag) are used for the above and shown in Fig. 14.20.

Mnemonic	Meaning	Format	Operation	Flags affected
CLD	Clear DF	CLD	(DF)←0	DF
STD	Set DF	STD	(DF)←1	DF

Fig. 14.20: Instructions for autoincrementing and autodecrementing in string instructions

Execution of CLD (this makes DF = 0) permits autoincrement mode while execution of STD (this makes DF = 1) permits autodecrement mode.

SI or DI are autoincremented/autodecremented by one if a byte of data is processed or by two if a word of data is processed.

37. Write down the equivalent string instructions for the following two.

(i)
```
MOV AL, [DI]
CMP AL, [SI]
DEC SI
DEC DI
```

(ii)
```
MOV AL, [SI]
MOV [DI], AL
INC SI
INC DI
```

Ans. The following two are the equivalent string instructions of the given ones:

(i)
```
STD
CMP SB
```

(ii)
```
CLD
MOV SB
```

38. Where do memory source and destination addresses reside in string instructions?

Ans. The memory source and destination addresses in such cases are register SI in the data segment and DI in the extra segment.

15

Programming Techniques

1. Write an ALP (assembly language programming) for addition of two 8-bit data BB H and 11 H.

Ans.
```
0200  MOV AL, BB H        : 8-bit data BB H into AL
0202  MOV CL, 11 H        : 8-bit data 11 H into CL
0204  ADD AL, CL          : Contents of AL and CL added
0206  HLT                 : Stop.
```
Comment : Result in AL = CC H.

2. Write an ALP for addition of two 16-bit data BB11 H and 1122 H.

Ans.
```
0200  MOV AX, BB11 H      : 16-bit data BB11 H into AX
0203  MOV CX, 1122 H      : 16-bit data 1122 H into CX
0206  ADD AX, CX          : Contents of AX and CX added
0208  HLT                 : Stop
```
Comment : Result in AX = CC33 H.

3. Write an ALP for addition of two 8-bit data BB H and 11 H. The first data has an offset address of 0304 H and displacement.

Ans.
```
0200  MOV BX, 0304 H      : Offset address put in BX
0203  MOV AL, 11 H        : 8-bit data 11H into AL
0205  ADD AL, [BX + 07]   : 8-bit data from offset + displacement added with AL
0207  HLT                 : Stop.
```
Comment : Result in AL = CC H.

4. Write an ALP that subtracts 1234 H existing in DX from the word beginning at memory location MEMWDS.

Ans.
```
0200  MOV DX, 1234 H      : 16-bit data 1234 H put into DX
0203  SUB MEMWDS, DX      : Subtract data word 1234 H existing in DX from the data
                           word pointed to by MEMWDS.
0208  HLT                 : Stop.
```
Comment : If MEMWDS points to 3000 H then,

[3001 H : 3000 H] ← [3001 H : 3000 H] – 1234 H

5. Write an ALP which multiplies two 8-bit data 21 H and 17 H.

Ans.
```
0200  MOV AL, 21 H        : 8-bit multiplicand 21 H put into AL
0202  MOV CL, 17 H        : 8-bit multiplier 17 H put into CL
0204  MUL CL              : Contents of CL and AL are multiplied and the result
                           stored in AX
```

```
0206  HLT                    : Stop.
Comment : Result in AX = 02F7 H.
```

6. Write an ALP for dividing 1234 H by 34 H.

Ans.
```
0200  MOV AX, 1234 H         : 16-bit dividend in 1234 H
0203  MOV CL, 34 H           : 8-bit divisor in 34 H
0205  DIV CL                 : Content of AX divided by content of CL
0207  HLT                    : Stop.
Comment: Result in AX with
            Quotient in AL = 59 H and
            Remainder in AH = 20 H.
```

7. Write ALP that saves the contents of 8086's flags in memory location having an offset 1212 H and then to reload the flags from the contents of the memory location having an offset 2121 H.

Ans.
```
0200  LAHF                   : Load AH from flags
0201  MOV [1212], A H        : Move the contents of AH to memory locations pointed to
                               by offset 1212 H
0205  MOV AH, [2121]         : Move the contents of memory locations pointed to offset
                               2121 H to AH
0209  SAHF                   : Store AH into flags
020A  HLT                    : Stop.
```

8. Write an ALP that transfers a block of 100 bytes of data. The source and destination memory blocks start at 3000 H and 4000 H memory locations respectively. The data segment register value is DSADDR.

Ans.
```
2000  MOV AX, DSADDR         : Move initial address of DS register into AX.
2003  MOV DS, AX             : DS loaded with AX
2005  MOV SI, 3000 H         : Source address put into SI.
2008  MOV DI, 4000 H         : Destination address put into DI.
200B  MOV CX, 64 H           : Count value for number of bytes put into CX register
200D  MOV AH, [SI]           : Source byte moved into AH
200F  MOV [DI], AH           : AH byte moved into destination address
2011  INC SI                 : Increment source address
2012  INC DI                 : Increment destination address
2013  DEC CX                 : Decrement CX count
2014  JNZ 200D               : Jump to 200D H until CX = 0
2017  HLT                    : Stop.
```

9. Write an ALP for ASCII addition of two numbers 2 H and 5 H.

Ans.
```
2000  MOV AL, 32 H           : ASCII code 32 H for number 2 H is moved into AL
2002  MOV BL, 35 H           : ASCII code 35 H for number 5 H is moved into BL
2004  AAA                    : ASCII adjust for addition
2005  HLT                    : Stop.
Result : (AL) = 07 H.
```

10. Write an ALP to find the average of two numbers.

Ans.
```
2000  MOV AL, 72 H           : Get 1st number 72 H in AL
2002  ADD AL, 78 H           : Add 2nd number 78 H with 72 H (in AL)
2004  ADC AH, 00 H           : Put the carry in AH
```

```
2006  SAR AX, 1              : Divide Sum by 2
2008  MOV [3000 H], AL       : Copy AL content in memory location 3000 H
200B  HLT                    : Stop.
```

11. Write an ALP for moving a block consisting of 10 bytes from memory locations starting from 5000 H to memory locations starting from 6000 H. Use LOOP instruction.

Ans.
```
2000  CLD                    : Clear direction flag
2001  MOV SI, 4000 H         : Source address put in SI
2004  MOV DI, 5000 H         : Destination address put in DI
2007  MOV CX, 000A H         : Put number of bytes to be transferred in CX.
200A  MOV SB                 : 1 byte copied from memory addressed by SI to addressed
                               by DI.
200B  LOOPNZ 200A H          : Loop till CX = 0
200D  HLT                    : Stop.
```

12. Write an ALP to find 2's complement of a string of 100 bytes.

Ans.
```
2000  CLD                    : Clear direction flag
2001  MOV SI, 4000 H         : Source address put in SI
2004  MOV DI, 5000 H         : Destination address put in DI
2007  MOV CX, 0064 H         : Put the number of bytes to be 2's complemented in CX
200A  LODSB                  : Data byte to AL and INC SI
200B  NEGAL                  : 2's Complement of AL
200D  STOSB                  : Current AL value into DI and INC DI
200E  LOOPNZ 200A H          : Loop till CX = 0.
2010  HLT                    : Stop.
```

13. Write an ALP for block move of 100 bytes using Repeat instruction.

Ans.
```
2000  CLD                    : Clear direction flag
2001  MOV SI, 4000 H         : Source address put in SI
2004  MOV DI, 5000 H         : Destination address put in DI
2007  MOV CX, 0064 H         : Put number of bytes to be block moved into CX
200A  REPNZ                  : Repeat till CX = 0
200B  MOVSB                  : Move data byte addressed by SI to DI.
2009  HLT                    : Stop.
```

14. Write an ALP to evaluate X (Y + Z), where X = 10 H, Y = 20 H and Z = 30 H.

Ans.
```
2000  MOV AL, 20 H           : 20 H put in AL
2002  MOV CL, 30 H           : 30 H put in CL
2004  ADD AL, CL             : AL and CL are added up and result in AL
2006  MOV CL, AL             : AL transferred in CL
2008  MOV AL, 10 H           : 10 H put in AL
200A  MUL CL                 : AL and CL are multiplied and result in AL
200C  MOV SI, 4000 H         : Source address in SI
200F  MOV SI, AL             : AL put in SI
2011  HLT                    : Stop.
```

15. Write an ALP that reverses the contents of the bytes TABLE through TABLE + N – 1.

Ans.
```
2000  MOV CL, N          : Number N put in CL
2002  MOV CH, 00 H       : 00 H put in CH
2004  MOV SI, TABLE      : Starting address of TABLE put in SI
2007  MOV DI, SI         : DI loaded with SI value
2009  SUB DI, 01         : 01 subtracted from DI
200B  ADD DI, CX         : CX value added to present DI value
200D  SHR CX, 01         : Divide CX count value by 2
200F  MOV AL, [SI]       : Move data pointed to by SI into AL
2011  XCHG AL, [DI]      : Exchange AL with data pointed to by DI
2013  MOV [SI], AL       : Save the exchanged data in SI
2014  INC SI             : Increment SI
2015  DEC DI             : Decrement DI
2016  LOOP 200F H        : Continue till CX = 0
2019  HLT                : Stop.
```

16. Write an ALP to find the maximum value of a byte from a string of bytes.

Ans.
```
2000  MOV SI, 3000 H     : Source address put in SI
2003  MOV CX, 0100 H     : Count value of bytes put in CX
2006  MOV AH, 00 H       : AH initialised with 00H
2008  CMP AH, [SI]       : AH compared with data pointed to by SI
200A  JAE 200E H         : Jump if AH is ≥ (SI) to 200E H
200C  MOV AH, [SI]       : Otherwise, move (SI) to AH
200E  INC SI             : Increment SI
200F  LOOPNZ 2008        : Loop unless CX ≠ 0
2011  MOV [SI], AH       : (AH) transferred to the memory location pointed to by SI
2013  HLT                : Stop.
```

17. Write an ALP to find the minimum value of a byte from a string of bytes.

Ans.
```
2000  MOV SI, 3000 H     : Source address put in SI
2003  MOV CX, 0100 H     : Count value of bytes put in CX
2006  MOV AH, 00 H       : AH initialised with 00H
2008  CMP AH, [SI]       : AH compared with data pointed to by SI
200A  JB 200E H          : Jump if (AH) < (SI) to 200E H
200C  MOV AH, [SI]       : Otherwise move (SI) to AH
200E  INC SI             : Increment SI
200F  LOOPNZ 2008        : Loop unless CX ≠ 0
2011  MOV [SI], AH       : (AH) transferred to the memory location pointed to by SI
2013  HLT                : Stop.
```

16

Modular Program Development and Assembler Directives

1. What is modular programming?

Ans. Instead of writing a large program in a single unit, it is better to write small programs—which are parts of the large program. Such small programs are called program modules or simply modules. Each such module can be separately written, tested and debugged. Once the debugging of the small programs is over, they can be linked together. Such methodology of developing a large program by linking the modules is called modular programming.

2. What are data coupling and control coupling?

Ans. Data coupling refers to how data/information are shared between two modules while control coupling refers to how the modules are entered and exited. Coupling depends on several factors like organisation of data as also whether the modules are assembled together or separately. The modular approach should be such that data coupling be minimized while control coupling is kept as simple as possible.

3. How modular programming helps assemblers?

Ans. Modular programming helps assembly language programming in the following ways :

- Use of macros-sections of code.
- Provide for procedures—i.e., subroutines.
- Helps data structuring such that the different modules can access them.

4. What is a procedure?

Ans. The procedure (or subroutine) is a set of codes that can be branched to and returned from.

The branch to a procedure is known as CALL and the return from the procedure is known as RETURN.

The RETURN is always made to the instruction just following the CALL, irrespective of where the CALL is located.

Procedures are instrumental to modular programming, although not all modules are procedures. Procedures have one disadvantage in that an extra code is needed to link them— normally referred to as linkage.

The CALL instruction pushes IP (and CS for a far call) onto the stack. When using procedures, one must remember that every CALL must have a RET. Near calls require near returns and far calls require far returns.

5. What are the two types of procedures?

Ans. There are two types of procedures. They are:

- Those that operate on the same set of data always.
- Those that operate on a new set of data each time they are called.

6. How are procedures delimited within the source code?

Ans. A procedure is delimited within a source code by placing a statement of the form < Procedure name > Proc < attribute > at the beginning of the procedure and the statement < Procedure name > ENDP at the end.

The procedure name acts as the identifier for calling the procedure and the attribute can be either NEAR or FAR—this attribute determines the type of RET statement.

7. Explain how a procedure and data from another module can be accessed.

Ans. A large program is generally divided into separate independent modules. The object codes for these modules are then linked together to generate a linked/executable file.

The assembly language directives: PUBLIC and EXTRN are used to enable the linker to access procedure and data from different modules. The PUBLIC directive lets the linker know that the variable/procedure can be accessed from other modules while the EXTRN directive lets the assembler know that the variable/procedure is not in the existing module but has to be accessed from another module. EXTRN directive also provides the linker with some added information about the procedure. For example,

```
EXTRN ROUTINE : FAR, TOKEN : BYTE
```

indicates to the linker that ROUTINE is a FAR procedure type and that TOKEN is a variable having type byte.

8. Discuss the technique of passing parameters to a procedure.

Ans. When calling a procedure, one or more parameters need to be passed to the procedure—an example being delay parameter. This parameter passing can be done by using one of the CPU registers like,

```
MOV CX, T
CALL DELAY
```

where, T represents delay parameter.

A second technique is to use a memory location like,

```
MOV TEMP, T
CALL DELAY
```

where, TEMP is representative of memory locations.

A third technique is to pass the address of the memory variable like,

```
MOV SI, POINTER
CALL DELAY
```

while in the procedure, it extracts the delay parameter by using the instruction MOV CX, [SI].

This way an entire table of values can be passed to a procedure. The above technique has the inherent disadvantage of a register or memory location being dedicated to hold the parameter when the procedure is called. This problem becomes more prominent when using nested procedures. One alternative is to use the stack to relieve registers/memory locations being dedicated like,

```
MOV CX, T
PUSH CX
CALL DELAY
```

The procedure then can pop off the parameters, when needed.

9. Explain the term Assembler Directive.

Ans. There are certain instructions in the assembly language program which are not a part of the instruction set. These special instructions are instructions to the assembler, linker and loader and control the manner in which a program assembles and lists itself. They come into play during the assembly of a program but do not generate any executable machine code.

As such these special instructions—which, as told, are not a part of the instruction set —are called assembler directives or pseudo-operations.

10. Give a tabular form of assembler directives.

Ans. Table 16.1 gives a summary of assembler directives.

Table 16.1: Summary of assembler directives

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 words 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
EXITM	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 assembling the program

11. Explain the following assembler directives (a) CODE (b) ASSUME (c) ALIGN

Ans. (a) CODE

It provides a shortcut in the definition of the code segment. The format is Code [name]

Here, the 'name' is not mandatory but is used to distinguish between different code segments where multiple type code segments are needed in a program.

(b) ASSUME

The four physical segments viz., CS, DS, SS and ES can be directly accessed by 8086 at any given point of time. Again 8086 may contain a number of logical segments which can be assigned as physical segments by the ASSUME directive. For example

ASSUME CS: Code, DS : Data, SS : Stack

(c) ALIGN

This directive forces the assembler to align the next segment to an address that is divisible by the number that follows the ALIGN directive. The general format is

ALIGN number

where number = 2, 4, 8, 16

ALIGN 4 forces the assembler to align the next segment at an address that is divisible by 4. The assembler fills the unused byte with 0 for data and with NOP for code.

Normally, ALIGN 2 is used to start a data segment on a word boundary while ALIGN 4 is used to start a data segment on a double word boundary.

12. Explain the DATA directive.

Ans. It is a shortcut definition to data segments. The directives DB, DW, DD, DR and DT are used to (a) define different types of variables or (b) to set aside one or more storage locations in memory-depending on the data type

DB — Define Byte
DW — Define Word
DD — Define Double word
DQ — Define Quadword
DT — Define Ten Bytes

ALPHA DB, 10 H, 16 H, 24 H; Declare array if 3 bytes names; ALPHA

13. Explain the following assembler directives : (a) DUP (b) END (c) EVEN

Ans. **(a) DUP:** The directive is used to initialise several locations and to assign values to these locations. Its format is: Name Data-Type Num DUP (value)

As an Example:

```
TABLE DOB 20 DUP(0) ; Reserve an array of 20
                    ; bytes of memory and initialise all 20
                    ; bytes with 0. Array is named TABLE
```

(b) END: This directive is put in the last line of a program and indicates the assembler that this is the end of a program module. Statement, if any, put after the END directive is ignored. A carriage return is obviously required after the END directive.

(c) EVEN: This directive instructs the assembler to advance its location counter in such a manner that the next defined data item or label is aligned on an even storage boundary. It is very effectively used to access 16 or 32-bits at a time. As an example.

```
EVEN LOOKUP DW 10 DUP (0)  ; Declares the array of 10 words
                           ; starting from an even address.
```

14. Discuss the MODEL directive.

Ans. This directive selects a particular standard memory model. Each memory model is characterised by having a maximum space with regard to availability of code and data. This different models are distinguished by the manner by which subroutines and data are reached by programs.

Table 16.2 gives an idea about the different models with regard to availability of code and data.

Table16.2: The different models

Model	Code segments	Data segments
Small	One	One
Medium	Multiple	One
Compact	One	Multiple
Large	Multiple	Multiple

15. Give a typical program format using assembler directives.

Ans. A typical program format using assembler directives is as shown below:

```
Line 1.   MODEL SMALL   ;  selects small model
Line 2.   DATA          ;  indicates data segment
          ...
          ...
          ...
Line 15.  CODE          ;  indicates start of code segment
          ...
          ...           ;  body of the program
          ...
Line 20.  END           ;  End of file
```

16. Discuss the PTR directive.

Ans. This directive assigns a specific type to a variable or a label and is used in situations where the type of the operand is not clear. The following examples will help explain the PTR directive more elaborately.

(a) The instruction INC [BX] does not tell the assembler whether to increment a byte or word pointed to by BX. This ambiguity is cleared with PTR directive.
INC BYTE PTR [BX] ; Increment the byte pointed to by [BX]
INC WORD [BX] ; Increment the word pointed to by [BX]

(b) An array of words can be accessed by the statement WORDS, as for examples
WORDS DW 1234 H, 8823 H, 12345 H, etc.
But PTR directive helps accessing a byte in an array, like,
MOV AH, BYTE PTR WORDS.

(c) PTR directive finds usage in indirect jump. For an instruction like JMP [BX], the assembler cannot decide of whether to code the instruction for a NEAR or FAR jump. This difficulty is overcome by PTR directive.

JMP WORD PTR [BX] and JMP DWORD PTR [BX] are examples of NEAR jump and FAR jump respectively.

17. What is a macro?

Ans. A macro, like a procedure, is a group of instructions that perform one task. The macro instructions are placed in the program by the macro assembler at the point it is invoked.

Use of macros helps in creating new instructions that will be recognised by the assembler. In fact libraries of macros can either be written or purchased and included in the source code which apparently expands the basic instruction set of 8086.

18. Show the general format of macros.

Ans. The general format of a macro is

NAME MACRO Arg 1 Arg 2 Arg 3

Statements .....

.....

ENDM

The format begins with NAME which is actually the name assigned to the MACRO. 'Arg's' represent the arguments of the macro. Arguments are optional in nature and allows the same macro to be used in different places within a program with different sets of data. Each of the arguments represent a particular constant, hence a CPU register, for instance, cannot be used.

All macros end with ENDM.

19. Explain macro definition, macro call and macro expansion.

Ans. Creation of macro involves insertion of a new opcode that can be used in the program. This code, often called prototype code, along with the statements for representing and terminating a macro is called macro definition.

The statements that follow a macro definition is called macro call.

When the assembler encounters a macro call, it replaces the call with macro's code. This replacement action is referred to as macro expansion.

20. Explain the INCLUDE file.

Ans. A special file, say MACRO.LIB can be created which would contain the definitions of all macros of the user. In such a case, the writing of each macro's definition at the head of the main program can be dispensed with. The INCLUDE file may look like.

INCLUDE MACRO.LIB

This statement forces the assembler to automatically include all the statements in the MACRO.LIB.

Sometimes it may be undesirable to include the INCLUDE statement when the INCLUDE file is very long and the user may not be using many of the macros in the file.

21. Explain local variables in a macro.

Ans. Within the body of a macro, local variables can be used. A local variable can be defined by using LOCAL directive and is available within the macro and not outside.

For example, a local variable can be used in a jump address. The jump address has to be defined as a local, an error message will be outputted by the assembler.

Local variable(s) must be defined immediately following the macro directive, with the help of local directives.

22. Explain Controlled Expansion (also called Conditional Assembly).

Ans. While inside the macro, facilities are available to either accept or reject a code during macro execution— i.e., expansion of a macro prototype code would depend on the type(s) of actual parameter(s) passed to it by the call. This facility of selecting a code that is to be assembled is called controlled expansion.

The conditional assembly statements in macro are:

IF-ELSE-ENDIF	Statement
REPEAT	Statement
WHILE	Statement
FOR	Statement

23. For the conditional assembly process, show the (a) forms used for the IF statement (b) relational operators used with WHILE and REPEAT.

Ans. Figure 16.1 and 16.2 show respectively the forms used for the IF statement and the relational operators used with WHILE and REPEAT.

Statement	Function
IF	If the expression is true
IFB	If argument is blank
IFE	If the expression is not true
OFDEF	If the label has been defined
IFNB	If argument is not blank
IFNDEF	If the label has not been defined
IFIDN	If argument 1 equals argument 2
IFDIFWWW	If argument 1 does not equal argument 2

Fig.16.1: Forms used for IF statements

Operator	Function
EQ	Equal
NE	Not Equal
LE	Less than or Equal
LT	Less than
GT	Greater than
GE	Greater than or Equal
NOT	Logical inversion
AND	Logical AND
OR	Logical OR
XOR	Logical XOR

Fig.16.2: Relational operators used with WHILE and REPEAT

24. Distinguish between macro and procedure.

Ans. A procedure is invoked with a CALL instruction and terminated with a RET instruction. Again the code for the procedure appears only once in the programs—irrespective of the number of times it appears.

A macro is invoked, on the other hand, during program assembly and not when the program is run. Whenever in the program the macro is required, assembler substitutes the defined sequence of instructions corresponding to the macro. Hence macro, if used quite a few number of times, would consume a lot of memory space than that would be required by procedure.

Macro does not require CALL–RET instructions and hence will be executed faster. Sometimes, depending on the macro size, the macro may require less number of codes than is required by the equivalent procedure.

17

Input/Output Interface of 8086

1. What are the two schemes employed for I/O addressing.

Ans. The two schemes employed for I/O addressing are Isolated I/O and Memory I/O.

2. Compare Isolated I/O and Memory mapped I/O.

Ans. The comparison is shown in Table 17.1

Table 17.1: Comparison between isolated and memory mapped I/O

Isolated I/O	Memory mapped I/O
1. I/O devices are treated separate from memory.	1. I/O devices are treated as part of memory.
2. Full 1 MB address space is available for use as memory.	2. Full 1 MB cannot be used as memory since I/O devices are treated as part of memory.
3. Separate instructions are provided in the instruction set to perform isolated I/O input-output operations. These maximise I/O operations.	3. No separate instructions are needed in this case to perform memory mapped I/O operations. Hence, the advantage is that many instructions and addressing modes are available for I/O operations.
4. Data transfer takes place between I/O port and AL or AX register only. This is certainly a disadvantage.	4. No such restriction in this case. Data transfer can take place between I/O port and any internal register. Here, the disadvantage is that it somewhat slows the I/O operations.

3. Draw the Isolated I/O memory and I/O address space.

Ans. In isolated I/O scheme, memory and I/O are treated separately. The 1 MB address space can be treated as memory address space ranging from 00000 H to FFFFF H, while the address range from 00000 H to 0FFFF H (i.e., 64 KB I/O addresses) can be treated as I/O address space as shown below in Fig. 17.1.

It should be remembered that two consecutive memory or I/O addresses could be accessed as a word-wide data.

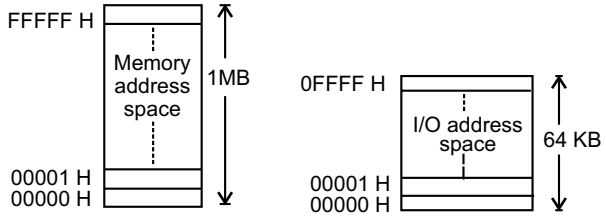


Fig. 17.1: Isolated I/O (a) memory address space (b) I/O address space

4. Draw and explain the memory mapped I/O scheme for 8086.

Ans. In this scheme, CPU looks to I/O ports as if it is part of memory. Some of the memory space is earmarked (dedicated) for I/O ports or addresses. The memory mapped I/O scheme is shown in Fig. 17.2 below in which the memory locations starting from C0000 H to C0FFF H (4 KB in all) and from D0000 H to D0FFF H (4 KB in all) are assigned to I/O devices.

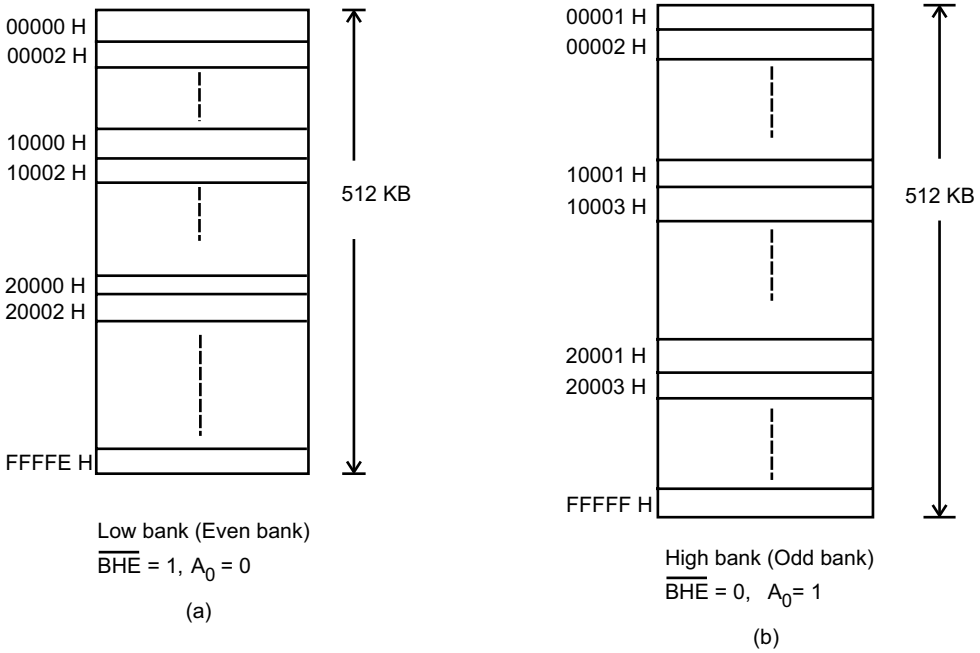


Fig. 17.2 : Memory mapped I/O scheme

5. Draw the (a) MIN and (b) MAX mode 8086 based I/O interface.

Ans. (a) The 8086 based Min mode I/O interface is shown below in Fig. 17.3.

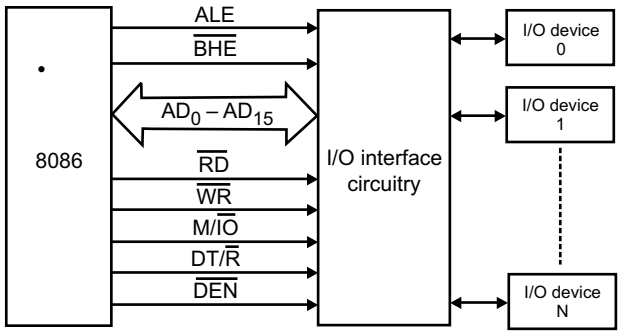


Fig. 17.3: The Minimum mode 8086 system I/O interface

It is seen that the lower two bytes $AD_0 - AD_{15}$ are used for input/output data transfers. The interface circuitry performs the following tasks.

- Selecting the particular I/O port
- Synchronise data transfer
- Latch the output data
- Sample the input data
- Voltage levels between the I/O devices and 8086 are made compatible.

(b) The 8086 based Max mode I/O interface is shown below in Fig.17.4

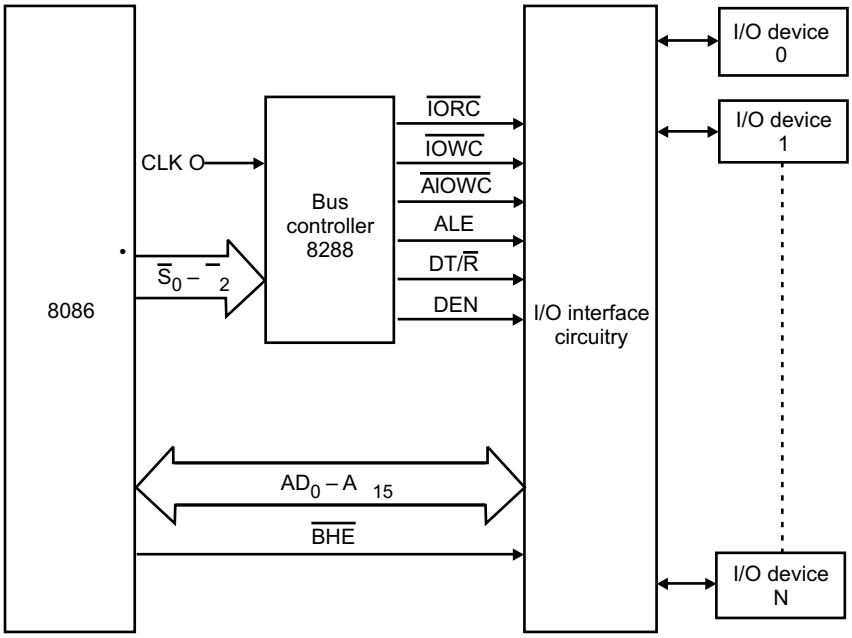


Fig. 17.4: The Maximum-mode 8086 system I/O interface

In this case the status codes $\overline{S_2} - \overline{S_0}$ outputted by 8086 are fed to the 8288 bus controller IC. The decoder circuit within 8288 decodes these three signals. For instance, 001 and 010 on $\overline{S_2}\ \overline{S_1}\ \overline{S_0}$ lines indicate 'Read I/O port' and 'Write I/O port' respectively. The first corresponds to $\overline{IORC}$ while the second corresponds to $\overline{IOWC}$ or $\overline{AIOWC}$ signals. These command signals are utilised to control data flow and its direction between I/O devices and the data bus.

6. What kind of I/O is used for IN and OUT instructions?

Ans. For 8086 based systems, isolated I/O is used with IN and OUT instructions. The IN and OUT instructions are of two types: direct I/O instructions and variable I/O instructions. The different types of instructions are tabulated in Fig.17.5.

Mnemonic	Meaning	Format	Operation
IN	Input direct	IN Acc, Port	(Acc)←(Port) Acc = AL or AX
	Input indirect (variable)	IN Acc, DX	(Acc)←((DX))
OUT	Output direct	OUT Port, Acc	(Port)←(Acc)
	Output indirect (variable)	OUT DX, Acc	((DX))←(Acc)

Fig. 17.5: Input/output instructions

7. Which register(s) is/are involved in data transfers?

Ans. Only AL (for 8-bits) or AX (for16-bits) register is involved in data transfer involving the 8086's CPU and I/O devices—thus they are also known as accumulator I/O.

8. Bring out the differences between direct I/O instructions and variable I/O instructions.

Ans. The differences between the two types of instructions are tabulated below in Table 17.2.

Table 17.2: Differences between direct and variable I/O instructions

Direct I/O	Variable I/O
1. Involves 8-bit address as part of instruction	1. Involves 16-bit address as part of instruction. This resides in DX register. It must be borne in mind that the value in DX register is not an offset, but the actual port address.
2. Can access a maximum of 2^8 = 255 byte addresses.	2. Can access a maximum of 2^{16} = 64 KB of addresses.

9. Give one example each of (a) direct I/O (b) variable I/O instruction.

Ans. **(a)** An example of direct I/O instruction is as follows:

```
IN AL, 0F2 H
```

On execution, the contents of the byte wide I/O port at address location F2 H will be put into AL register.

(b) An example of this type is:

```
MOV  DX, 0C00F H
IN  AL, DX
```

On execution, at first DX register is loaded with the input port having address C00F H. The second instruction ensures that the port content is moved over to AL register.

10. Draw the (a) in port and (b) out port bus cycle of 8086.

Ans. **(a)** The input bus cycle of 8086 is shown in the Fig.17.6. In the first T state (i.e., T_1), address comes out via A_0–A_{19}, along with $\overline{BHE}$ signal. Also ALE signal goes high in T_1. The high to low transition on ALE at the end of T_1 latches the address bus. $M/\overline{IO}$ signal goes low at the beginning of T_1. $\overline{RD}$ line goes low in T_2 while data transfer occurs in T_3. $DT/\overline{R}$ goes low at the beginning of T_1 and $\overline{DEN}$ signal becomes active in T_2 which tells the I/O interface circuit when to put data on the data bus.

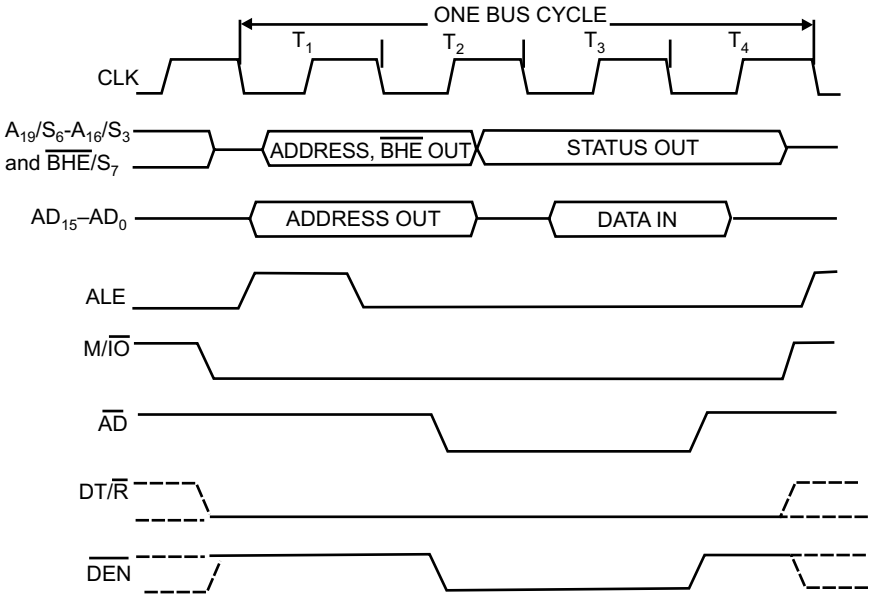


Fig. 17.6 : The Input bus cycle of 8086

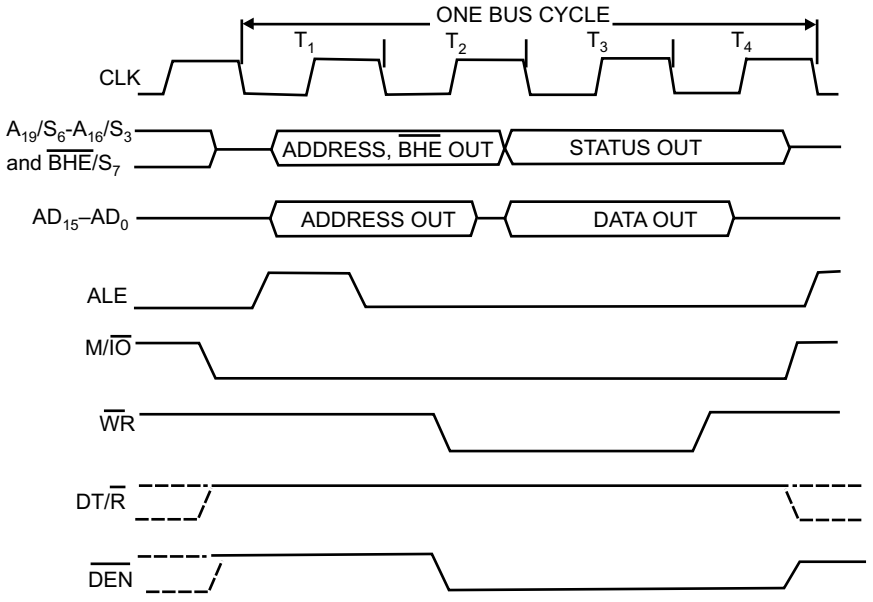


Fig. 17.7: The Output bus cycle of 8086

(b) The output bus cycle shown in Fig. 17.7.

The main differences between the output bus cycle and the just discussed input bus cycle are:

- $\overline{WR}$ Signal becomes active earlier than the $\overline{RD}$ signal. Hence in this case valid data is put on the data bus in T_2 state.
- $\overline{DEN}$ signal becomes active in T_1, while the same occurs in T_2 state for input bus cycle.

18

8086 Interrupts

1. How many interrupts can be implemented using 8086 μP?

Ans. A total of 256 interrupts can be implemented using 8086 μP.

2. Mention and tabulate the different types of interrupts that 8086 can implement.

Ans. 8086 μP can implement seven different types of interrupts.

- NMI and INTR are external interrupts implemented via *Hardware*.
- INT n, INTO and INT3 (breakpoint instruction) are software interrupts implemented through *Program*.
- The 'divide-by-0' and 'Single-step' are interrupts *initiated by CPU*.

Table 18.1 shows the seven interrupt types implemented by 8086.

Table 18.1: The Seven different types of 8086 interrupts

Name	Initiated by:	Maskable?	Trigger	Priority	Acknowledge signal?	Vector table address–	Interrupt latency
NMI	External hardware	No	↑Edge, hold 2 T states min.	2	None	00008H–0000BH	Current instruction + 51 T states
INTR	External hardware	Yes via IF	High level until acknowledged	3	$\overline{\text{INTA}}$	n * 4[b]	Current instruction + 61 T states
INT n	Internal via software	No	None	1	None	n * 4	51 T states
INT 3 (break point)	Internal via software	No	None	1	None	0000CH–0000FH	52 T states
INTO	Internal via software	No	None	1	None	00010H–00013H	53 T states
Divide-by-0	Internal via CPU	Yes via OF	None	1	None	00000H–00003H	51 T states
Single-step	Internal via CPU	Yes via TF	None	4	None	00004H–00007H	51 T states

a. All interrupt types cause the flags, CS, and IP registers to be pushed onto the stack. In addition, the IF and TF flags are cleared.

b. n is an 8-bit type number read during the second INTA pulse.

3. Distinguish between the two hardware interrupts of 8086.

Ans. The distinction between the two hardware interrupts of 8086 are as follows, shown in Table 18.2.

Table18.2: Comparison of NMI and INTR interrupts

NMI	INTR
1. Non-maskable type.	1. Maskable type.
2. Higher priority.	2. Lower priority.
3. Edge triggered interrupt initiated on Low to High transition.	3. Level triggered interrupt.
4. Must remain high for more than 2 CLK cycles.	4. Sampled during last CLK cycle of each instruction.
5. The rising edge of NMI input is latched on-chip and is serviced at the end of current instruction.	5. No latching. Must stay high until acknowledged by CPU.
6. No acknowledgement.	6. Acknowledged by INTA output signal.

4. How many bytes are needed to store the starting addresses of ISS for 8086 µP?

Ans. 8086 µP can implement 256 different interrupts. To store the starting address of a single ISS (Interrupt Service Subroutine), four bytes of memory space are required—two bytes to store the value of CS and two bytes to store the IP value. Thus to store the starting address of 256 ISS, in all 256 × 4 = 1024 bytes = 1 KB will be required.

5. Indicate the number of memory spaces needed in stack when an interrupt occurs.

Ans. When an interrupt occurs, before moving over to starting address of the corresponding ISS, the following are pushed into the stack: the contents of the flag register, CS and IP. Since each one of the three are 2 bytes, hence a total of 6 bytes of memory space is needed in the stack to accommodate the flag register, CS and IP contents.

6. What are meant by interrupt pointer and interrupt pointer table?

Ans. The starting address of an ISS in the 1 KB memory space is known as the interrupt pointer or interrupt vector corresponding to that interrupt.

The 1 KB memory space needed to store the starting addresses of all the 256 ISS is called the interrupt pointer table.

7. Write down the steps, sequentially carried out by the systems when an interrupt occurs.

Ans. When an interrupt occurs (hardware or software), the following things happen:

- The contents of flags register, CS and IP are pushed on to the stack.
- TF and IF are cleared which disable single step and INTR interrupts respectively.
- Program jumps to the starting address of ISS.
- At the end of ISS, when IRET is executed in the last line, the contents of flag register, CS and IP are popped out of the stack and placed in the respective registers.
- When the flags are restored, IF and TF get back their previous values.

8. Draw and discuss the interrupt pointer table for 8086 µP.

Ans. The interrupt pointer table for 8086 is shown in Fig. 18.1.

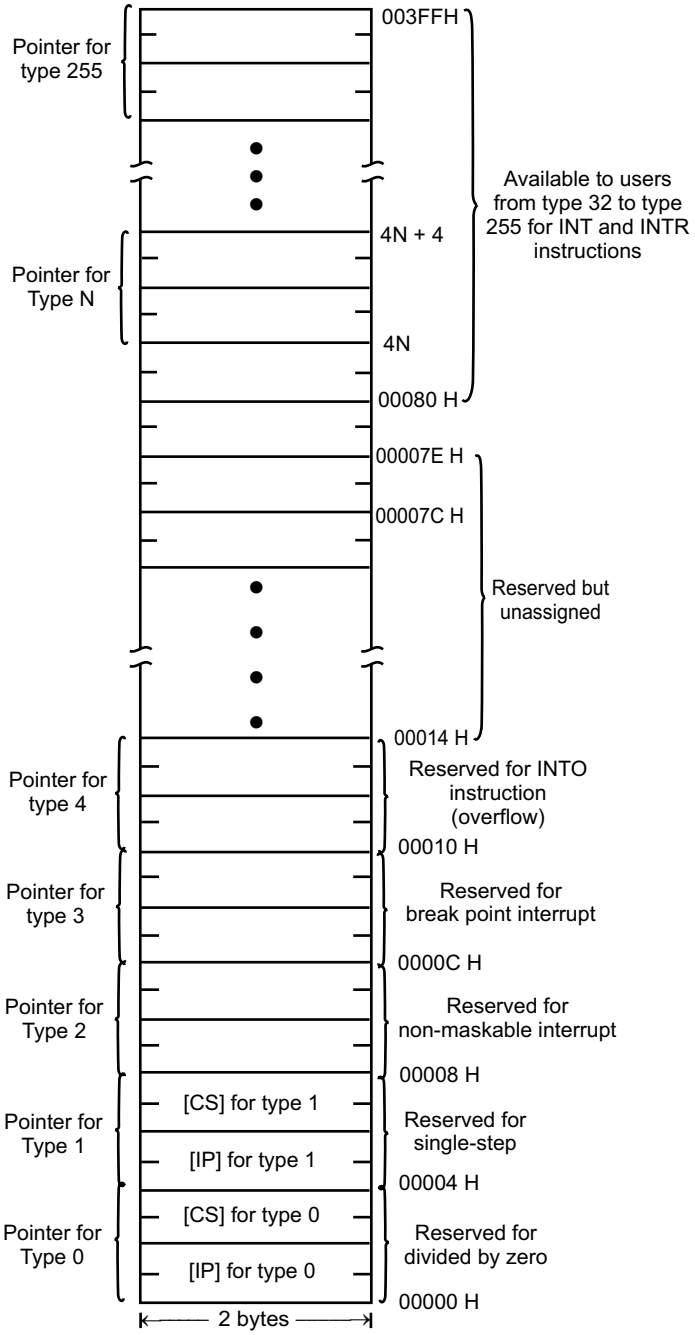


Fig. 18.1: Interrupt Pointer Table for intel 8086

The 256 interrupt pointers are stored in memory locations starting from 00000 H to 003FF H (1 KB memory space). The number assigned to an interrupt pointer is called

the Type of the corresponding interrupt. As for example, Type 0 interrupt, Type 1 interrupt ... Type 255 interrupt. Type 0 interrupt has a memory address 00000 H, Type 1 has a memory address 00004 H, while Type 255 has a memory address 003FF H. The first five pointers (Type 0 to Type 4) are dedicated pointers used for divide by zero, single step, NMI, break point and overflow interrupts respectively. The next 27 pointers (Type 5 to Type 31) are reserved pointers—reserved for some special interrupts. The remaining 224 interrupts—from Type 32 to Type 255 are available to the programmer for handling hardware and software interrupts.

9. Discuss the priority of interrupts of 8086.

Ans. 8086 tests for the occurence of interrupts in the following hierarchical sequence:

- Internal interrupts (divide-by-0, single step, break point and overflow)
- Non-maskable interrupt—via NMI
- Software interrupts—via INTn
- External hardware interrupt—via INTR

Hence, internal interrupts belong to the highest priority group and internal hardware interrupts are the lowest priority group. Again, different interrupts are given different priorities by assigning a type number corresponding to each priority—starting from Type 0 (highest priority interrupt) to Type 255 (lowest priority interrupt). Thus, Type 40 interrupt is having more priority than Type 41 interrupt. If we presume that at any instant a Type 40 interrupt is in progress, then it can be interrupted by any software interrupt, the non-maskable interrupt, all internal interrupts or any external interrupt with a Type number less than 40.

10. Outline the events that take place when 8086 processes an interrupt.

Ans. Fig. 18.2 shows the manner in which 8086 processes an interrupt while the following are the events that take place sequentially when the processor receives an interrupt (from an *external* device via INT 32 through INT 255:

- Receiving an interrupt request (from an external device)
- Generation of interrupt acknowledge bus cycles.
- Servicing the Interrupt Service Subroutine (corresponding to the external device which has interrupted the CPU.)

Again the difference between simultaneous interrupt and interrupt within an ISS is to be understood. Occurrence of more than one interrupt within the *same instruction* is called simultaneous interrupt while if an interrupt occurs while the ISS is in progress, it is an interrupt occurring within an ISS.

Internal interrupts (except single step) have priority over simultaneous external requests. For example, let the current instruction causes a divide-by-zero interrupt when an INTR (a hardware interrupt) occurs, the former will be serviced. Again, if simultaneous interrupts occur on INTR and NMI, then NMI will be serviced first.

For simultaneous interrupts occurring, the priority structure of Fig.18.2 will be honoured, with the highest priority interrupt being serviced first.

Although it has been commented that software interrupts get priority over external hardware interrupts, even then if an interrupt on NMI occurs as soon as the interrupt's ISS begins, it (i.e., NMI) will be recognised and hence serviced.

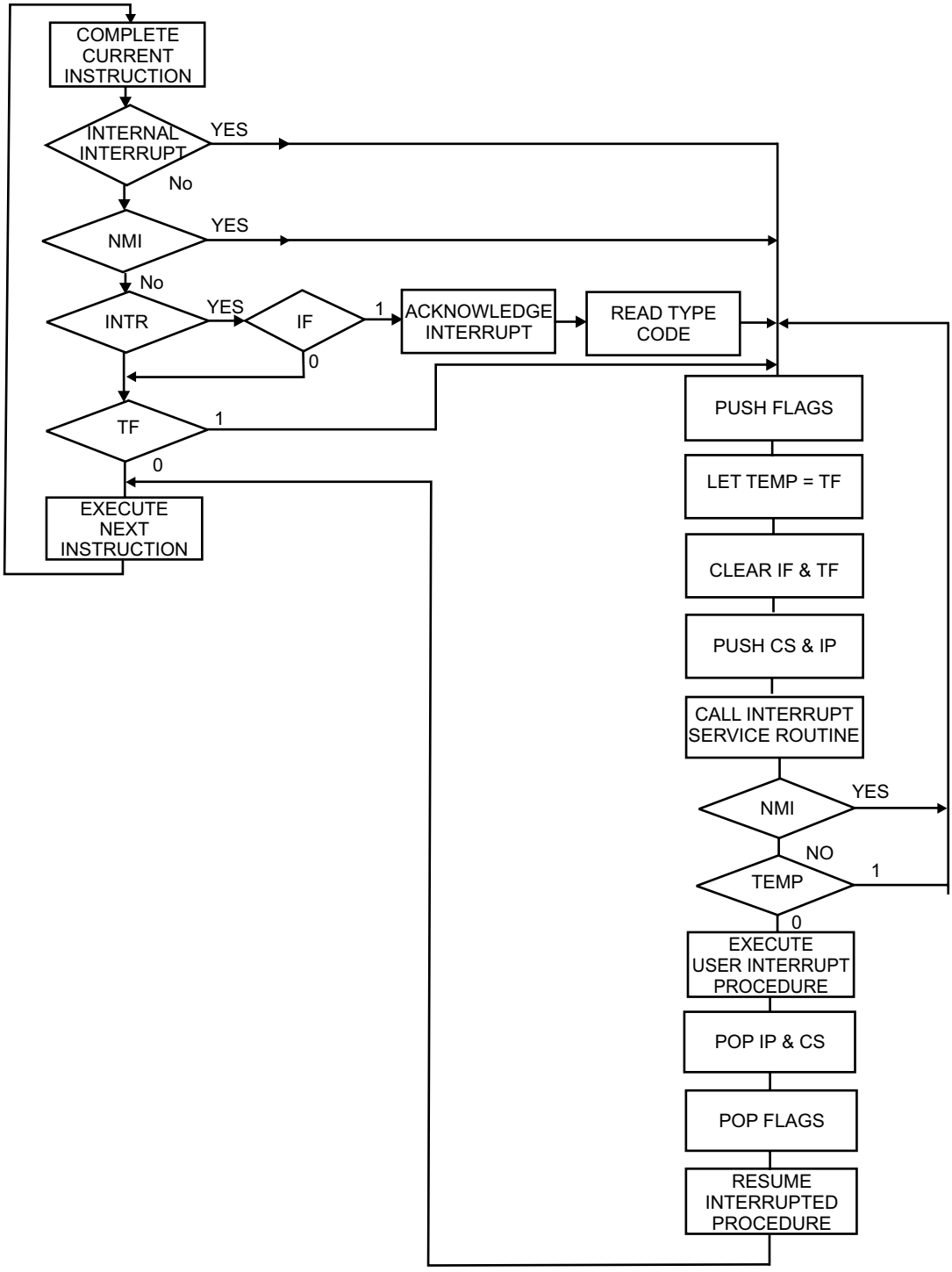


Fig.18.2: Interrupt processing sequence of the 8086 microprocessors

11. List the different interrupt instructions associated with 8086 μP.

Ans. Table 18.3 lists the different interrupts of 8086 μP along with a brief description of their functions.

Table18.3 : The different interrupt instructions of 8086 μP

Mnemonic	Meaning	Format	Operation	Flags affected
CLI	Clear interrupt flag	CLI	0→(IF)	IF
STI	Set interrupt flag	STI	1→(IF)	IF
INT n	Type n software interrupt	INT n	(Flags)→((SP) − 2) 0→TF, IF. (CS)→((SP) − 4) (2 + 4, n)→(CS) (IP)→((SP)–6) (4 . n)→(IP)	TF, IF
IRET	Interrupt return	IRET	((SP))→(IP) ((SP)+2)→(CS) ((SP)+4)→(Flags) (SP)+6→(SP)	All
INTO	Interrupt on overflow	INTO	INT 4 steps	TF, IF
HLT	Halt	HLT	Wait for an external interrupt or reset to occur	None
WAIT	Wait	WAIT	Wait for $\overline{\text{TEST}}$ input to go active	None

12. Show the internal interrupts and their priorities.

Ans. The internal interrupts are: Divide-by-0, single step, break point and overflow corresponding to Type 0, Type 1, Type 3 and Type 4 interrupts respectively.

Since a type with lesser number has higher priority than a type with more number, thus the mentioned internal interrupts can be arranged in a decreasing priority mode, with highest priority mentioned first: Divide-by-0, single step, break point, overflow.

13. What are the characteristics associated with internal interrupts?

Ans. The following are the characteristics associated with internal interrupts:

- The interrupt type code is either contained in the instruction itself or is predefined.
- No $\overline{\text{INTA}}$ bus cycles are generated, as in the case of INTR interrupt input.
- Apart from single step interrupt, no other internal interrupt can be disabled.
- Internal interrupts, except single step have higher priority than external interrupts.

14. Discuss the two interrupts HLT and WAIT.

Ans. On execution of HLT (halt) instruction by 8086, CPU suspends its instruction execution and enters into an idle state. It waits for either an external hardware interrupt or a reset input (interrupt). When any one of these occurs, CPU starts executing again.

When the WAIT instruction is executed by 8086, it internally checks the logic level existing at its $\overline{\text{TEST}}$ input. If $\overline{\text{TEST}}$ is at logic 1 state, then CPU goes into an idle state. When $\overline{\text{TEST}}$ input assumes a zero state, execution resumes from the next sequential

instruction in the program. $\overline{\text{TEST}}$ input is normally connected to the BUSY output signal of 8087 NDP.

15. Mention the addresses at which CS_{40} and IP_{40} corresponding to vector 40 would be stored in memory.

Ans. INT 40, for its storage, requires four memory locations—two for IP_{40} and two for CS_{40}.

The addresses are calculated as follows:

$$4 \times 40 = 160_{10} = 1010\ 0000_2 = A0\ H.$$

Thus, IP_{40} is stored starting at 000A0 H and CS_{40} is stored starting at 000A2 H.

16. Explain in detail the external hardware interrupt sequence.

Ans. The external device can request for service via INT 32 through INT 255 by pulling the corresponding INT n (n = 32 to 255) high. The interrupt request gives rise to generation of interrupt acknowledge bus cycles and then moving into ISS corresponding to the device which has interrupted the system. The presently requested interrupt is recognised provided no higher priority interrupt is pending and IF is already set via software.

Once the interrupt is recognised (since for any INTR to be recognised, the corresponding INT must stay high till the last clock cycle of the presently executed instruction). 8086 initiates interrupt acknowledge bus cycles, shown in Fig. 18.3.

During T_1 of the first interrupt bus cycle, ALE is put to low state and remains so till the end of the cycle. During the whole of this cycle, address/data bus is driven into Z-state. During T_2 and T_3 of this first interrupt bus cycle, $\overline{\text{INTA}}$ is put to low state—indicating that the request for service has been granted so that the requesting device can withdraw its high logic which is connected to INTR pin 8086.

$\overline{\text{LOCK}}$ signal is of importance only in maximum mode. This signal goes low during T_2 of the first INTA bus cycle and is maintained in zero state until T_2 of the second INTA bus cycle.

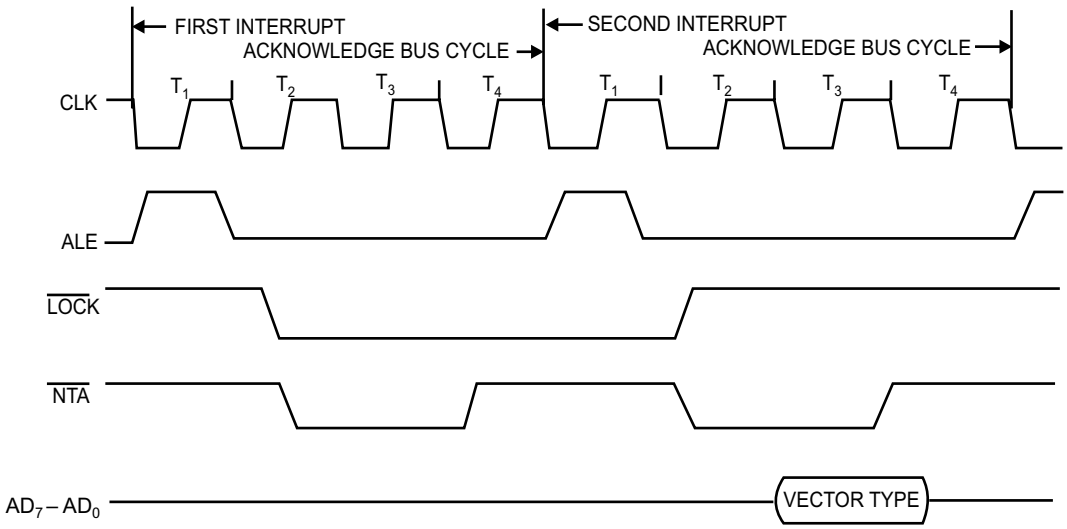


Fig. 18.3: Interrupt acknowledge bus cycle

8086 is prevented from accepting a HOLD request. The $\overline{\text{LOCK}}$ output, in conjunction with external logic, is used to lock off other devices from the system bus. This ensures completion of the current interrupt till its completion.

During the second interrupt bus cycle, the external circuit puts in the interrupt code (32_{10} = 20 H through 255_{10} = FF H) on the data bus AD_0– AD_7 during T_3 and T_4 and is read by 8086.

Before moving to ISS, CPU saves the contents of the flag register along with the current CS and IP values. Now by reading the number from the data bus, the corresponding CS and IP values are placed in them (for instance, if the external device has interrupted via INT 60, then CS_{60} and IP_{60} would be loaded into CS and IP register respectively). Thus the ISS would be run to its completion because before moving into ISS, IF and single-stepping have been disabled.

In the last line of ISR, an IRET instruction is there, which on its execution pops the old CS and IP values from the stack and put them in CS and IP registers. This thus ensures that the main program starts at the very memory location that was left off because of ISS.

17. Indicate two applications where NMI interrupt can be applied.

Ans. NMI is a non-maskable hardware interrupt, i.e., it cannot be masked or disabled. Hence, it is used for very important system exigencies like (a) detection of power failure or (b) detection of memory read error cases.

18. Draw a circuit that will terminate the INTR when interrupt request has been acknowledged.

Ans. Fig. 18.4 makes INTR input of 8086 to go into 1 state once the interrupt request comes from some external agency. The falling edge of the peripheral clocks the flip-flop which makes INTR to become 1. The first $\overline{\text{INTR}}$ pulse then resets Q, making INTR to become 0. This ensures that no second interrupt request is recognised by the system. The reset input sees to it that INTR remains in the 0 state when the system is reset.

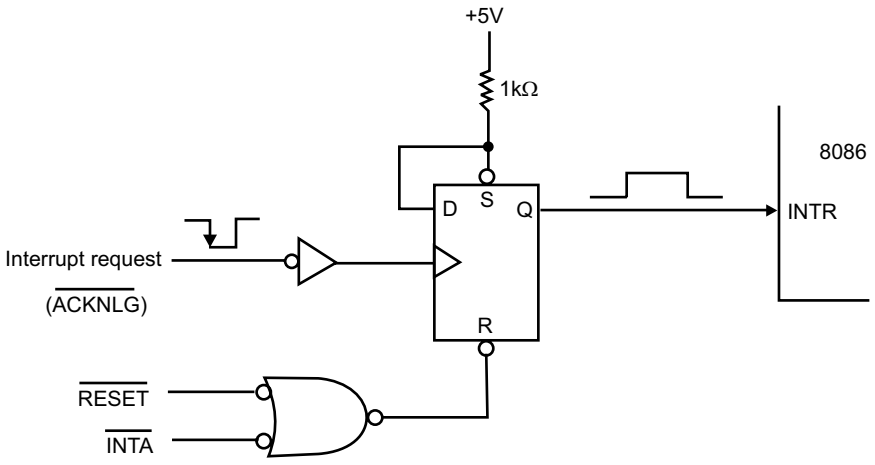


Fig. 18.4: Termination of INTR request once interrupt has been acknowledged

19. Discuss the following (a) Type 0 interrupt (b) Type 1 interrupt (c) Type 2 interrupt (d) Type 3 interrupt and (e) Type 4 interrupt.

Ans. (a) Type 0 interrupt (or Divide-by-zero interrupt)

If the quotient resulting from a DIV (divide) instruction or an IDIV (integer divide) instruction is too large such that it cannot be accommodated in the destination register, a divide error occurs. Then 8086 perform a Type 0 interrupt. This then passes the control to a service subroutine at addresses corresponding to IP_0 and CS_0 at 0000 H and 0002 H respectively in the pointer table.

(b) Type 1 interrupt (Single Step interrupt)

The single step interrupt will be enabled only if the trap flag (TF) bit is set (= 1). The TF bit can be set/reset by software.

Single step control is used for debugging in assembly language. In this mode the processor executes one instruction and then stops. The contents of various registers and memory locations can be examined. If the results are found to be ok, then a command can be inserted for execution of the next instruction. Trap flag cannot be set directly. This is done by pushing the flags on the stack, changes are made and then they are popped.

(c) Type 2 interrupt (non-maskable NMI interrupt)

Type 2 interrupt is the non-maskable NMI interrupt and is used for some emergency situations like power failure. When power fails, an external circuit detects this and sends an interrupt signal via NMI pin of 8086. The DC supply remains on for atleast 50 ms via capacitor banks so that the program and data remaining in RAM locations can be saved, which were being executed at the time of power failure.

(d) Type 3 interrupt (break point interrupt)

Type 3 interrupt is a break point interrupt. The program runs up to the break point when the interrupt occurs. This is achieved by inserting INT 3 at the point the break is desired. The ISS corresponding to Type 3 interrupt saves the register contents in the stack and can also be displayed on CRT and the control is returned to the user. This is used as a software debugging tool, like single stepping method.

(e) Type 4 interrupt (overflow interrupt)

The software instruction INTO (interrupt on overflow) is inserted in a program immediately after an arithmetic operation is performed. Insertion of INTO implements a Type 4 interrupt. When the signed result of an arithmetic operation on two signed numbers is too large to be stored in the destination register or else in a memory location, an overflow occurs and the OF (overflow flag) is set. This initiates INT4 instruction and the program control moves over to the starting address of the ISS, which corresponds to IP_4 and CS_4. These two are stored at address locations 0010 H and 0012 H respectively.

20. In what way the INTO instruction is different from others?

Ans. The INTO instruction is different in that no type number is needed to be mentioned.

To explain the difference, for executing any INT instruction, type no. is needed, like INT 10, INT 23, etc.

To explain further,

Opcode	Operand	Object Code	Mnemonic
INT	Type	CD 23	INT 23 H (assuming Type 23 H is employed)
INTO	none	CE	INT

21. Draw the schemes of (a) Min and (b) Max mode 8086 system external hardware interrupt interface and explain.

Ans. (a) The scheme of interconnections of Min-mode 8086 system external hardware interrupt interface is shown below in Fig. 18.5.

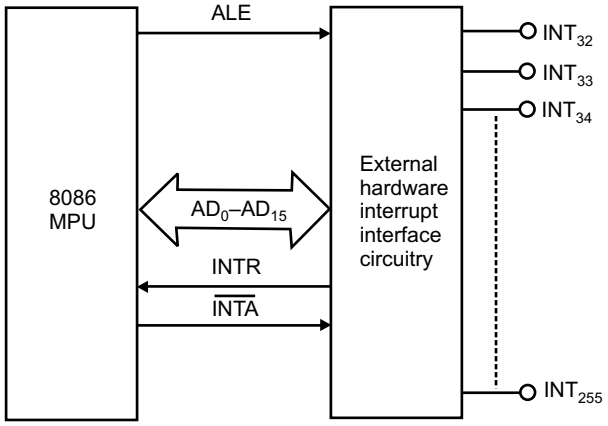


Fig. 18.5: Minimum-mode external hardware interrupt interface

The interconnecting signals to be considered for 8086 are ALE, INTR, $\overline{\text{INTA}}$ and the data bus $AD_0 - AD_{15}$.

The external device requests the service of 8086 via the INTR line. INTR is level triggered and must stay at logic 1 until recognised by the processor. Two interrupt acknowledge bus cycles are generated in response to INTR. At the end of the first bus cycle, the INTR should be removed so that it does not interrupt the 8086 a second time and the ISS can run without interruption. In this second bus cycle CPU puts the type number on the data bus of the active interrupt.

(b) The scheme of interconnections of Max-mode 8086 system external hardware interrupt interface is shown below in Fig. 18.6.

In this mode, the bus controller IC 8288 generates the $\overline{\text{INTA}}$ and ALE signals. $\overline{\text{INTA}}$ is generated when at the input of 8288, a status 000 H is applied via the status lines $\overline{S_2}$ $\overline{S_1}$ $\overline{S_0}$.

The $\overline{\text{LOCK}}$ signal in the figure is the bus priority lock signal and is the input to bus arbiter circuit. This circuit ensures that no other device can take control of the system buses until the presently run interrupt acknowledge cycle is completed.

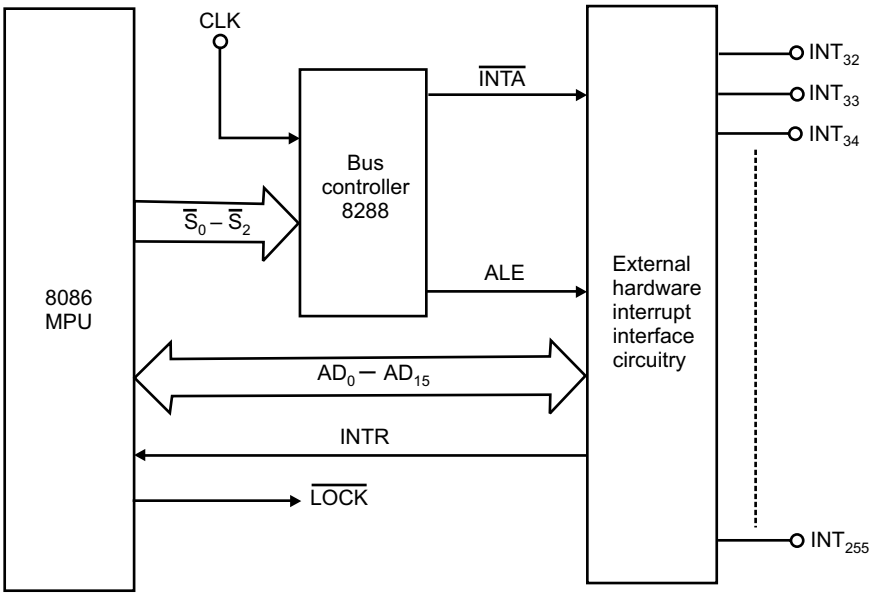


Fig.18.6: Maximum-mode 8086 system external hardware interrupt interface