MU

MICROPROCESSOR

Code - EMO43A

easy-solutions

Mumbai University Paper Solutions

Strictly as per the New Revised Syllabus (Rev - 2016) of
Mumbai University w.e.f. academic year 2018-2019
(As per Choice Based Credit and Grading System)

MU

MICROPROCESSOR

Semester V - Computer Engineering

Chapterwise Paper Solution upto May 2019.

TechKnowledge™
Publications

easy – solutions

MUMBAI

Microprocessor

Semester V – Computer Engineering

Strictly as per the Choice Based Credit and Grading System
(Revise 2016) of Mumbai University w.e.f. academic year 2018-2019

TechKnowledge™ Publications

EM043A

Microprocessor

(Semester V – Computer Engineering) (MU)

Edition 2019

This edition is for sale in India, Bangladesh, Bhutan, Maldives, Nepal, Pakistan, Sri Lanka and designated countries in South-East Asia. Sale and purchase of this book outside of these countries is unauthorized by the publisher.

Printed at : 37/2, Ashtvinayak Industrial Estate, Near Pari Company,
Narhe, Pune, Maharashtra State India,
Pune – 411041

Published by

TechKnowledge Publications

Head Office : B/5, First floor, Maniratna Complex, Taware Colony, Aranyeshwar Corner,
Pune - 411 009. Maharashtra State, India
Ph : 91-20-24221234, 91-20-24225678.
Email : info@techknowledgebooks.com,
Website : www.techknowledgebooks.com

(Book Code : EMO43A)

INDEX

Table of Contents

SYLLABUS

Microprocessor

Module No.	Unit No.	Topics
1.0		**The Intel Microprocessors 8086/8088 Architecture**
	1.1	• 8086/8088 CPU Architecture, Programmer's Model • Functional Pin Diagram. • Memory Segmentation. • Banking in 8086. • Demultiplexing of Address/Data bus. • Study of 8284 Clock Generator. • Study of 8288 Bus Controller. • Functioning of 8086 in Minimum mode and Maximum mode. • Timing diagrams for Read and Write operations in minimum and maximum mode.
Module No.	**Unit No.**	**Topics**
2.0		**Instruction Set and Programming**
	2.1	• Addressing Modes • Instruction set – Data Transfer Instructions, String Instructions, Logical Instructions, Arithmetic Instructions, Transfer of Control Instructions, Processor Control Instructions. • Assembler Directives and Assembly Language Programming, Macros, Procedures. • Mixed Language Programming with C Language and Assembly Language. • Programming based on DOS and BIOS Interrupts (INT 21H, INT 10H)
3.0		**8086 Interrupts**
	3.1	• Types of interrupts. • Interrupt Service Routine. • Interrupt Vector Table. • Servicing of Interrupts by 8086 microprocessor. • Programmable Interrupt Controller 8259 – Block Diagram, Interfacing the 8259 in single and cascaded mode, Operating modes, programs for 8259 using ICWs and OCWs.
4.0		**Peripherals and their interfacing with 8086**
	4.1	Memory Interfacing - RAM and ROM Decoding Techniques – Partial and Absolute.
	4.2	8255-PPI – Block diagram, Functional PIN Diagram, CWR, operating modes, interfacing with 8086.

	4.3	8253 PIT - Block diagram, Functional PIN Diagram, CWR, operating modes, interfacing with 8086.
	4.4	8257-DMAC – Block diagram, Functional PIN Diagram, Register organization, DMA operations and transfer modes.
5.0		**Intel 80386DX Processor**
	5.1	• Architecture of 80386 microprocessor. • 80386 registers – General purpose Registers, EFLAGS and Control registers. • Real mode, Protected mode, virtual 8086 mode. • 80386 memory management in Protected Mode – Descriptors and selectors, descriptor tables, the memory paging mechanism.
6.0		**Pentium Processor**
	6.1	Pentium Architecture. Superscalar Operation, Integer & Floating Point Pipeline Stages, Branch Prediction Logic, Cache Organisation and MESI Model.

❑❑❑

Microprocessor

Statistical Analysis

Chapter No.	Dec. 2018	May 2019
Chapter 1	–	–
Chapter 2	10 Marks	05 Marks
Chapter 3	15 Marks	20 Marks
Chapter 4	10 Marks	10 Marks
Chapter 5	–	–
Chapter 6	15 Marks	10 Marks
Chapter 7	10 Marks	05 Marks
Chapter 8	05 Marks	10 Marks
Chapter 9	05 Marks	10 Marks
Chapter 10	–	05 Marks
Chapter 11	10 Marks	–
Chapter 12	–	–
Chapter 13	–	–
Chapter 14	15 Marks	15 Marks
Chapter 15	25 Marks	15 Marks

Dec. 2018

Chapter 2 : The Intel Microprocessors 8086/8088 Architecture [Total Marks - 10]

Q. 6(a) Explain segmentation of 8086 microprocessor. Give its advantages. **(10 Marks)**

Ans. :

(A) Segmentation of 8086 microprocessor

- 8086 has 20-bit address bus while the registers are of 16-bit. To access a memory location, provide 20-bit address is provided while the registers are 16-bit; this is made possible using segmentation.
- Segmentation in 8086 refers to division of the 1MB main memory into segments or blocks of 64KB each. This is done so as to access a segment of the memory using the 16-bit address or pointer registers.
- There are four registers that point to the beginning of a segment and are called as **segment registers**. They are:
  1. The Code Segment (CS) register
  2. The Stack Segment (SS) register
  3. The Extra Segment (ES) register
  4. The Data Segment (DS) register
- 8086 has 16-bit registers while the memory access requires generating of a 20-bit address on the address bus. These segment registers have specific task to point to the beginning of specific segments.
- Code Segment (CS) points to the beginning of Code segment (that stores the programs or codes) Stack Segment (SS) points to the beginning of stack segment. Data Segment (DS) and Extra Segment (ES) are used to point to data segments, wherein ES is used only during the string instruction execution.
- The segment registers do the task of pointing to the starting of a segment while the pointer or index registers (BX, SI, DI, SP, BP, IP) point to a location within the segment.
- The 16-bit address provided by the segment register is suffixed with 4-bit zeroes or shifted left four times. Thus producing a 20-bit address.

- A pointer (index or offset) register of 16-bits can select a location within the segment and hence the segment is of 64KB (2^{16}) memory locations.
- The CS is used to point to the beginning of Code segment and the Instruction pointer (IP) along with CS points to the next instruction to be executed. This is done by shifting the 16-bit value of CS left by 4 bits and then adding the value of IP with the shifted value as shown Fig. 1-Q. 6(a).

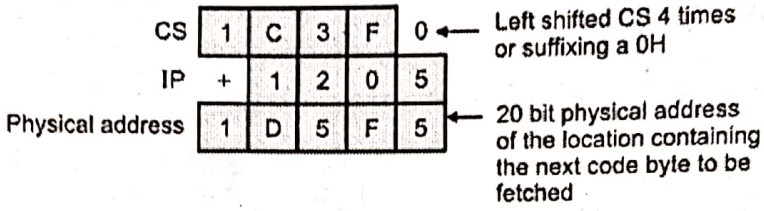


Fig. 1-Q.6(a) : Computation

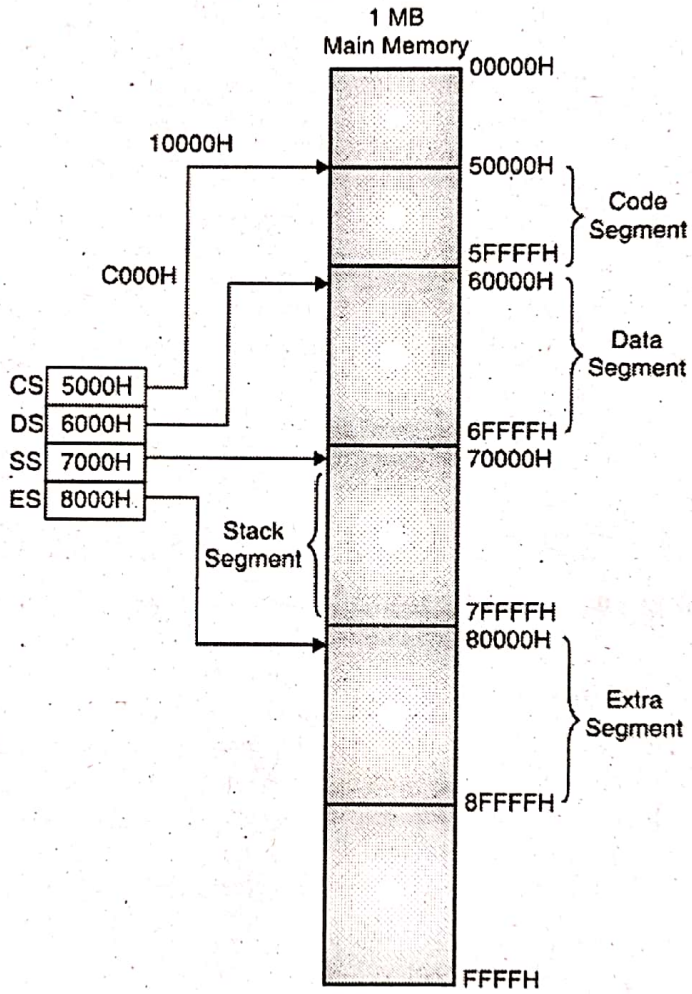


Fig. 2-Q. 6(a) : Segments in 8086

(B) Advantages of Segmentation

1. The programmer can access a memory that required 20-bit address, by using 16-bit registers only.
2. The programs, data and stack are stored in separate blocks in memory and hence the three are organized in a modular fashion.
3. It also help in object oriented programming to store data of an object.
4. Sharing of data or passing of data from one program to another is easily possible due to segmentation.
5. The segmentation makes data relocatable as the program uses only offset register pointers while the segment points to the base of a segment.

Chapter 3 : Operating Modes [Total Marks - 15]

Q. 1(a) Draw and explain memory read machine cycle timing diagram in minimum mode of 8086. (5 Marks)

Ans.:

- Fig. 1-Q. 1(a) shows the timing diagram for the memory read machine cycle.
- The sequence of operations during the read machine cycle are as follows :

Step 1 : The 8086 will make $M/\overline{IO} = 1$ if the read is from memory and $M/\overline{IO} = 0$ if the read is from the I/O device.

Step 2 : At about the same time the ALE output is asserted to 1.

Step 3 : Make $\overline{BHE}$ low/high and send out the desired address on AD_0 to AD_{15} and A_{16} to A_{19} lines.

Step 4 : Pull down ALE (make it 0). The address is latched into external latch.

Step 5 : Remove the address from AD_0 to AD_{15} lines and put them in the input mode (float them).

Step 6 : Assert the $\overline{RD}$ (read) signal low. This will put the data from the addressed memory location or I/O port on to the data bus.

Step 7 : Insert the "wait" T- states if the 8086 READY input is made low before or during the T_2 state of a machine cycle.

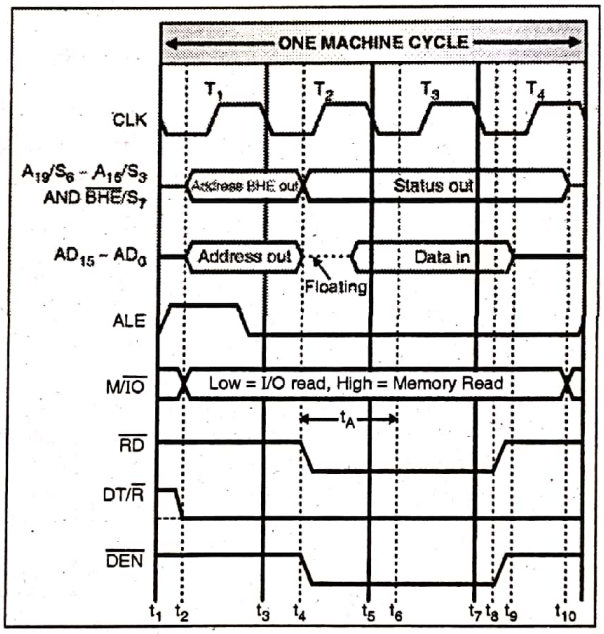


Fig. 1- Q. 1(a) : Timing diagram for the read machine cycle of 8086

Step 9 : Complete the "Read" cycle by making the $\overline{RD}$ line high (inactive).

Step 10 : For larger systems programmer need to use the data buffers. (8286 transceivers). Then the DT/$\overline{R}$ and $\overline{DEN}$ signals of 8086 are connected to 8086 and enabled at the appropriate time.

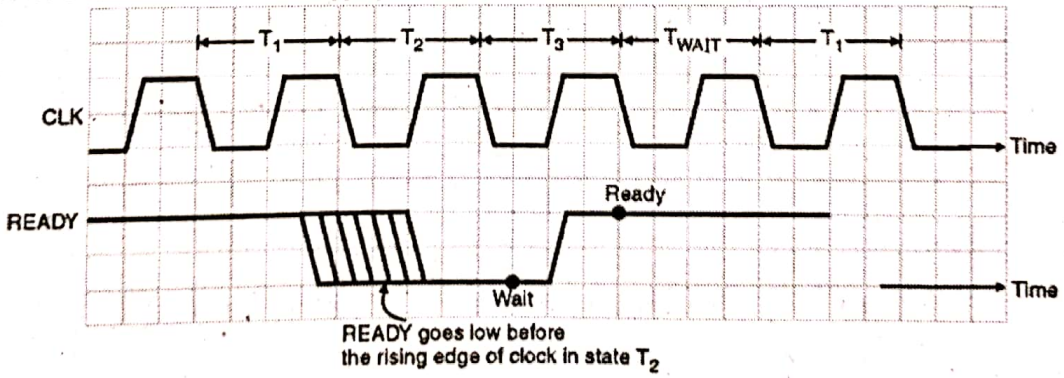


Fig. 2-Q. 1(a) : WAIT T-states

- **Memory access time (t_A) :** The address to data time or the time gap between the processor providing the address and the memory or I/O device providing the data is called as the access time of the memory or I/O device.
- **Concept of wait T-states :** It is used to synchronize slower devices. If a particular memory or I/O device is slower i.e. has a greater value of access time, then it needs to disable the READY pin of the microprocessor. This causes the microprocessor to insert wait states in between the machine cycle giving time for the device to place its data on the data bus. The name is given as wait states as the microprocessor waits for the device. The processor waits until the READY pin is enabled again. Fig. 2- Q. 1(a) shows wait states inserted in between of a machine cycle.

Q. 2(a) Explain the maximum mode configuration of 8086 microprocessor. (10 Marks)

Ans. :

- Fig. 1-Q. 2(a) shows the block diagram of the 8086 system in maximum mode.
- Additional circuitry is required to generate the control signals. The additional circuitry from the status signals ($S_2 - S_0$) to produces I/O and memory transfer signals. The Intel 8288 bus controller is used for this purpose.
- It generates the control signals required to direct the data flow and for controlling the latches 8282 and transreceivers 8286.
- It generates the control signals like, $\overline{MRDC}$, $\overline{MWTC}$, $\overline{AMWC}$, $\overline{AIOWC}$, $\overline{IORC}$, $\overline{IOWC}$ signals.
- The $\overline{MRDC}$ and $\overline{MWTC}$ are memory read command and memory write command signals. They instruct the memory to accept or send data on the data bus.
- The $\overline{IORC}$ and $\overline{IOWC}$ are I/O read command and I/O write command signals. They instruct the I/O device to read or write data to and from addressed port on the data bus.
- The $\overline{AIOWC}$ and $\overline{AMWTC}$ are advanced I/O write command and advanced memory write command signals. These signals are similar to the $\overline{IOWC}$ and $\overline{MWTC}$ signals except that, they are activated one clock signal earlier to the $\overline{IOWC}$ and $\overline{MWTC}$ signals.

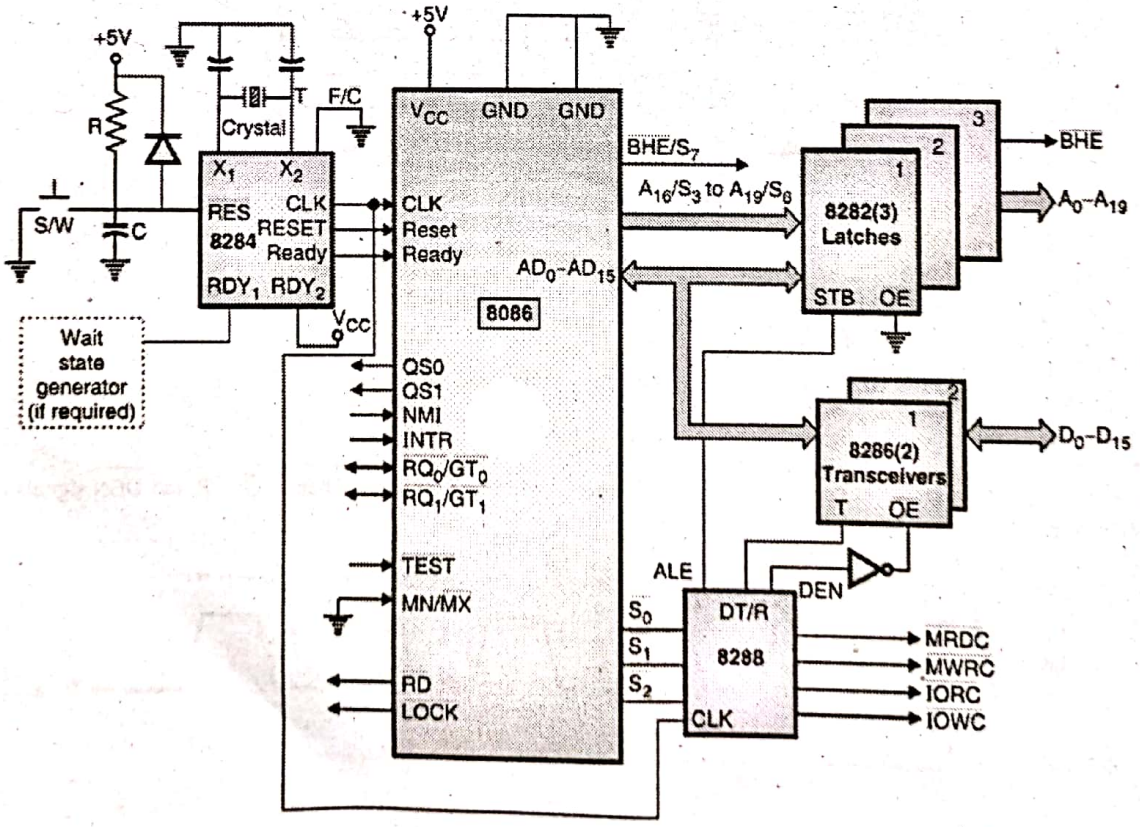


Fig. 1- Q. 2(a) : Block diagram of maximum mode minimum system

Chapter 4 : 8086/8088 Addressing Modes and Instruction Set [Total Marks - 10]

Q. 6(b) Explain different addressing modes of 8086 microprocessor. (10 Marks)

Ans. :

- Addressing modes (methods) refer to the different methods of addressing (selecting) the operands.
- Addressing modes of 8086 are as follows :

1. Register addressing mode
2. Immediate addressing mode
3. Memory addressing modes
4. String addressing mode
5. Implied addressing mode
6. I/O addressing modes

- Fig. 1-Q. 6(b) shows the classification of addressing modes of 8086.

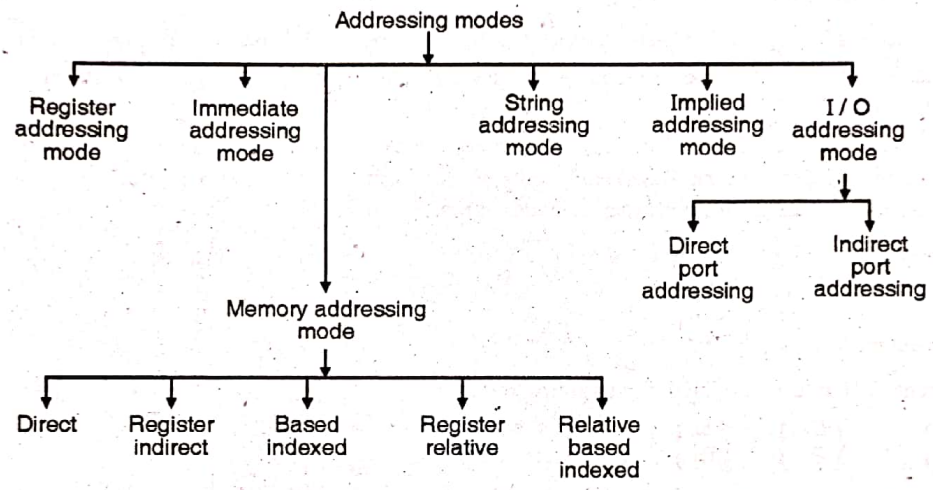


Fig. 1- Q. 6(b) : Addressing modes of 8086

1. Register Addressing Mode

- In this mode of addressing, operand is in the register, and instruction specifies the particular register as shown in Fig. 2- Q. 6. (b).
- The advantage of this addressing mode is that the access is faster.
- Registers may be used as source operands, destination operands or both.
- The registers may be 8/16 bit.
- **E.g. MOV AX , BX**

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

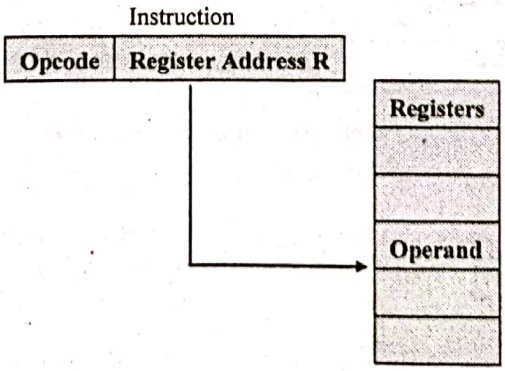


Fig. 2- Q. 6(b) : Register addressing mode

2. Immediate Operand Addressing Mode

- In this case the operand is in the instruction itself. It is said to be immediate addressing mode as the operand is in the immediate next location of the OPCODE. Fig. 3- Q. 6(b) shows format of instruction encoded with immediate operand.

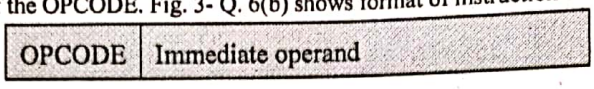


Fig. 3-Q. 6(b) : Instruction encoded with an immediate operand

- The operand in this case could be either 8-bit or 16-bit.
- **E.g. MOV CL, 02 H**

  This instruction copies the immediate number 02H in the CL register.

3. Memory Addressing Mode

- For memory accesses, the processor needs to generate a 20-bit address. The registers in 8086 are of 16-bit.
- In segmentation, 8086 produces the 20-bit address by special method i.e. segment register multiplied by 10H and adding to it the effective address.
- The effective address can either be a direct 16-bit address or can have various components i.e. base register value, index register value and the displacement. Based on the different combinations there are various addressing modes. Once we get EA (effective address), we can calculate PA (physical address) as,

$$PA = \text{Segment} : \text{Offset}$$

$$\Downarrow \qquad \Downarrow$$

$$= \text{Segment register} : EA$$

$$= \text{Segment register} : \text{BASE} + \text{INDEX} + \text{DISPLACEMENT}$$

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

- **Effective Address :** The address effective from the starting of the segment is called as the effective address. For example, if the effective address is 10, then it indicates that the location to be accesses is 10^{th} from the starting of the segment.

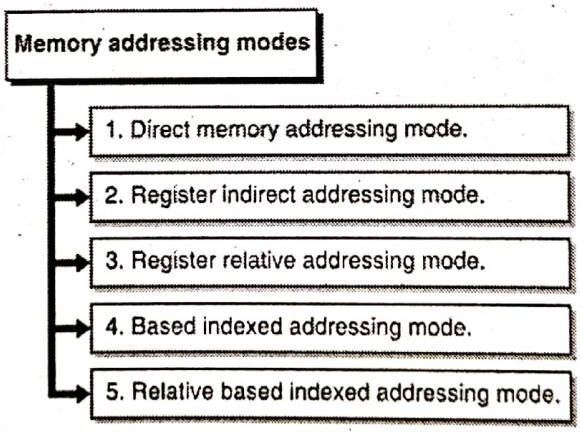


Fig. 4-Q. 6(b) : Memory addressing modes

(i) Direct Memory Addressing Mode

- In this mode, the 16-bit effective address EA is directly given in the instruction. The physical address is generated by adding this 16-bit direct address to segment register *10 H as shown in the Fig. 5-Q. 6(b).

$$PA = \text{segment} : EA$$

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

– **E.g. MOV [1023], AL**

The contents of AL are copied to memory location whose effective address is 1023H i.e. the physical address = DS *10H +1023.

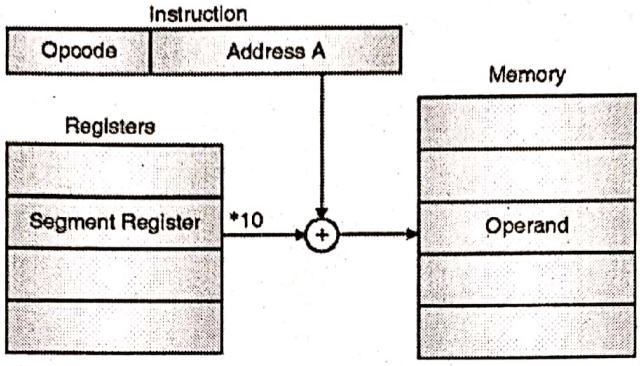


Fig. 5-Q. 6(b) : Direct addressing.

(ii) Register Indirect Addressing Mode

– In this addressing mode the effective address is given by a base register or an index register, specified in the instruction. This effective address is added with the segment register * 10H to generate the physical address as shown in Fig. 6-Q. 6(b).

$$\therefore EA = \begin{Bmatrix} (BX) \\ (BP) \\ (SI) \\ (DI) \end{Bmatrix}$$

Segment : EA

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

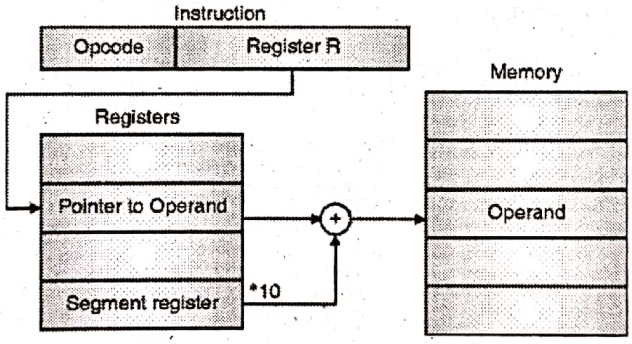


Fig. 6-Q. 6(b) : Register indirect addressing modes

– **E.g. MOV [SI], AL**

The contents of AL register are copied to memory location whose effective address is given by SI i.e. the physical address = DS *10H + SI.

(iii) Register Relative Addressing Mode

– In this addressing mode the effective address is given by a base register or index register along with an 8-bit displacement, specified in the instruction. This effective address is added with the segment register * 10H to generate the physical address as shown in Fig. 7-Q. 6(b).

$$\therefore EA = \begin{Bmatrix} (BX) \\ (BP) \\ (SI) \\ (DI) \end{Bmatrix} + \begin{Bmatrix} \text{8 bit displacement} \\ \text{(sign extended)} \\ \text{16 bit displacement} \end{Bmatrix}$$

$$\therefore \quad PA = \text{Segment} : EA$$

$$= \begin{Bmatrix} CS \\ ES \\ DS \\ SS \end{Bmatrix} : \begin{Bmatrix} (BX) \\ (BP) \\ (SI) \\ (DI) \end{Bmatrix} + \begin{Bmatrix} 8/16 \text{ bit} \\ \text{offset} \end{Bmatrix}$$

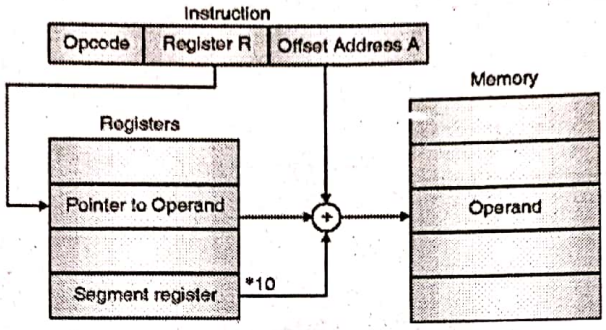


Fig. 7-Q. 6(b) : Register relative addressing mode

– **E.g. MOV [BX + 10], AL**

This instruction copies the contents of AL register to memory location whose effective address is given by BX + 10H i.e. the physical address = DS *10H + BX + 10H.

(iv) Based Indexed Addressing Mode

– In this addressing mode the effective address is given by a base register and an index register, specified in the instruction. This effective address is added with the segment register * 10H to generate the physical address as shown in Fig. 8-Q. 6(b).

$$\therefore EA = \{\text{Base register}\} + \{\text{Index register}\}$$

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

$$PA = \text{Segment register} : EA$$

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

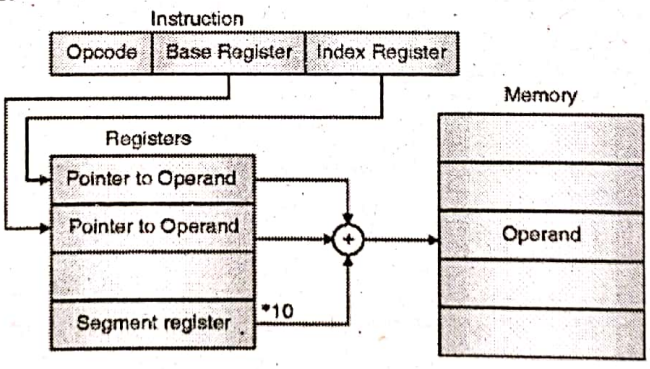


Fig. 8-Q. 6(b) : Based indexed addressing mode

– **E.g. MOV [BX + SI], AL**

This instruction copies the contents of AL register to memory location whose effective address is given by BX + SI i.e. the physical address = DS *10H + BX + SI.

(v) Relative Based Indexed Addressing Mode

- In this addressing mode the effective address is given by a base register and an index register along with an 8-bit displacement, specified in the instruction. This effective address is added with the segment register * 10H to generate the physical address as shown in Fig. 9-Q. 6(b).

$$\therefore EA = \{\text{Base register}\} + \{\text{Index register}\} + \left\{\begin{matrix}\text{8 bit displacement}\\ \text{(sign extended)}\\ \text{16 bit displacement}\end{matrix}\right\}$$

$$= \left\{\begin{matrix}(BX)\\(BP)\end{matrix}\right\} + \left\{\begin{matrix}(SI)\\(DI)\end{matrix}\right\} + \left\{\begin{matrix}\text{8/16 bit}\\ \text{displacement}\end{matrix}\right\}$$

$$PA = \text{Segment register :}$$

$$EA = \left\{\begin{matrix}CS\\SS\\DS\\ES\end{matrix}\right\} : \left\{\begin{matrix}(BX)\\(BP)\end{matrix}\right\} + \left\{\begin{matrix}(SI)\\(DI)\end{matrix}\right\} + \left\{\begin{matrix}\text{8/16 bit}\\ \text{displacement}\end{matrix}\right\}$$

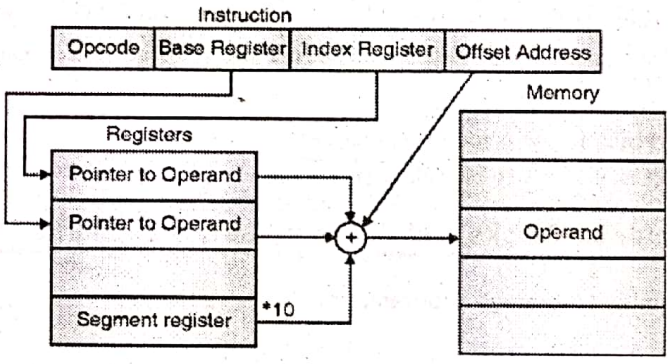


Fig. 9-Q. 6(b) : Relative Based Indexed Addressing Mode

- **E.g. MOV CX, [BX + SI + 0400]**

  This instruction copies the contents of AL register to memory location whose effective address is given by BX + SI + 04H i.e. the physical address = DS *10H + BX + SI + 04H.

4. String Addressing Mode

- String instructions use a different addressing mode wherein the pointers SI and DI along with segment registers DS and ES, respectively are used to access the source and destination memory locations.
- The memory location pointer by DS:SI is used as source while the memory location pointed by ES:DI is used as a destination.
- These pointers are also automatically incremented or decremented according to the value of Direction Flag.

5. Implied Addressing Mode

- The operand or reference to operand is not specified in the instruction. Instead the operand is obvious in the mnemonic or the instruction.
- E.g. XLAT

  CMA

  STC

  STD

6. I/O Addressing Mode

This addressing mode is basically used for IOs, it categorized in :

(a) Memory mapped I/O

(b) I/O mapped I/O

(a) Memory mapped I/O

- Memory mapped I/O refers to an I/O location mapped in memory i.e. given a memory address.
- In case of a memory mapped I/O device or I/O location, the benefit is that many instructions can access this data directly and hence giving a ease of access.
- The disadvantage of such I/O locations is that the access of I/O locations being normally slowers, it makes the processor to wait.

(b) I/O mapped I/O

If I/Os are mapped in ***I/O map I/O,*** then 8086 supports two different addressing modes :

(i) Direct port addressing (ii) Indirect port addressing

(i) Direct Port addressing : The address of I/O device or I/O location is given in the instruction itself. The limitation of this is that the direct address given in the instruction can be of a maximum of 8-bits and hence only 256 I/O locations can be accessed.

(ii) Indirect port addressing : Here a pointer register i.e. the register DX is used to give the address of the I/O location. Since the register DX is of 16-bit, 16-bit address supporting a huge range of 65536 I/O locations

Chapter 6 : Stacks and Subroutines [Total Marks - 15]

Q. 1(b) Write a short note on mixed language programming. (5 Marks)

Ans. :

- 'C' generates an object code that is extremely fast and compact, but it is not as fast as the object code generated by a good programmer using assembly language.
- There are special cases where a function is coded in assembly language to reduce execution time.
- **For Example :** The Floating Point math package must be coded in assembly language as it is used frequently and its execution speed will have great effect on the overall speed of the program that uses it. There are also occasions when some hardware devices need exact timing and then it is necessary to write assembly level programs to meet such strict timing restrictions.
- In addition, certain instructions cannot be executed in Higher Level Languages like C.
- **For Example :** C does not have an instruction for performing bit-wise rotation operation. Thus in spite of C being very powerful, routines must be written in assembly language to :
  - Increase the speed and efficiency of the routine.
  - Perform Machine specific functions not available in Microsoft C or in Turbo C.
  - Use third party routines.
- There are 2 ways of combining C and Assembly language.

(A) Method 1 :

- In this method Built-In-Inline assembler is used to include assembly language routines in the C-program, without any need for a specific assembler.
- Such assembly language routines are called in-line assembly.
- They are compiled right along with C Routines rather than being assembled separately and then linked together using linker modules provided by the C Compiler.
- Turbo C (TC) has inline assembler.

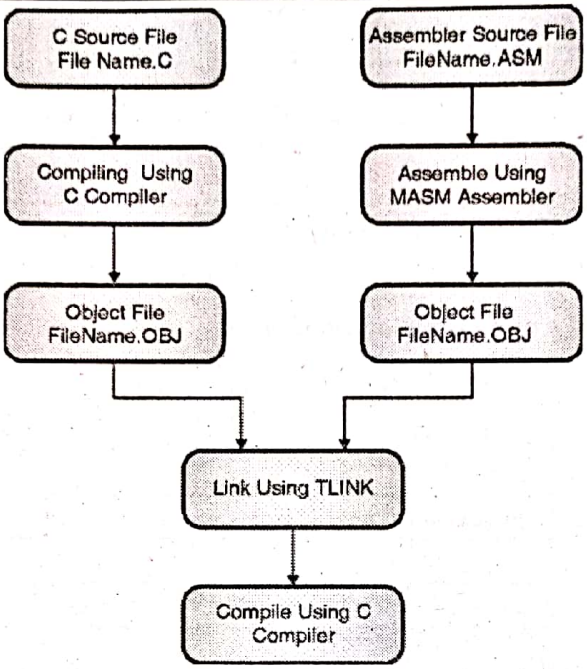


Fig. 1-Q.1(b) : Combining C and assembly

(B) Method 2

- There are times when programs written in one language have to call modules written in other languages. This is called as mixed language programming.
- **For Example :** when a particular sub-routine is available in a language, different from the language currently used in a program, or when algorithms are described in a different language, users need to use more than one language.
- Mixed language calls involve calling functions in separate modules.
- Instead of compiling all source programs using the same compiler, different compilers or assemblers are used as per the language used in the program.
- Microsoft C supports Mixed Language Programming.
- Therefore, it can combine assembly language routines in C as a separate language.
- C program calls assembly language routines that are separately assembled by MASM (MASM assembler) or TASM (Turbo assembler).
- These assembled modules are linked with the compiled C modules to get the combine executable file.
- Fig. 1-Q. 1(b) shows Compile, Assemble, and link processes using C compiler, MASM Assembler and TLINK.

Q. 5(a) Differentiate procedure and macro. Write a program to find the factorial of a number using procedure. **(10 Marks)**

Ans. :

(A) Differentiation between Procedure and Macro

Sr. No.	Procedure	Macro
1.	It resembles a call function of high level language. The processor branches to the procedure on call proc, instruction and returns back to the caller program after executing the procedure.	When the assembler comes across the instruction "CALL MACRO", it replaces this instruction with the group of instructions placed in the corresponding macro.
2.	Since the processor branches to another memory location and returns back, it consumes some time to store and fetch back the return address. Hence it has a latency period.	Macro does not required any latency period.

Sr. No.	Procedure	Macro
3.	Since the assembler stores the instructions of procedure only once in the memory, the program consumes less space in memory.	Since the assembler replaces all "Call macro" instruction by the group of instructions in the macro, the program consumes more space in memory.
4.	Procedures are to be used for repetitive task, if the task is very large (i.e. it has many instructions).	Macros are to be used for repetitive task; if the task is small (i.e. it has less number of instructions).

(B) Program to find factorial of a number using procedure

Label	Instruction	Comment
	.model small	
	.data	
	numdw 08h	
	.code	
	mov ax, @data	initialize data segment
	mov ds, ax	
	mov ax, 01	initialise ax=1
	mov bx, num	load the number in cx
	call fact	call procedure
	mov di, ax	store the lsb of result in di
	mov bp, 2	initialise count for no of times display is called
	movbx, dx	store msb of result in regbx
	movbx, di	store lsb of result in bx
	Dec bp	decrement bp
	mov ah, 4ch	
	int 21h	
	factproc near	function for finding the factorial
	cmpbx, 01	isbx=1?
	jz l11	if yes, ax=1
l12:	Mul bx	find factorial
	Dec bx	decrement bx
	cmp bx, 01	multiply till bx=1
	jne l12	
	Ret	
l11:	mov ax, 01	initialise ax=1
	Ret	return to called program
	fact endp	end procedure
	End	end program

Chapter 7:8086 Interrupt Structure [Total Marks - 10]

Q. 5(b) Explain the Interrupt structure of 8086 microprocessor. **(10 Marks)**

Ans. :

Interrupt structure of 8086 Microprocessor

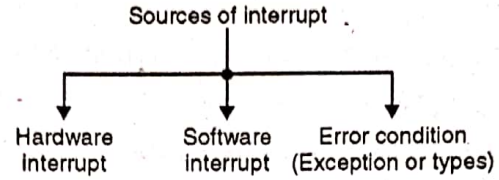


Fig. 1-Q. 5(b) : Interrupt structure of 8086 microprocessor

1. Hardware Interrupt

- In this type of interrupt, physical pins are provided in the chip. In 8086 there are two pins :
  - (i) NMI (Non maskable interrupt).
  - (ii) INTR.
- NMI is non maskable i.e. microprocessor has to service this interrupt, it cannot avoid it. Whereas INTR is maskable, if IF flag in flag register is '0', microprocessor will not recognise interrupt available on the pin.

2. Software Interrupt

Software interrupt, in 8086 we have INT instruction. When INT instruction is executed interrupt will occur.

3. Error Conditions (Exception Or Types)

- 8086 supports division, multiplication, addition etc. if user asks microprocessor to divide any number by ZERO, then you know that dividing any number by ZERO produces answer '**∞(infinity)**'.
- So in this case microprocessor will generate an interrupt **"Automatically"** and interrupt current execution. In ISR, user can display message **"Divide by zero error"**. Instead of showing the answer as "**∞(infinity)**".
- So internally generated errors produces an interrupt for microprocessor, normally referred as **"TYPE"** by Intel engineer and referred as **"Exception"** by motorola engineer.
- Thus 8086 has a simple and versatile interrupt system. Every interrupt is assigned a **"type code"** that identifies it to the CPU.
- The 8086 can handle upto 256 different interrupt types. Interrupts may be initiated by devices external to the CPU; in addition, they also may be triggered by software interrupt introductions and under certain condition, by the CPU itself. Fig.2-Q.5(b) shows interrupt sources for 8086.

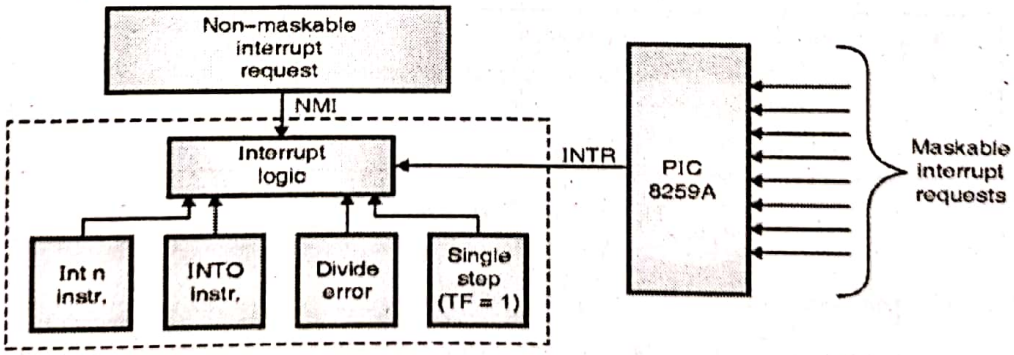


Fig. 2-Q. 5(b) : Interrupt sources

- In Fig. 2-Q. 5(b) 8086 have two lines that external device may use to signal interrupts. The INTR line is usually driven by an Intel 8259A (PIC), which in turn connected to the devices that need interrupt services.

Chapter 8: IC 8259 Programmable Interrupt Controller (PIC) [Total Marks - 05]

Q. 1(d) Give formats of initialization command words (ICW's) of 8259 PIC. **(5 Marks)**

Ans. :

The Fig.1-Q.1(d) below show the formats of ICWs.

1. ICW1

- ICW1 is compulsory as seen in Fig. 1-Q. 1(d). The address bit, A_0 must be '0' while giving the Initialization control word 1 to the 8259 chip.
- **A_7 to A_5(D_7 to D_5) :** The three MSBs i.e. D_7 to D_5 are required when interfaced with 8085. In case of 8086, these bits are not required for 8259 interfaced with 8086.
- **D_4 :** This bit should always be kept at logic '1'.
- **LTIM (D_3) :** This bit is used t indicate the interrupts are to be level triggered or edge triggered. If this bit is kept at '1' then the interrupts IR0 to IR7 are level triggered else they are edge triggered.

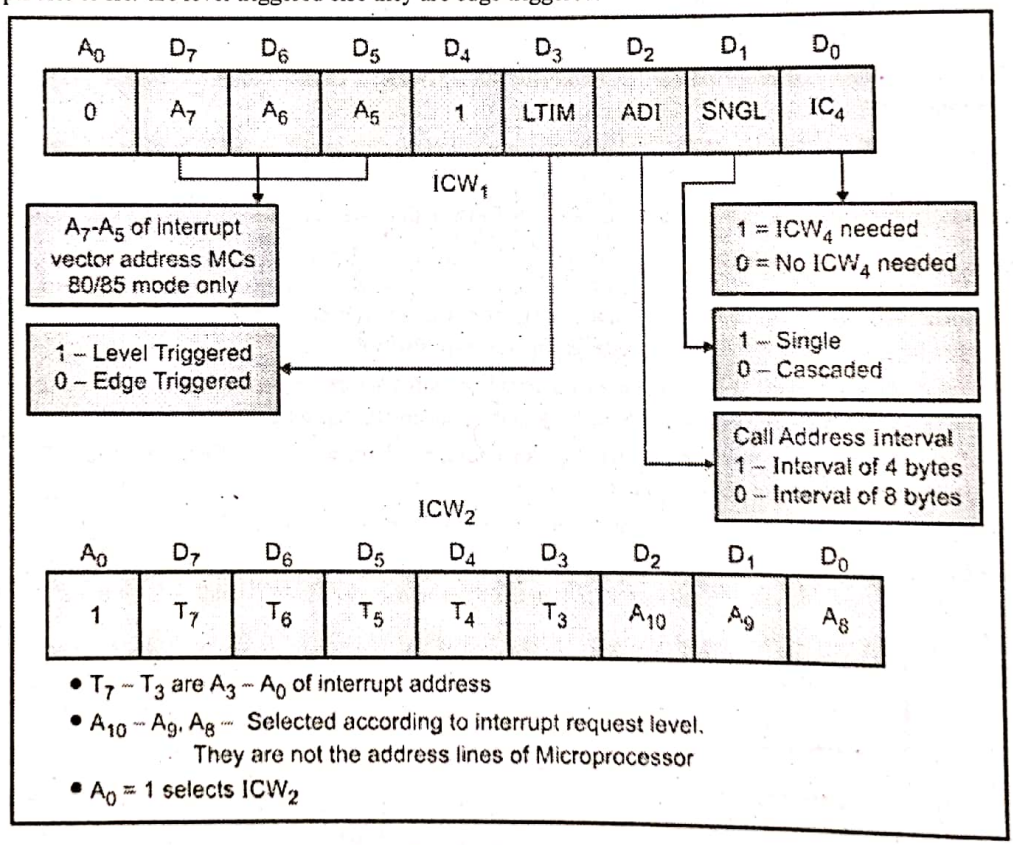


Fig.1-Q.1(d) : Format of ICWs

- **ADI(D_2) :** This bit is again required for 8085 and not required for 8086
- **SNGL (D_1) :** This bit is used to indicate the 8259 is in single mode or cascaded mode. If this bit is '1', then 8259 is in single mode else it is in cascaded mode. If in cascaded mode then ICW3 will be required.
- **IC4 (D_0) :** This bit indicates the requirement of ICW4.

2. ICW2

- ICW2 is compulsory as seen in Fig.1-Q.1(d). The address bit, A_0 must be '1' while giving the Initialization control word 2 to the 8259 chip.
- **T_7 to T_3 (D_7 to D_3) :** The five bits of ICW2 are used to indicate the interrupt type to be given to 8086 when an interrupt occurs on a pin of 8259. The last three bits to make the interrupt type eight bit value are taken as 000 for IR0 to 111 for IR7. Hence the interrupt type corresponding to a particular interrupt on 8259 pin is generated by taking the five bits T_7 to T_3 and concatenated with the three bits 000 to 111.
- **D_2 to D_0 :** These bits are not required when interfaced with 8086.

3. ICW3

- ICW3 is required on in cascaded mode as seen in Fig. 2-Q.1(d). The address bit, A_0 must be '1' while giving the Initialization control word 3 to the 8259 chip. ICW3 has a different structure for both master 8259 as well as slave 8259.
- In the Fig. 2(a)-Q.1(d) for master, each of the bit is used to indicate whether a slave is connected of the corresponding interrupt request (IR) pin or not. A '1' indicates that a slave is connected to the corresponding Interrupt request pin, while a '0' indicates no slave is connected.
- For slave, the ICW3 contains the ID of the device. This is required for the slave to compare when the acknowledgement is given by the processor, to realize whether the acknowledgement is meant for the same slave or another slave. The ID is provided by the master 8259 on receipt of acknowledgement from the microprocessor on the cascaded pins.

Master mode ICW_3

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

Sn = 1 – IRn Input has a slave
= 0 – IRn Input does not have a slave

(a)

Slave mode ICW_3

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	ID_2	ID_1	ID_0

$D_2D_1D_0$ - 000 to 111 for IR_0 to IR_7 or slave 1 to slave 8

(b)

ICW_4

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	SFNM	BUF	M/S	AEOI	µPM

(c)

Fig. 2-Q.1(d)

4. ICW4

- ICW4 is required if IC4 bit of ICW1 was made '1' as seen in Fig.2(c)-Q.1(d). The address bit, A_0 must be '1' while giving the Initialization control word 4 to the 8259 chip.
- **D_7 to D_5 :** The three MSBs i.e. D_7 to D_5 are not required for 8259 interfaced with 8086.

- **SFNM(D_4) :** This bit indicates whether the 8259 has to operate in SFNM mode or FNM mode. If this bit is '1', then 8259 has to operate in SFNM mode, else FNM mode.
- **BUF and M/S (D_3) :** These bits are used to indicate whether 8259 is in buffered mode or not. In case of buffered mode, M/S indicates it's a master or a slave. If the BUF bit is '1' and If the M/S bit is '1', it indicates buffered master, else buffered slave.
- **AEOI(D_1) :** This bit is used to indicate whether Automatic end of interrupt (EOI) or normal end of interrupt.
- **µPM (D_0) :** This bit is used to indicate the 8259 is connected to 8085 or 8086. If this bit is '1' then 8259 is connected to 8086 else connected to 8085.

Chapter 9: 8255 Programmable Peripheral Interface [Total Marks - 05]

Q. 4(b)(i) Explain the I/O mode control word format of 8255 PPI. **(5 Marks)**

Ans.:

There are three I/O modes of operation

1. Mode 0 - Basic I/O
2. Mode 1 - Strobed I/O
3. Mode 2 - Bi-directional I/O

The I/O modes are programmed using control register. The control word format of I/O modes is shown in Fig. 1- Q. 4(b).

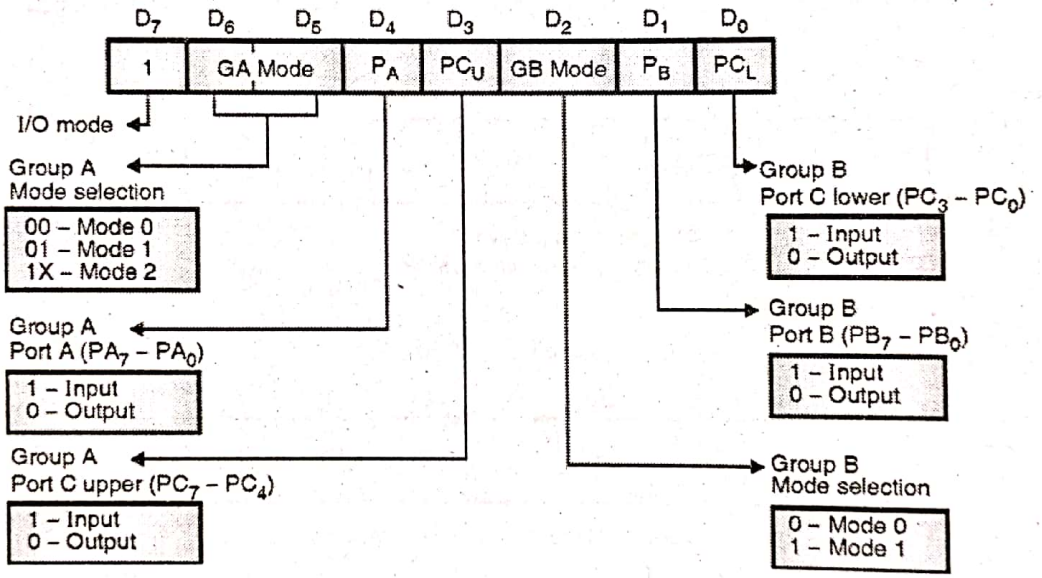


Fig. 1- Q.4(b) : I/O modes control word format

Function of each bit is as follows :

1. **D_7 :** When the bit D_7= 1 then I/O mode is selected, if D_7 = 0 then BSR mode is selected. The function of bits D_0 to D_6 is dependent on mode (I/O mode or BSR mode).
2. **D_6 and D_5 :** In I/O mode the bits D_6 and D_5 specifies the different I/O modes for group A i.e. Mode 0, Mode 1 and Mode 2 for port A and port C upper.
3. **D_4 and D_3 :** In I/O mode the bits D_4 and D_3 selects the port function for group A. If these bits = 1 the respective port specified is used as input port. But if bit = 0, the port is used as output port.
4. **D_2 :** In I/O mode the bit D_2 specifies the different I/O modes for group B i.e. Mode 0 and Mode 1 for port B and port C lower.
5. **D_1 and D_0 :** In I/O mode the bits D_1 and D_0 selects the port function for group B. If these bits = 1 the respective port specified is used as input port. But if bit = 0, the port is used as output port.

Chapter 11: 8257 DMAC [Total Marks - 10]

Q. 3(b) Draw and explain the block diagram of 8257 DMA controller. **(10 Marks)**

Ans. :

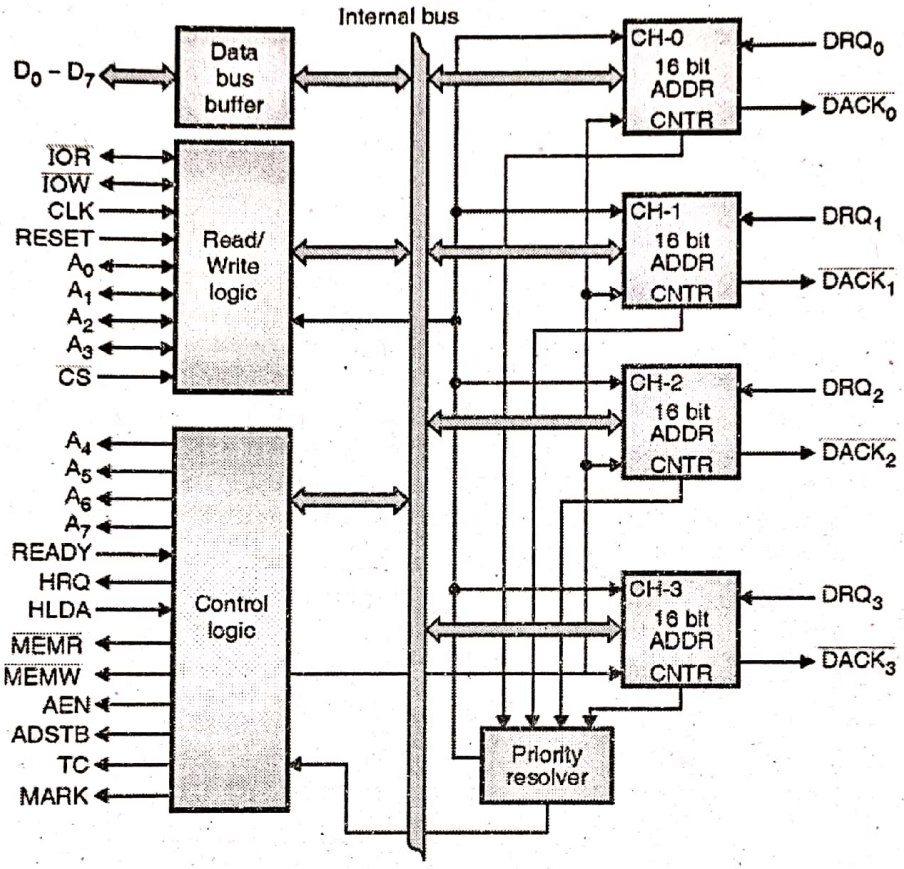


Fig. 1- Q. 3(b) : 8257 Functional Block Diagram

1. DMA Channels

- The 8257 provides four separate DMA channels (labeled CH-0 to CH-3).
- Each channel includes two sixteen-bit registers.
  - (i) DMA address registers
  - (ii) Terminal count register . Both registers must be initialized before a channel is enabled.
- The value loaded into the low-order 14-bits of the terminal count register specifies the number of DMA cycles minus one before the Terminal Count (TC) output is activated. For instance, a terminal count of 0 would cause the TC output be active in the first DMA cycle for that channel.
- If N = the number of desired DMA cycles, load the value N-1 into the low-order 14-bits of the terminal count register.
- The most significant two bits of the terminal count register specify the type of DMA operation for that channel.
- Each channel accepts a DMA Request (DRQn) input and provides a DMA Acknowledge (DACKn) output.

(a) DMA request (DRQ 0 – DRQ 3)

- These are individual asynchronous channel request inputs used by the peripherals to obtain a DMA cycle.
- If not in the rotating priority mode than DRQ 0 has the highest priority and DRQ 3 has the lowest.
- For multiple DMA cycles (Burst mode) the request line is held high until the DMA acknowledge of the last cycle arrives.

(b) DMA acknowledge ($\overline{DACK\ 0 - DACK\ 3}$)

- An active low level on the acknowledge output informs the peripheral connected to that channel that it has been selected for a DMA cycle.
- The $\overline{DACK}$ output acts as a "chip select" for the peripheral device requesting service.
- This line goes active (low) and inactive (high) once for each byte transferred even if a burst of data is being transferred.

2. Data Bus Buffer

This three-state, bi-directional, eight bit buffer interfaces the 8257 to the system data bus.

(a) Data Bus Lines ($D_0 - D_7$)

- These are bi-directional three-state lines. When the 8257 is being programmed by the CPU, eight – bits of data for a DMA address register a terminal count register or the Mode Set register are received on the data bus.
- When the CPU reads a DMA address register, a terminal count register or the Status register, the data is sent to the CPU over the data bus.
- During DMA cycles (when the 8357 is the bus master), the 8257 will output the most significant eight-bits of the memory address (from one of the DMA address registers) to the memory address (from one of the DMA address registers) to the 8212 latch via the data bus.
- These address bits will be transferred at the beginning of the DMA cycle; the bus will then be released to handled the memory data transfer during the balanced of the DMA cycle.

BIT 15	BIT 14	Type of DMA Operation
0	0	Verity DMA Cycle
0	1	Write DMA Cycle
1	0	Read DMA Cycle
1	1	(Illegal)

3. Read / Write Logic

- When the CPU is programming or reading one of the 8257's registers (i.e., when the 8257 is a "slave" device on the system bus), the Read / Write Logic accepts the I/O.
- Read ($\overline{I/OR}$) Write ($\overline{I/QW}$) signal, decodes the least significant four address bits, ($A_0 - A_3$), and either writes the contents of the data bus into the addressed register (if $\overline{I/OW}$ is true) or places the contents of the addressed register onto the data bus (if $\overline{I/OR}$ is true).
- During DMA cycles (i.e., when the 8257 is the bus "master"). The Read / Write Logic accepts the I/O read and memory write (DMA write cycle) or I/ O Write and memory read (DMA read cycle) signals which control the data link with the peripheral that has been granted the DMA cycle.
- During DMA transfers Non-DMA I / O devices should be de-selected (disabled) using "AEN" signal to inhibit I / O device decoding of the memory address as an erroneous device address.

(a) I / O Read ($\overline{I/OR}$)

- An active-low, bi-directional three-state line. In the 'slave' mode. It is an input which allows the 8-bit status register or the upper/lower byte of a 16-bit DMA address register or terminal count register to be read.
- In the 'master' mode. $\overline{I/OR}$ is a control output which is used to access data from a peripheral during the DMA write cycle.

(b) **I/O write ($\overline{I/OW}$)**

- An active-low, bi-directional three-state line.
- In the 'slave' mode, it is an input which allows the contents of the data bus to be loaded into the 8-bit mode set register or the upper/lower byte of a 16-bit DMA address register or terminal count register.
- IN the 'master' mode, ($\overline{I/OW}$) is a control output which allows data to be output to a peripheral during a DMA read cycle.

(c) **Clock input (CLK) :** Generally from an Intel® 8224 clock generator device. (ϕ 2 TTL) or Intel® 8085A CLK output.

(d) **Reset (RESET) :** An asynchronous input (generally from an 8224 or 8085 device) which disables all DMA channels by clearing the mode register and 3-states all control lines.

(e) **Address lines ($A_0 - A_3$) :** These least significant four address lines are bi-directional. In the 'slave' mode they are inputs which select one of the registers to be read or programmed. In the 'master' mode, they are outputs which constitute the least significant four bits of the 16-bit memory address generated by the 8257.

(f) **Chip select ($\overline{CS}$) :** An active-low input which enables the I/O read or I/O write input when the 8257 is being read or programmed in the 'slave' mode. In the 'master' mode. $\overline{CS}$ is automatically disabled to prevent the chip from selecting itself while performing the DMA function.

4. Control logic

This block controls the sequence of operations during all DMA cycles by generating the appropriate control signals and the 16-bit address that specifies the memory location to be accessed.

(a) **Address lines ($A_4 - A_7$) :** These four address lines are three-state outputs which constitute bits 4 through 7 of the 16-bit memory address generated by the 8257 during all DMA cycles.

(b) **Ready (READY) :** This asynchronous input is used to elongate the memory read and write cycles in the 8257 with wait states if the selected memory requires longer cycles. READY must conform to specified setup and hold times.

(c) **Hold acknowledge (HRQ) :** This input from the CPU indicates that the 8257 has acquired control of the system bus.

(d) **Memory read ($\overline{MEMR}$) :** This active-low three-state output is used to read data from the addressed memory location during DMA read cycles.

(e) **Memory write ($\overline{MEMW}$) :** This active-low three-state output is used to write data into the addressed memory location during DMA write cycles.

(f) **Address strobe (ADSTB) :** This output strobes the most significant byte of the memory address into the 8212 device from the data bus.

(g) **Address enable (AEN) :** This output is used to disable (float) the system data bus and the system control bus.

- It also be used to disable (float) the system address bus by use of an enable on the address bus drivers in systems to inhibit non-DMA devices from responding during DMA cycles.
- It is also used to isolate the 8257 data bus from the system data bus to facilitate the transfer of the most significant DMA address bits over the 8257 data I/O pins without subjecting the system data bus to any timing constraints for the transfer.
- When the 8257 is used in an I/O device structure as opposed to memory mapped, this AEN output should be used to disable the selection of an I/O device when the DMA address is on the address bus.
- The I/O device selection should be determined by the DMA acknowledge outputs for the 4 channels.

(h) **Terminal count (TC) :** This output notifies the currently selected peripheral that the present DMA cycle should be the last cycle for this data block.

- If the TC STOP bit in the mode set register is set, the selected channel will be automatically disabled at the end of that DMA cycle.
- TC is activated when the 14-bit value in the selected channel's terminal count register equals zero.

- Recall that the low-order 14-bits of the terminal count register should be loaded with the values (n–1). Where n = the desired number of the DMA cycles.

(i) **Modulo 128 mark (MARK) :** This output notifies the selected peripheral that the current DMA cycle is the 128^{th} cycle since the previous MARK output.

5. Mode set register

- When set, the various bits in the mode set register enable each of the four DMA channels and four different options for the 8257 :

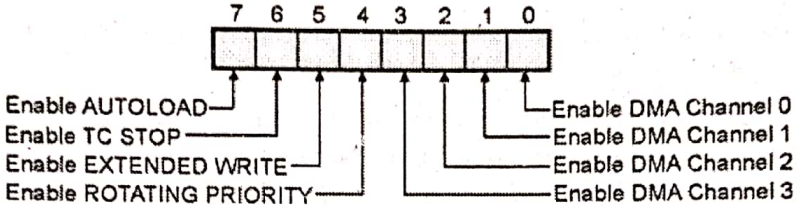


Fig. 2- Q. 3(b) : Mode register

- The mode set register is normally programmed by the CPU after the DMA address register(s) and terminal count register(s) are initialized.
- The mode set register is cleared by the RESET input, thus disabling all options, inhibiting all channels and preventing bus conflicts on power-up.
- A channel should not be left enabled unless its DMA address and terminal count registers contain valid values; otherwise, an inadvertent DMA request (DRQn) from a peripheral could initiate a DMA cycle that would destroy memory data.

6. Status register

- The eight-bit status register indicates which channels have reached a terminal count condition and includes the update flag described previously.

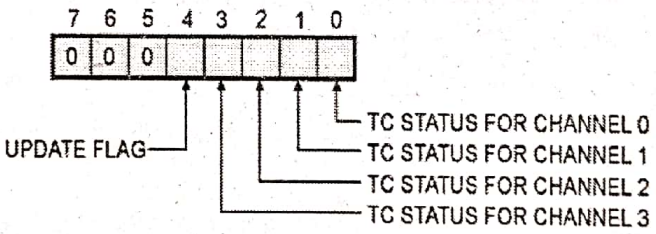


Fig. 3- Q. 3(b) : Status Register

- The TC status bits are set when the terminal count (TC) output is activated for that channel.
- These bits remain set until the status register is read or the 8257 is reset. The UPDATE FLAG, where it is not affected by a status register read operation.
- The UPDATE FLAG can be cleared by resetting the 8257, by changing to the non-auto load mode (i.e., by resetting the AUTO LOAD bit in the mode set register) or it can be left to clear itself at the Completion of the update cycle.
- The purpose of the UPDATE FLAG is to prevent the CPU from inadvertently skipping a data block by overwriting a starting address on terminal count in the channel 3 registers before those parameters are properly auto-loaded into channel 2.
- The user is cautioned against reading the TC status register and using this information to re-enable channels that have not completed operation.
- Unless the DMA channels are inhibited a channel could reach terminal count (TC) between the status read and the mode write.
- DMA can be inhibited by a hardware gate on the HRQ line or by disabling channels with a mode word before reading the TC status.

Chapter 14 : Intel 80386DX Processor [Total Marks - 15]

Q. 1(c) Explain flag register of 80386 microprocessor. (5 Marks)

Ans. :

The flag register of the x86 family is as shown in the Fig. 1- Q.1(c)

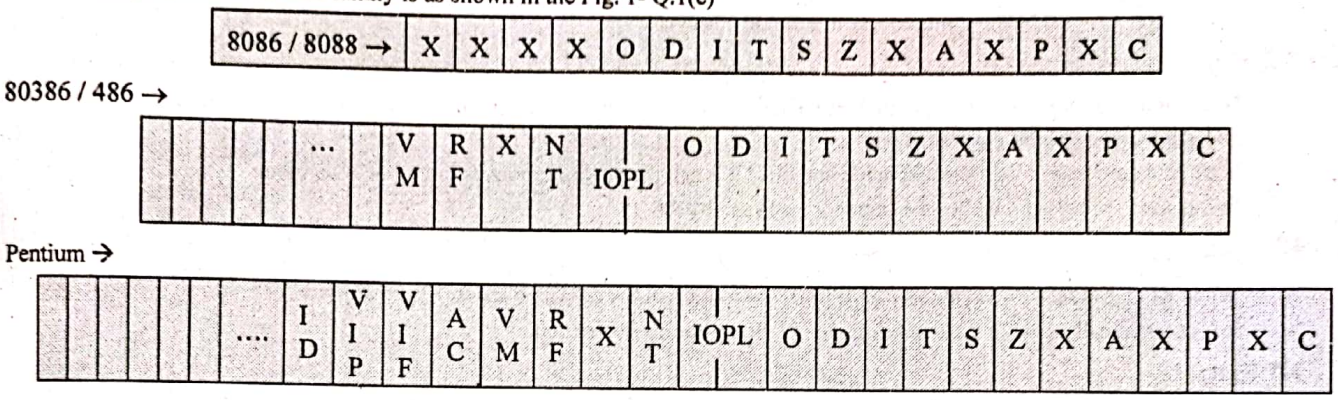


Fig. 1- Q.1(c) : Flag Register

Identification Flag (ID)	**System Flag**
Virtual Interrupt Pending (VIP)	**System Flag**
Virtual Interrupt Flag (VIF)	**System Flag**
Alignment Check (AC)	**System Flag**
Virtual 8086 Mode (VM)	**System Flag**
Resume Flag (RF)	**System Flag**
Nested Task (NT)	**System Flag**
I/O Privilege Level (IOPL)	**System Flag**
Overflow Flag (OF)	**Status Flag**
Direction Flag (DF)	**Control Flag**
Interrupt Enable Flag (IF)	**System Flag**
Trap Flag (TF)	**System Flag**
Sign Flag (SF)	**Status Flag**
Zero Flag (ZF)	**Status Flag**
Auxiliary Carry (AF)	**Status Flag**
Parity Flag (PF)	**Status Flag**
Carry Flag (CF)	**Status Flag**

1. ID Flag

The ability to the programmer to set and reset the ID flag indicates that the processor supports the CPUID instruction.

2. VIP Flag

The VIP is used to indicate about a pending interrupt when another interrupt was in service during the operation of Pentium processor in virtual mode.

3. VIF Flag

The VIF is a virtual image of the IF flag. It is used to enable or disable the interrupt when the Pentium processor is operating in virtual mode.

4. AC Flag

- Setting the AC flag and the AM bit in the control register 0 (CR0) enables alignment checking on memory references.
- An alignment check exception is generated when a reference is made to an unaligned operand.
- Unaligned accesses are those wherein the entire operand is not in the same row in the memory banks i.e. the operand is divided into multiple rows and hence requires multiple bus cycles.
- To have efficient memory accesses, words, double words and quad words of data must be stored at what is called as aligned boundaries. Aligned data permits access in a single bus cycle.

5. VM Flag

- When the processor enters in virtual 8086 mode, which is an emulation of the programming environment of the 8086 microprocessor, this bit is set to '1'.
- When the processor is in protected mode and this bit is set, the processor moves to virtual 8086 mode.
- In this mode processor handles the segment loads as in 8086.

6. RF Flag

- When RF = 1, it ignores the debug exception on execution of the next instruction.
- It is automatically reset at the successful completion of every instruction.

7. NT Flag

- If NT = 1, it indicates that the currently executing task is nested within another task.
- It has a valid link to caller task i.e. this task is executed using the call instruction.

8. IOPL Flag

- The IOPL encoded values indicates the privilege level at which the task should be executed to access the I/O device.
- Privilege levels are used in protected mode to maintain multiple tasks being executed at different privileges and hence it can access things accordingly.

9. OF Flag

- The OF = 1 indicates that the operation resulted in signed overflow.
- Sign overflow occurs when a operation results in carry / borrow in the sign bit but doesn't result in a carry / borrow out of the high order bit or vice-versa.

10. DF Flag

- DF defines whether ESI and/or EDI registers are auto-incremented or auto-decremented during the execution of string instructions.
- Auto-increment occurs if DF = 0; auto-decrement occurs if DF = 1.

11. IF Flag

- When IF = 1, it allows recognition of external maskable interrupt INTR pin.
- When IF = 0, external maskable interrupt on INTR are not recognized.

12. TF Flag

- When TF = 1, the processor is put into single-step mode used for debugging.
- In this mode, processor generates a single stepping interrupt after each instruction, which allows a program to be inspected as it executes.

13. SF Flag

- SF = 1, if the MSB of the result is 1, i.e. the result is negative in case of a signed operation.
- SF copies the MSB i.e. the bit 7, 15, 31, for 8-, 16-, and 32-bit operations respectively.

14. ZF Flag

The ZF = 1 only if all bits of the result are zero; else ZF = 0.

15. AF

- Auxiliary Carry Flag also called as half way carry and is used for BCD operations.
- For 8-, 16-, or 32-bit operations, AF is set according to the carryout of bit 3 in each case.

16. PF Flag

- The PF = 1, if the lower 8 bits of the operation contain an even number of 1s (i.e. even parity).
- The PF = 0, if the lower 8 bits have odd parity.

17. CF Flag

- The CF = 1, if the operation resulted in a carryout of the MSB; else CF = 0.
- For 8-, 16-, or 32-bit operations, CF is set according to the carryout of bit 7, 15, or 31, respectively.

Q. 4(a) Explain the modes of operation of 80386 microprocessor ? **(10 Marks)**

Ans. :

The processing modes of the 80386 also determine the features that are accessible. There are three processing modes

1. Real - Address Mode. 2. Virtual 8086 Mode
3. Protected Mode.

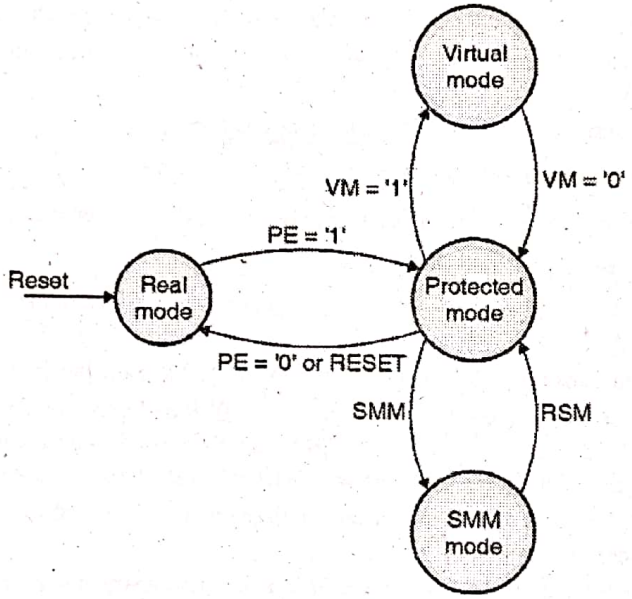


Fig. 1- Q. 4(a)

1. Real Mode and Virtual Mode

- Real address mode (often called just "real mode") is the mode of the processor immediately after RESET.
- In real mode the 80386 will appear to programmers as a fast 8086 with some new instructions.
- Most applications of the 80386 will use real mode for initialization only.
- Virtual 8086 mode (also called V86 mode) is a dynamic mode in the sense that the processor can switch repeatedly and rapidly between V86 mode and protected mode.
- The CPU enters V86 mode from protected mode to execute an 8086 program, then leaves V86 mode and enters protected mode to continue executing a native 80386 program.
- The features that are available to applications programs in protected mode and to all programs in V86 mode are the same.
- The 80386 provides a 1 Mbyte + 64 Kbytes – 16 Bytes memory space for an 8086 program.
- Memory location access is performed as in the 8086, i.e. the 16 – bit value in a segment selector is shifted left by four bits to form the base address of a segment.
- But, unlike the 8086, the resulting linear address may have up to 21 significant bits.
- There is a possibility of a carry when the base address is added to the effective address. On the 8086, the carried bit is truncated, whereas on the 80386 the carried bit is stored in bit position 20 of the linear address.

2. Protected Mode

- This mode is mainly meant for multi tasking operations.
- Multiple tasks running simultaneously using separate code, data and stack segment.
- It also takes care of proper authentication of a task to access a particular segment.

Chapter 15 : Intel P5 Microarchitecture [Total Marks :25]

Q. 3(a) Explain the branch prediction logic used in Pentium processor. **(10 Marks)**

Ans. :

- BPL (Branch Prediction Logic) reduces the flushing problem of pipelining and avoids pipeline and EU stall if branching prediction done is correct. Else if predicted wrong then there is a performance penalty.
- If predicted wrong for a branching instruction in 'U' pipeline a penalty of three cycle is incurred while a 4-cycle penalty is incurred in case predicted wrong for branch instruction in 'V' pipeline, for an unconditional jump.
- If conditional jump or call instruction is predicted wrong in either pipeline the 3-clock penalty is incurred.
- Branch prediction Mechanism is implemented using look aside set associative type cache with 256 entries. This cache is referred to as the branch target buffer (BTB).

  The directory for each branch instruction contains the following information :

  1. A valid bit that indicates whether the entry is in use.
  2. Two history bits that track how often the branch has been taken each time that it entered the pipeline before.
  3. Memory address that the branch instruction was fetched form.
- If the respective directory entry is valid, the target address of the branch is also stored in the corresponding data entry in the branch target buffer.
- The BTB is a look-aside cache that sits off to the side of D1 stage of two pipeline and monitors for branch instructions.
- The first time a branch instruction is encountered in either pipeline, the BTB performs look up in the cache. Since instruction has not been seen before, it results in a BTB miss. This means that BTB has no history on this instruction and hence predicts that branch will not be taken (even in case of unconditional jumps) and hence prefetcher is instructed to continue fetching sequentially.
- If the execution unit decides the branch is to be taken, the next instruction to be executed should be the one fetched from the branch target address else sequential execution continues.
- The EU gives a feedback to the BPL for record updating in BTB. A directory entry is made containing the source memory address and history bits to indicate branch is taken 100% times (in this case), and the corresponding target address is stored in the data entry field of the BTB.

  The history bits indicates one of four possible states :

1. Strongly taken

- The history bits are marked to this first time an entry is made and branching is done.
- If the state is here and EU indicates that the branching is taken then it remains here.
- If the state is here and EU indicates that the branching is not taken then is degraded to weakly taken.

2. Weakly taken

- If the state is here and EU indicates that the branching is taken then is upgraded to Strongly taken.
- If the state is here and EU indicates that the branching is not taken then is degraded to weakly not taken.

3. Weakly not taken

- If the state is here and EU indicates that the branching is taken then is upgraded to Weakly taken.
- If the state is here and EU indicates that the branching is not taken then is degraded to Strongly not taken.

4. Strongly not taken

- If the state is here and EU indicates that the branching is taken then is upgraded to Weakly not taken.
- If the state is here and EU indicates that the branching is not taken then remains here.

- If the branch was correctly predicted to be taken history bits are upgraded and no further action necessary i.e. correct instructions are already in the pipeline.
- If branch was incorrectly predicted to be taken, the history bits are downgraded and the pipeline needs to be flushed and switching of prefetcher queue takes place.
- If the branch was correctly predicted not to be taken, history bits are downgraded and no action required.
- If incorrectly predicted not to be taken then history bits are upgraded and the queue is flushed and instructions fetched from previous prefetch queue that contains sequential instructions. Hence time is saved because there are two prefetch queues.

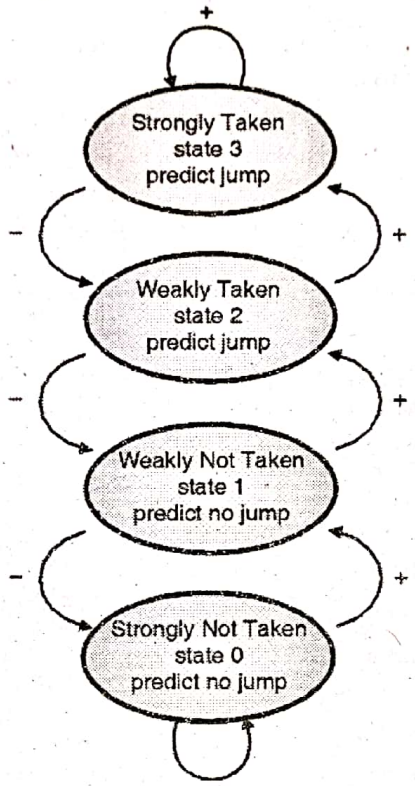


Fig. 1-Q. 3(a) : State transition diagram of BTB entry

Q. 4(b)(ii) Explain an instruction issue algorithm of Pentium processor. (5 Marks)

Ans. :

- The instructions of Pentium processor are pairable if the following rules are followed.
- The Pentium processor incorporates 2 integer pipeline designated as 'U' and 'V' pipelines.
- 'U' pipeline is the primary pipeline and its execution unit incorporates a barrel shifter while 'V' pipeline execution unit lacks this.
- Only simple instructions can be executed in 'V' pipeline while all other instructions are executed in 'U' pipeline
- Simple instructions are the ones that take 2 or 3 clock cycles only.
- The list of some simple instructions are :

MOV reg, reg / mem / imm
MOV mem, reg / imm
ALU reg, reg / mem / imm
ALU mem, reg / imm
(ALU instructions refer to ADD, AND, CMP, OR, TEST, XOR, etc.)
INC reg / mem
DEC reg / mem
PUSH reg / mem
POP reg

LEA reg / mem

JMP / CALL / Jconditional near

NOP

- Both pipelines are supplied by a steady stream of instructions by the prefetcher, which in turn is supplied by the code cache.
- Since 'V' pipeline has no barrel shifter, some instruction can execute only in 'U' pipeline.
- The prefetch queue in use delivers the first instruction to the 'U' pipeline while next to the 'V' pipeline.
- Instructions are pairable only if :
  1. Both instructions are simple
  2. Instruction must not have register contention
- When the instruction is fetched for the first time, the size is taken as one byte.
- When multibyte instruction is executed for the first time, the D1 stage provides a feedback to the code cache about instruction length.
- The boundary information of these instructions is then stored in the cache directory. Next time when code cache gives a line it also provides the prefetcher with the information about the instruction boundary.
- Based on this problem we can formulate the instruction issue algorithm are formulated as below :
  1. Decode two consecutive instructions I1 and I2
  2. If all the following conditions are true the two instructions are pairable and issue I1 to 'U' pipeline and I2 to 'V' pipeline; else they are not pairable, and only the instruction I1 is to be given to 'U' pipeline. The conditions are:
     (a) I1 and I2 are simple instructions
     (b) I1 is not a jump instruction
     (c) Destination of I1 is not the source of I2
     (d) Destination of I1 is not the destination of I2

Q. 2(b) Design 8086 based system for following specifications :
(i) 8086 in minimum mode with clock frequency 5MHz.
(ii) 64 KB EPROM using 16 KB*4 chips
(iii) 16 KB RAM using 8 KB*2 chips **(10 Marks)**

Ans.:

Step 1 : Total EPROM required = 64 KB

Chip size available = 16 KB (IC 27128 i.e. 16 KB EPROM)

$$\text{Number of chips required} = \frac{64\text{ KB}}{16\text{ KB}} = 4$$

$$\text{Number of sets required} = \frac{\text{No . of Chips}}{\text{No . of Banks}}$$

$$= \frac{4}{2} = 2$$

SET 1 : Ending address of SET 1 = FFFFFH

SET size = Chip size × 2

= 16 KB × 2 = 32 KB

$32\text{ KB} \Rightarrow 2^{15}$

i.e. $\frac{0000}{0}\ \frac{0111}{7}\ \frac{1111}{F}\ \frac{1111}{F}\ \frac{1111}{F}$

Starting address = Ending address – SET size.

= FFFFFH – 07FFF = F8000H

SET 1 : Ending address of SET 1 = F7FFFH

SET size = Chip size × 2

= 16 KB × 2 = 32 KB

32 KB $\Rightarrow 2^{15}$

i.e. $\frac{0000}{0}\ \frac{0111}{7}\ \frac{1111}{F}\ \frac{1111}{F}\ \frac{1111}{F}$

i.e. $\frac{0000}{0}\ \frac{0011}{3}\ \frac{1111}{F}\ \frac{1111}{F}\ \frac{1111}{F}$

Starting address = Ending address – SET size.

= F7FFFH – 07FFF = F0000H

		Even bank	Odd bank
ROM SET 1	Starting Address	F8000H	F8001H
	Ending Address	FFFFEH	FFFFFH
ROM SET 2	Starting Address	F0000H	F0001H
	Ending Address	F7FFEH	F7FFFH

Step 2 : Total SRAM required = 16 KB

Chip size available = 8 KB

∴ No. of chips = 2 chips

∴ No. of sets = 1 set.

Set 1 : Starting address = 00000H

Set size = 8 KB * 2 = 16 KB

$\Rightarrow$ 03FFFH.

∴ Ending address = 03FFFH

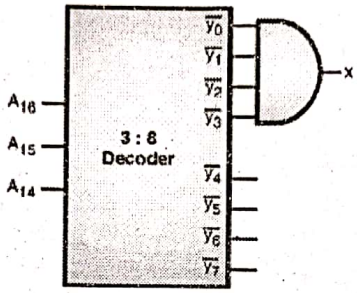
	Even Bank	Odd Bank
Starting Address	00000H	00001H
Ending Address	03FFEH	03FFFH

Step 3 : Memory Map :

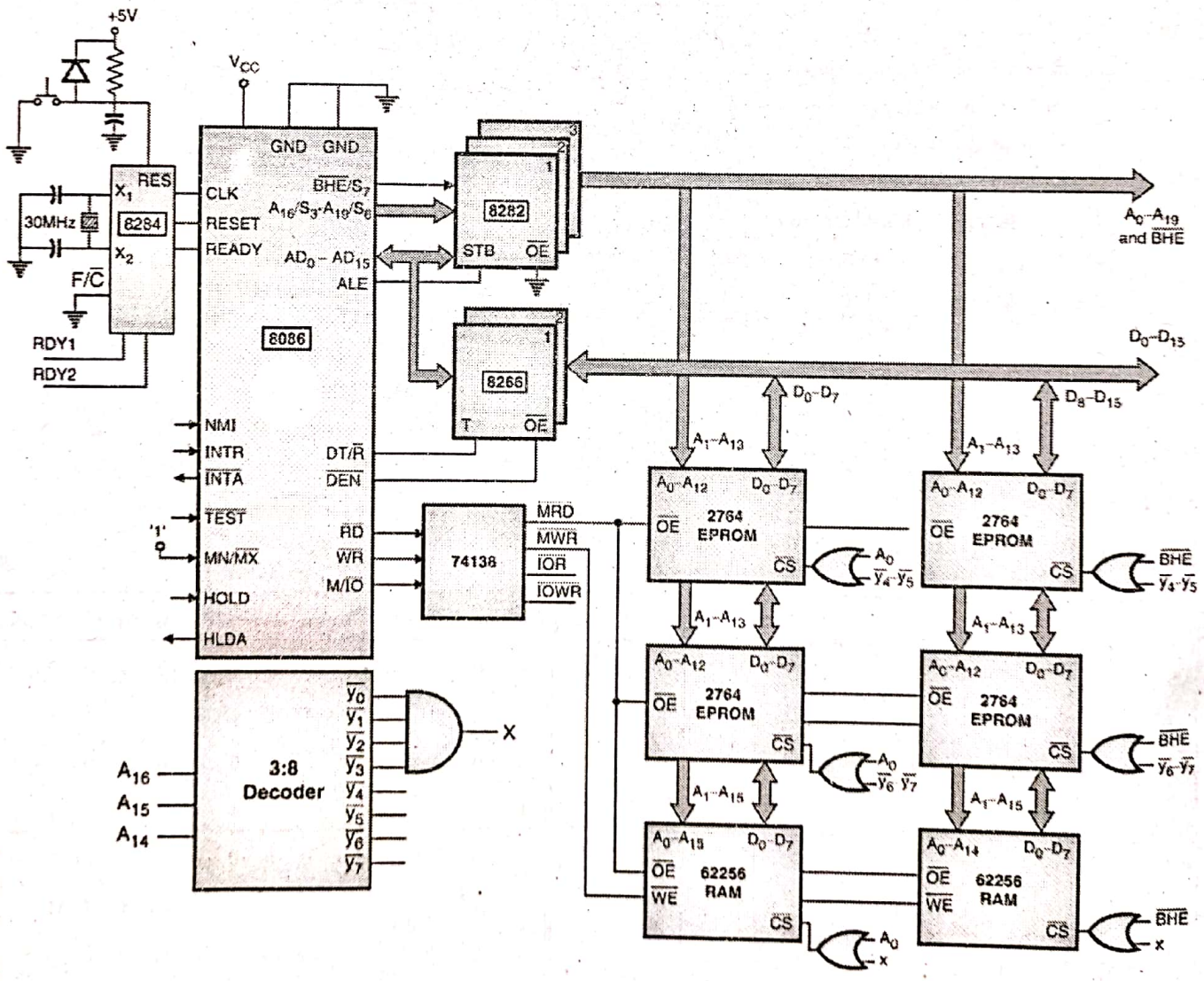
EA $\Rightarrow$ Ending address, SA $\Rightarrow$ Starting address, EB $\Rightarrow$ Even Bank, OB $\Rightarrow$ Odd Bank

			A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
RAM Set-1 $\bar{y}_0.\bar{y}_1.\bar{y}_2.\bar{y}_3$	EB	SA = 00000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		EA = 03FFEH	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	OB	SA = 00001H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		EA = 03FFFH	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ROM Set-1	EB	SA = F0000H	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		EA = F7FFEH	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	OB	SA = F0001H	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		EA = F7FFFH	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

ROM Set-1 $\cdot \overline{y_7}$	EB	SA = F8000H	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
		EA = FFFFEH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	OB	SA = F8001H	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
		EA = FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1



Step 4 : Final Implementation :



May 2019

Chapter 2 : The Intel Microprocessors 8086/8088 Architecture[Total Marks - 5]

Q. 1(a) Give the advantages of memory segmentation of 8086 microprocessor. (5 Marks)

Ans. :

The advantages of Segmentation are as follows :

1. The most important advantage is that the programmer can access a memory that required 20-bit address, by using 16-bit registers only.
2. The programs, data and stack are stored in separate blocks in memory and hence the three are organized in a modular fashion.
3. It also help in object oriented programming to store data of an object
4. Sharing of data or passing of data from one program to another is easily possible due to segmentation
5. The segmentation makes data relocatable as the program uses only offset register pointers while the segment points to the base of a segment.

Chapter 3 : Operating Modes [Total Marks – 20]

Q. 2(a) Explain minimum mode configuration of 8086 microprocessor. (10 marks)

Ans.:

- The minimum mode system block diagram is as shown in Fig. 1- Q.2(a)
- In Fig. 1- Q.2(a) the system in the minimum mode contains the support chips such as 8282, 8284 and if necessary, the 8286 data buffers.
- In Fig. 1- Q.2(a) the signals $AD_0 - AD_{15}$, $A_{16} / S_3 - A_{19} / S_6$ and $\overline{BHE} / S_7$ are multiplexed.
- These signals are demultiplexed by external latches and the ALE signal. The latches are generally buffered output D-type flipflops like 8282, these latches provide increased output drive capacity.
- If the microprocessor system has several devices that are interfaced with it, then to increase the current sourcing/sinking it is essential to use drives and transreceivers for data bus. Intel 8286 is used for this purpose. They are controlled by two signals $\overline{DEN}$ and $DT/\overline{R}$. To service 16 data, two 8286 transreceiver ICs are required.
- The $\overline{DEN}$ signal indicates that valid data is available on the data bus, while the $DT/\overline{R}$ is responsible for indicating the direction of data to or from the processor. At the time of data transfer the $\overline{OE}$ (Output Enable) pin should be active low.
- The 8284 clock generator provides a clock pulses at constant frequency. The clock generator is synchronized with the READY signal and some external signals. The $\overline{RES}$ signal, initializes the system with clock pulses.
- The status on the lines $M/\overline{IO}$, $\overline{RD}$ and $\overline{WR}$ will decide the type of operation I/O read, I/O write, memory read or memory write. The HOLD and HLDA signals are used to interface with other bus masters.
- The INTR and $\overline{INTA}$ signals are used to increase the interrupt handling capacity of the 8086.

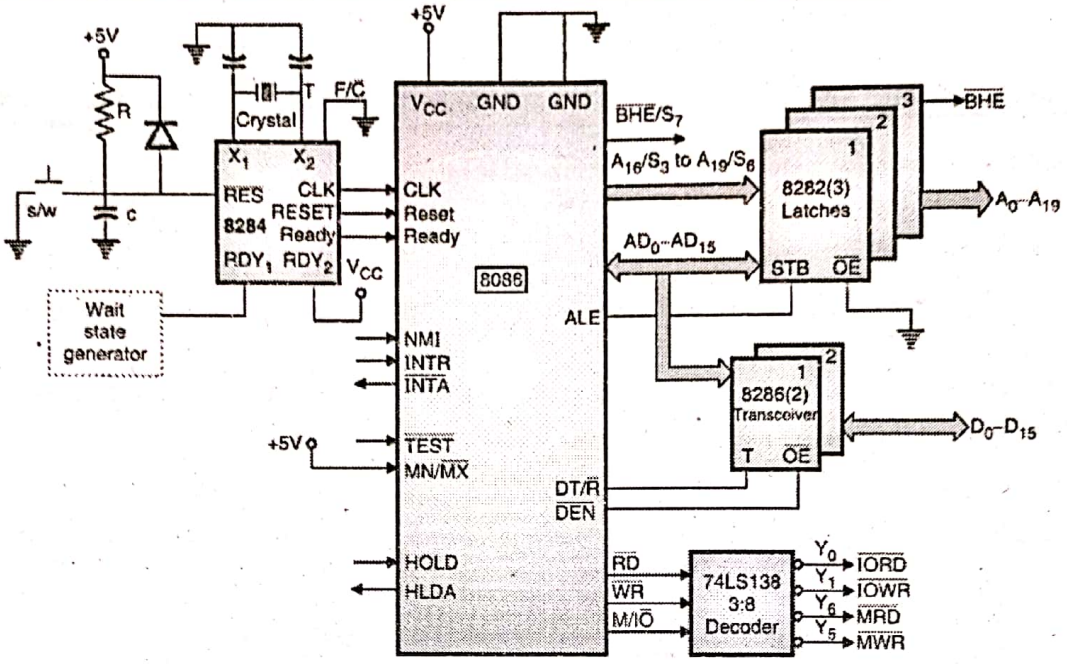


Fig. 1- Q.2(a) : Block diagram of minimum system in minimum mode

Q. 6(a) Draw and explain memory read and memory write machine cycle timing diagrams in maximum mode of 8086. (10 marks)

Ans.:

1. Read Cycle

- Fig. 1- Q.6(a) is the timing diagram which shows the activities of various signals during the read operation
- The sequence of operations during the read machine cycle in maximum mode are as follows :

Step 1 : The 8086 will make $M/\overline{IO} = 1$ if the read is from memory and $M/\overline{IO} = 0$ if the read is from the I/O device.

Step 2 : At about the same time the ALE output is asserted to 1.

Step 3 : Make $\overline{BHE}$ low/high and send out the desired address on AD_0 to AD_{15} and A_{16} to A_{19} lines.

Step 4 : Pull down ALE (make it 0). The address is latched into external latch.

Step 5 : Remove the address from AD_0 to AD_{15} lines and put them in the input mode (float them).

Step 6 : Assert the $\overline{RD}$ (read) signal low. This will put the data from the addressed memory location or I/O port on to the data bus.

Step 7 : Insert the "wait" T- states if the 8086 READY input is made low before or during the T_2 state of a machine cycle.

Step 8 : As soon as READY input goes high, 8086 comes out of the wait T-states and completes the machine cycle.

Step 9 : Complete the "Read" cycle by making the $\overline{RD}$ line high (inactive).

Step 10 : For larger systems we need to use the data buffers (8286 transceivers). Then the $DT/\overline{R}$ and $\overline{DEN}$ signals of 8086 are connected to 8086 and enabled at the appropriate time.

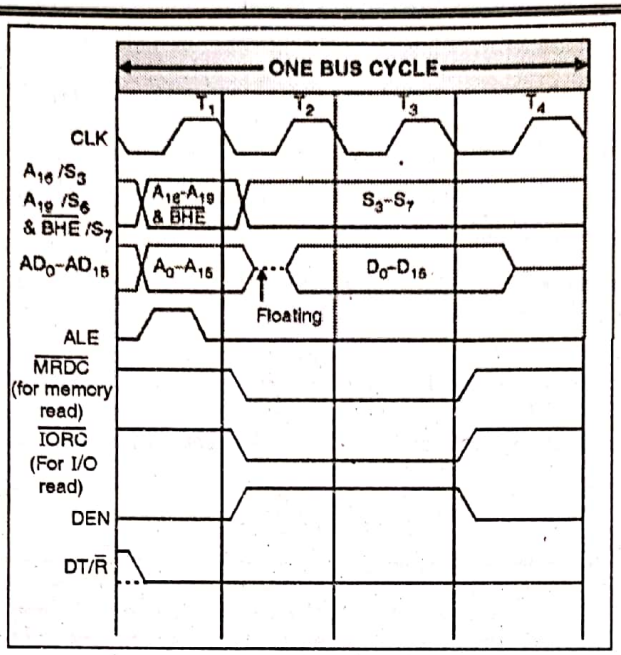


Fig. 1- Q. 6(a) : Maximum mode read bus cycle

- **Memory access time (t_A) :** The address to data time or the time gap between the processor providing the address and the memory or I/O device providing the data is called as the access time of the memory or I/O device.
- **Concept of wait T-states :** It is used to synchronize slower devices. If a particular memory or I/O device is slower i.e. has a greater value of access time, then it needs to disable the READY pin of the microprocessor. This causes the microprocessor to insert wait states in between the machine cycle giving time for the device to place its data on the data bus. The name is given as wait states as the microprocessor waits for the device. The processor waits until the READY pin is enabled again. **Fig. 2- Q. 6(a)** shows wait states inserted in between of a machine cycle.

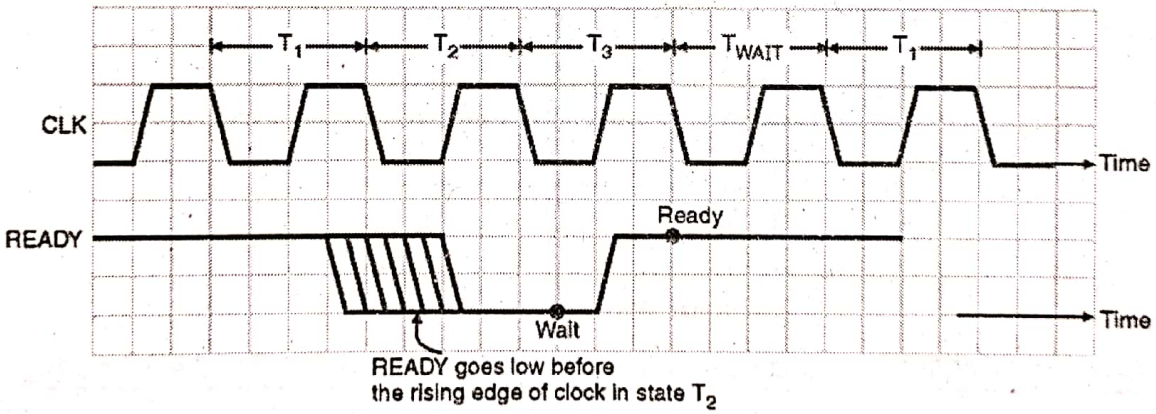


Fig. 2- Q. 6(a) : WAIT T-states

- Fig. 3- Q.6(a) shows the write machine cycle in maximum mode.
- The sequence of operations during the write machine cycle in maximum mode are as follows :

Step 1 : The 8086 will make $M/\overline{IO} = 1$ if the write is from memory and $M/\overline{IO} = 0$ if the write is from the I/O device.

Step 2 : At about the same time the ALE output is asserted to 1.

Step 3 : Make $\overline{BHE}$ low/high and send out the desired address on AD_0 to AD_{15} and A_{16} to A_{19} lines.

Step 4 : Pull down ALE (make it 0). The address is latched into external latch.

Step 5 : Remove the address from AD_0 to AD_{15} lines and place the data on them.

Step 6 : Assert the $\overline{WR}$ (write) signal low. This will put the data from the addressed memory location or I/O port on to the data bus.

Step 7 : Insert the "wait" T- states if the 8086 READY input is made low before or during the T_2 state of a machine cycle.

Step 8 : As soon as READY input goes high, 8086 comes out of the wait T-states and completes the machine cycle.

Step 9 : Complete the "Write" cycle by making the $\overline{WR}$ line high (inactive).

Step 10 : For larger systems we need to use the data buffers. (8286 transceivers). Then the DT/$\overline{R}$ and $\overline{DEN}$ signals of 8086 are connected to 8086 and enabled at the appropriate time.

The cycle is identical to read and write cycle of 8086 in minimum mode of operation. The few differences we have those are as follows :

(i) Status lines $\overline{S_2}\,\overline{S_0}$ lines are taken into account. These lines are active for T_1 and T_2 cycle. After that they are inactive.

(ii) ALE, Memory Read, I/O Read, DT/$\overline{R}$, DEN are generated by 8288 bus controller. They are not generated by microprocessor directly.

2. Write Cycle

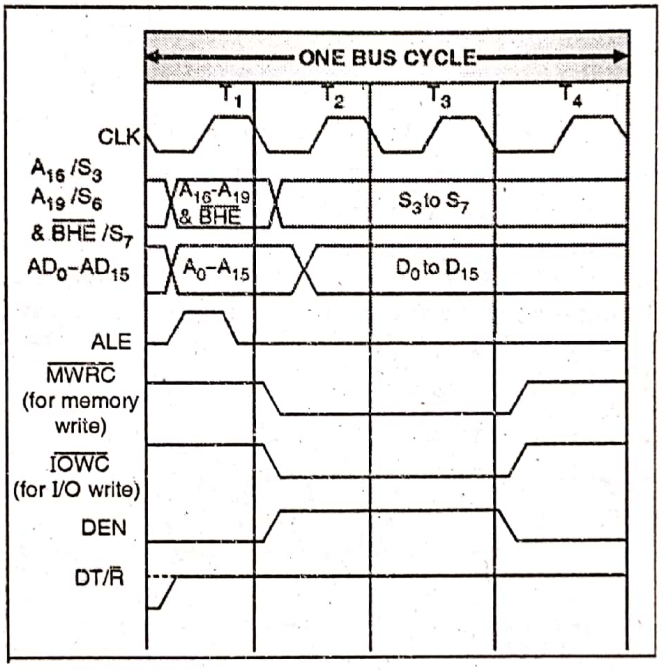


Fig. 3- Q. 6(a) : Maximum mode write cycle

Chapter 4 : 8086/8088 Addressing Modes and Instruction Set [Total Marks - 10]

Q. 5(a) Explain different addressing modes of 8086 microprocessor. (10 Marks)

Ans. :

- Addressing modes (methods) refer to the different methods of addressing (selecting) the operands.
- Addressing modes of 8086 are as follows :

1. Register addressing mode
2. Immediate addressing mode
3. Memory addressing modes
4. String addressing mode
5. Implied addressing mode
6. I/O addressing modes

– Fig. 1-Q. 5(a) shows the classification of addressing modes of 8086.

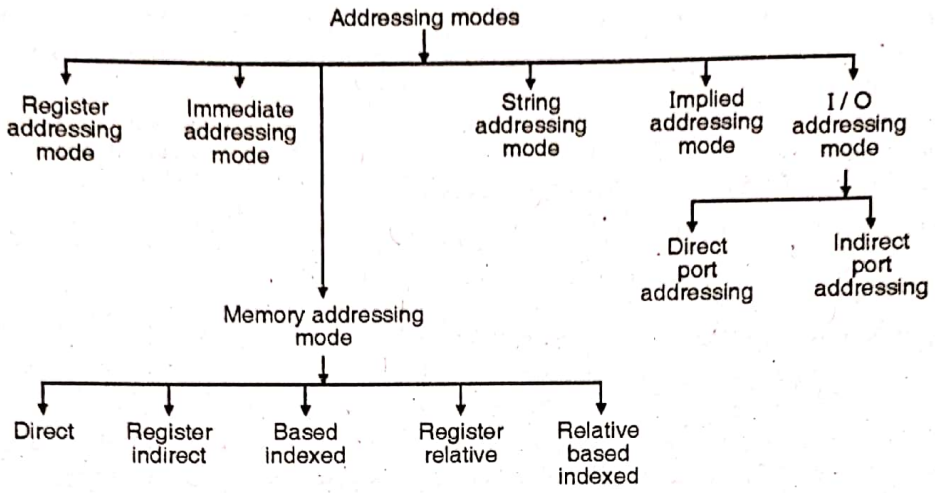


Fig. 1- Q.5(a) : Addressing modes of 8086

1. Register Addressing Mode

- In this mode of addressing, operand is in the register, and instruction specifies the particular register as shown in Fig. 2- Q.5(a).
- The advantage of this addressing mode is that the access is faster.
- Registers may be used as source operands, destination operands or both.
- The registers may be 8/16 bit.
- **E.g. MOV AX , BX**

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

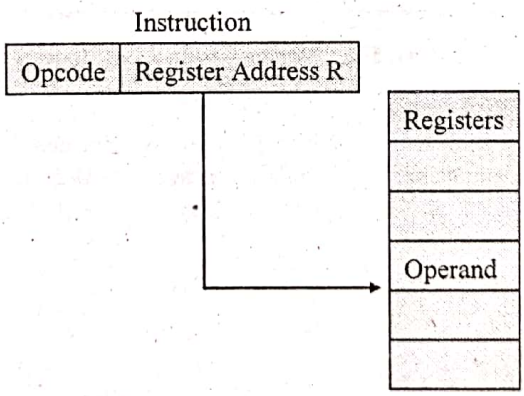


Fig. 2- Q. 5(a) : Register addressing

2. Immediate Operand Addressing Mode

- In this case the operand is in the instruction itself. It is said to be immediate addressing mode as the operand is in the immediate next location of the OPCODE. Fig. 3- Q. 5(a) shows format of instruction encoded with immediate operand.

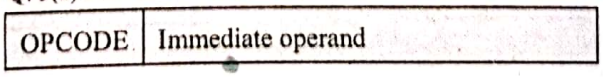


Fig. 3- Q. 5(a) : Instruction encoded with an immediate operand

- The operand in this case could be either 8-bit or 16-bit.
- **E.g. MOV CL, 02 H**

  This instruction copies the immediate number 02H in the CL register.

3. Memory Addressing Mode :

- For memory accesses, the processor needs to generate a 20-bit address. The registers in 8086 are of 16-bit.
- In segmentation, that 8086 produces the 20-bit address by special method i.e. segment register multiplied by 10H and adding to it the effective address.
- The effective address can either be a direct 16-bit address or can have various components i.e. base register value, index register value and the displacement. Based on the different combinations there are various addressing modes. Once we get EA (effective address), we can calculate PA (physical address) as,

PA = Segment :Offset
⇓ ⇓
= Segment register :EA
= Segment register :BASE + INDEX + DISPLACEMENT

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

- **Effective Address :** The address effective from the starting of the segment is called as the effective address. For example, if the effective address is 10, then it indicates that the location to be accesses is 10th from the starting of the segment.

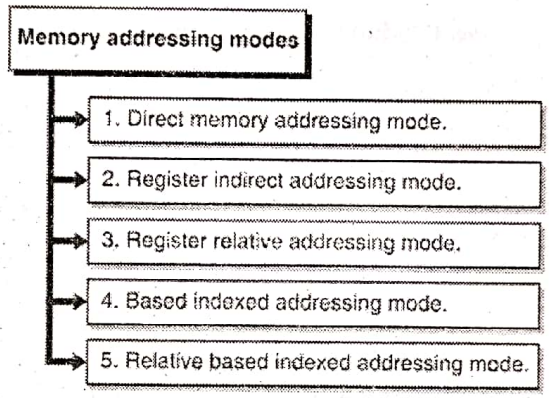


Fig. 4- Q. 5(a) : Memory Addressing Modes

(i) Direct Memory Addressing Mode

- In this mode, the 16-bit effective address EA is directly given in the instruction. The physical address is generated by adding this 16-bit direct address to segment register *10 H as shown in the Fig. 5- Q. 5(a).

PA = segment : EA

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

- **E.g. MOV [1023], AL**

The contents of AL are copied to memory location whose effective address is 1023H i.e. the physical address = DS *10H +1023.

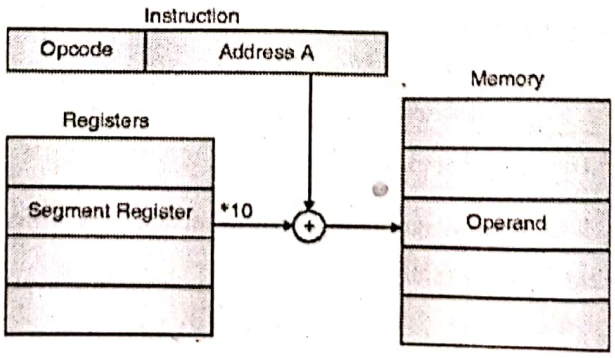


Fig. 5- Q. 5(a) : Direct addressing

(ii) Register Indirect Addressing Mode

– In this addressing mode the effective address is given by a base register or an index register, specified in the instruction. This effective address is added with the segment register * 10H to generate the physical address as shown in Fig. 6- Q. 5(a).

$$\therefore EA = \begin{Bmatrix} (BX) \\ (BP) \\ (SI) \\ (DI) \end{Bmatrix}$$

Segment : EA

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

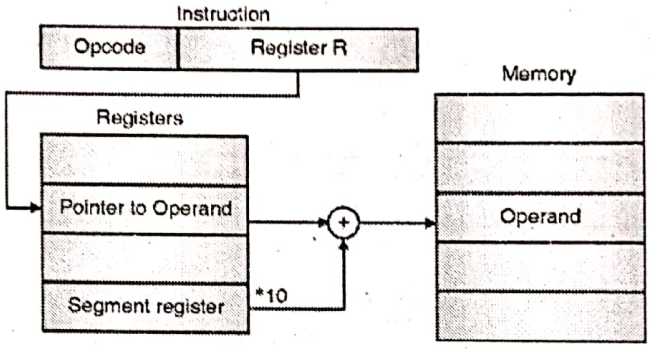


Fig. 6- Q. 5(a) : Register indirect addressing modes

– **E.g. MOV [SI], AL**

The contents of AL register are copied to memory location whose effective address is given by SI i.e. the physical address = DS *10H + SI.

(iii) Register Relative Addressing Mode

– In this addressing mode the effective address is given by a base register or index register along with an 8-bit displacement, specified in the instruction. This effective address is added with the segment register * 10H to generate the physical address as shown in Fig. 7-Q. 5(a).

$$\therefore EA = \begin{Bmatrix} (BX) \\ (BP) \\ (SI) \\ (DI) \end{Bmatrix} + \begin{Bmatrix} \text{8 bit displacement} \\ \text{(sign extended)} \\ \text{16 bit displacement} \end{Bmatrix}$$

$$\therefore PA = \text{Segment : EA}$$

$$= \begin{Bmatrix} CS \\ ES \\ DS \\ SS \end{Bmatrix} : \begin{Bmatrix} (BX) \\ (BP) \\ (SI) \\ (DI) \end{Bmatrix} + \begin{Bmatrix} \text{8/16 bit} \\ \text{offset} \end{Bmatrix}$$

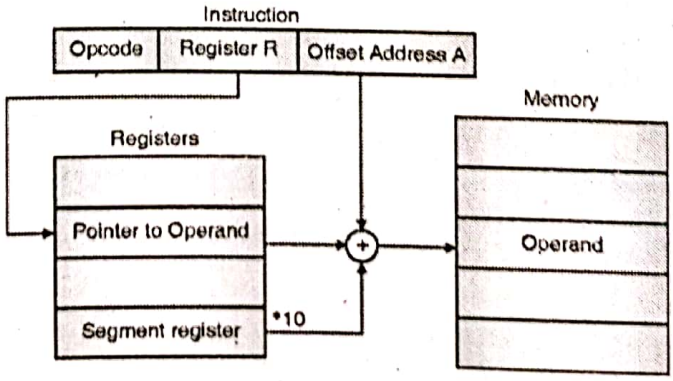


Fig. 7-Q. 5(a) : Register relative addressing mode

– **E.g. MOV [BX + 10], AL**

This instruction copies the contents of AL register to memory location whose effective address is given by BX + 10H i.e. the physical address = DS *10H + BX + 10H.

(iv) Based Indexed Addressing Mode

– In this addressing mode the effective address is given by a base register and an index register, specified in the instruction. This effective address is added with the segment register * 10H to generate the physical address as shown in Fig. 8- Q. 5(a).

$$\therefore EA = \{\text{Base register}\} + \{\text{Index register}\}$$

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

$$PA = \text{Segment register} : EA$$

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

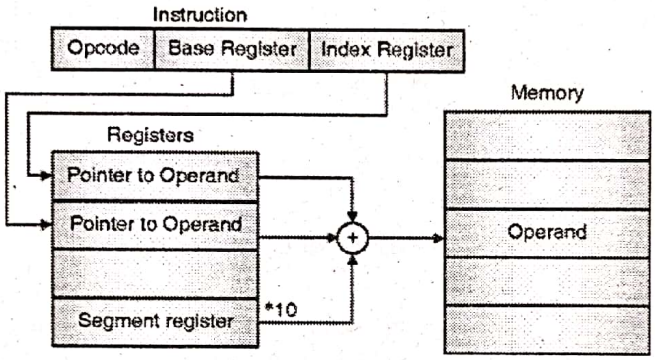


Fig. 8- Q. 5(a) : Based indexed addressing mode

– **E.g. MOV [BX + SI], AL**

This instruction copies the contents of AL register to memory location whose effective address is given by BX + SI i.e. the physical address = DS *10H + BX + SI

(v) Relative Based Indexed Addressing Modes

– In this addressing mode the effective address is given by a base register and an index register along with an 8-bit displacement, specified in the instruction. This effective address is added with the segment register * 10H to generate the physical address as shown in Fig. 9-Q. 5(a).

$$\therefore EA = \{\text{Base register}\} + \{\text{Index register}\} + \begin{Bmatrix}\text{8 bit displacement}\\\text{(sign extended)}\\\text{16 bit displacement}\end{Bmatrix}$$

$$= \begin{Bmatrix}(BX)\\(BP)\end{Bmatrix} + \begin{Bmatrix}(SI)\\(DI)\end{Bmatrix} + \begin{Bmatrix}\text{8/16 bit}\\\text{displacement}\end{Bmatrix}$$

$$PA = \text{Segment register} :$$

$$EA = \begin{Bmatrix}CS\\SS\\DS\\ES\end{Bmatrix} : \begin{Bmatrix}(BX)\\(BP)\end{Bmatrix} + \begin{Bmatrix}(SI)\\(DI)\end{Bmatrix} + \begin{Bmatrix}\text{8/16 bit}\\\text{displacement}\end{Bmatrix}$$

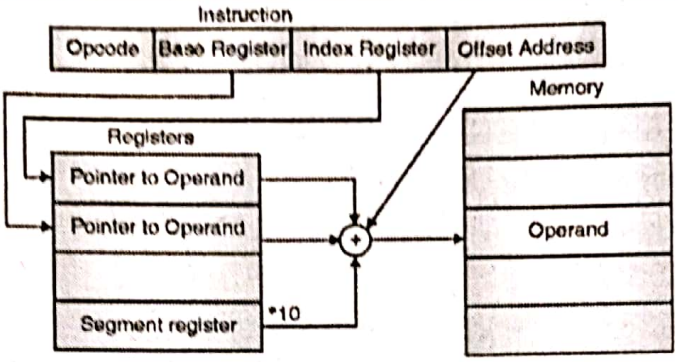


Fig. 9- Q. 5(a) : Relative Based Indexed Addressing Mode

- **E.g. MOV CX, [BX + SI + 0400]**

  This instruction copies the contents of AL register to memory location whose effective address is given by BX + SI + 04H i.e. the physical address = DS *10H + BX + SI + 04H.

4. String Addressing Mode

- String instructions use a different addressing mode wherein the pointers SI and DI along with segment registers DS and ES, respectively are used to access the source and destination memory locations.
- The memory location pointer by DS:SI is used as source while the memory location pointed by ES:DI is used as a destination. These pointers are also automatically incremented or decremented according to the value of Direction Flag.

5. Implied Addressing Mode

The operand or reference to operand is not specified in the instruction. Instead the operand is obvious in the mnemonic or the instruction.

E.g. XLAT

CMA

STC

STD

6. I/O Addressing Mode

This addressing mode is basically used for IOs, it categorized in :

(i) Memory mapped I/O

(ii) I/O mapped I/O

(i) Memory mapped I/O

- Memory mapped I/O refers to an I/O location mapped in memory i.e. given a memory address.
- In case of a memory mapped I/O device or I/O location, the benefit is that many instructions can access this data directly and hence giving a ease of access.
- The disadvantage of such I/O locations is that the access of I/O locations being normally slowers, it makes the processor to wait.

(ii) I/O mapped I/O

- If I/Os are mapped in ***I/O map I/O***, then 8086 supports two different addressing modes :

  (a) Direct port addressing. (b) Indirect port addressing.

- **Direct Port addressing :** The address of I/O device or I/O location is given in the instruction itself. The limitation of this is that the direct address given in the instruction can be of a maximum of 8-bits and hence only 256 I/O locations can be accessed.

- **Indirect port addressing :** Here a pointer register i.e. the register DX is used to give the address of the I/O location. Since the register DX is of 16-bit, 16-bit address supports a huge range of 65536 I/O locations.

Chapter 6 : Stacks and Subroutines[Total Marks - 10]

Q. 1(b) Differentiate Procedure and macro with example. **(5 Marks)**

Ans. :

Sr. No.	Procedure	Macro
1.	It resembles a call function of high level language. The processor branches to the procedure on call proc, instruction and returns back to the caller program after executing the procedure.	When the assembler comes across the instruction "CALL MACRO", it replaces this instruction with the group of instructions placed in the corresponding macro.
2.	Since the processor branches to another memory location and returns back, it consumes some time to store and fetch back the return address. Hence it has a latency period.	Macro does not required any latency period.
3.	Since the assembler stores the instructions of procedure only once in the memory, the program consumes less space in memory.	Since the assembler replaces all "Call macro" instruction by the group of instructions in the macro, the program consumes more space in memory.
4.	Procedures are to be used for repetitive task, if the task is very large (i.e. it has many instructions).	Macros are to be used for repetitive task, if the task is small (i.e. it has less number of instructions).

Q. 3(a)(i) Write a short note on mixed language programming. **(5 Marks)**

Ans. :

- 'C' generates an object code that is extremely fast and compact, but it is not as fast as the object code generated by a good programmer using assembly language.
- There are special cases where a function is coded in assembly language to reduce execution time.
- **For example :** The Floating Point math package must be coded in assembly language as it is used frequently and its execution speed will have great effect on the overall speed of the program that uses it. There are also occasions when some hardware devices need exact timing and then it is necessary to write assembly level programs to meet such strict timing restrictions.
- In addition, certain instructions cannot be executed in Higher Level Languages like C.
- **For example :** C does not have an instruction for performing bit-wise rotation operation. Thus in spite of C being very powerful, routines must be written in assembly language to :
  - Increase the speed and efficiency of the routine.
  - Perform Machine specific functions not available in Microsoft C or in Turbo C.
  - Use third party routines.

There are 2 ways of combining C and Assembly language.

Method 1

- In this method built-In-Inline assembler is used to include assembly language routines in the C-program, without any need for a specific assembler.
- Such assembly language routines are called in-line assembly.
- They are compiled right along with C Routines rather than being assembled separately and then linked together using linker modules provided by the C Compiler.
- Turbo C (TC) has inline assembler.

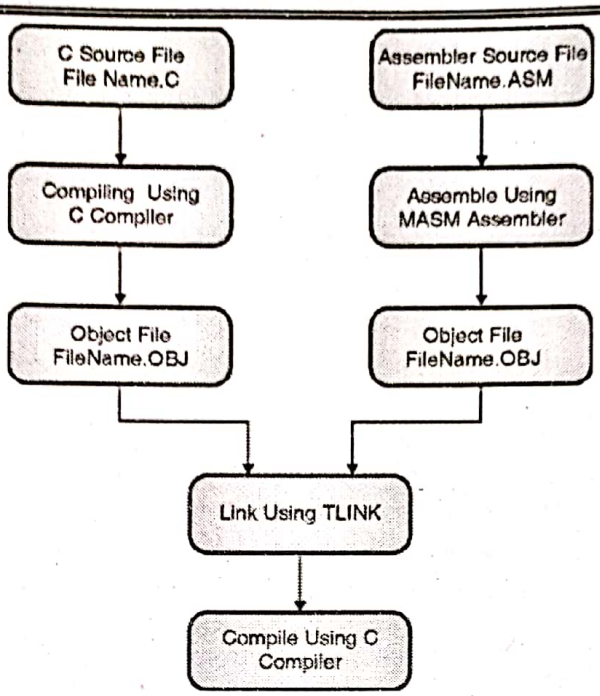


Fig. 1- Q. 3(a) : Combining C and assembly

Method 2

- There are times when programs written in one language have to call modules written in other languages. This is called as mixed language programming.
- **For example :** When a particular sub-routine is available in a language, different from the language currently used in a program, or when algorithms are described in a different language, users to use more than one language
- Mixed language calls involve calling functions in separate modules.
- Instead of compiling all source programs using the same compiler, different compilers or assemblers are used as per the language used in the program.
- Microsoft C supports Mixed Language Programming.
- Therefore, it can combine assembly language routines in C as a separate language.
- C program calls assembly language routines that are separately assembled by MASM (MASM assembler) or TASM (Turbo assembler).
- These assembled modules are linked with the compiled C modules to get the combine executable file.
- Fig. 1- Q. 3(a) shows Compile, Assemble, and link processes using C compiler, MASM Assembler and TLINK.

Chapter 7 : 8086 Interrupt Structure [Total Marks - 5]

Q.6(b) (i) Explain Types of Interrupt . **(5 marks)**

Ans. :

Interrupt structure of 8086 microprocessor

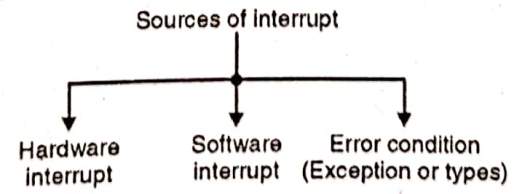


Fig. 1- Q. 6(b) : Interrupt structure of 8086 microprocessor

1. Hardware Interrupt

- In this type of interrupt, physical pins are provided in the chip. In 8086 there are two pins :
  (i) NMI (Non maskable interrupt)
  (ii) INTR
- NMI is non maskable i.e. microprocessor has to service this interrupt, it cannot avoid it. Whereas INTR is maskable, if flag in flag register is '0', microprocessor will not recognise interrupt available on the pin.

2. Software Interrupt

Software interrupt, in 8086 we have INT instruction. When INT instruction is executed interrupt will occur.

3. Error Conditions (Exception Or Types)

- 8086 supports division, multiplication, addition etc. if user asks microprocessor to divide any number by ZERO, then you know that dividing any number by ZERO produces answer '**∞(infinity)**'.
- So in this case microprocessor will generate an interrupt **"Automatically"** and interrupt current execution. In ISR, user can display message **"Divide by zero error"**. Instead of showing the answer as **"∞(infinity)"**.
- So internally generated errors produces an interrupt for microprocessor, normally referred as **"TYPE"** by Intel engineer and referred as **"Exception"** by motorola engineer.
- Thus 8086 has a simple and versatile interrupt system. Every interrupt is assigned a **"type code"** that identifies it to the CPU.
- The 8086 can handle upto 256 different interrupt types. Interrupts may be initiated by devices external to the CPU; in addition, they also may be triggered by software interrupt introductions and under certain condition, by the CPU itself Fig. 2-Q. 6(b) shows interrupt sources for 8086.

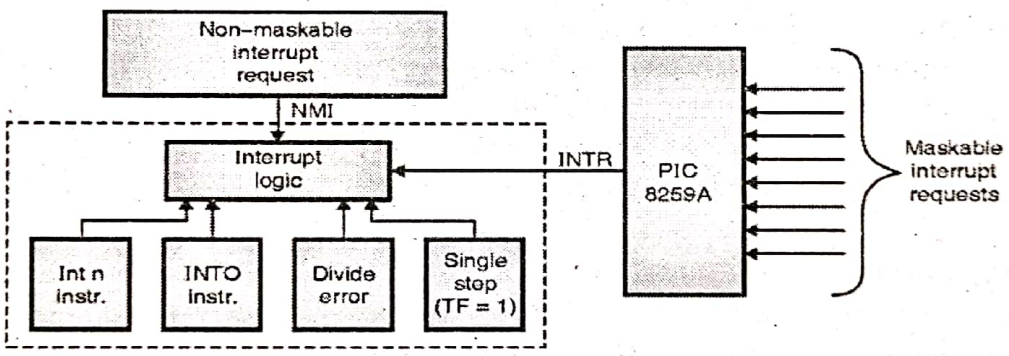


Fig. 2-Q. 6(b) : Interrupt sources for 8086

- In Fig. 2-Q. 6(b) 8086 have two lines that external device may use to signal interrupts.
- The INTR line is usually driven by an Intel 8259A (PIC), which in turn connected to the devices that need interrupt services

Chapter 8: IC 8259 Programmable Interrupt Controller (PIC) [Total Marks - 10]

Q. 5(b) Explain the operation of three 8259 PIC in cascade mode. (10 Marks)

Ans. :

- To cascaded 8259, the INT pin of the slaves are connected to interrupt request pins (IR0 – IR7) and $\overline{INTA}$ to the $\overline{INTA}$ of master 8259.
- The CAS2 – CAS0 lines work as output for the master and input for the slave. Fig. 1-Q. 5(b) shows cascaded 8259 interfaced with 8086 in maximum mode.

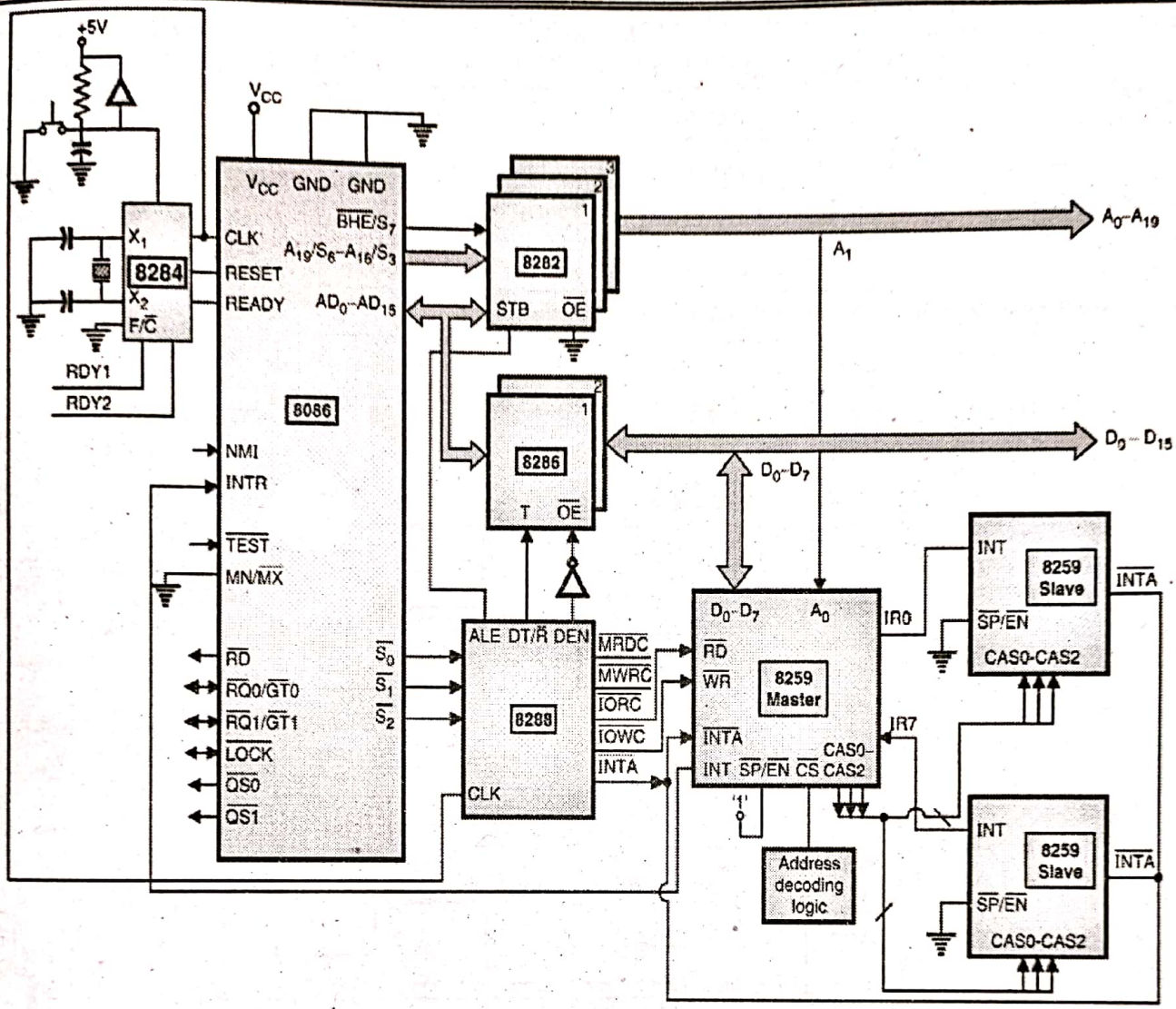


Fig. 1- Q. 5(b) : Interfacing 8259 with 8086 (8259 – cascaded, 8086- maximum mode)

Chapter 9: 8255 Programmable Peripheral Interface[Total Marks - 10]

Q. 3(b) Draw and explain the block diagram of 8255 Programmable Peripheral Interface (PPI) with control word format. (10 Marks)

Ans. :

The block diagram of 8255 is as shown in Fig.1-Q.3(b).

1. Data Bus Buffer

- This is an 8-bit bi-directional buffer used to interface the internal data bus of 8255 with the external (system) data bus.
- The CPU transfers data to and from the 8255 through this buffer.

2. Read/Write Control Logic

- It accepts address and control signals from the μP.
- The Control signals determine whether it is a read or a write operation and also select or reset the 8255 chip.
- The address bits (A1, A0) are used to select the ports or the Control Word Register as are follows :

A1	A0	Selection	Sample address
0	0	Port A	80 H (i.e. 1000 0000)
0	1	Port B	81 H (i.e. 1000 0001)
1	0	Port C	82 H (i.e. 1000 0010)
1	1	Control Word	83 H (i.e. 1000 0011)

- The Ports are controlled by their respective Group Control Registers.

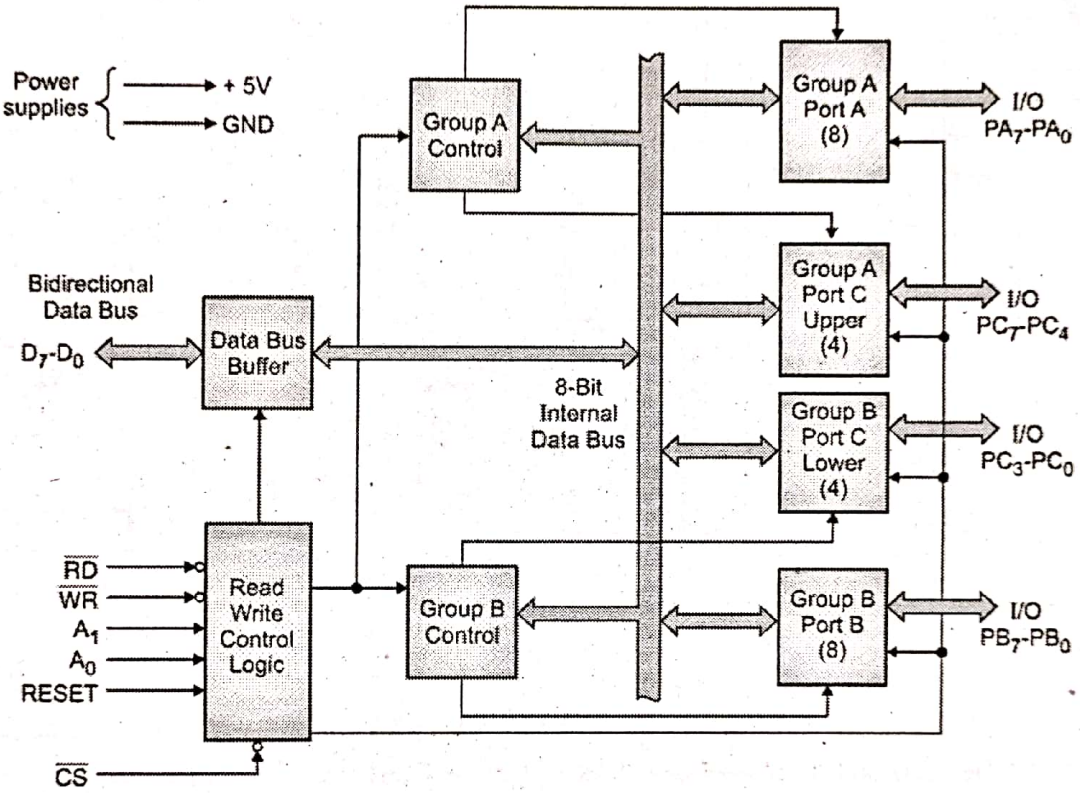


Fig. 1-Q.3(b) : Block Diagram of 8255

3. Group A Control

- This Control block controls Port A and Port C_{Upper} i.e. PC7-PC4.
- It accepts Control signals from the Control Word and forwards them to the respective Ports.

4. Group B Control

- This Control block controls Port B and Port C_{Lower} i.e. PC3-PC0.
- It accepts Control signals from the Control Word and forwards them to the respective Ports.

5. Port A, Port B, Port C

- These are 8-bit bi-directional Ports.
- They can be programmed to work in the various modes as follows:

  Port A : Mode 0, 1 and 2

  Port B : Mode 0 and 1

  Port C : Mode 0 and BSR mode

Chapter 10: 8253/8254 Programmable Interval timer [Total Marks - 5]

Q. 6(b)(ii) Explain the following : Modes of 8253/8254 Programmable Interval timer. **(5 Marks)**

Ans. :

The programmable timer IC provides following modes of operations.

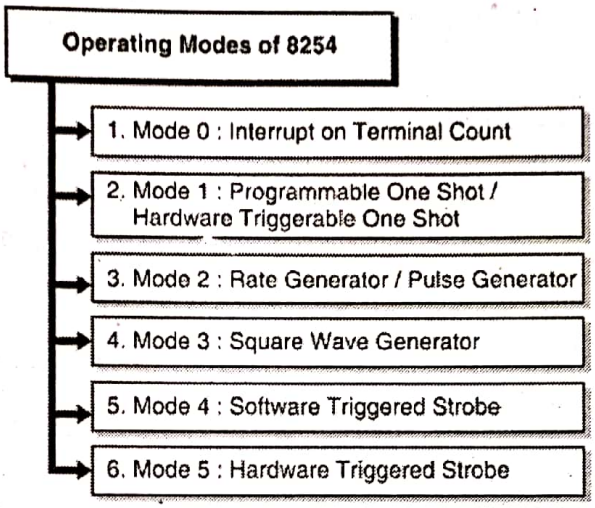


Fig. 1-Q. 6(b) : Operating modes of 8254

1. Mode 0: Interrupt on Terminal Count

(i) OUT pin initially LOW.

(ii) Count value is loaded.

(iii) GATE pin is made HIGH, so counting enabled.

(iv) During counting OUT pin remains LOW.

(v) On Terminal count OUT becomes HIGH and remains HIGH.

(vi) During counting if GATE is made low, it disables counting. And when made HIGH again counting resumes.

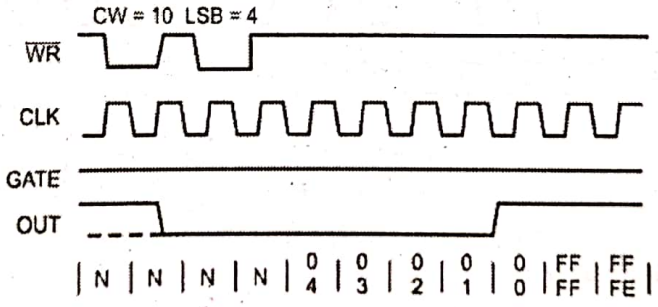


Fig. 2-Q. 6(b) : Mode 0 timing diagram

2. Mode 1 : Monostable Multivibrator

(i) OUT pin initially HIGH.

(ii) Count value is loaded.

(iii) GATE pin is given RISIN EDGE, so counting enabled.

(iv) During counting OUT pin remains LOW.

(v) On Terminal count OUT becomes HIGH and remains HIGH.

(vi) During counting if GATE is made low, it does not affect counting.

(vii) And when GATE is given another rising edge the count restarts.

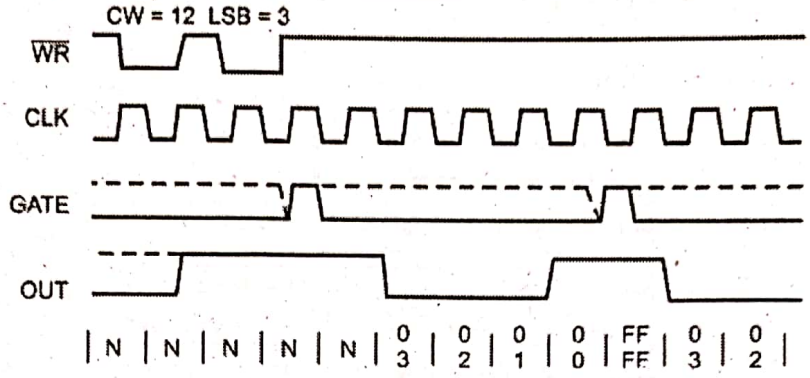


Fig. 3-Q. 6(b) : Mode 1 timing diagram

3. Mode 2 : Rate Generator

(i) OUT pin initially HIGH.

(ii) Count value is loaded.

(iii) GATE pin is made HIGH, so counting enabled.

(iv) During counting OUT pin remains HIGH.

(v) One cycle before the Terminal count OUT becomes LOW.

(vi) The count is reloaded and the above process repeats

(vii) During counting if GATE is made low, it disables counting. And when made HIGH again counting restarts.

(viii)Rate generator mode is also called as mod-n or divide by n counter.

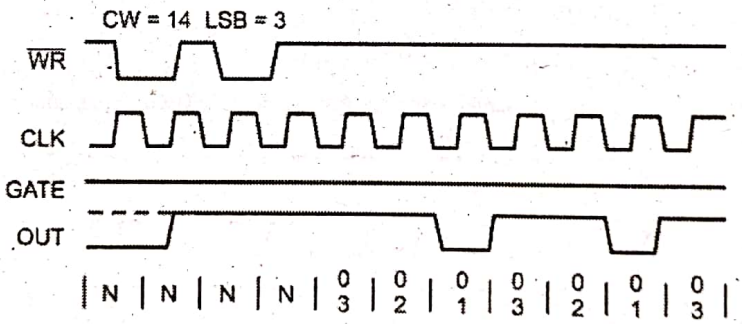


Fig. 4-Q. 6(b) : Mode 2 timing diagram

4. Mode 3 : Square Wave Generator

(i) OUT pin initially HIGH.

(ii) Count value is loaded.

(iii) GATE pin is made HIGH, so counting enabled.

(iv) OUT pin remains HIGH for half the count i.e. n/2 and remains low for the remaining half.

(v) On Terminal count the count is reloaded and the process repeats.

(vi) During counting if GATE is made low, it disables counting. And when made HIGH again counting restarts.

(vii) If the count is ODD, the OUT pin remains HIGH for (n+1)/2 count and low for (n-1)/2 counts.

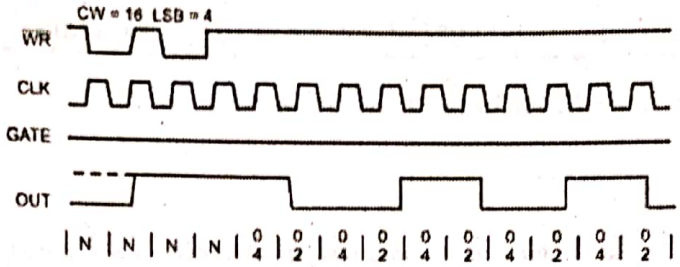


Fig. 5-Q. 6(b) : Mode 3 timing diagram

5. Mode 4 : Software Triggered Strobe

(i) OUT pin initially HIGH.

(ii) Count value is loaded.

(iii) GATE pin is made HIGH, so counting enabled.

(iv) During counting OUT pin remains LOW.

(v) On Terminal count OUT becomes LOW for 1 cycle after that again becomes HIGH and remains HIGH.

(vi) During counting if GATE is made low, it disables counting. And when made HIGH again counting restarts.

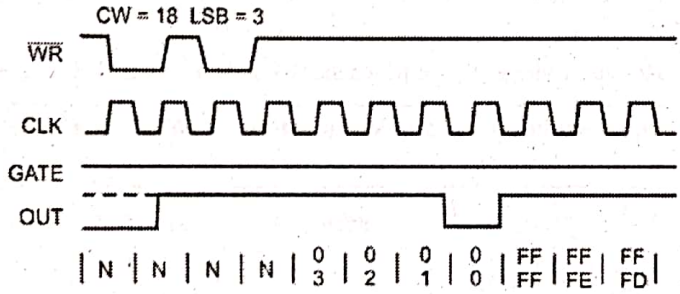


Fig. 6-Q. 6(b) : Mode 4 timing diagram

6. Mode 5 : Hardware Triggered Strobe

(i) OUT pin initially HIGH.

(ii) Count value is loaded.

(iii) Counting enabled when a trigger (rising edge) applied to the GATE pin.

(iv) During counting OUT pin remains HIGH.

(v) On Terminal count OUT becomes LOW for 1 cycle after that becomes HIGH and remains HIGH.

(vi) During counting if GATE is given another trigger the counting restarts.

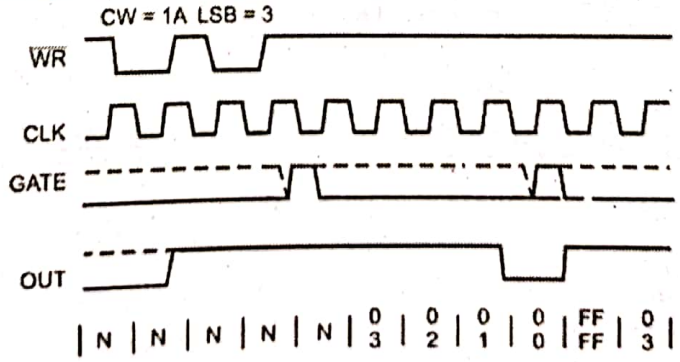


Fig. 7-Q. 6(b) : Mode 5 timing diagram

Chapter 14 : Intel 80386DX Processor [Total Marks - 15]

Q. 1(c) Explain VM, RF, IOPL and NT flags of 80386 microprocessor. **(5 Marks)**

Ans. :

1. VM Flag

When the processor enters in virtual 8086 mode, which is an emulation of the programming environment of the 8086 microprocessor, this bit is set to '1'. When the processor is in protected mode and this bit is set, the processor moves to virtual 8086 mode. In mode processor handles the segment loads as in 8086.

2. RF Flag

When RF = 1, it ignores the debug exception on execution of the next instruction. It is automatically reset at the successful completion of every instruction.

3. NT Flag

If NT = 1, it indicates that the currently executing task is nested within another task and it has a valid link to caller task i.e. this task is executed using the call instruction.

4. IOPL Flag

The IOPL encoded values indicates the privilege level at which the task should be executed to access the I/O device.

Q. 4(a) Differentiate Real mode, Protected mode and Virtual mode of 80386 microprocessor. **(10 Marks)**

Ans. :

Sr. No.	Real Mode	Protected Mode	Virtual Mode
1.	Only one task can be executed at any given instant.	Multiple tasks can be executed simultaneously.	Only one task can be executed at any given instant.
2.	Switching between real and protected mode requires complicated process.	Protected mode switching with virtual mode is easier compared to real mode.	Switching between virtual and protected mode is easy compared to that of real mode.
3.	Maximum memory accessible is 1MB + 64KB – 16 bytes.	Memory accessible is entire 4GB.	Memory accessible is entire 4GB.
4.	Memory addressing is similar to that of 8086.	Memory addressing is done using descriptors and selectors.	Memory accessing virtually seems to be similar to that of 8086.
5.	No protection amongst tasks.	Protection is implemented amongst tasks	No protection amongst tasks.

Chapter 15: Intel P5 Microarchitecture [Total Marks - 15]

Q. 1(d) Explain an instruction issue algorithm of Pentium processor. **(5 Marks)**

Ans. :

- The instructions of Pentium processor are pairable if the following rules are followed.
- The Pentium processor incorporates 2 integer pipeline designated as 'U' and 'V' pipelines.
- 'U' pipeline is the primary pipeline and its execution unit incorporates a barrel shifter while 'V' pipeline execution unit lacks this.
- Only simple instructions can be executed in 'V' pipeline while all other instructions are executed in 'U' pipeline
- Simple instructions are the ones that take 2 or 3 clock cycles only.

- The following is a list of some simple instructions:

  MOV reg, reg / mem / imm

  MOV mem, reg / imm

  ALU reg, reg / mem / imm

  ALU mem, reg / imm

  (ALU instructions refer to ADD, AND, CMP, OR, TEST, XOR, etc.)

  INC reg / mem

  DEC reg / mem

  PUSH reg / mem

  POP reg

  LEA reg / mem

  JMP / CALL / Jconditional near

  NOP

- Both pipelines are supplied by a steady stream of instructions by the prefetcher, which in turn is supplied by the code cache.
- Since 'V' pipeline has no barrel shifter, some instruction can execute only in 'U' pipeline. The prefetch queue in use delivers the first instruction to the 'U' pipeline while next to the 'V' pipeline.
- Instructions are pairable only if :
  - Both instructions are simple
  - Instruction must not have register contention
- When the instruction is fetched for the first time, the size is taken as one byte. When multibyte instruction is executed for the first time, the D1 stage provides a feedback to the code cache about instruction length.
- The boundary information of these instructions is then stored in the cache directory. Next time when code cache gives a line it also provides the prefetcher with the information about the instruction boundary.
- Based on this problem we can formulate are formulated.
  1. Decode two consecutive instructions I1 and I2
  2. If all the following conditions are true the two instructions are pairable and issue I1 to 'U' pipeline and I2 to 'V' pipeline; else they are not pairable, and only the instruction I1 is to be given to 'U' pipeline. The conditions are:
     (a) I1 and I2 are simple instructions
     (b) I1 is not a jump instruction
     (c) Destination of I1 is not the source of I2
     (d) Destination of I1 is not the destination of I2

Q. 2(b) Explain cache organization of Pentium processor. (10 Marks)

Ans. :

- The code cache is 8KB in size, organized as two-way set-associative mapping configuration. The cache ways are referred to as way zero and way one as shown in Fig. 1-Q. 2(b) and 2-Q. 2(b).
- Each cache line is 32 bytes wide and the bus connected from this cache to the prefetcher is also 256 bits (32 bytes), allowing 32 bytes to be delivered to the prefetch queue during a single prefetch.
  1. Each cache way contains 128 cache lines with a associated 128 entry directory with each of the cache ways.
  2. The cache directories are triple ported, to support split line access and snooping.
  3. The directory entry consists of a 20-bit tag field to identify the page in the memory; a state bit that indicates whether the line in cache contains valid or invalid information and a parity bit used to detect errors when reading each entry.

4. The directories are accessed by the address issued by the prefetcher. When the prefetcher initiates a spilt-line access, the two line addresses are submitted to the code cache. Address bits A_{11}-A_5 from the prefetcher identify the set where the target line may reside in cache, and are used as index into the cache directories. The lower portion of the prefetcher address (A4:A0) identifies a byte within the line.

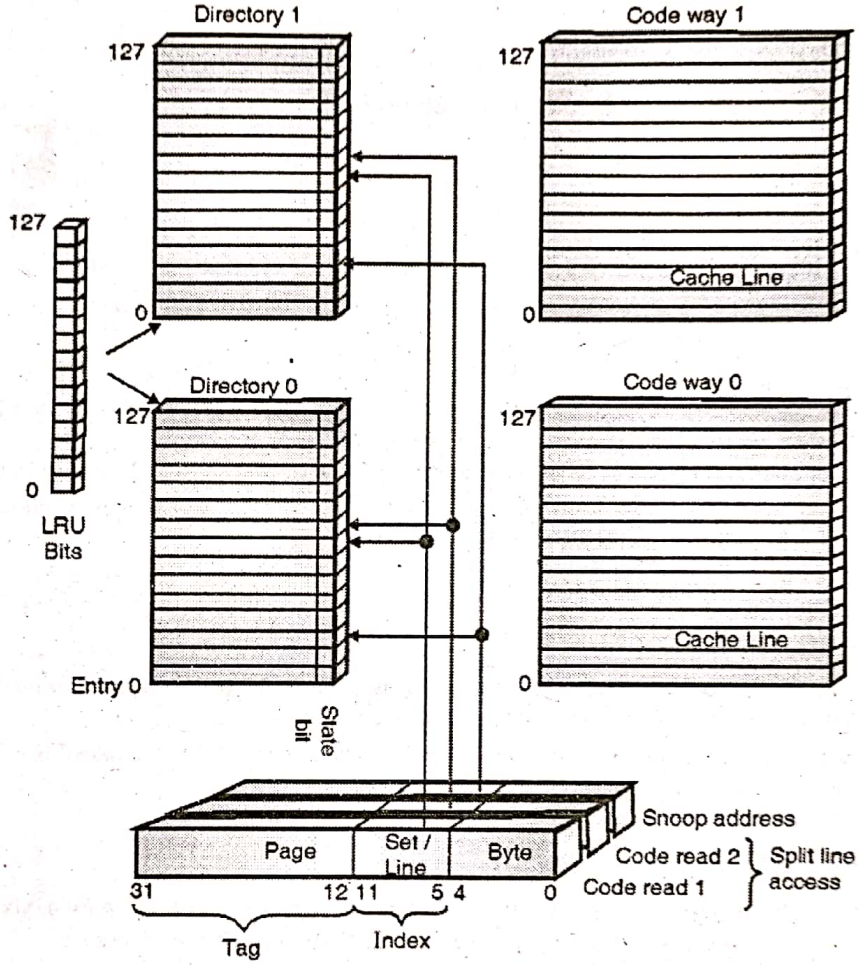


Fig. 1-Q. 2(b) : Pentium code cache organization

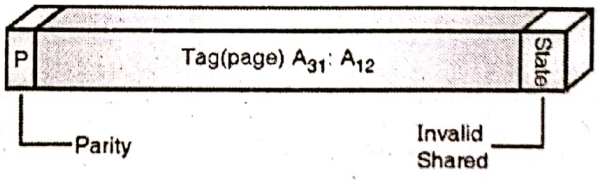


Fig. 2-Q. 2(b) : Code cache directory entry

5. Each cache line holds four quadwords of information. Accesses made to memory, caused by code cache misses, always result in the transfer of four quadwords from memory to cache. Each quadword is associated with a parity bit for error checking as shown in the Fig. 3-Q. 2(b).

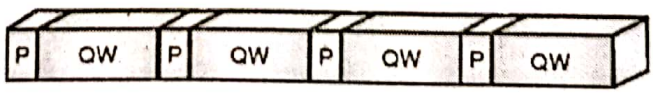


Fig. 3-Q. 2(b) : Line structure in code cache

6. The code cache is designed to permit two simultaneous prefetch accesses: one to the upper half of one line and another to the lower half of the next line; this helps in accessing an instruction that resides in two adjacent cache lines in a single cycle.

7. A snoop address can be presented to the cache directory at the same time that the split-line access is occurring on the third port of the cache directory. On a snoop hit, the snoop process does not read or write the cache lines, but may result in the invalidation of a cache line.

1. Line Storage Algorithm

- The code cache considers the 4GB memory space to be divided pages of 4kB each (since each way of the code cache is 4KB), and 1M such pages. Each page is divided into 128 lines, each of 32 bytes.
- When the prefetcher issues a request for an instruction, the code cache checks the directory to decide whether it has a copy of that line from the required page of memory.
- If the code cache doesn't have a copy, it issues a cache line-fill request to bus unit. The bus unit fetches the line from the L2 cache or system memory and places it in the L1 cache and makes a directory entry to track its presence.
- Suppose that the line was fetched from line no. 8 of a memory page 20. The code cache uses the line number, 8, to index into 8^{th} entry of its two directories and then takes one of the following actions :
  - (a) If either of the directory entries is marked 'invalid', the target page address, 20, is saved in that directory entry as the tag address and the new line is placed here. The state bit is set to shared state. The LRU bit associated with pair of directory entries is complemented to indicate that the entry 8 in the opposite directory is now the less recently used of the pair.
  - (b) If neither of the entries are invalid, the code cache will replace the entry currently pointed to by the LRU bit associated with this pair of directory entries. This is called to as a cache line replacement. The target page address, i.e. 20, is saved in the corresponding directory entry as the tag address. The state bit is set to 'shared' state. The LRU bit associated with this pair of directory entries is complemented to indicate that the entry 8 in the opposite directory is now the less recently used of the pair.

2. Inquire Cycles

- Inquire cycles are performed by the Pentium processor L1 cache when another bus master either reads or writes from main memory.
- This is done to ensure cache consistency between the contents of the internal data and code caches and system memory. The inquire cycles are run in the following cases.

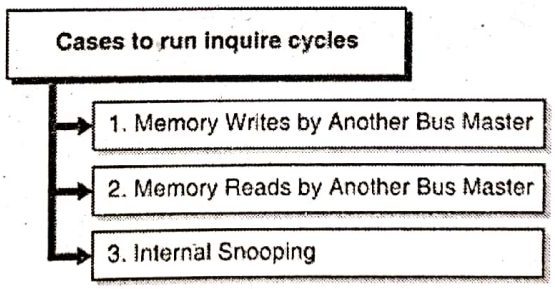


Fig. 4-Q.2(b) : Cases to inquire cycles

(i) Memory Writes by Another Bus Master

(a) When another bus master initiates a write to a location in memory, L2 cache controller directs the Pentium processor to snoop the address bus only in case if L2 cache has a snoop hit.

(b) The code cache performs a lookup to determine if it has a copy of the line that is being updated by the bus master.

(c) If it results in a snoop miss, the code cache takes no action.

(d) If a snoop hit results, the cache line is invalidated.

(ii) Memory Reads by Another Bus Master

(a) The same action is taken by L2 cache controller when another bus master initiates a memory read bus cycle.

(b) No action is taken by the code cache for either a snoop hit or miss.

(iii) Internal Snooping

(a) When the data cache initiates either a read or write operation, the code cache snoops the address as it is passed from the data cache to the bus unit.

(b) If snoop hit is detected, the code cache directory entry for that line is immediately invalidated so as to maintain consistency, when the processor is operating in the write-back mode and modified code is being run.

3. Split Line Access

(i) In a Pentium processor, the instruction length varies from 1 byte to 15 bytes and hence multi-byte instructions may reside in two sequential lines stored in the code cache. A code cache miss results in a 32-byte cache line-fill, if it's a cacheable address.

(ii) When the prefetcher finds that the instruction is residing in two lines, prefetcher must perform two sequential cache accesses in order to get the instruction from the code cache but this would impact performance.

(iii) The Pentium processor incorporates a special concept called as split-line access, permitting upper half of one line and lower half of the next to be fetched from the code cache in a single cycle.

(iv) But, when the split line is read from the cache, the information is not properly aligned.

(v) These bytes of the instruction must be rotated so that the prefetch queue receives the instruction in the proper order. This is done by a byte rotation mechanism as shown in the Fig. 5-Q. 2(b).

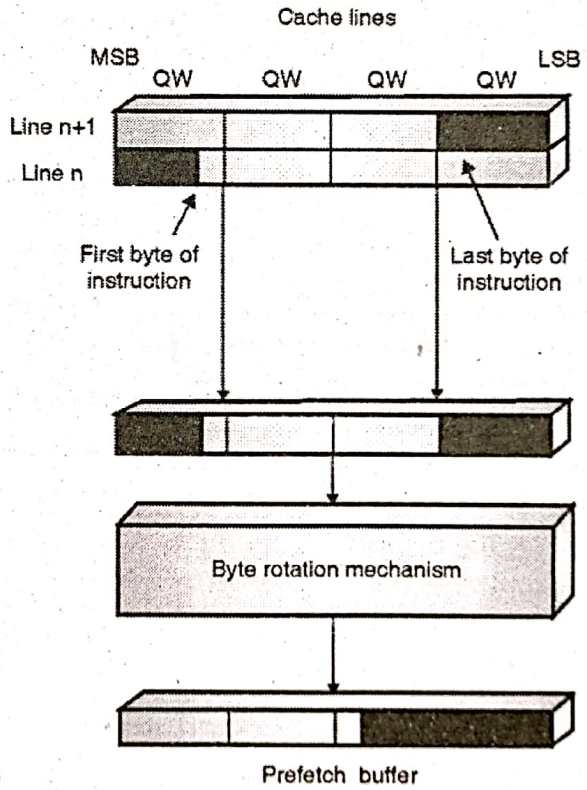


Fig. 5-Q. 2(b) : Split line access

(vi) In order for split-line access to work efficiently, instruction boundaries of the cache line need to be known.

This permits the prefetcher to track where instructions start within a given line and hence direct the code cache to fetch an entire line, or the lower half of one and the upper half of the next.

(vii) When an instruction is decoded for the first time, the length of the instruction is feedback by the D1 unit to the code cache.

❑❑❑

Dec. 2018

Q. 1 (a) Draw and explain memory read machine cycle timing diagram in minimum mode of 8086. **(5 Marks)**

(b) Write a short note on mixed language programming. **(5 Marks)**

(c) Explain flag register of 80386 microprocessor. **(5 Marks)**

(d) Give formats of initialization command words(ICW's) of 8259 PIC. **(5 Marks)**

Q. 2 (a) Explain the maximum mode configuration of 8086 microprocessor. **(10 Marks)**

(b) Design 8086 based system for following specifications:

(i) 8086 in minimum mode with clock frequency 5MHz.

(ii) 64 KB EPROM using 16KB*8 chips

(iii) I6 KB RAM using 8 KB*8 chips **(10 Marks)**

Q. 3 (a) Explain the branch prediction logic used in Pentium processor . **(10 Marks)**

(b) Draw and explain the block diagram of 8257 DMA controller. **(10 Marks)**

(b) (i) Explain the I/O mode control word format of 8255 PPI . **(5 Marks)**

(ii) Explain an instruction issue algorithm of Pentium processor. **(5 Marks)**

Q. 5 (a) Differentiate procedure and macro. Write a program to find the factorial of a number using procedure. **(10 Marks)**

(b) Explain the interrupt structure of 8086 microprocessor. **(10 Marks)**

Q. 6 (a) Explain segmentation of 8086 microprocessor. Give its advantages. **(10 Marks)**

(b) Explain different addressing modes of 8086 microprocessor. **(10 Marks)**

May 2019

Q. 1 (a) Give the advantages of memory segmentation of 8086 microprocessor. **(5 Marks)**

(b) Differentiate Procedure and macro with example. **(5 Marks)**

(c) Explain VM, RF, IOPL and NT flags of 80386 microprocessor. **(5 Marks)**

(d) Explain an instruction issue algorithm of Pentium processor. **(5 Marks)**

Q. 2 (a) Explain minimum mode configuration of 8086 microprocessor. **(10 Marks)**

(b) Explain cache organization of Pentium processor. **(10 Marks)**

Q. 3 (a) (i) Write a short note on mixed language programming. **(5 Marks)**

(ii) Write a program to find the largest number from an array. **(5 Marks)**

(b) Draw and explain the block diagram of 8255 Programmable Peripheral Interface (PPI) with control word formats. **(10 Marks)**

Q. 4 (a) Differentiate Real Mode, Protected Mode and virtual 8086 mode of 80386 microprocessor. **(10 Marks)**

(b) Design 8086 based system for following specifications : **(10 Marks)**

(i) 8086 in minimum mode with clock frequency 5MHz.

(ii) 128 KB EPROM using 32KB*8 chips.

(iii) 32 KB RAM using 16KB*8 chips.

Q. 5 (a) Explain different addressing modes of 8086 microprocessor. **(10 Marks)**

(b) Explain the operation of three 8259 PIC in cascaded mode. **(10 Marks)**

Q. 6 (a) Draw and explain memory read and memory write machine cycle timing diagrams in maximum mode of 8086. **(10 Marks)**

(b) Explain the following :

(i) Types of interrupts. **(5 Marks)**

(ii) Modes of 8253 Programmable Interval timer. **(5 Marks)**

❑❑❑

Your Success is Our Goal

Semester V - Computer Engineering

Computer Networks

Database Management System

MICROPROCESSOR

Theory of Computer Science

Multimedia System (Dept. Elective I)

Advance Operating System (Dept. Elective I)

Distributors

MUMBAI

Student's Agencies (I) Pvt. Ltd.
102, Konark Shram, Ground Floor, Behind Everest Building,156 Tardeo Road, Mumbai.
M :℡91672 90777.

Vidyarthi Sales Agencies
Shop. No. 5, Hendre Mansion, Khotachiwadi,157/159, J.S.S Road, Girgaum, Mumbai. M :℡98197 76110.

Bharat Sales Agency
Goregaonkar Lane, Behind Central Plaza Cinema, Charni Road, Mumbai. M :℡86572 92797

Ved Book Distributors - Mr. Sachin Waingade
(For Library Orders)
M : 80975 71421 / 92208 77214.
E : mumbai@techknowledgebooks.com

EM043A Price ₹ 45/-

BOOKS ARE AVAILABLE AT ALL LEADING BOOKSELLERS !!

B-50