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).

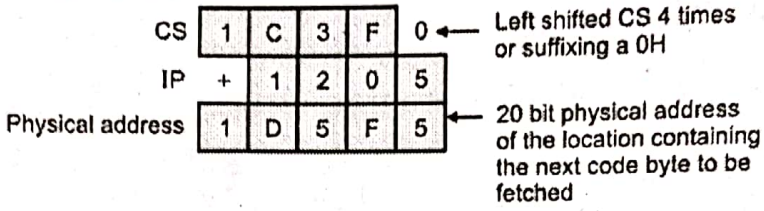


Fig. 1-Q.6(a) : Computation

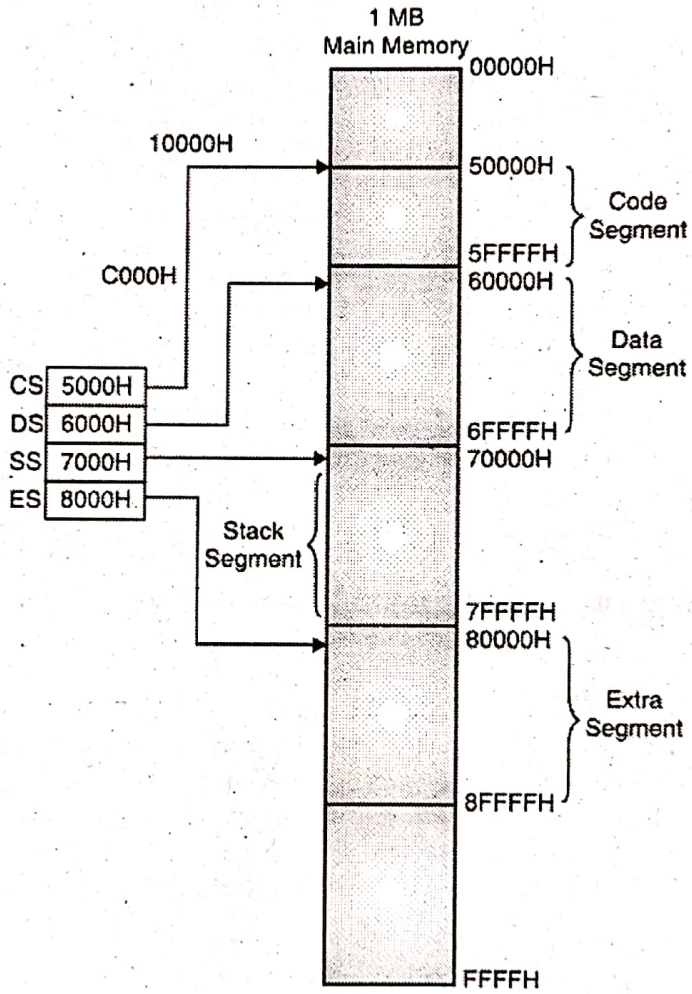


Fig. 2-Q. 6(a) : Segments in 8086

(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.

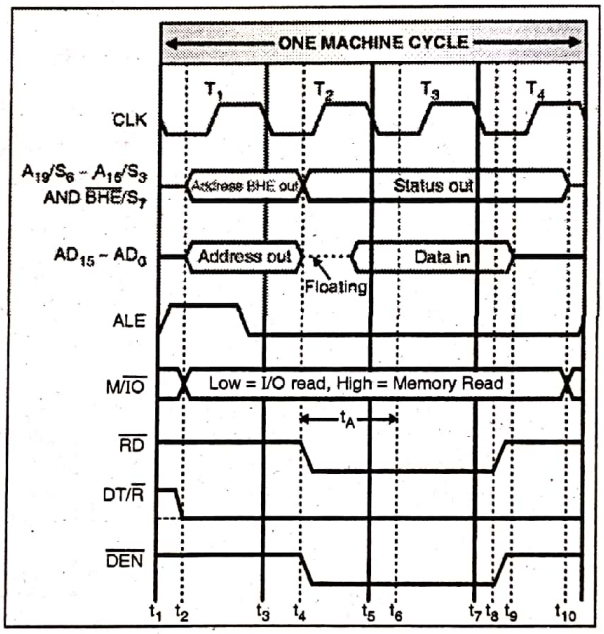


Fig. 1- Q. 1(a) : Timing diagram for the read machine cycle of 8086

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.

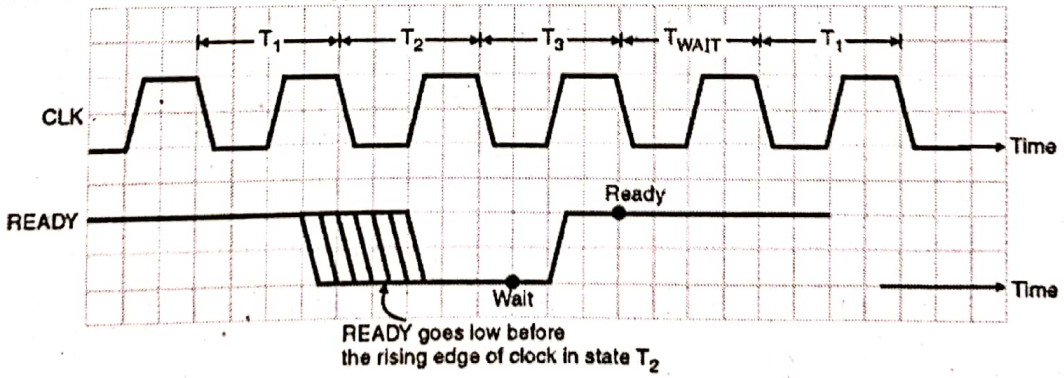


Fig. 2-Q. 1(a) : WAIT T-states

- **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.

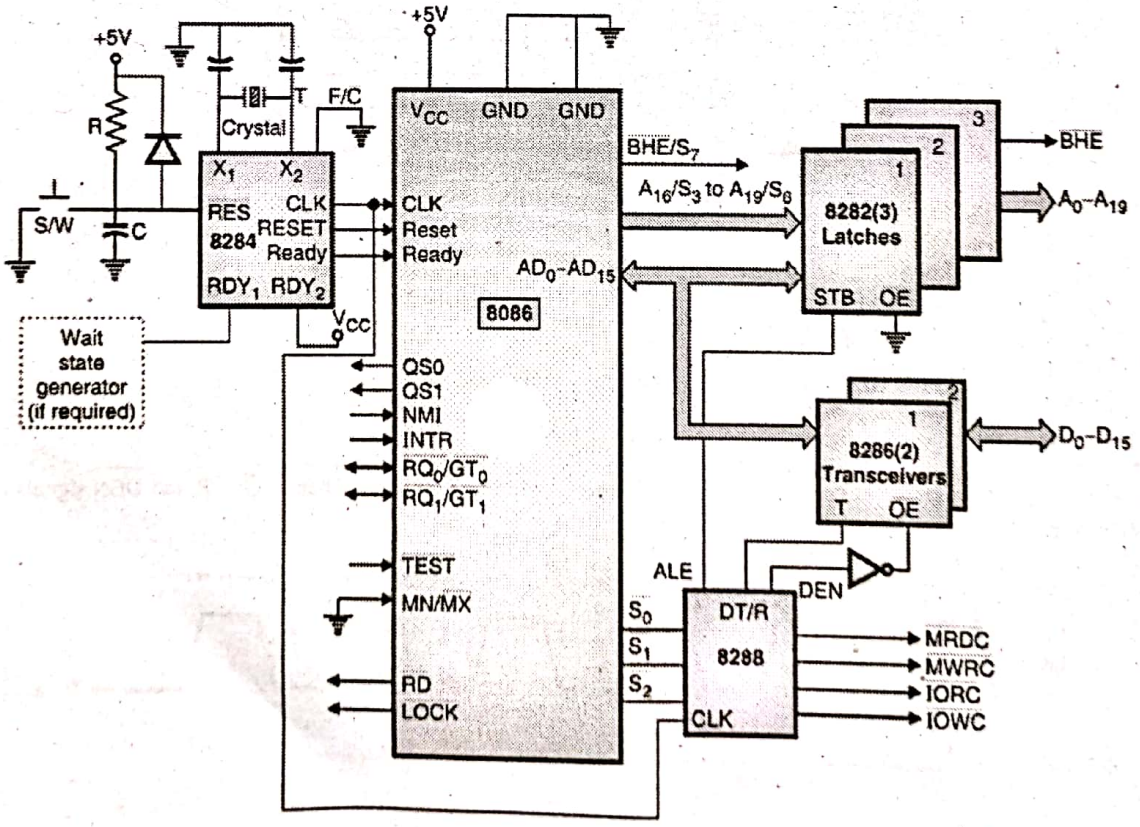


Fig. 1- Q. 2(a) : Block diagram of maximum mode minimum system

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.

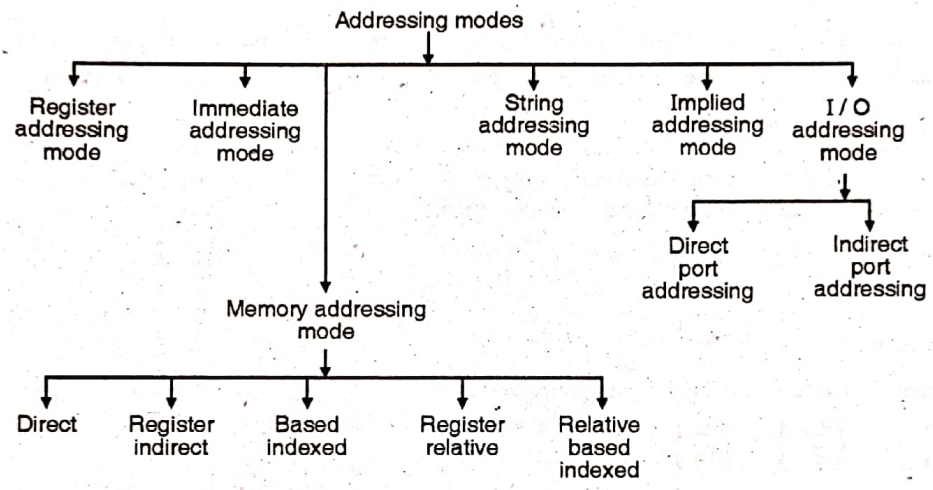


Fig. 1- Q. 6(b) : Addressing modes of 8086

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.

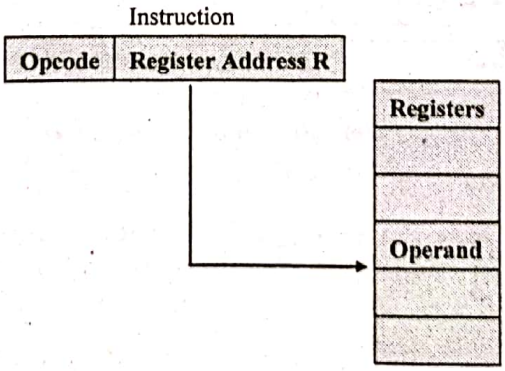


Fig. 2- Q. 6(b) : Register addressing mode

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.

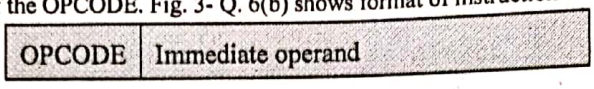


Fig. 3-Q. 6(b) : Instruction encoded with an immediate operand

- 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.

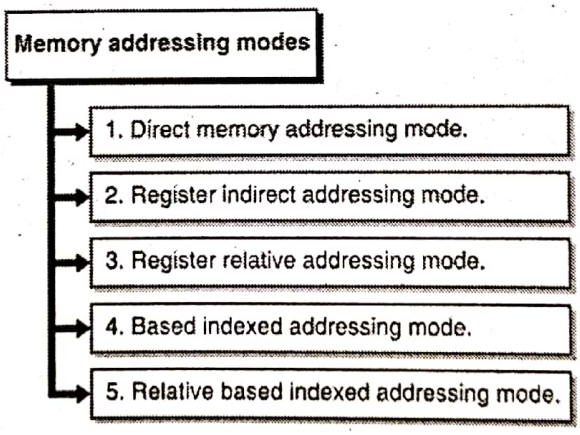


Fig. 4-Q. 6(b) : Memory addressing modes

(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.

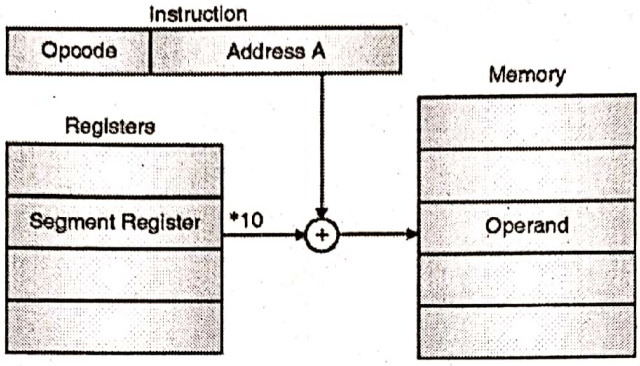


Fig. 5-Q. 6(b) : Direct addressing.

(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

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

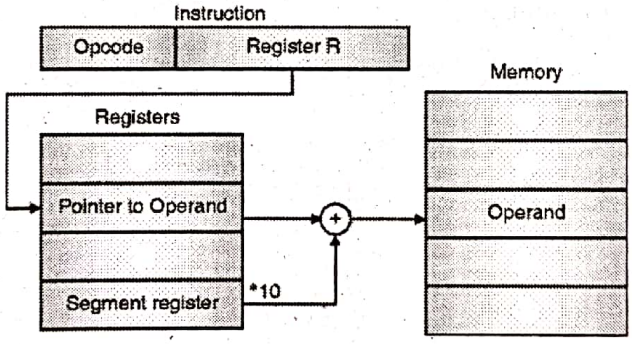


Fig. 6-Q. 6(b) : Register indirect addressing modes

– **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).

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

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

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

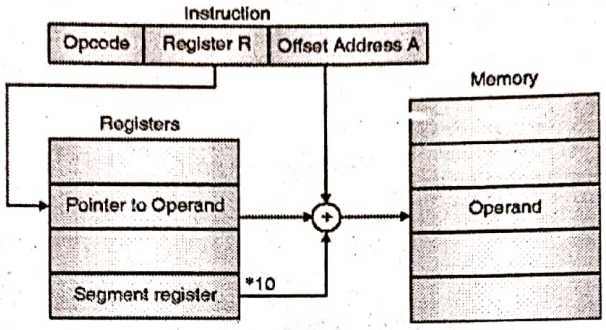


Fig. 7-Q. 6(b) : Register relative addressing mode

– **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).

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

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

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

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

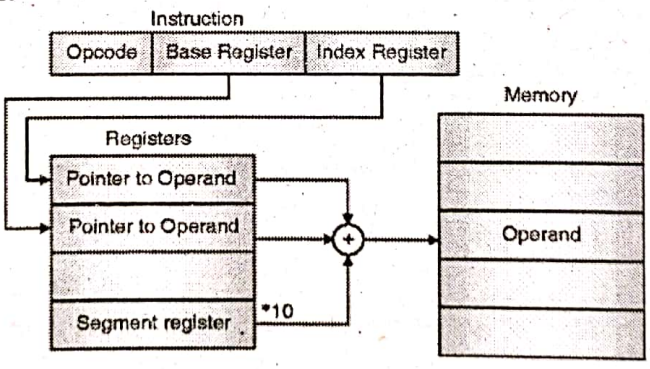


Fig. 8-Q. 6(b) : Based indexed addressing mode

– **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).

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

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

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

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

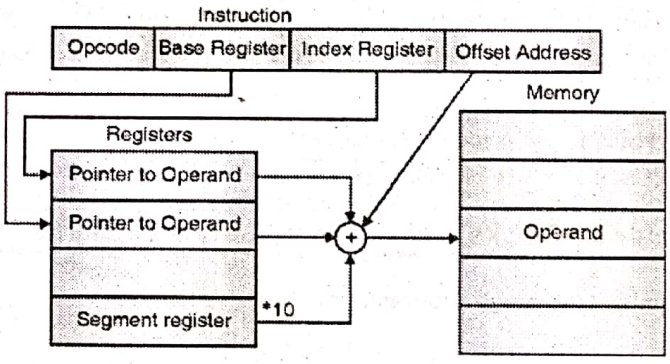


Fig. 9-Q. 6(b) : Relative Based Indexed Addressing Mode

- **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.

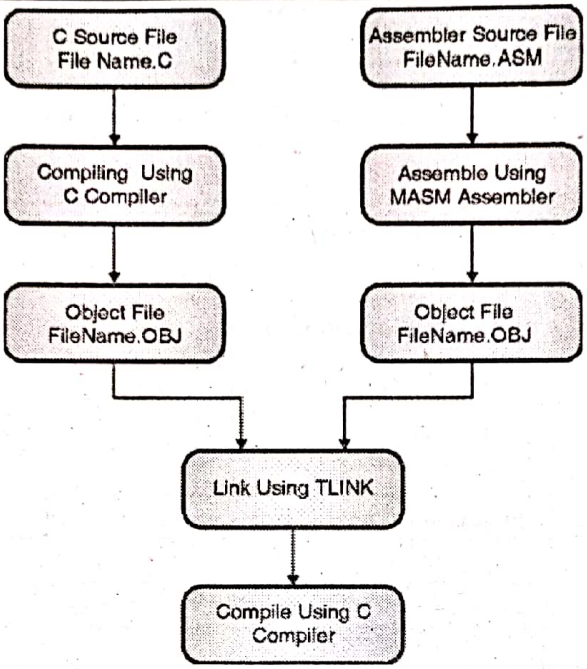


Fig. 1-Q.1(b) : Combining C and assembly

(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

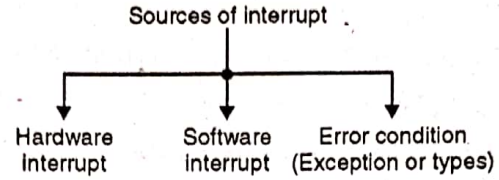


Fig. 1-Q. 5(b) : Interrupt structure of 8086 microprocessor

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.

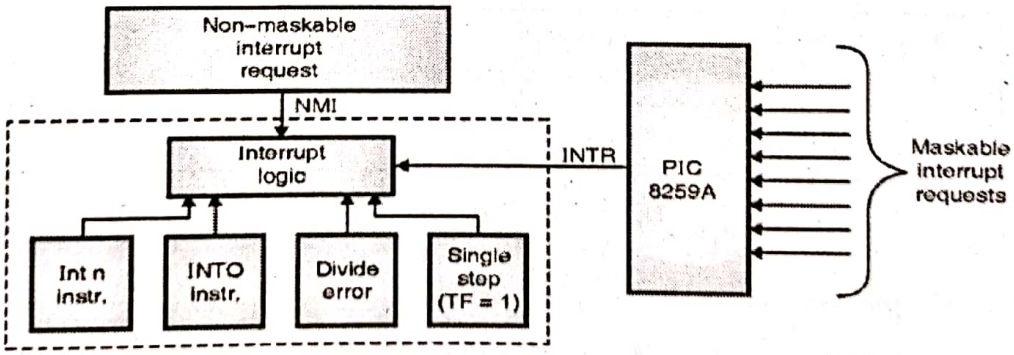


Fig. 2-Q. 5(b) : Interrupt sources

- 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.

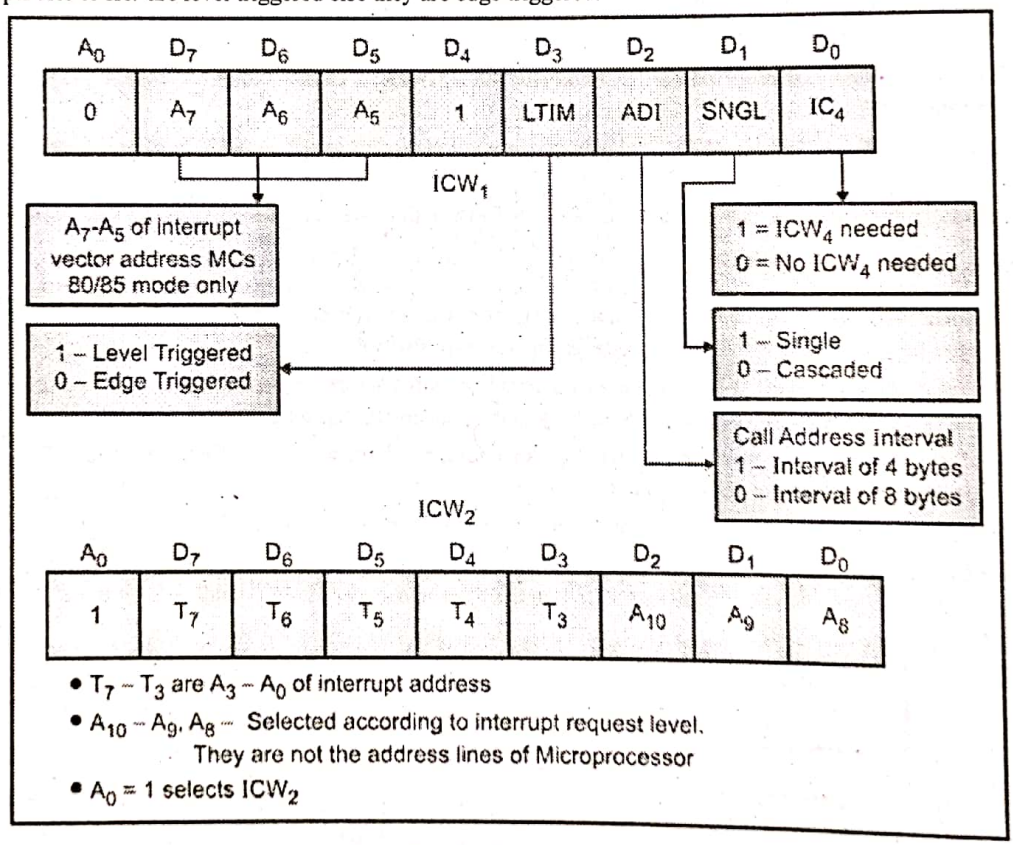


Fig.1-Q.1(d) : Format of ICWs

- **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).

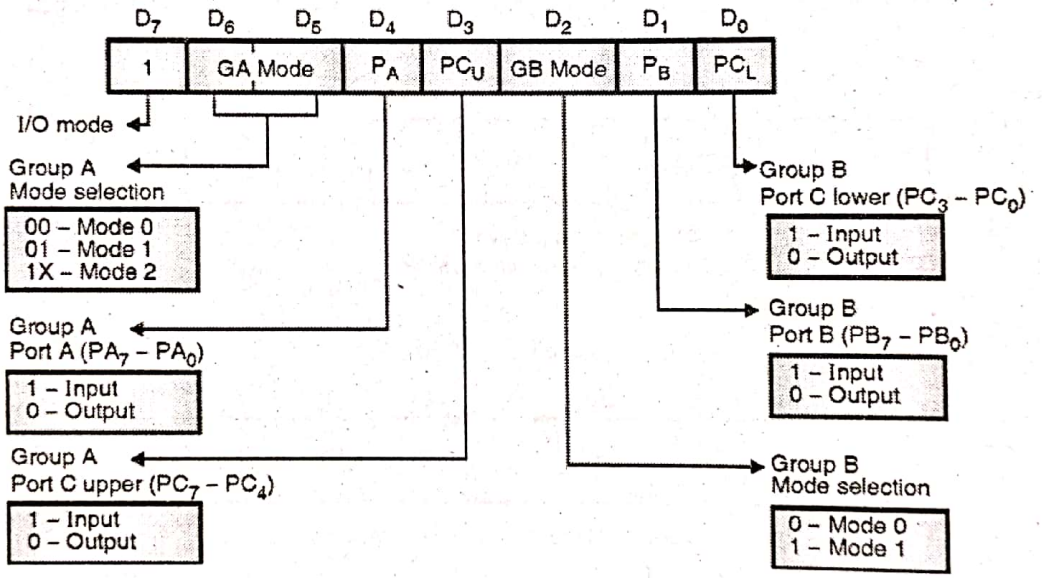


Fig. 1- Q.4(b) : I/O modes control word format

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. :

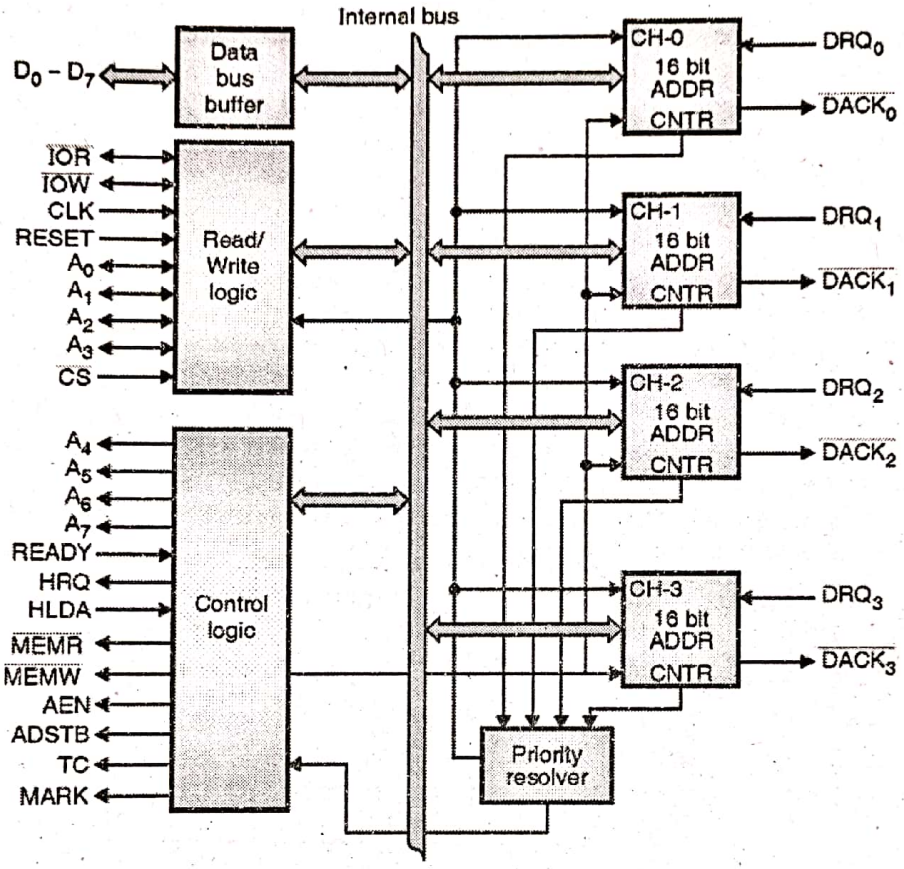


Fig. 1- Q. 3(b) : 8257 Functional Block Diagram

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 :

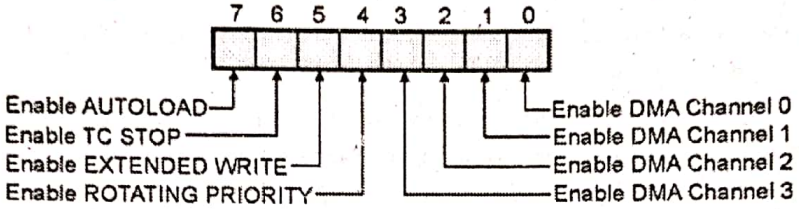


Fig. 2- Q. 3(b) : Mode register

- 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.

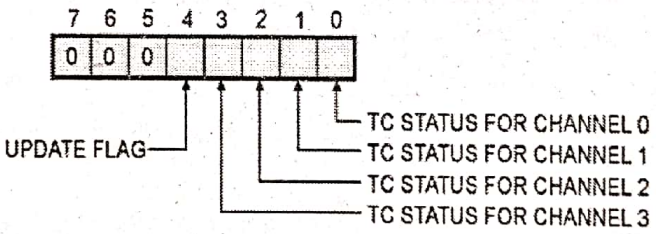


Fig. 3- Q. 3(b) : Status Register

- 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)

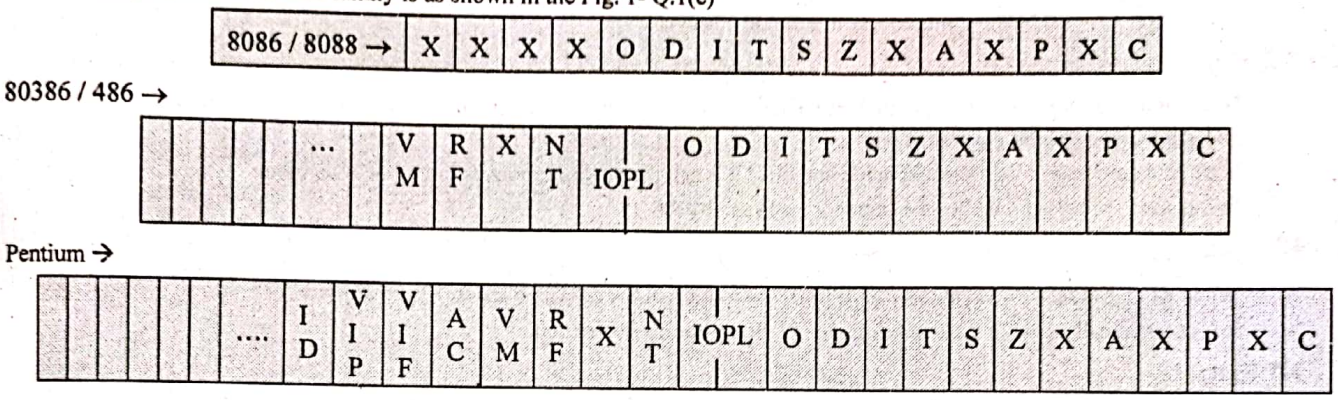


Fig. 1- Q.1(c) : Flag Register

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.

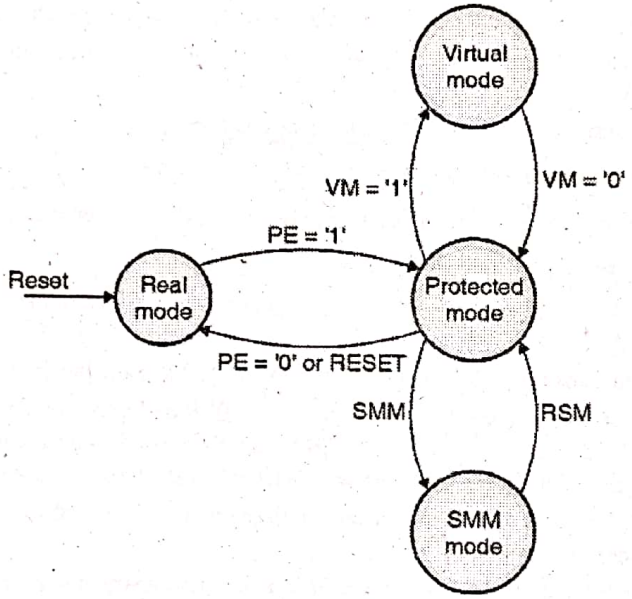


Fig. 1- Q. 4(a)

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.

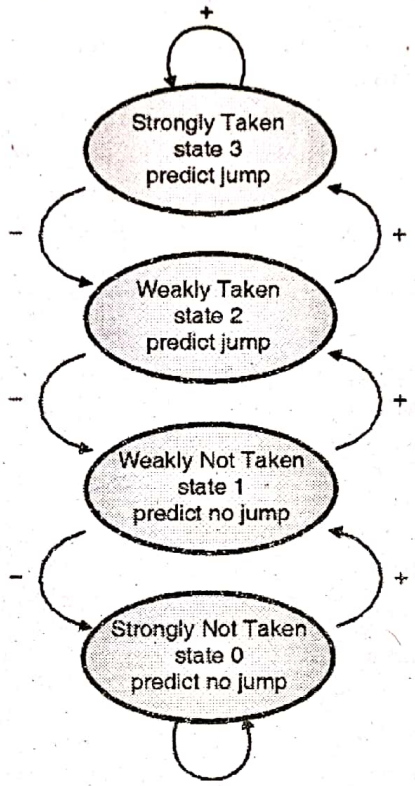


Fig. 1-Q. 3(a) : State transition diagram of BTB entry

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

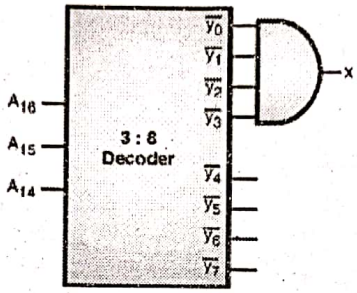
	Even Bank	Odd Bank
Starting Address	00000H	00001H
Ending Address	03FFEH	03FFFH

Step 3 : Memory Map :

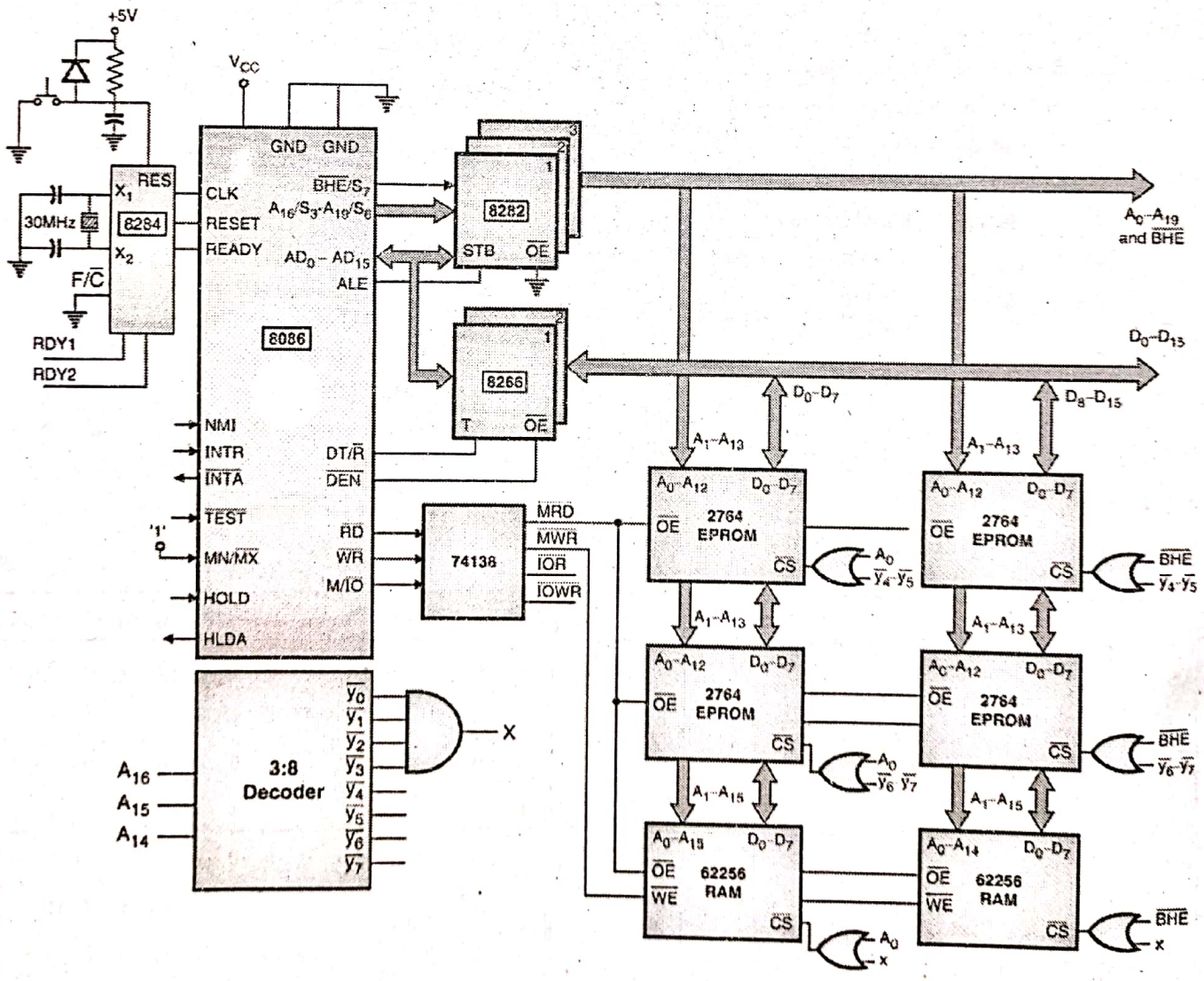
EA $\Rightarrow$ Ending address, SA $\Rightarrow$ Starting address, EB $\Rightarrow$ Even Bank, OB $\Rightarrow$ Odd Bank

			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.

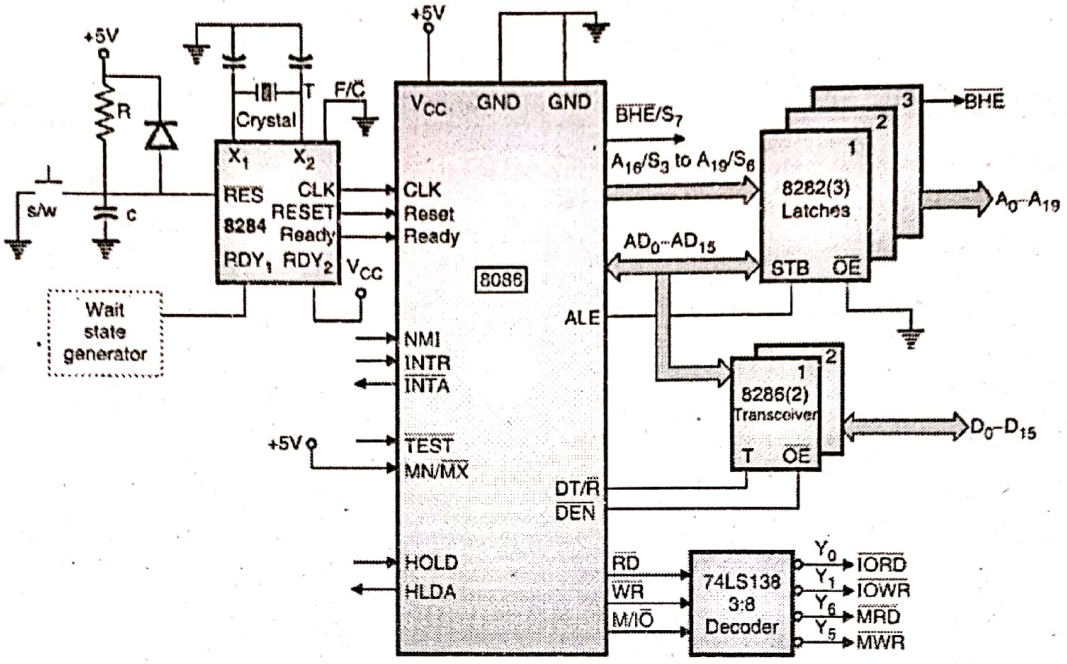


Fig. 1- Q.2(a) : Block diagram of minimum system in minimum mode

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.

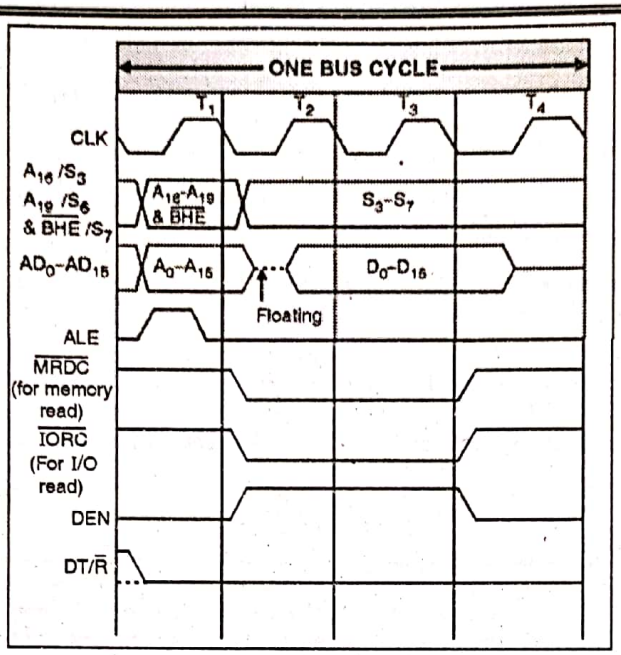


Fig. 1- Q. 6(a) : Maximum mode read bus cycle

- **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.

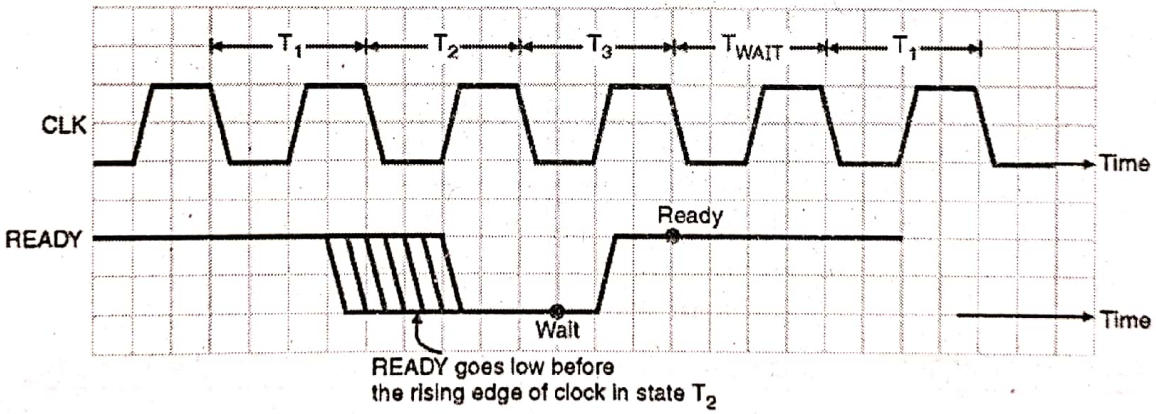


Fig. 2- Q. 6(a) : WAIT T-states

- 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

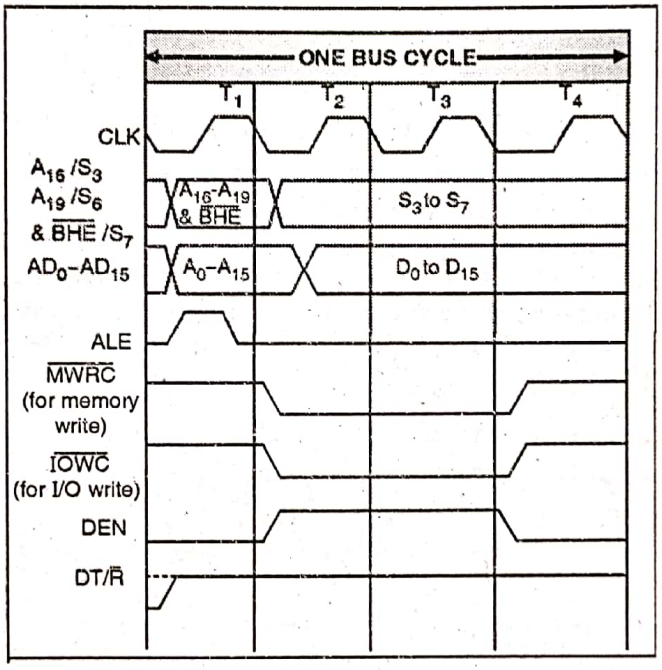


Fig. 3- Q. 6(a) : Maximum mode write cycle

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.

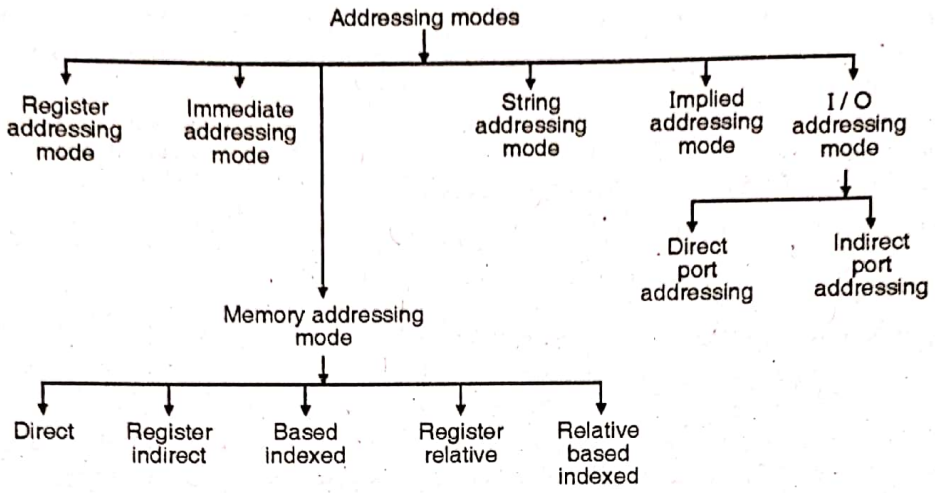


Fig. 1- Q.5(a) : Addressing modes of 8086

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.

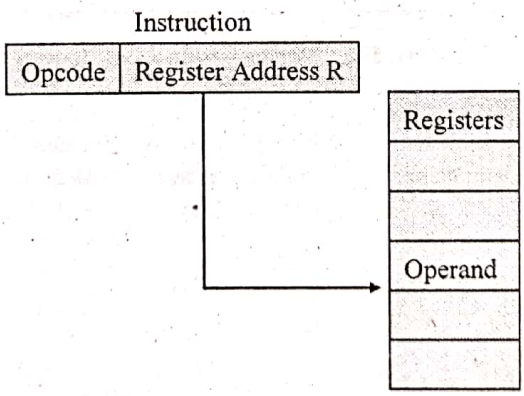


Fig. 2- Q. 5(a) : Register addressing

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.

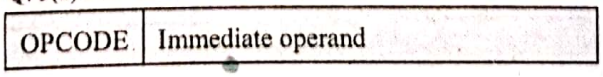


Fig. 3- Q. 5(a) : Instruction encoded with an immediate operand

- 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.

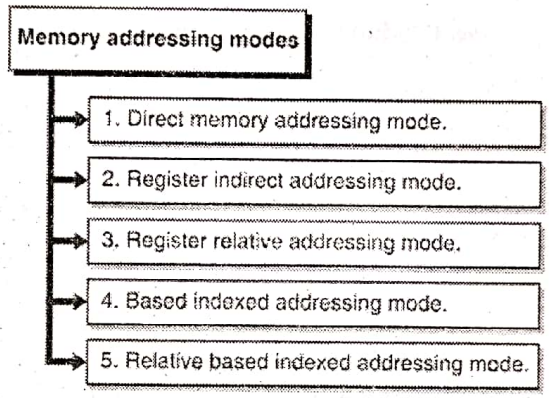


Fig. 4- Q. 5(a) : Memory Addressing Modes

(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.

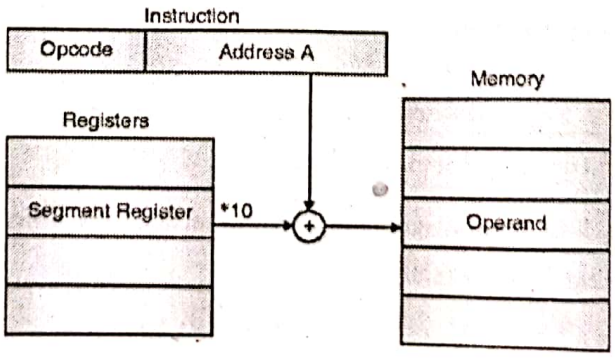


Fig. 5- Q. 5(a) : Direct addressing

(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

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

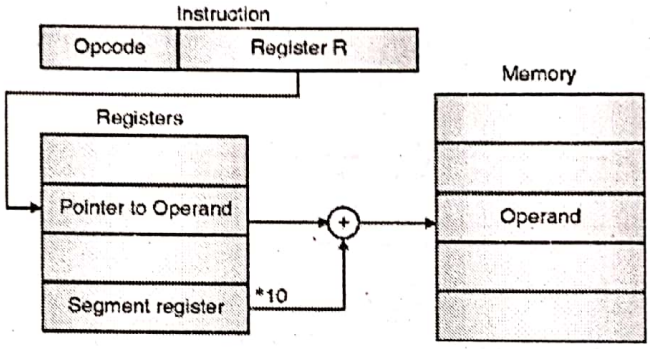


Fig. 6- Q. 5(a) : Register indirect addressing modes

- **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).

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

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

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

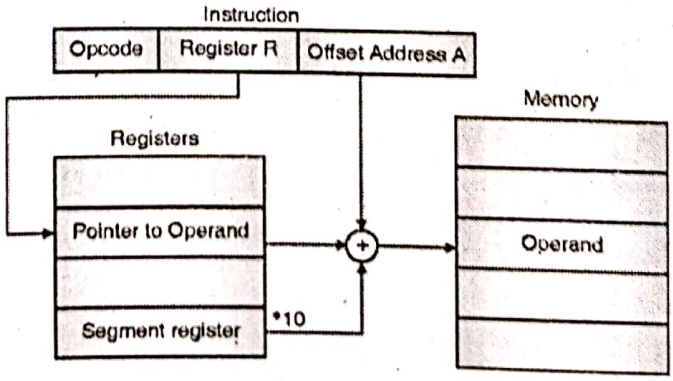


Fig. 7-Q. 5(a) : Register relative addressing mode

– **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).

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

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

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

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

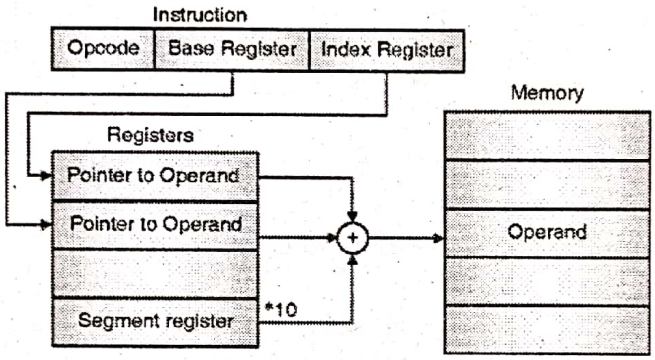


Fig. 8- Q. 5(a) : Based indexed addressing mode

– **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).

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

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

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

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

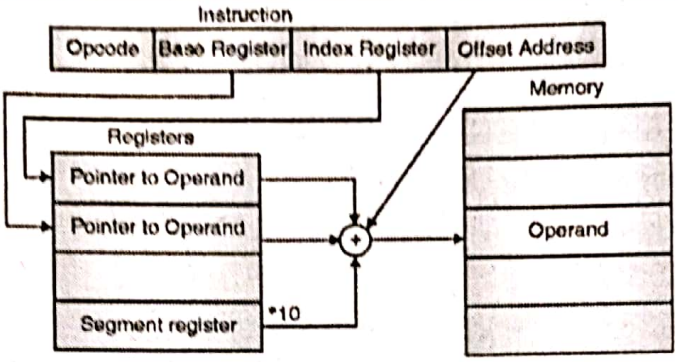


Fig. 9- Q. 5(a) : Relative Based Indexed Addressing Mode

- **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.

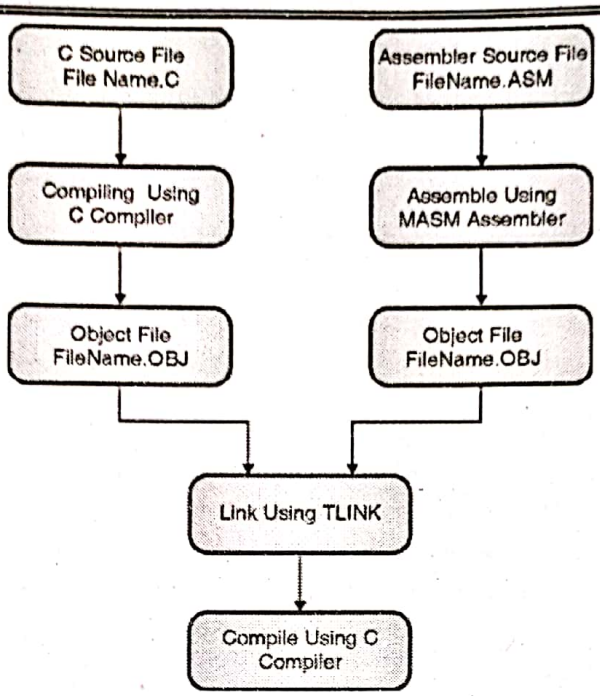


Fig. 1- Q. 3(a) : Combining C and assembly

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

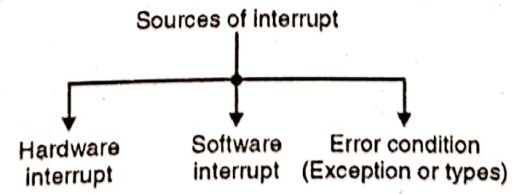


Fig. 1- Q. 6(b) : Interrupt structure of 8086 microprocessor

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.

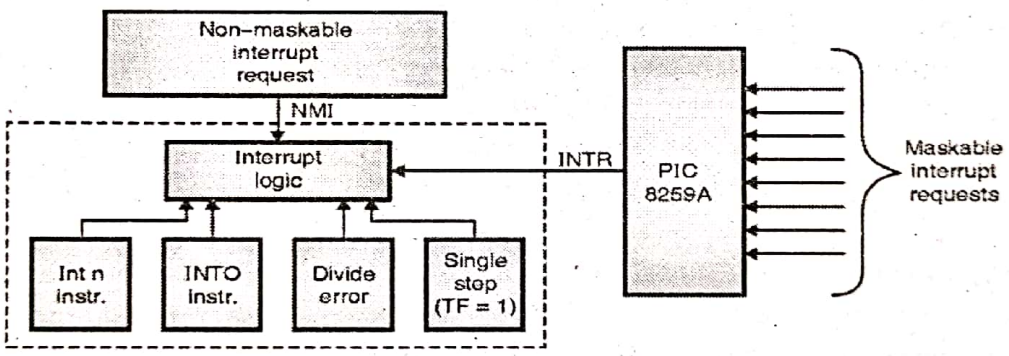


Fig. 2-Q. 6(b) : Interrupt sources for 8086

- 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.

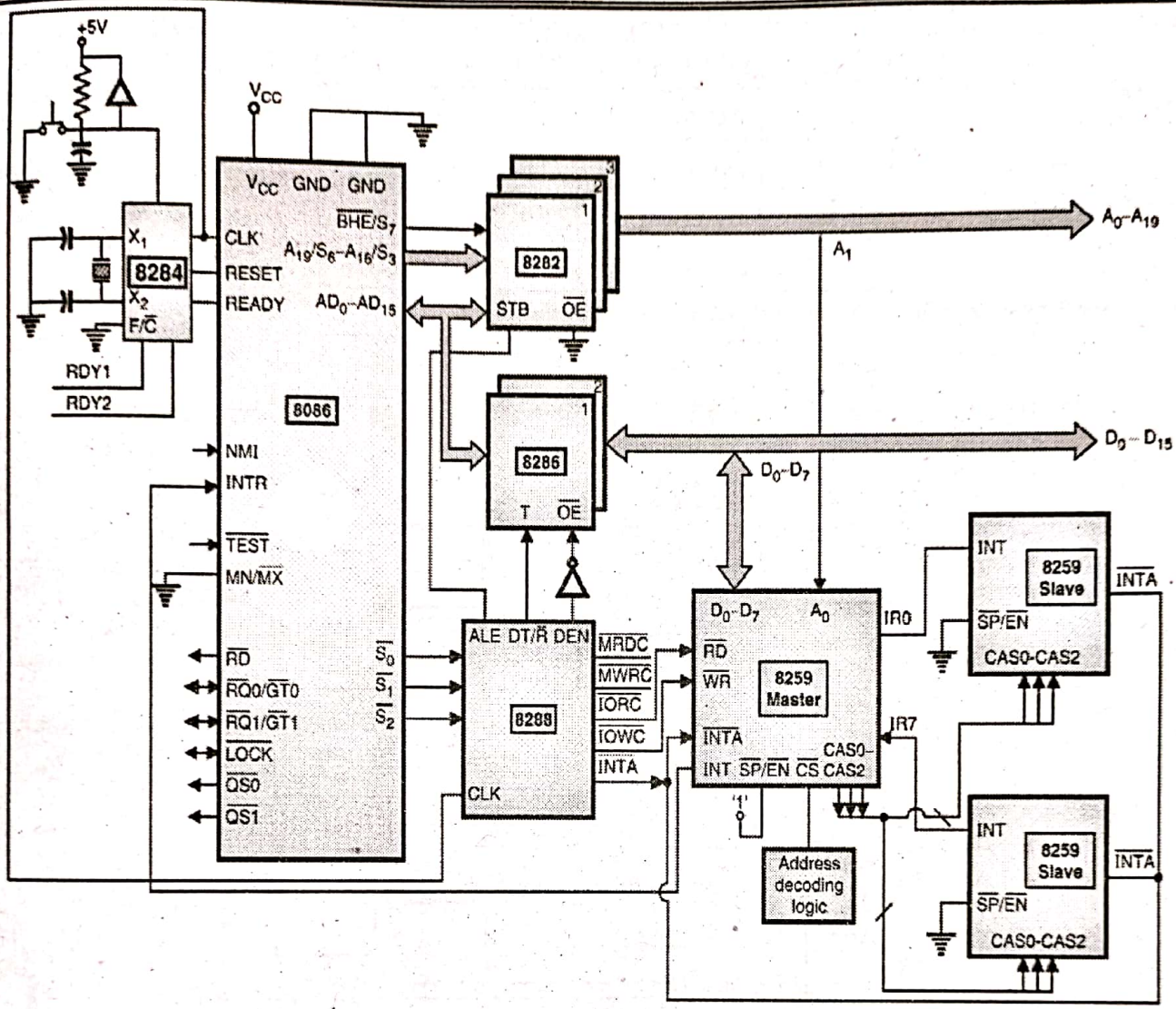


Fig. 1- Q. 5(b) : Interfacing 8259 with 8086 (8259 – cascaded, 8086- maximum mode)

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.

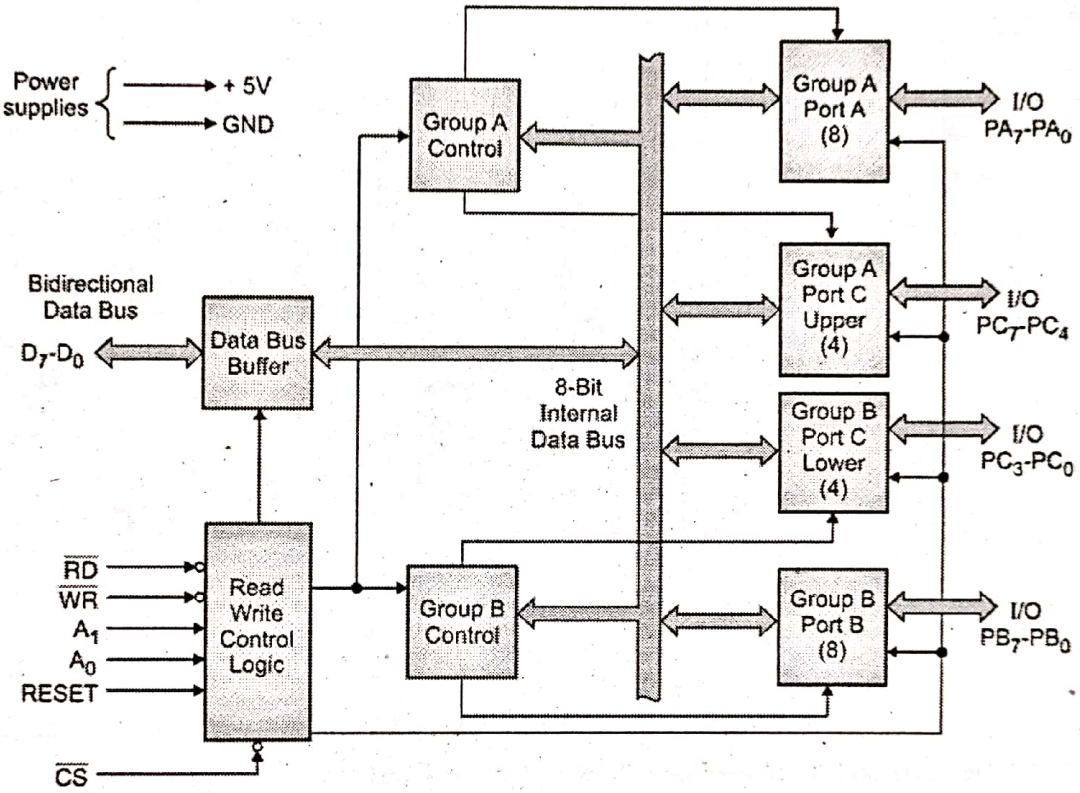


Fig. 1-Q.3(b) : Block Diagram of 8255

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.

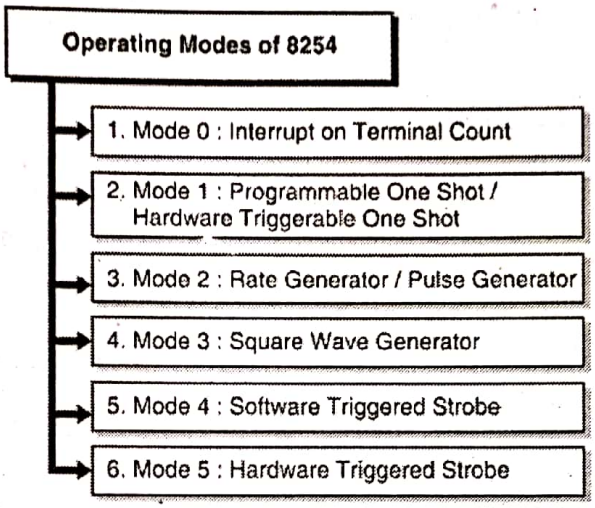


Fig. 1-Q. 6(b) : Operating modes of 8254

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.

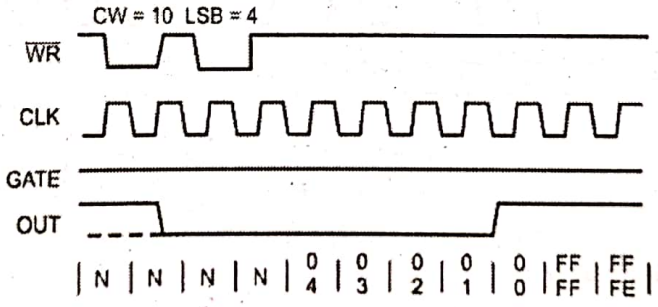


Fig. 2-Q. 6(b) : Mode 0 timing diagram

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.

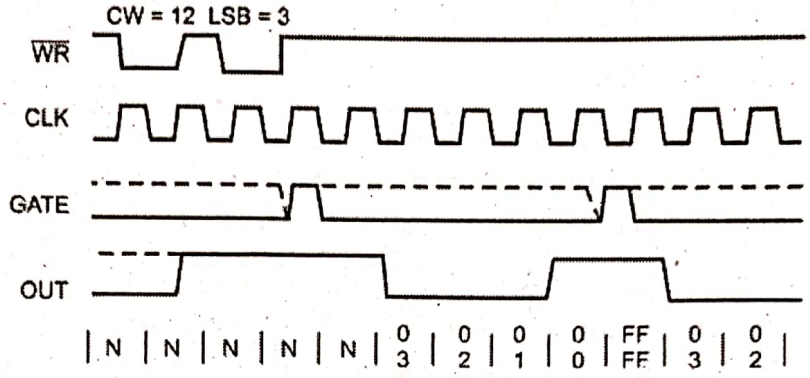


Fig. 3-Q. 6(b) : Mode 1 timing diagram

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.

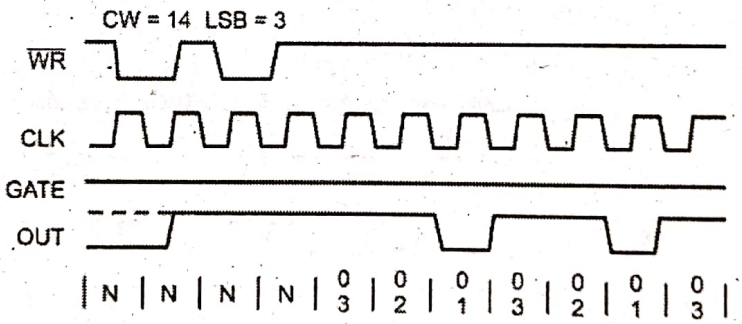


Fig. 4-Q. 6(b) : Mode 2 timing diagram

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.

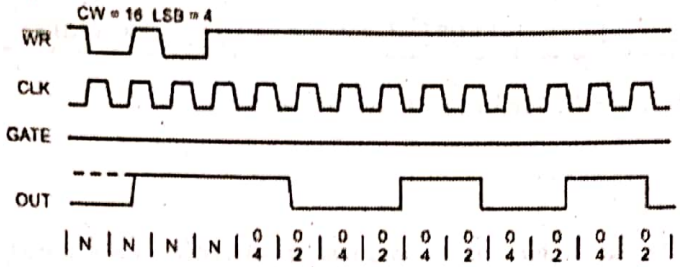


Fig. 5-Q. 6(b) : Mode 3 timing diagram

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.

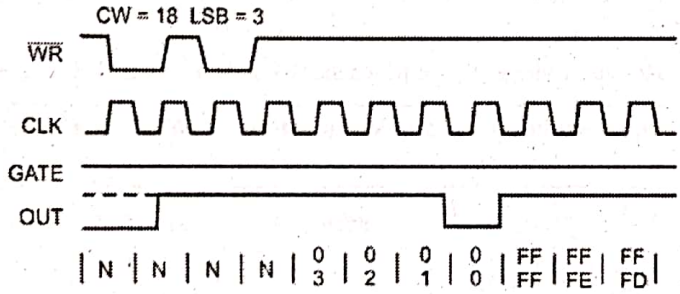


Fig. 6-Q. 6(b) : Mode 4 timing diagram

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.

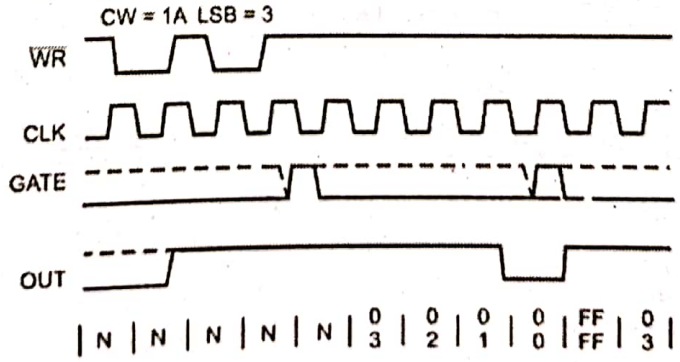


Fig. 7-Q. 6(b) : Mode 5 timing diagram

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.

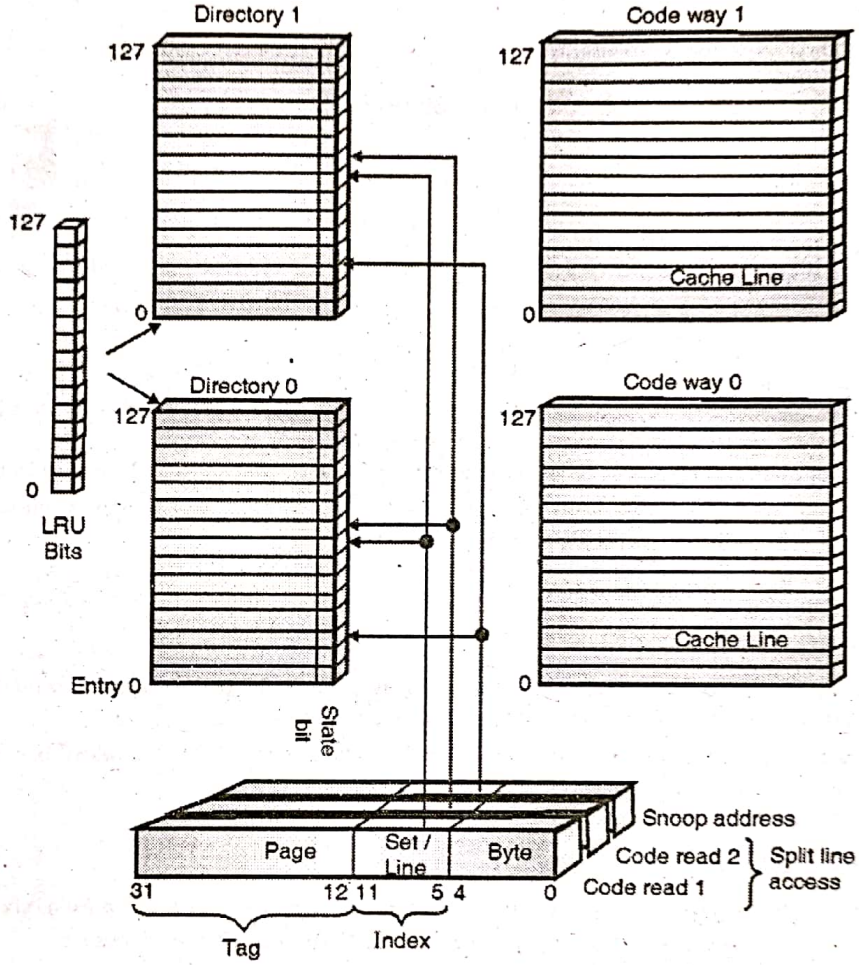


Fig. 1-Q. 2(b) : Pentium code cache organization

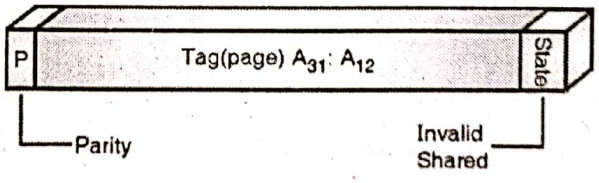


Fig. 2-Q. 2(b) : Code cache directory entry

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).

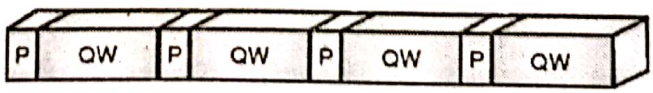


Fig. 3-Q. 2(b) : Line structure in code cache

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.

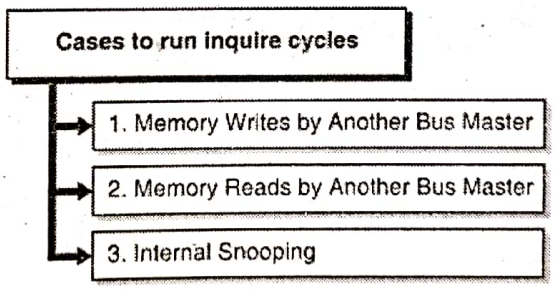


Fig. 4-Q.2(b) : Cases to inquire cycles

(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).

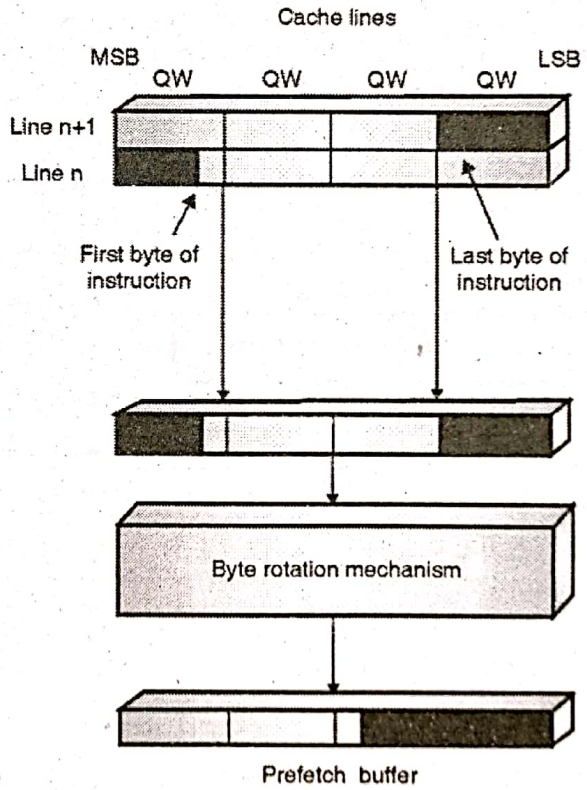


Fig. 5-Q. 2(b) : Split line access

(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

Microprocessor

Chapter 1 : Introduction

Q. 1 Define (a) Opcode (b) Interrupt.

Ans. :

(a) **Opcode :** A binary code, that indicates the operation to be performed is called as an Opcode.

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

Q. 2 Enlist the characteristics of a microprocessor.

Ans. :

The power of the microprocessor is determined by the following characteristics of the microprocessor :

1. A processor is n-bit processor implies that
   a) Its ALU is n-bit, i.e. the ALU can perform n-bit operation simultaneously.
   b) Its internal data bus and register size is n-bit.
   c) Its external data bus is also n-bit (in most of the cases).
2. **Processing capability :** It depends upon the number of instructions and flexibility of each instruction.
3. **Word length :** It depends upon the width of internal data bus, registers, ALU etc .
4. **Clock frequency:** The processing speed of microprocessor depends upon clock frequency. The program execution speed is also determined by this parameter. The maximum clock frequency depends upon technology adopted in microprocessor fabrication.
5. **Width of the data bus :** This parameter decides word length of the microcomputer. This is the width of the external data bus .
6. **Width of the address bus :** This parameter decides the memory addressing capability of the microprocessor. The maximum size of the memory is decided by this parameter.
7. **I/O addressing capability :** The maximum number of the I/O ports accessed by the microprocessor depends upon the width of the I/O address provided in the I/O instruction.
8. **Data types :** The microprocessor handles various types of data formats like binary, BCD, ASCII, integers, real numbers, signed numbers and unsigned numbers etc.
9. **Interrupt capability :** Interrupts are used to handle unpredictable and random events in the microcomputer. It is used to interrupt the microprocessor. It is also used to speed up the I/O programs. It improves the throughput of the system.

Q. 3 Write short note on basic functions of microprocessor. May 14

Ans. :

Fig. 1.1 shows the architecture of microprocessor. This architecture is divided in different groups as follows :

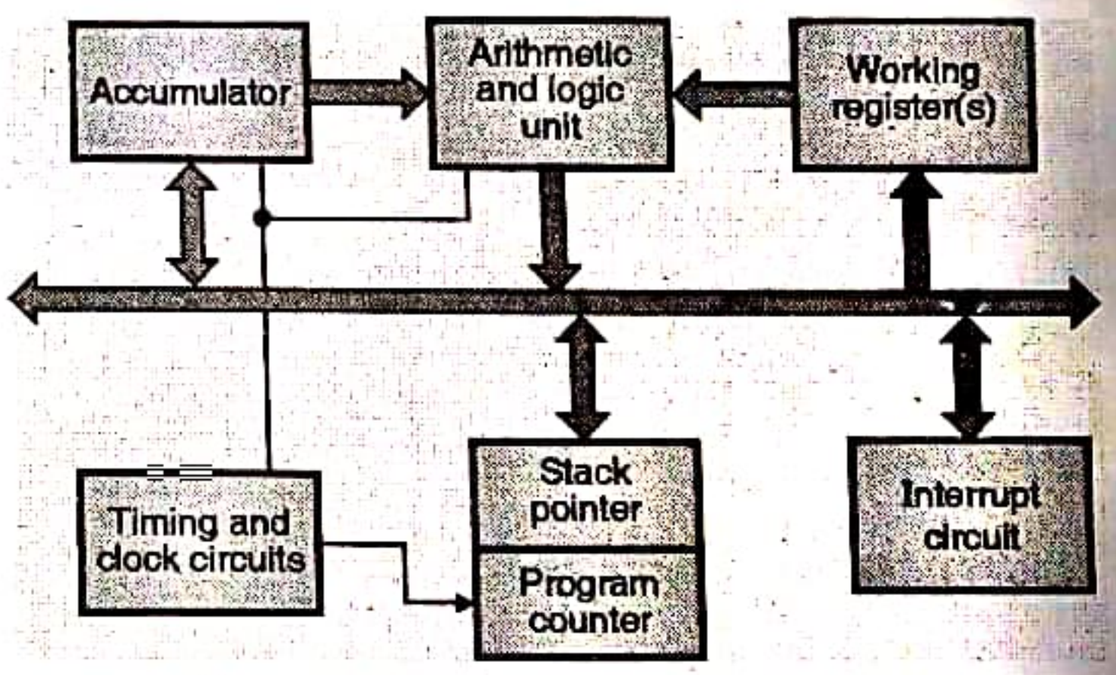


Fig. 1.1 : General architecture of a microprocessor

1. Register Section

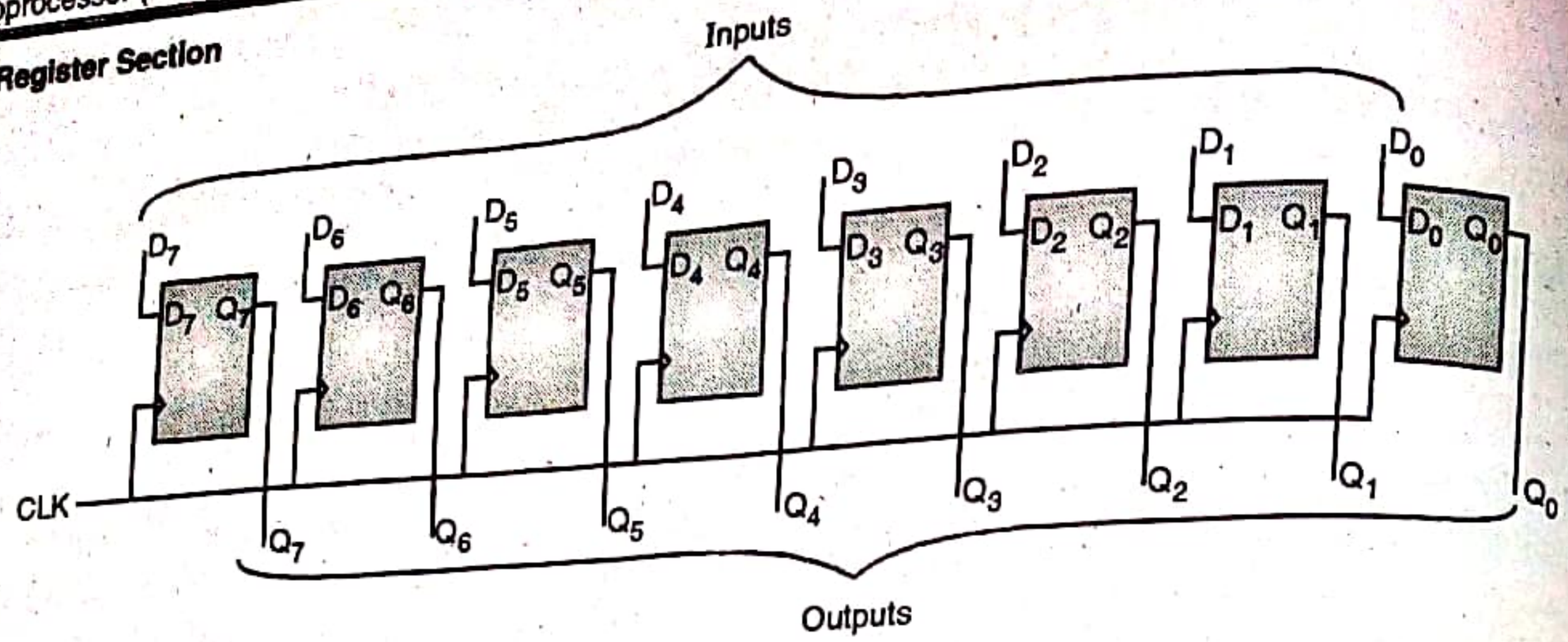


Fig. 1.2 : 8 bit register

It consists of PIPO (Parallel In Parallel Out) register as shown in Fig. 1.2. This section is also called as scratch pad memory. It stores data and address of memory. The register organization affects the length of program, the execution time of program and simplification of the program.

To achieve better performance, the number of registers should be large. The architecture of microcomputer depends upon the number and type of the registers used in microprocessor. It consists 8-bit registers or 16 bit registers. The register section varies from microprocessor to microprocessor. The registers are used to store the data and address.

These registers are classified as :

(i) Temporary registers

(ii) General purpose registers

(iii) Special purpose registers.

2. Arithmetic and Logical Unit

This section processes data i.e. it performs arithmetic and logical operations. It performs arithmetic operations like addition, subtraction and logical operations like ANDing, ORing, EX-ORing, etc. The ALU is not available to the user. Its word length depends upon the width of an internal data bus. The ALU is controlled by timing and control circuits. It accepts operands from memory or register. It stores result of arithmetic and logic operations in register or memory. It provides status of result to the flag register. Flag register shows status of result. ALU looks after the branching decisions.

3. Interrupt Control

This block accepts different interrupt request inputs. When a valid interrupt request is present it informs control logic to take action in response to each signal.

4. Timing and Control Unit

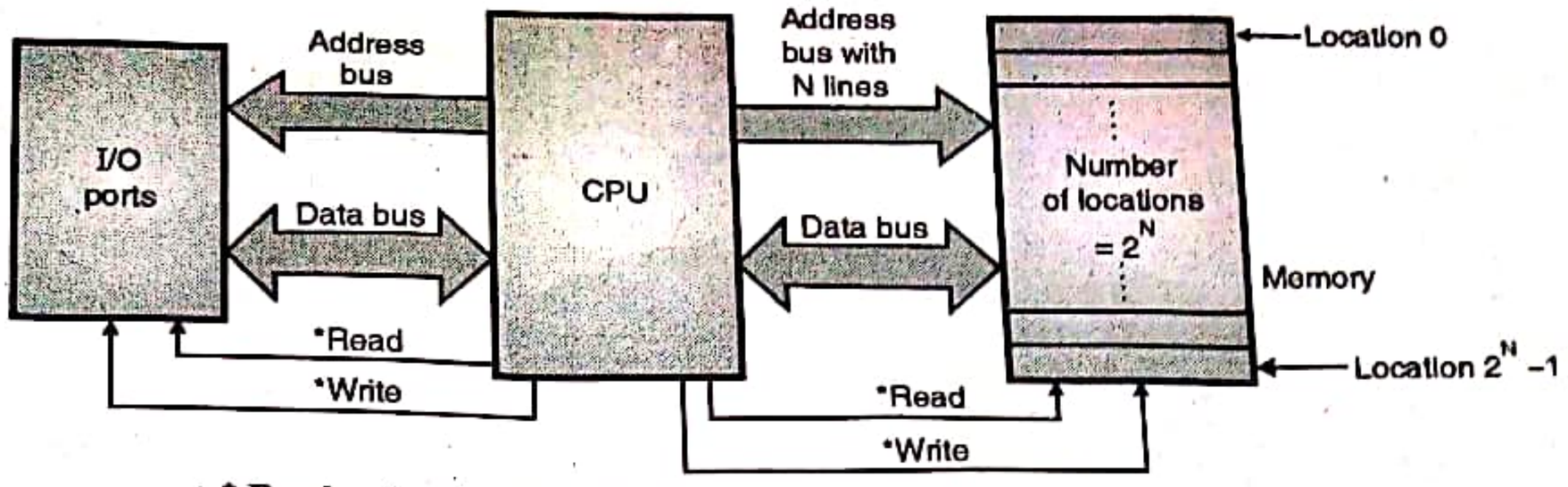
This is a control section of microprocessor made up of synchronous sequential logic circuit. It controls all internal and external circuits. It operates with reference to clock signal. This accepts information from instruction decoder and generates microsteps to perform it. In addition to this, the block accepts clock inputs, performs sequencing and synchronizing operations.

The synchronization is required for communication between microprocessor and peripheral devices. To implement this it uses different status and control signals. The basic operation of a microprocessor is regulated by this unit. It synchronizes all the data transfers. This unit takes appropriate actions in response to external control signals.

Q. 4 What are the functions of the microprocessor buses?

Ans. : Address Bus

The bus over which the CPU sends out the address of the memory location is called as the **address bus**. The address bus carries the address of the memory location to be written to or read from. The address bus may consist of 16, 20, 24 or 32 parallel signal lines. If there are N address lines, then it can directly address 2^N memory locations.



*** Read and write signals are a part of control bus**

Fig. 1.3 : The three types of buses and their utility

As shown in Fig. 1.3, the address bus is a unidirectional bus (indicated by the arrow). The address bus is also used to send the I/O port address on the address bus. When the CPU reads data from or writes data to a port, then it sends the port address out on the address bus.

Data Bus

The data bus consists of 8, 16 or 32 parallel lines. The data bus is a bi-directional bus. That means the data can get transferred from CPU to memory and vice versa. The data bus also connects the I/O ports and CPU. So the CPU can write data to or read it from the memory or I/O ports. The number of data lines used in the data bus is equal to the size of data word that can be written or read simultaneously. Many devices in the system will connect their outputs to the same data bus. But only one device at a time will have its output enabled, so that there is no data contention (collision of information). In order to completely eliminate the possibility of more than one devices putting their outputs simultaneously on the data bus, all the devices getting connected to data bus should have **three state outputs**. viz. logic '0', logic '1' and high impedance state. This will make it possible to disable the outputs of devices which should be stopped from connecting their outputs to the data bus.

Control Bus

The control bus is used for sending control signals to the memory and input/output devices, as shown in Fig. 1.3.

The CPU sends signals on the control bus to enable the outputs of addressed memory devices or I/O port devices.

Some of the control bus signals are as follows :

1. Memory read 2. Memory write 3. I/O read 4. I/O write

Chapter 2 : Architecture of 8086 Microprocessor

Q. 1 Compare the 8085 and 8086 microprocessor w.r.t. architecture, instruction set, speed, memory organization. May 11

Ans. : Comparison of 8085 and 8086 microprocessor

Sr. No.	Parameter	8085	8086
1.	**Size**	It is an 8 bit Microprocessor.	It is 16 bit Microprocessor.
2.	**Address Bus**	Its address bus is of 16 bits.	Its address bus is of 20 bits.
3.	**Memory**	It can access upto 2^{16} = 65,536 bytes i.e. 64 Kbytes of memory.	It can access upto 2^{20} = 1,048,576 bytes i.e. 1 MB of memory.
4.	**Instruction Queue**	It does not have an instruction queue.	It has a 6 byte instruction queue
5.	**Pipelining**	It does not support pipelined architecture.	It supports 2 stage pipelined architecture.
6.	**Multiprocessor Support**	It does not have multiprocessing support.	It supports multiprocessing. It has compatibility with further processors like Intel 80386, 80486 and Pentium.
7.	**I/Os**	It can address 2^8 = 256 I/Os.	It can address 2^{16} = 65,536 I/Os.

Sr. No.	Parameter	8085	8086
8.	Coprocessor Interface	It does not have coprocessor interface.	It has coprocessor interface. Coprocessor 8087 can be interfaced with 8086.
9.	Arithmetic support	It only supports integer and decimal arithmetic.	It supports integer, decimal and ASCII arithmetic.
10.	Multiplication and Division	It does not have instruction that computes the multiplication and division operation.	It supports instructions that compute the multiplication and division operation.
11.	Check speed	It operates on 3 MHz, 5 MHz or 6 MHz. i.e. it operates on low clock speed.	It operates on 5 MHz, 8 MHz or 10 MHz. i.e. it operates on high clock speed.
12.	External Hardware	It requires less external hardware.	It requires more external hardware.
13.	Operating modes	It supports single operating mode.	It supports two different operating modes : maximum mode and minimum mode.
14.	Addressing modes	It supports 5 addressing modes.	It supports the addressing modes supported by microprocessor 8085. It also supports some additional addressing modes.
15.	Cost	Its cost is low.	Its cost is high.
16.	Multiplexed pins	It has very few multiplexed pins.	It has number of multiplexed pins than 8085.
17.	Memory Segmentation	The memory spaces not segmented.	The memory space is segmented.
18.	Interrupts	8085 provides 8 software vectored interrupts and five hardware interrupts.	8086 has 256 software vectored interrupts and 2 hardware interrupts.

Q. 2 Explain architecture of 8086. Dec. 13, May 14

Ans. :

Architecture of 8086

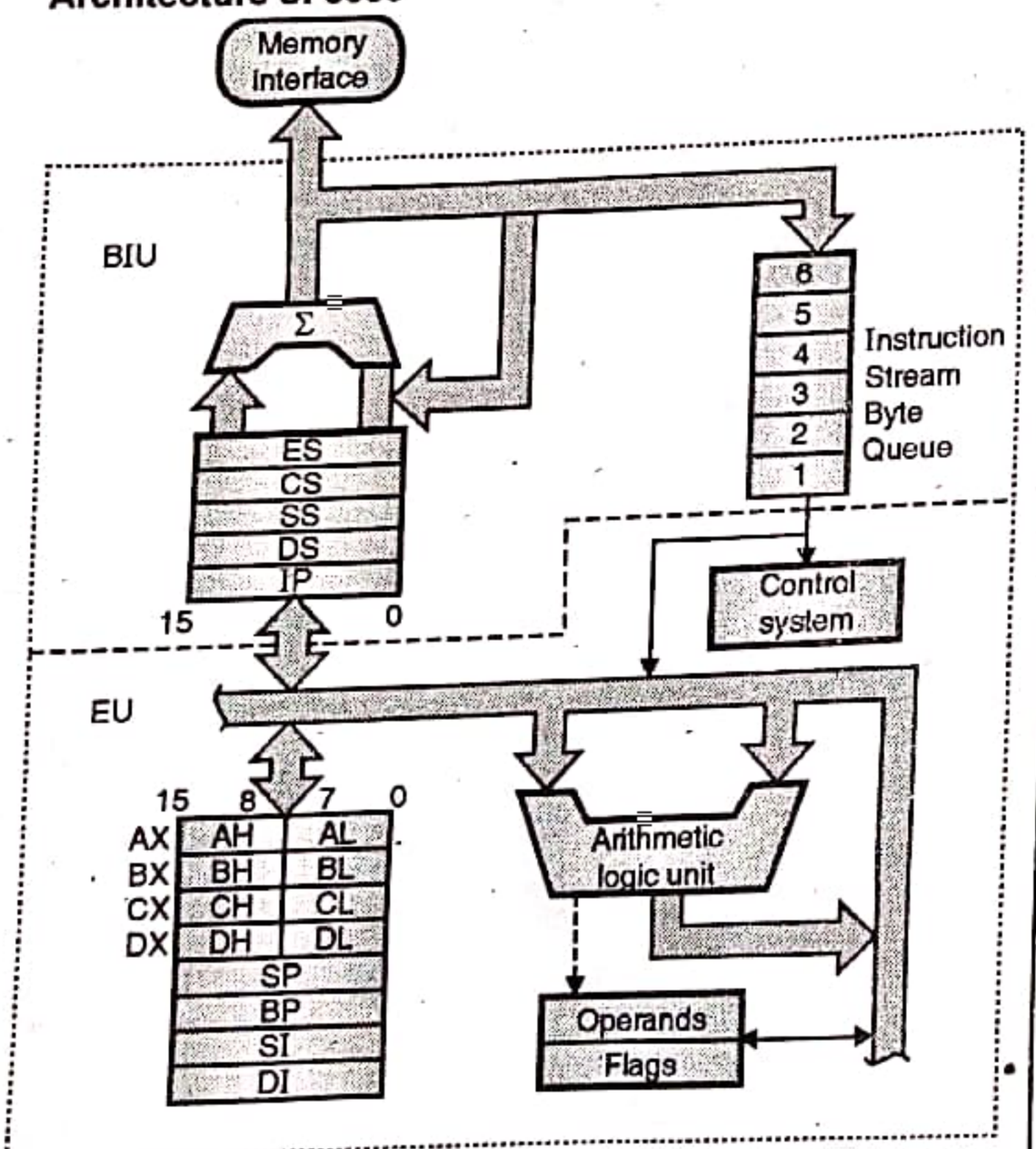


Fig. 2.1 : 8086 internal architecture

The Execution Unit (EU)

Main function of EU is **decoding** and **execution** of the instructions. In order to carry out its tasks it has the following units :

1. Arithmetic Logic Unit (ALU)

The ALU in the EU is a 16 bit unit i.e. it can perform 16-bit operation simultaneously.

It is capable of performing a variety of arithmetic and logic operations such as add, subtract, AND, OR, NOT, EX-OR, increment, decrement, shift etc.

2. Flag Register

Flag register is a part of EU. It is a 16-bit register with each bit corresponding to a flipflop. A flag is a flipflop. It indicates some condition produced by the execution of an instruction. For example the Zero Flag (ZF) will set if the result of execution of an instruction is zero.

A flag can control certain operations of EU. Fig. 2.2 shows the details of the 16 bit flag register of 8086 CPU. As shown, it consists of nine active flags out of sixteen. The remaining seven flags marked "U" are undefined flags.

3. General Purpose Registers

As shown in Fig. 2.2, the execution unit (EU) has four general purpose 16-bit registers. Each one of them can be used for temporary storage of 8 bit data, 16-bit data or 32-bit data. 8-bit Registers - AH, AL, BH, BL, CH, CL, DH, DL. Any of these registers can be used as an 8-bit operand. 16-bit Registers - AX, BX, CX, DX, SI, DI, SP, BP. Any of these registers can be used as a 16-bit operand. 32-bit Registers - DX : AX together can be used for 32-bit operand. These registers can be used for general purpose computing when their other specialized functions do not interfere.

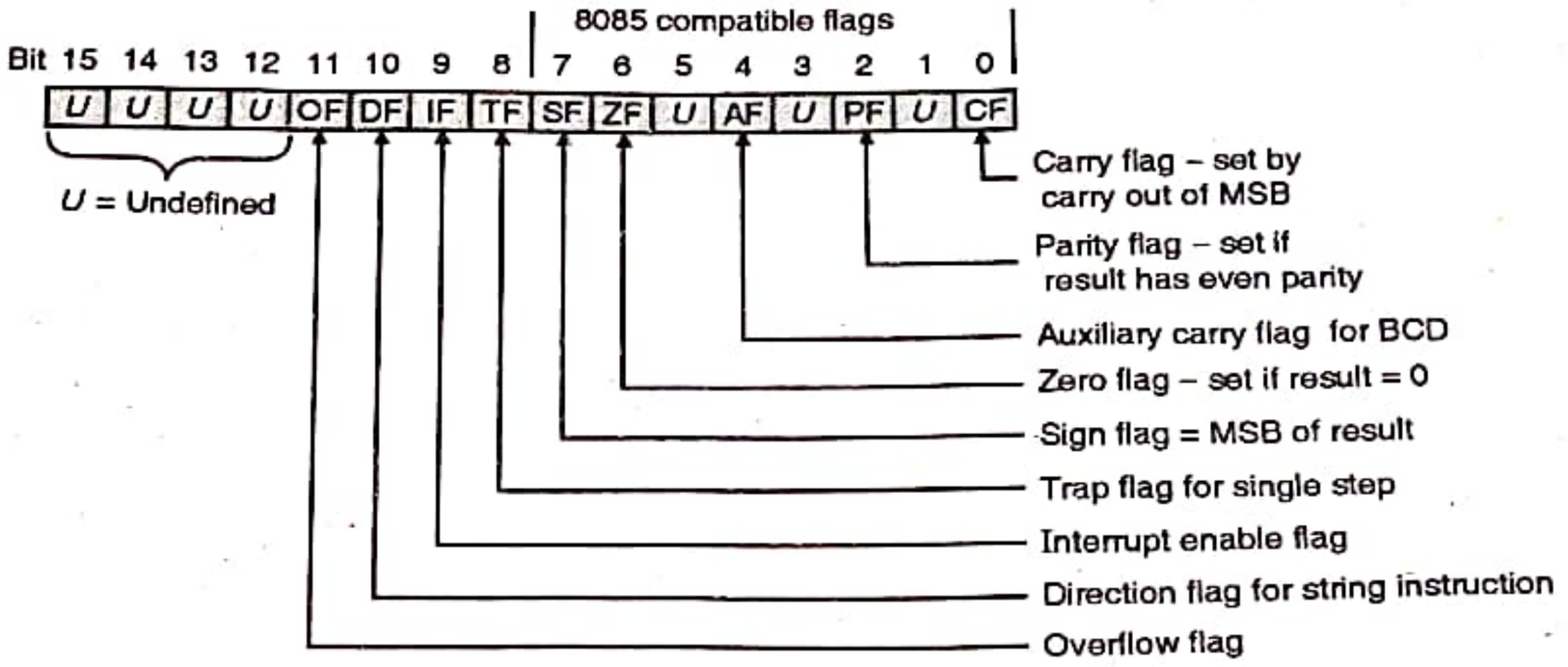


Fig. 2.2 : 8086 flag register format

4. Control Circuitry

The control circuit is a part of EU. It is used for directing the internal operations.

5. A Decoder

"The process of translation from instructions into action is known as decoding" A decoder in the Execution Unit (EU) is used for translating the instructions fetched from the memory into a series of actions. The EU will actually carry out these actions.

6. Pointer and Index Register

The execution unit also contains the following 16 bit registers. These registers can be used as general purpose 16-bit registers. But mainly they are used to hold the 16 bit offset of data word in one of the segments as given below :

(1) **Base Pointer (BP) register :** This is a 16 bit register in the E.U. It holds the 16 bit offset relative to the Stack Segment (SS) register. But BP has a specific use. BP is used whenever we pass a parameter by way of stack. The BP register can also be used as an offset register in the addressing mode called **base addressing mode**.

(2) **Stack Pointer (SP) register :** This is also a 16-bit register in the E.U. It holds the 16-bit offset address relative to Stack Segment (SS) register. SP is used for sequential access of stack segment. It always points to the top of stack. It is mainly used during instructions like PUSH, POP, CALL, RETURN etc.

(3) **Source Index (SI) register :** This register is used for holding the offset of a data word in the Data Segment (DS). The physical address of a location in the data segment can be generated by adding a hardwired zero to the segment base (16 bit) held by the Data Segment (DS) register and then by adding the offset in the SI register to it.

(4) **Destination Index (DI) register :** This register is used for holding the 16 bit offset of a data word in the Extra Segment (ES). The Source Index (SI) register and destination index (DI) register are used for the string related instructions.

For example if we want to move a block of data from memory to memory, then Source Index (SI) register can be used to Point to the source memory address and Destination Index (DI) register is used to point to the destination memory address.

The Bus Interfacing Unit (BIU)

The bus interface unit performs all the activities related to Bus. Specifically BIU has the following **five functions :**

1. Instruction **fetching** (reading) from primary memory. (performed over the system bus)
2. R/W of data operand from/to primary memory. (performed over the system bus)
3. I/O of data from/to peripheral ports. (performed over the system bus)
4. Address generation for memory reference
5. Instruction queuing in an instruction queue.

Q. 3 Explain the memory segmentation of 8086.

Dec. 17

Ans. : Memory segmentation of 8086

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.

The BIU contains four special purpose registers called as segment registers. They are :

1. The Code Segment (CS) register
2. The Stack Segment (SS) register
3. The Extra Segment (ES) register and
4. The Data Segment (DS) register

The purpose of using these segment registers are explained as follows :

1. All these are 16 bit registers.
2. The number of address lines in 8086 is 20. So the 8086 BIU will send out a 20 bit address in order to access one of the 1,048,576 or 1 Mb memory locations.
3. But it is interesting to note that the 8086 does not work the whole 1,048,576 byte (1M.byte) memory at any given time. However it works with only four 65,536 (64k-byte) segments within the whole 1 M-byte memory.

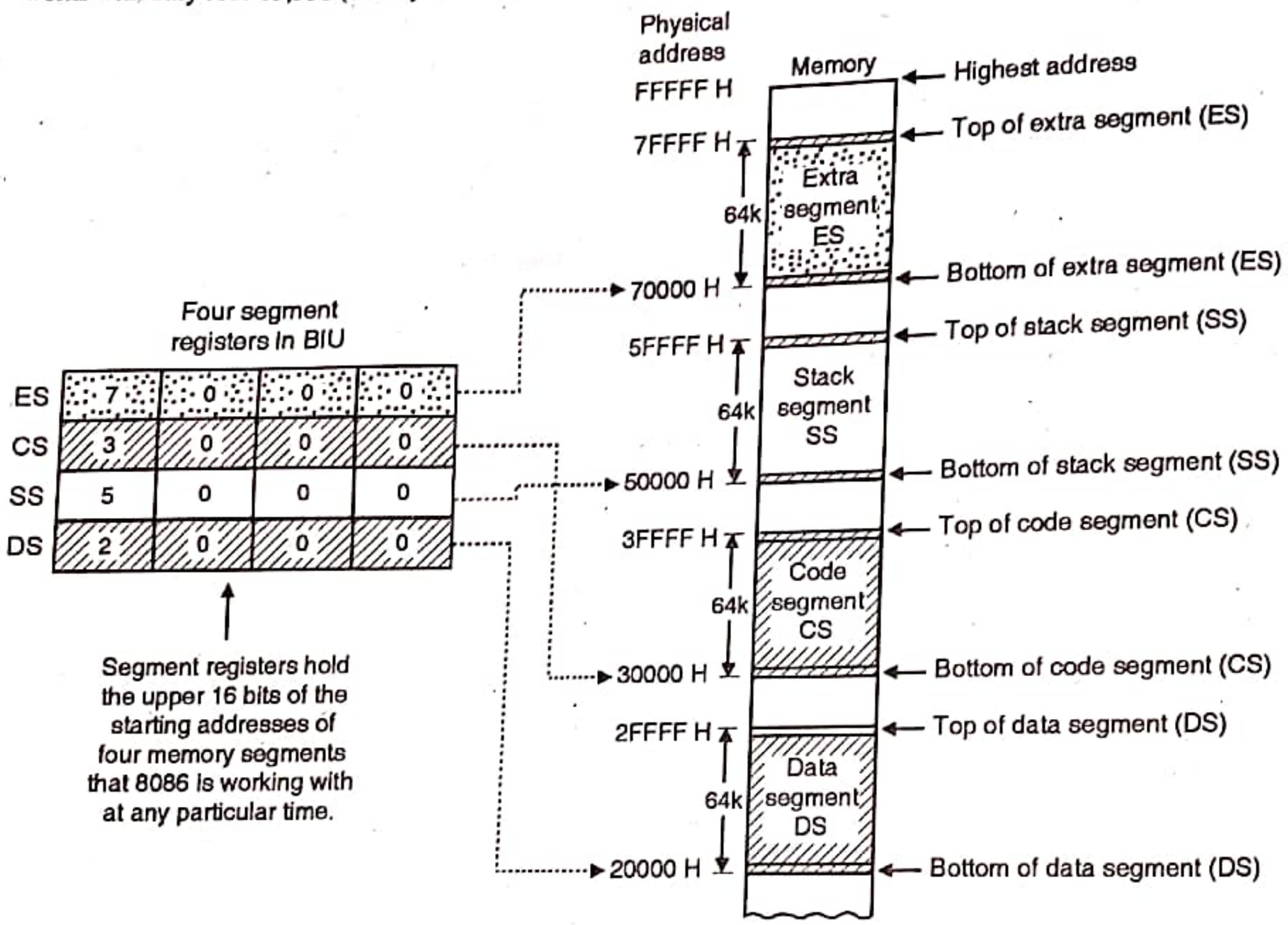


Fig. 2.3 : One way of positioning four 64 k byte segments within the 1 M byte memory space of an 8086

The four segment registers actually hold (contain) the upper 16 bits of the starting addresses of the four memory segments of 64 k byte each with which the 8086 is working at that instant of time.

These starting addresses will always be changing. They are not fix. Fig. 2.3 shows one of the possible ways to position the four 64 k byte segments within the 1-M byte memory space of 8086. There is no restriction on the locations of these segments in the memory. These segments can be separate from each other as shown in Fig. 2.3 or they can overlap. The starting address or base address of the data segment is 20000H. The upper 16-bits of this i.e. 2000 are loaded into the Data Segment register (DS). Similarly upper 16 bits of the starting addresses of the other segments are stored into the corresponding segment registers.

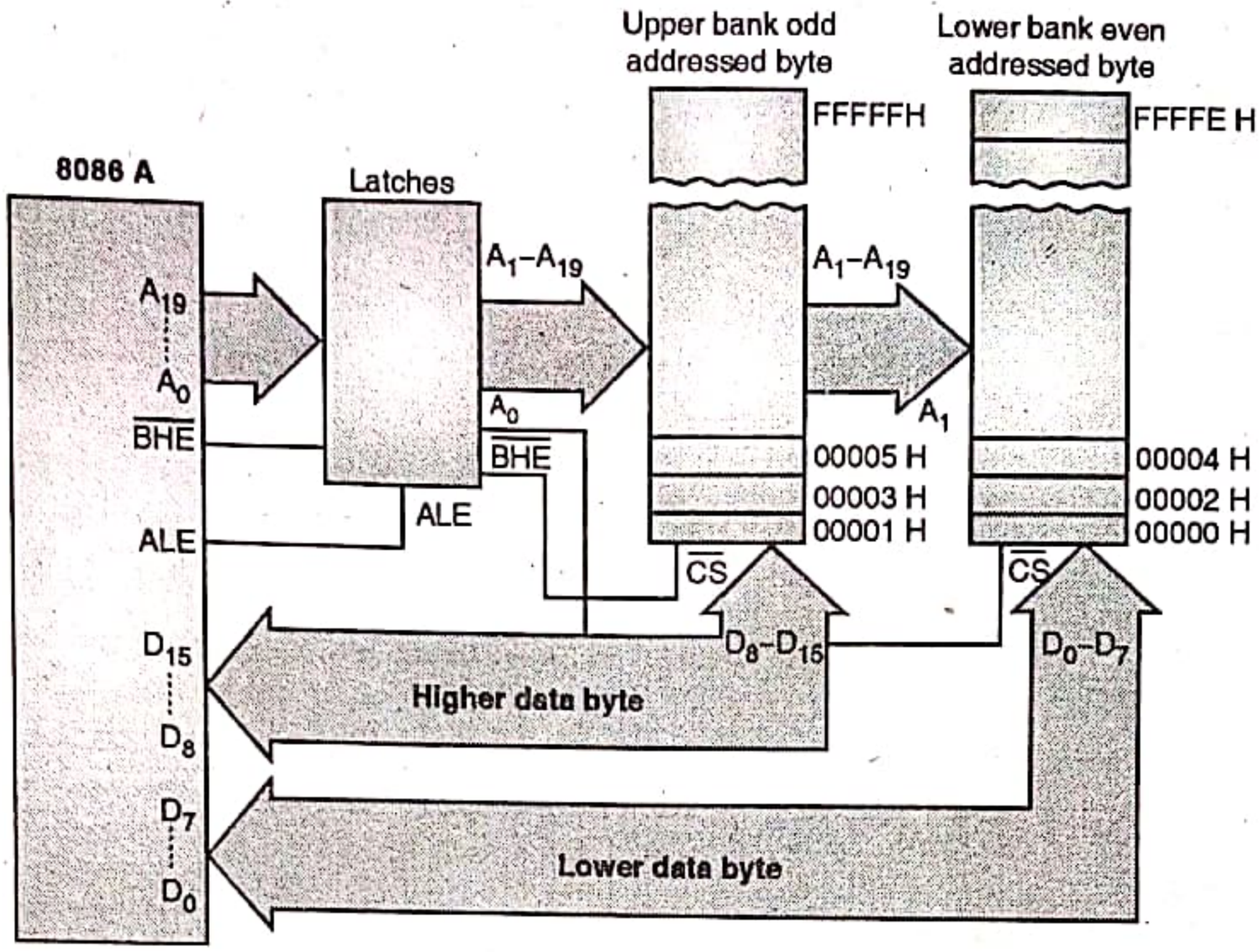
The segment overlapping usually takes place for small programs which do not need all the 64 k bytes in each segment. The segments can adjacent, disjoint, partially overlapping or fully overlapping. In the users program there can be many segments. But 8086 can deal with only four of them at any given time, because it has only four segment registers. Whenever the segment orientation is to be changed, to change the base addresses and load the upper 16 bits into the corresponding segment registers.

Q. 4 Write short note on memory banking in 8086. May 12

Ans. : Memory banking in 8086

8086 has a 16-bit data bus hence it should be able to access 16-bit data in one operation. But the memory chips available are normally such that each location has 8-bits i.e. a byte. Hence, to read 16-bits it needs to access 2 memory locations.

If both these memory locations are in same memory chip then the address bus will have to provide two addresses sequentially and will require double the time also 16-bits would not be accessed simultaneously. To solve this problem, the memory of 8086 is divided into two banks each bank provides 8-bits. One bank contains all even addresses called the "Even bank", while the other is called "Odd bank" containing all odd addresses. Fig. 2.4 shows the different accesses possible for 8086.



(a)

Memory location	Address	Data Type	$\overline{BHE}$	A_0	Bus Cycles	Data Lines Used
Even	00000	Byte	1	0	One	$D_0 - D_7$
Even	00000	Word	0	0	One	$D_0 - D_{15}$
Odd	00001	Byte	0	1	One	$D_8 - D_{15}$
Odd	00001	Word	0	1	First	$D_0 - D_7$
			1	0	Second	$D_8 - D_{15}$

(b)

Fig. 2.4 : Memory banking

Case I : Byte access from an even location

Since the address in even A_0 is '0'. $\overline{BHE}$ signal is kept disabled as only even bank access is required. The access is completed in one cycle. The data is accessed on data lines D_0 to D_7. For e.g. when the byte required is from an even address like (00000)H, it implies one byte from 00000H i.e. first location of even bank as shown in the Fig. 2.4(a).

Case II : Byte access from an odd location

Since the address in odd A_0 is '1'. $\overline{BHE}$ signal is kept enabled as odd bank access is required. The access is completed in one cycle. The data is accessed on data lines D_8 to D_{15}. For e.g. when the byte required is from an odd address like (00001)H, it implies one byte from 00001H i.e. first location of odd bank as shown in the Fig. 2.4(a).

Case III : Word access from an even location

Since the address in even A_0 is '0'. $\overline{BHE}$ signal is kept enabled as odd bank access is also required. The access is completed in one cycle. The data is accessed on data lines D_0 to D_7 (from even bank) and D_8 to D_{15} (from odd bank).

For e.g. when the word required is from an even address like (00000)H, it implies.

i) One byte from 00000H i.e. first location of even bank as shown in the Fig. 2.4(a).

ii) One byte from 00001H i.e. first location of odd bank as shown in the Fig. 2.4(a).

Hence, this is an aligned access and can be done in one cycle.

Case IV : Word access from an odd location

Since the address in odd A_0 is '1'.

$\overline{BHE}$ signal is kept enabled as odd bank access is also required. The access is completed in two cycles. The data is accessed on data lines D_8 to D_{15} in the first access from odd bank. The data is accessed on data lines D_0 to D_7 in the second access from the even bank. This access is not possible in the same cycle as it is a unaligned access. Unaligned access is one wherein the data to be accessed are in different rows of the two memory chips. For e.g. when the word required is from an odd address like (00001)H, it implies.

(i) One byte from 00001H i.e. first location of odd bank as shown in the Fig. 2.4(a).

(ii) And another byte from 00002H i.e. second location of even bank as shown in Fig. 2.4(a).

Since the locations in the two banks are not same, two accesses are required.

Chapter 3 : Operating Modes

Q. 1 Explain power on reset circuit used in 8086 system. Dec. 11, Dec. 15, Dec. 17

Ans. : Power on reset circuit

When power supply is switched on, there are some spikes in its output for some time. These spikes cause change in the default value of the registers. The default value of all the registers of 8086 is 0000H except for CS is FFFFH. Hence the address of the first instruction fetched by the processor is FFFF0H.

The change in these register values alter the behavior of the processor, especially if CS or IP change. This makes it necessary to implement power on reset circuit, which should reset the processor during these spikes and maintain register to their default values.

Operation of power on circuit

Fig. 3.1 shows power-on-reset circuit.

Initially the capacitor has no charge across it, when system is switched on. Hence it provides logic '0' on $\overline{RES}$ pin and reset is activated. Slowly the capacitor is charged through the resistor and makes the reset pin to logic '1'. The charging time of capacitor is such that by this time the spikes of power supply are vanished.

This disables the reset, and the processor switches on.

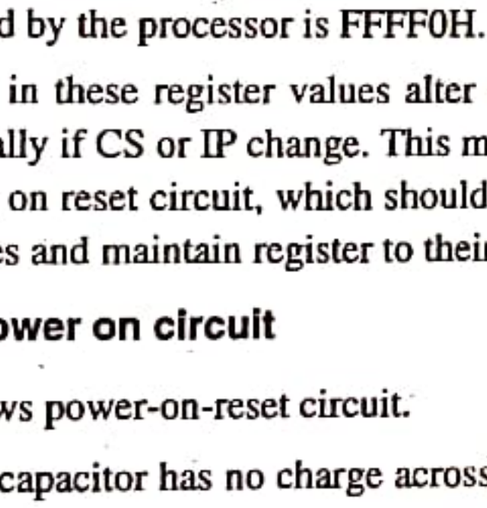


Fig. 3.1

The diode is connected across the resistor to discharge the capacitor rapidly when power supply is switched off.

The switch is for manual reset to discharge the capacitor, by connecting it to ground.

Q. 2 Write short note on 8288 bus controller May 12, Dec. 12, May 13, Dec. 15, Dec. 16

Ans. :

8288 bus controller

The 8288 bus controller is a 20 pin bipolar IC. It is used in the maximum mode configuration of 8086. The 8288 is used in the medium to large 8086 processing systems.

The 8288 bus controller accepts the CLK signal along with $\overline{S_0}$, $\overline{S_1}$ and $\overline{S_2}$ outputs of 8086 and generates the command, control and timing signals at its output.

It also provides the bipolar bus drive capability and optimizes the system performance.

Fig. 3.2(a) shows the simplified internal block diagram of 8288 bus controller and Fig. 3.2(b) shows the pin diagram of 8288.

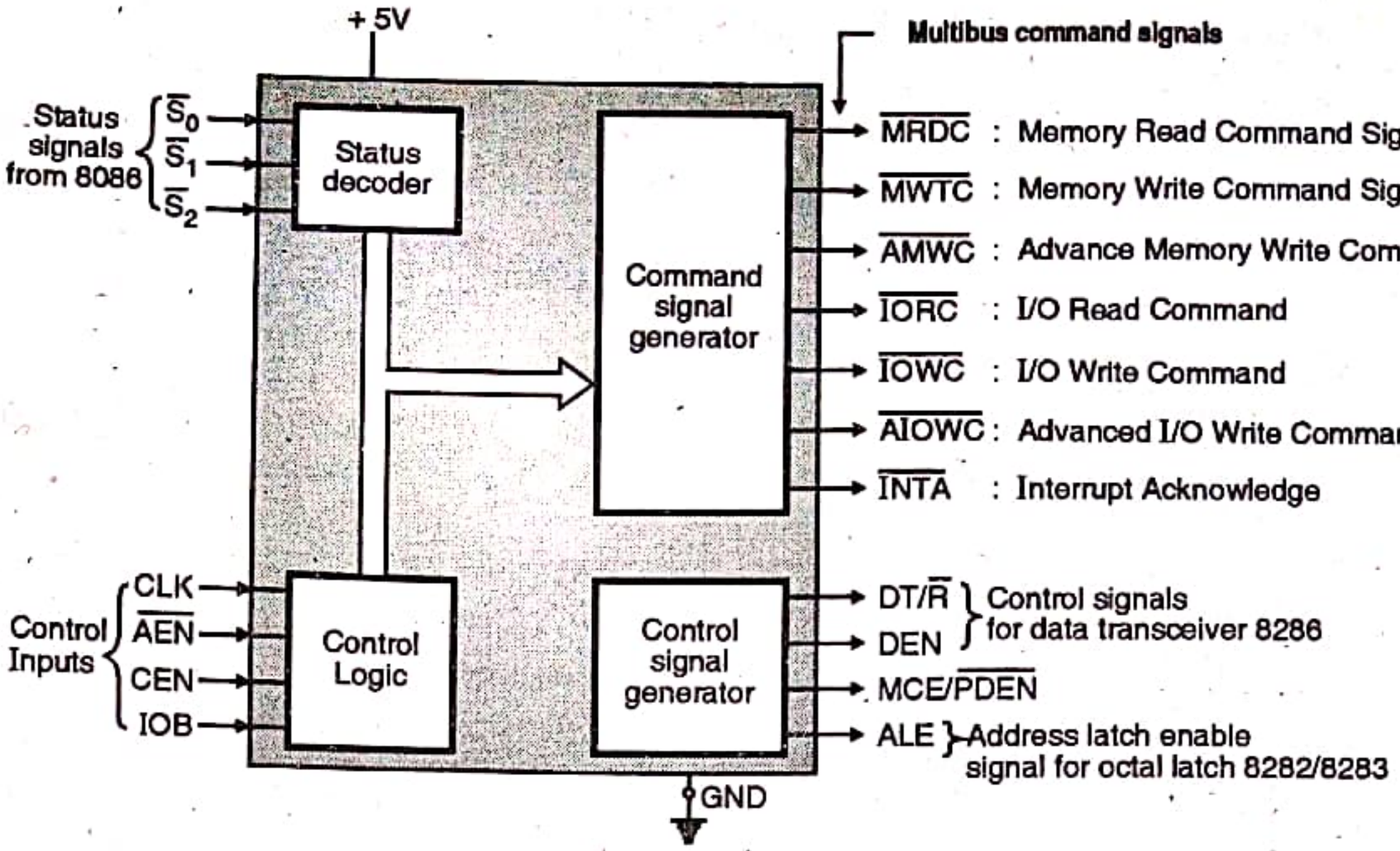


Fig. 3.2(a) : Simplified internal block diagram of 8288 bus controller

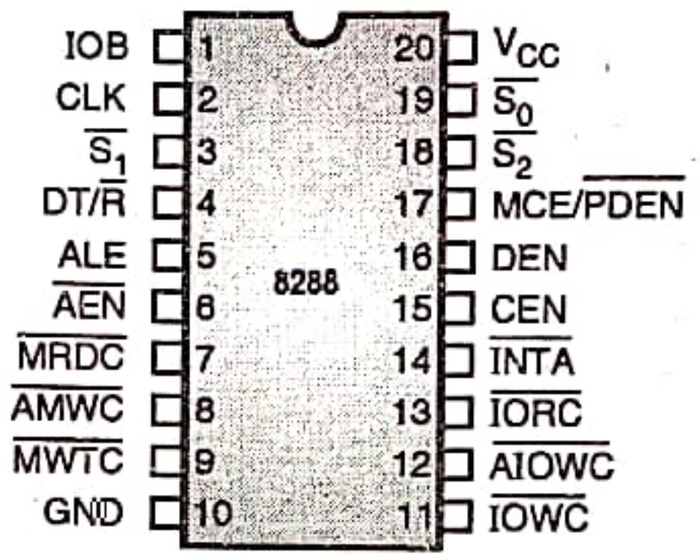


Fig. 3.2(b) : Pin diagram of 8288 bus controller

Pin Definitions

Name	I/O	Function
V_{CC}		+5V supply.
GND		Ground
$\overline{S}_0$, $\overline{S}_1$, $\overline{S}_2$	I	Status Input Pins : These pins are the status input pins from the 8086/8088 processors. The 8288 decodes these inputs to generate command and control signals at the appropriate time. When these pins are not in use (passive) they are all HIGH.
CLK	I	Clock : This is a clock signal from the 8284 clock generator and serves to establish synchronization when command and control signals are generated.
ALE	O	Address Latch Enable : This signal serves to strobe an address into the address latches. This signal is active HIGH and latching occurs on the falling (HIGH to LOW) transition. ALE is intended for use with transparent D type latches.
DEN	O	Data Enable : This signal serves to enable data transceivers onto either the local or system data bus. This signal is active HIGH.
DT/$\overline{R}$	O	Data Transmit/Receive : This signal establishes the direction of data flow through the transceivers. A HIGH on this line indicates Transmit (write to I/O or memory) and a LOW indicates Receive (read).

Name	I/O	Function
$\overline{AEN}$	I	Address Enable : $\overline{AEN}$ enables command outputs of the 8288. $\overline{AEN}$ does not affect the I/O command lines if the 8288 is in the I/O Bus mode (IOB tied HIGH).
CEN	I	Command Enable : When this signal is LOW all 8288 command outputs and the DEN & $\overline{PDEN}$ control outputs are forced to their inactive T-state. When this signal is HIGH, these outputs are enabled.
IOB	I	I/O Bus Mode : When the IOB is strapped HIGH the 8288 functions in the I/O Bus mode. When it is strapped LOW, the 8288 functions in the System Bus mode.
$\overline{IOWC}$	O	I/O Write Command : This command line instructs an I/O device to read from the data bus. This signal is active LOW.
$\overline{IORC}$	O	I/O Read Command : This command line instructs an I/O device to drive its data onto the data bus. This signal is active LOW.
$\overline{AIOWC}$	O	Advanced I/O Write Command : The $\overline{AIOWC}$ issues an I/O Write command earlier in the machine cycle to give I/O devices an early indication of a write instruction. Its timing is the same as a read command signal. $\overline{AIOWC}$ is active LOW.
$\overline{MWTC}$	O	Memory Write Command : This command line instructs the memory to record the data present on the data bus. This signal is active LOW.
$\overline{MRDC}$	O	Memory Read Command : This command line instructs the memory to drive its data onto the data bus. This signal is active LOW.
$\overline{AMWC}$	O	Advanced Memory Write Command : The $\overline{AMWC}$ issues a memory write command earlier in the machine cycle to give memory devices an early indication of a write instruction. Its timing is the same as a read command signal. $\overline{AMWC}$ is active LOW.
$\overline{INTA}$	O	Interrupt Acknowledge : This command line tells an interrupting device that its interrupt has been acknowledged and that it should drive vectoring information onto the data bus. This signal is active LOW.
MCE /$\overline{PDEN}$	O	This is a dual function pin. MCE (when IOB is tied LOW) : Master Cascade Enable occurs during an interrupt sequence and serves to read a Cascade Address from a master PIC (Priority Interrupt Controller) onto the data bus. The MCE signal is active HIGH. $\overline{PDEN}$ (when IOB is tied HIGH) : Peripheral Data Enable enables the data bus transceiver for the I/O bus during I/O instructions. It performs the same function for the I/O bus that DEN performs for the system bus. $\overline{PDEN}$ is active LOW.

Q. 3 Draw and explain timing diagram for read operation in minimum mode of 8086.

May 12, Dec. 14, May 16, Dec. 17

Ans. :

Fig. 3.3 shows the timing diagram which shows the activities of various signals during the read operation.

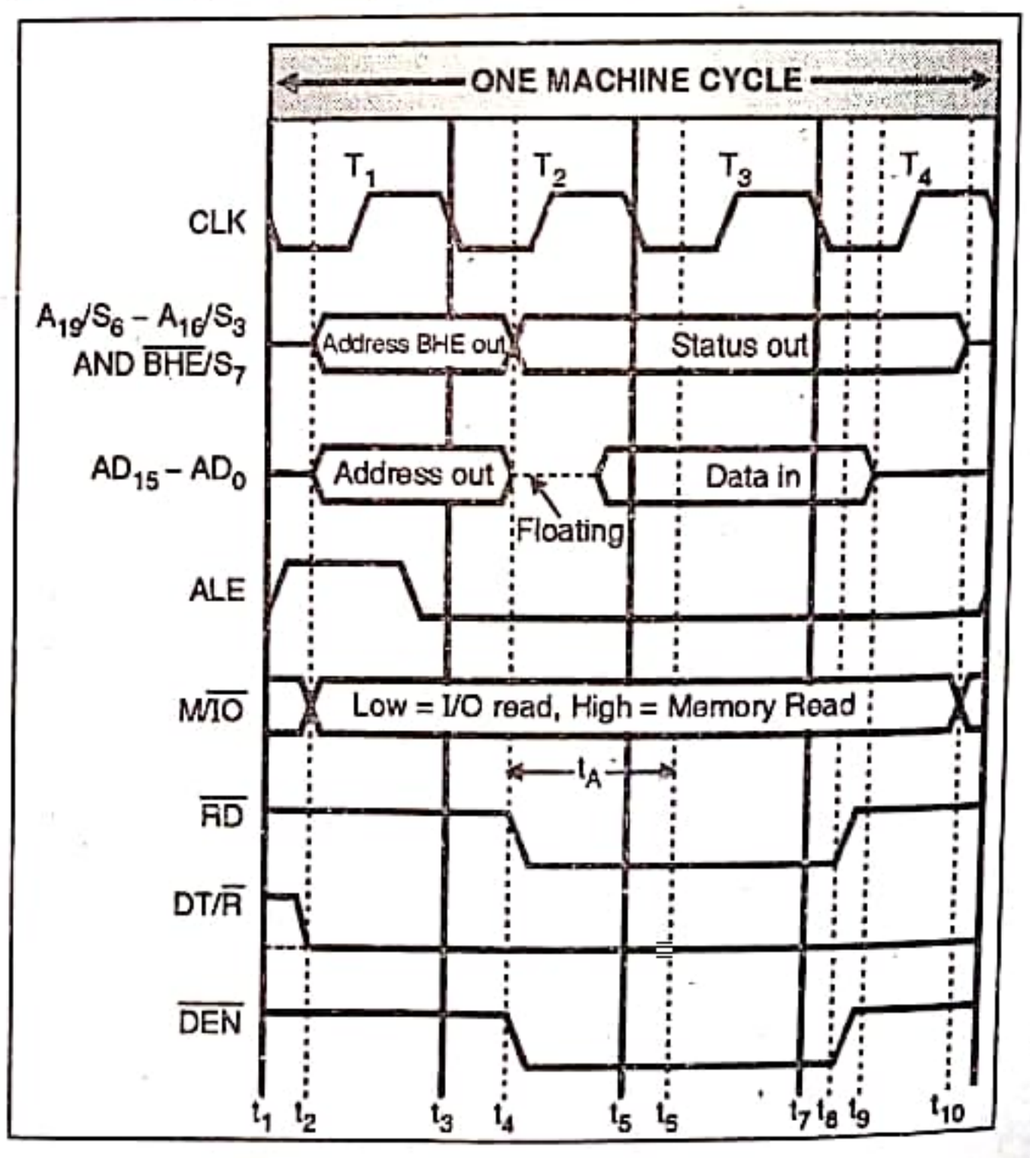


Fig. 3.3 : Timing diagrams for the read machine cycle of 8086

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.

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.

Q. 4 Draw and explain timing diagram for write operation in minimum mode of 8086. May 15

Ans. :

Timing diagram in minimum mode of 8086

The timing diagram for the write machine cycle is shown in Fig. 3.4.

1. During the T_1 state, the 8086 asserts $M/\overline{IO}$ low if it has to write to an I/O port and high if the memory location is to be written.
2. At about the same time, it will push the ALE output high. This will enable the address latches.
3. 8086 then outputs the $\overline{BHE}$ and the address of the desired port or memory location on the lines AD_0 to AD_{15} and A_{16} to A_{19}. Note that the address lines A_{16} to A_{19} will always be low if a port is to be written because the port address is always going to be a 16 bit address.
4. After some time the ALE output is pulled down. The address gets latched into the external latches.
5. The 8086 removes the address information from $AD_0 - AD_{15}$ and outputs the desired data on the data bus.
6. It then asserts the $\overline{WR}$ signal low. The low $\overline{WR}$ signal will turn on the memory or port where the data is to be written.
7. 8086 then gives some time for the addressed I/O port or memory to accept the data from the data bus and then raises the $\overline{WR}$ output high and floats the data bus.
8. If the addressed memory or port devices cannot accept the data within the normal machine cycle, then the external hardware set up to produce a low READY input.
9. If the READY input is pulsed low before or during the T-state T_2 of the write machine cycle, then the 8086 will introduce WAIT T-states after T_3 as long as the READY input is held low.
10. During the WAIT T-states, the signals on the data bus, address bus and control bus do not change.
11. If the READY input is made high before the end of a WAIT T-state, then 8086 will continue with the T_4 state as soon as it finishes the WAIT T-state.
12. If the system is large enough then it will need the external data buffers (IC 8286). Then the $DT/\overline{R}$ is used to decide the direction of data transfer.

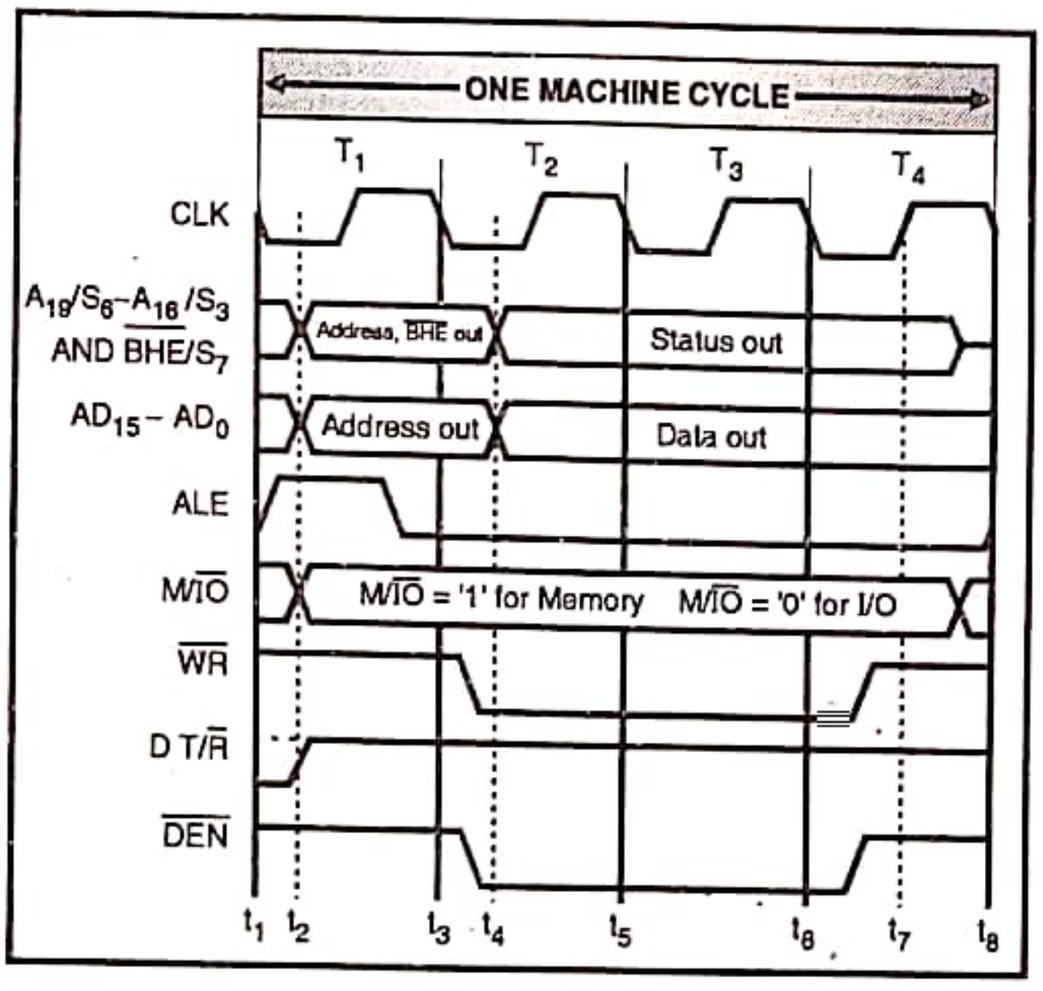


Fig. 3.4 : Write machine cycle in the 8086 minimum mode with wait T-state

13. During the write cycle, the 8086 pushes the DT/$\overline{R}$ high to configure the external data buffers in the "transmit" mode.

    Then the $\overline{DEN}$ signal is asserted low so as to enable the buffers.

14. The data output from 8086 is then passed through the data buffers to the addressed memory or I/O port.

Q. 5 Explain maximum mode of 8086 microprocessor. Dec. 17

Ans. :

Maximum mode of 8086 microprocessor :

Fig. 3.5 shows the block diagram of the 8086 system in maximum mode. The use of the bus controller IC 8288 along with the other support chips.

Additional circuitry is required to translate the control signals. The additional circuitry converts the status signals ($S_2 - S_0$) to 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 signals $\overline{MRDC}$, $\overline{MWTC}$, $\overline{AMWC}$, $\overline{AIOWC}$, $\overline{IORC}$, $\overline{IOWC}$ 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.

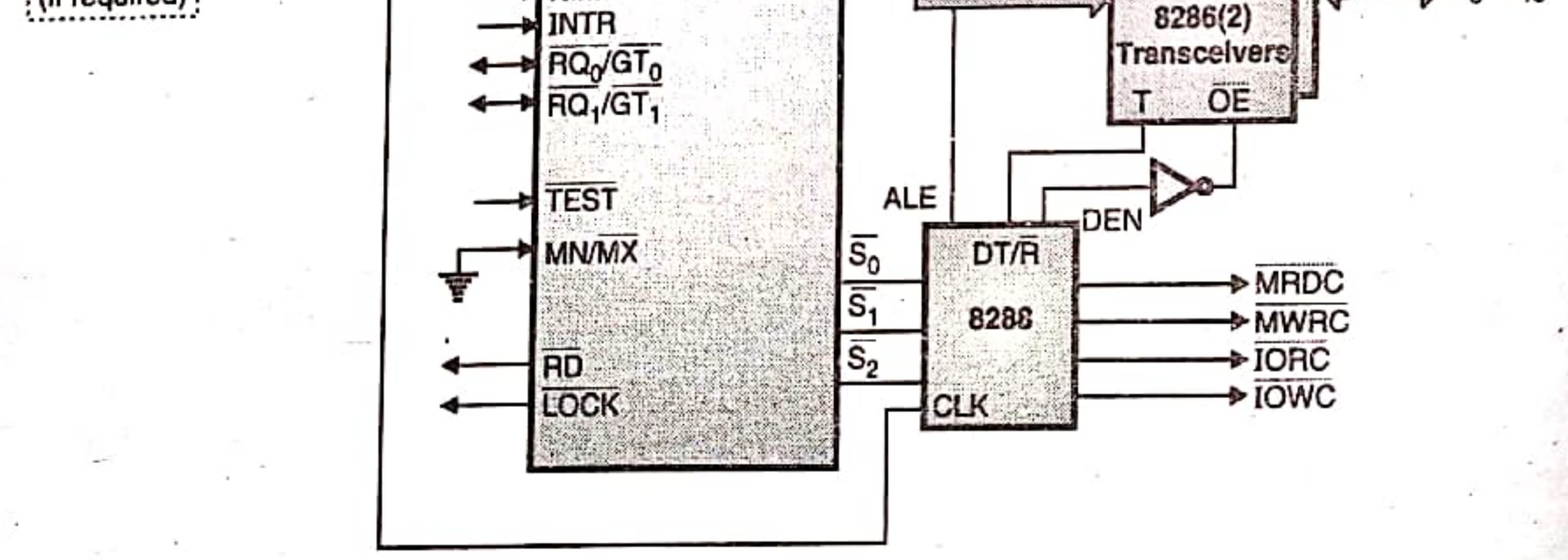


Fig. 3.5 : Block diagram of maximum mode minimum system

Q. 6 Draw timing diagram of read operation on 8086 based system. May 16

Ans. :

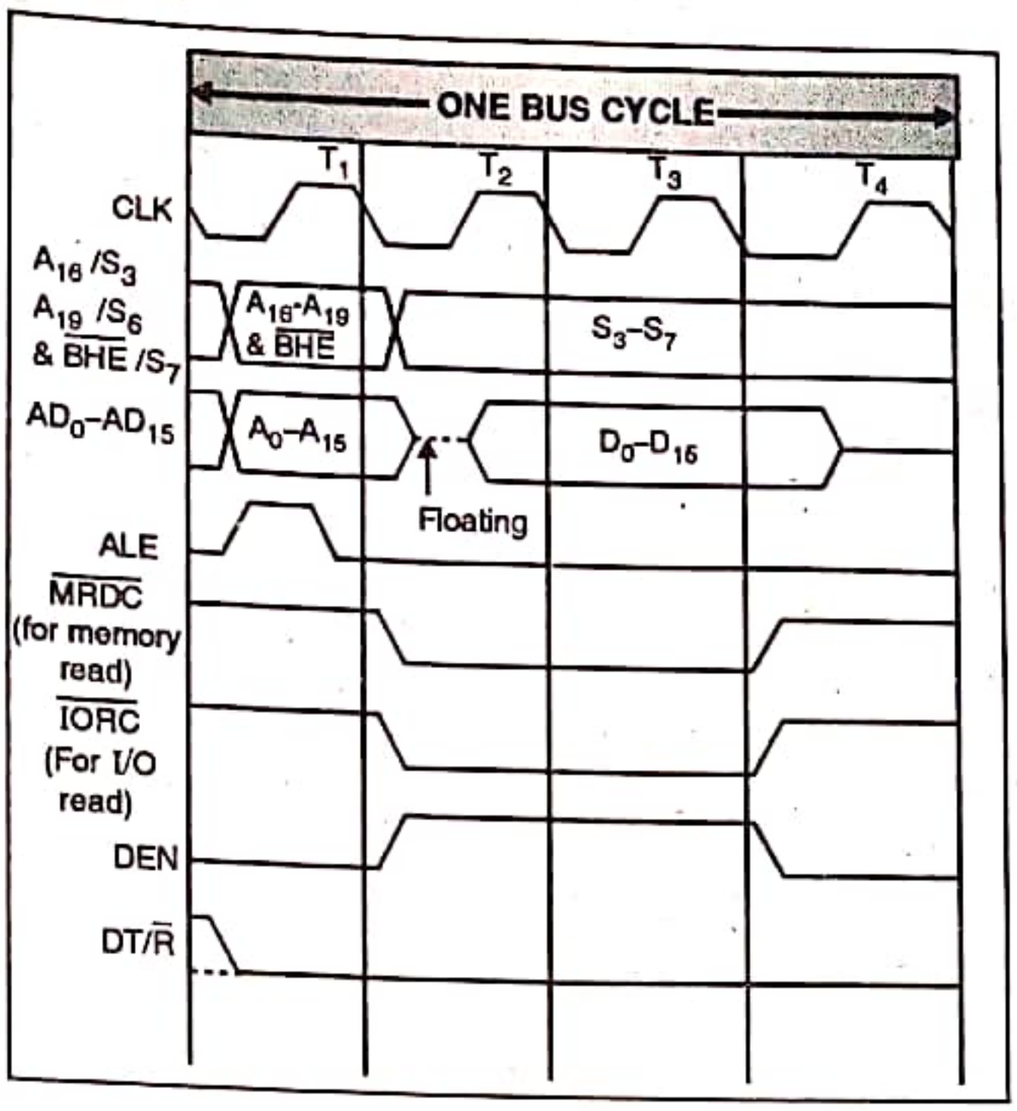


Fig. 3.6 : Maximum mode read bus cycle

Chapter 4 : 8086/8088 Addressing Modes and Instruction Set

Q. 1 Explain programming model of 8086. May 16

Ans. : Programmer's model of 8086

The 8086 programmer's model is a picture of the processor as available to the programmer. These are the registers used to hold numbers and addresses, as well as indicate status and act as controls.

Data Registers (16 bits each for AX, BX, CX and DX)

8086 Programmer's Model

BIU registers		ES		Extra Segment
		CS		Code Segment
		SS		Stack Segment
		DS		Data Segment
		IP		Instruction Pointer
EU registers	AX	AH	AL	Accumulator
	BX	BH	BL	Base Register
	CX	CH	CL	Count Register
	DX	DH	DL	Data Register
		SP		Stack Pointer
		BP		Base Pointer
		SI		Source Index Register
		DI		Destination Index Register
		FLAGS		

Fig. 4.1 : Programmer's model of 8086

For 16-bit data registers can be accessed by names AX, BX, CX and DX, Individual bytes of these registers can be accessed by name, AH, AL, BH, BL, CH, CL, DH and DL.

Address Registers

The segment register are CS, DS, ES and SS and offset is stored in instruction pointer IP or stack pointer SP or base registers BX or BP, or one of the index registers SI or DI.

The CS:IP pair gives the address of the next instruction to be executed in the program sequence. The SS:SP pair gives the address of the top of the stack, a temporary storage area often automatically used by the computer. SP is used for sequential access of the stack. The SS:BP pair is used as a pointer into the stack. BP is used for random access of the stack. The DS:SI pair is used as a source pointer and ES : DI pair is used as destination pointer for string instructions. For all other instructions DI register is used with DS segment register.

The DS:SI and ES:DI registers are used as general purpose pointers for copying and data processing. The dotted lines below show common default couplings. These defaults can be overridden by a notation such as DS:[BP], where DS will be used in place of the default SS. The BX data register not shown is also used as a pointer in the data segment (by default).

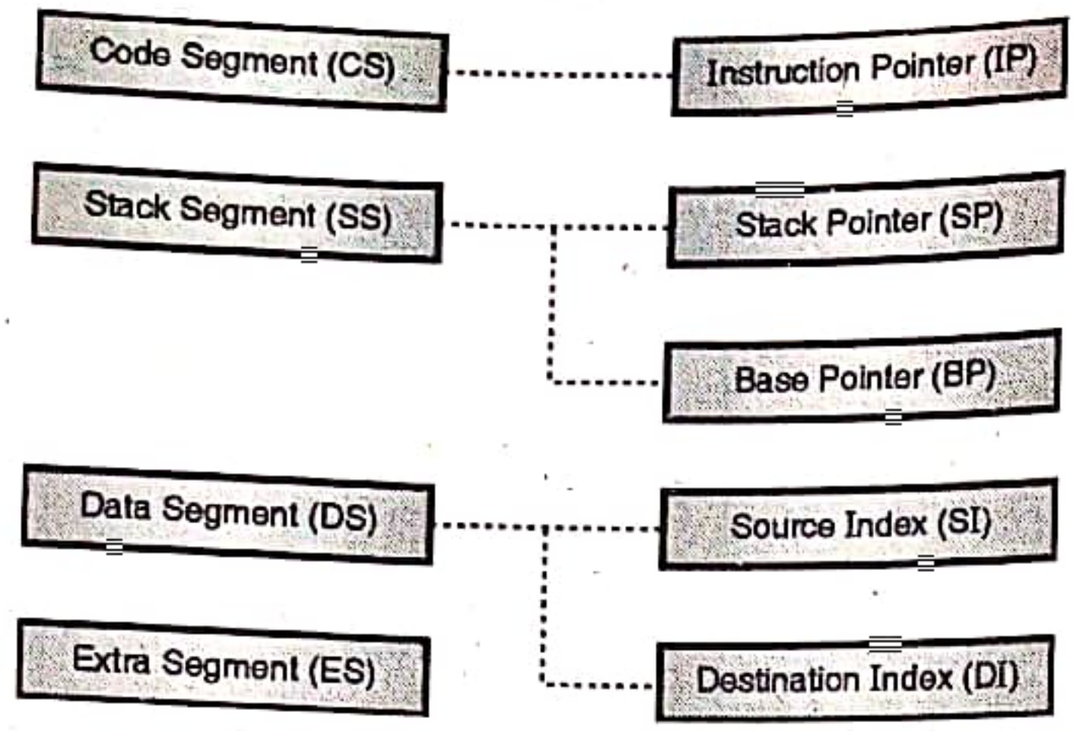


Fig. 4.2 : Address registers of 8086

Flag register

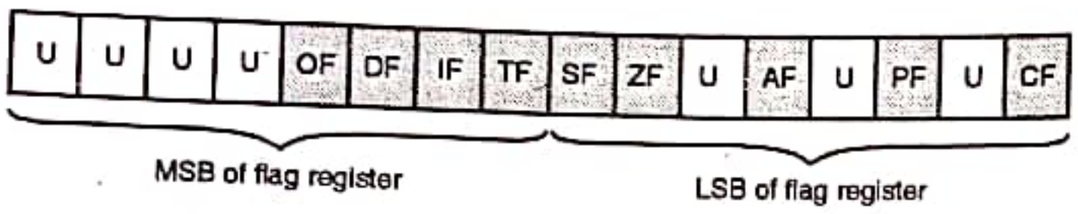


Fig. 4.3 : Flag register of 8086

CF = Carry Out

U = It can be 1 or 0. It is undefined

PF = Parity of last operation

AF = Auxiliary carry (used in BCD arithmetic)

ZF = Zero result

SF = Sign bit from last Operation

TF = Trap Flag – do single instruction

IF = Interrupt Enable Flag

DF = Direction flag

OF = Overflow (error for signed numbers)

Q. 2 Write short note on addressing modes in the 8086. May 11, Dec. 12, Dec. 13, Dec. 16 Dec. 17

Ans. : Addressing modes in the 8086

When 8086 executes an instruction, it performs specific function on data. The data is normally referred as **operands**. Operands may be contained in registers, within the instruction itself, in memory or in I/O ports.

To access these different types of operands, 8086 has to **address** memory or I/O. The address of memory and I/Os can be calculated in several different ways, normally referred as **Addressing modes**. These addressing modes greatly extend the flexibility and convenience of the instruction set.

Addressing modes (methods) refer to the different methods of addressing (selecting) the operands.

The addressing modes of 8086 are as follows :

1. Register Addressing Mode

In this mode of addressing, data is in the register, and instruction specifies the particular register as shown in Fig. 4.4. This addressing mode is normally preferred because the instructions are compact and fastest executing of all instruction forms.

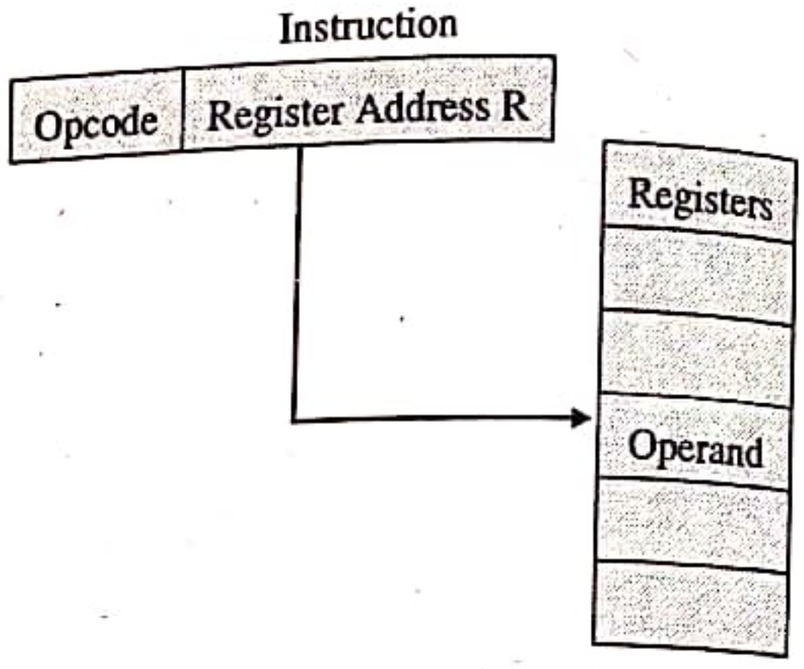


Fig. 4.4 : Register addressing

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.

2. Immediate Operand Addressing Mode

Immediate operand is nothing but **constant** data contained in an **instruction**. Thus if a source operand is part of instruction instead of register or memory, it is referred as immediate addressing mode. Fig. 4.5 shows format of instruction encoded with immediate operand.

Fig. 4.5 : Instruction encoded with an immediate operand

The immediate data may be 8 bits (byte) or 16 bits (word) in length. Immediate operand can be accessed quickly because they are available directly in instruction queue, like a register operand. Hence, there is no need of external bus and bus cycles to obtain the data.

Example :

e.g. MOV CL, 02 H MOV CL, 2

This instruction copies the immediate number 2H in the CL register.

3. Memory Addressing Modes

In this addressing mode, memory operands **must be** transferred to or from the CPU **over the BUS**. Whenever EU needs to read or write a memory operand, it must pass an offset value to the BIU. The BIU adds offset to the shifted contents of segment register, which produces 20 bit physical address and after that it executes the bus cycle(s) needed to access the operand.

The different memory addressing modes available are :

1. Direct memory addressing mode
2. Register indirect addressing mode
3. Based indexed addressing mode
4. Based indexed addressing mode
5. Relative based indexed addressing mode.

4. String Addressing Mode

String instructions do not use the normal memory addressing modes to access their operands. When a string instruction is executed, SI is assumed to point to the first byte or word of the source string and by default DS is assumed segment register. DI will point to the first byte or word of the destination string and ES is assumed to be by default segment register. In a repeated string operation, the CPU automatically adjust the SI and DI to obtain subsequent bytes or word. The automatic adjustment is done with the help of DF (Direction Flag) in flag register, and achieved by MOD and R/M bits in instruction format.

5. I/O Addressing Mode in 8086

This addressing mode is basically used for IOs.

1) **Memory mapped I/O :** If an I/O port is memory mapped, any of the memory operand addressing modes used to access the port. For example, a group of terminals can be accessed as an **array**. String instructions also used to transfer data to memory mapped I/O ports with in appropriate hardware interface.

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

   1. Direct port addressing.
   2. Indirect port addressing.

In direct port addressing, the port number is an 8 bit **immediate** operand. This allows fixed access to ports numbered 0-255. Indirect port addressing is similar to register indirect addressing of memory operands. The port number is taken forms register DX and can range from 0000H to FFFFH (0 to 65535 decimal). By previously adjusting the contents of register DX, one instruction can access any port in the IO space. A group of adjacent ports can be accessed using a simple software loop that adjusts the value in DX.

6. Implied Addressing Mode

The instructions which do not have operands come under implied addressing mode.

e.g. **XLAT**
CMA
STC
STD

These instructions do not have operands.

Q. 3 Explain the LAHF and STOSB instructions in 8086. Dec. 15

Ans. :

1. LAHF - Load AH register from flags

(Copy lower byte of flag register to AH)

Mnemonic	LAHF
Algorithm	AH = flag register's lower byte

Flags	No flags are affected.
Addr. Mode	Implied Addressing mode

Operation	AH ← Lower byte of flag register	The lower byte of 8086 flag register is copied to the AH register

2. STOS/STOSB/STOSW

(Store byte or word in string)

Mnemonic	STOSB/STOSW STOSB (For byte operation) or STOSW (for word operation)

Flags	No flags are affected

Algorithm	For STOSB ES : [DI] = AL If DF = 0 then DI = DI + 1 else DI = DI – 1 For STOSW ES : [DI] = AX If DF = 0 then DI = DI + 2 else DI = DI – 2
Addr. Mode	String addressing mode
Operation	It transfers (copies) a byte or word from register AL or AX to the string element addressed in ES by DI. Depending on DF flag, DI is automatically incremented or decremented.
Example	e.g. : MOV DI, OFFSET STR 1 ; DI points to top of string STR 1. CLD ; DF = 0 MOV AX, 00H STOSW ; Store string by AX value

May 12,

Q. 4 Explain CMP instruction with one example.

Ans. :

CMP - Compare byte or word

Mnemonic	CMP destination, source
Algorithm	Destination – source
Flags	AF, OF, SF, ZF and PF, CF are updated according to the result
Addr. Mode	Register addressing mode

Operation

This instruction compares a word/byte from source with byte/word from destination. The comparison is done by subtracting the source byte or word from the destination byte or word.

The result is not stored in either of the destination or source. The destination and source remain unchanged, only flags are updated.

The source may be register, memory location or an immediate number.

The destination may be a register or memory location.

Compare	CF	ZF	SF	
Source > destination	1	0	1	Subtraction required borrow, so CF = 1
Source < destination	0	0	0	No borrow required CF = 0
Source = destination	0	1	0	Result of subtraction is zero

Example : CMP BH, CL

Compares a byte in CL with byte in BH.

Let CL → 05 H and BH → 04 H, CF → 0, ZF → 0, SF → 0:

```
                =  0000 0100     → BH
                −  0000 0101     → CL
              ------------------
Carry flag → 0 [ ]  0000 0100    → Result of subtraction
Carry flag → 1 [ ]  1111 1011    ← 2's complement of the result i.e. result of comparison of BH and CL registers
```

Q. 5 Explain the SAR Intel 8086 assembly language instructions with example.

Dec. 11, May 12, Dec. 12, May 13

Ans. :

SAR - Shift arithmetic right byte or word

Mnemonic	SAR destination, count
Algorithm	Shift all bits right, the bit that goes off is set to CF. The sign bit that is inserted to the leftmost position has the same value as before shift.

Flags	All flags are affected
Addr. Mode	Immediate addressing mode

Operation	MSB → MSB → LSB → CF

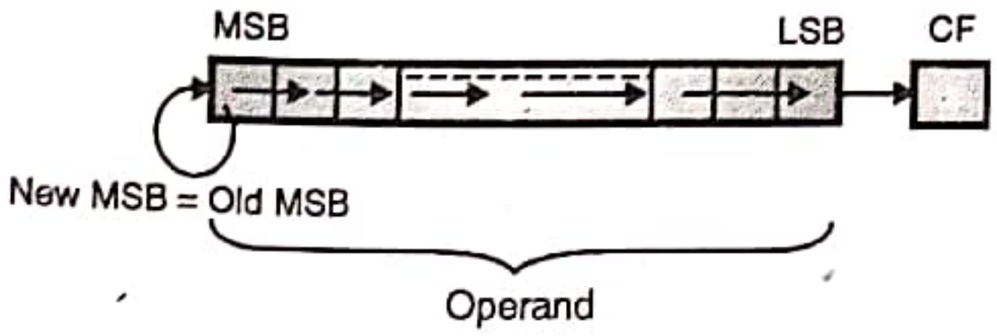


Fig. 4.6

This instruction shifts each bit is specified destination, same number of bit position of the right. Number of bits to be shifted depends upon count. As bit is shifted out of the MSB position, a copy of old MSB is put in the New MSB position. i.e. the sign bit is copied into the MSB. LSB is moved into CF.

Bits shifted into CF previously will be lost.

The count can be an any immediate number or specified in the CX register

Negative shifts are illegal.

Q. 6 Briefly explain string instructions of 8086.

May 17

Ans. : String instruction of 8086

8086/8088 provides special instructions which performs some string related activities. Five basic string operations, called **primitives** or **string primitives**. These primitives allow strings of bytes or words to be operated on, one element (byte or word) at a time. Strings of upto 64k bytes may be manipulated with these instructions.

Instructions are available to move, compare and scan for a value, as well as for moving string elements to and from the accumulator.

The five basic primitives (MOV, Compare, scan, load and store).

1) Operation B OR 2) Operation W.

The 1st and 2nd form explicitly determines B (byte) and W (word) operations respectively.

Table 4.1 : String instructions

REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero
REP	Repeat
MOVSB/MOVSW	Move byte or word string
CMPSB/CMPSW	Compare byte or word string
SCASB/SCASW	Scan byte or word string
LODSB/LODSW	Load byte or word string
STOSB/STOSW	Store byte or word string

The string instructions operate quite similarly in many respects, the common characteristics are :

1. A string instruction have a :
   (a) Source operand
   (b) Destination operand
   (c) Or both (source and destination).
2. It is assumed that source string resides in the current data segment.
3. A destination string must be in the current extra segment.
4. As mentioned, microprocessor checks the attributes of the operands to determine if the elements of strings are bytes or words.

5. Normally, the content of register SI (source index) is used as an offset, to address the current element of the source string.
6. The content of register DI (destination index) is taken as the offset of the current destination string element.
7. Register SI and DI must be initialised to point to the source / destination strings before executing the string instruction.
8. The string instructions automatically updates SI and/or DI in anticipation of processing the next string element. For this user have to set/reset DF flag to determine whether pointer should autoincrement (DF = 0) or should autodecrement (DF = 1). If byte strings are being processed, SI and/or DI is adjusted by 1, while the adjustment is 2 for word string.

Table 4.2 : String instruction register and flag use

SI	Index (offset) for source string
DI	Index (offset) for destination string
CX	Repetition counter
AL/AX	Scan value destination for LODS Source for STOS
DF	0 = autoincrement SI, DI
	1 = autodecrement SI, DI
ZF	Scan/compare terminator

9. If you use Repeat prefix, then register CX (counter or count register) is decremented by 1, after each repetition of the string instructions. In short CX is used as counter to repeat operation number of times, therefore CX should be initialized before string instruction is executed. If CX = 0, the string instruction is not executed, and control goes to the following instruction.

Q. 7 Explain the NOP instruction in Intel 8086 assembly language with example.

Dec. 11, May 13

Ans. :

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

Chapter 5 : Assembly Language Programming

Q. 1 Explain assembler directives of 8086.

May 12, Dec. 12

Ans. : Assembler directives of 8086

These are the statements that direct the assembler to do something. As the name says, it direct the assembler to do a task.

The speciality of these statements is that they are effective only during the assembly of a program but they do not generate any code that is machine executable. Divide the assembler directives into two categories namely the general purpose directives and the special directives.

They are classified into the following categories based on the functions performed by them.

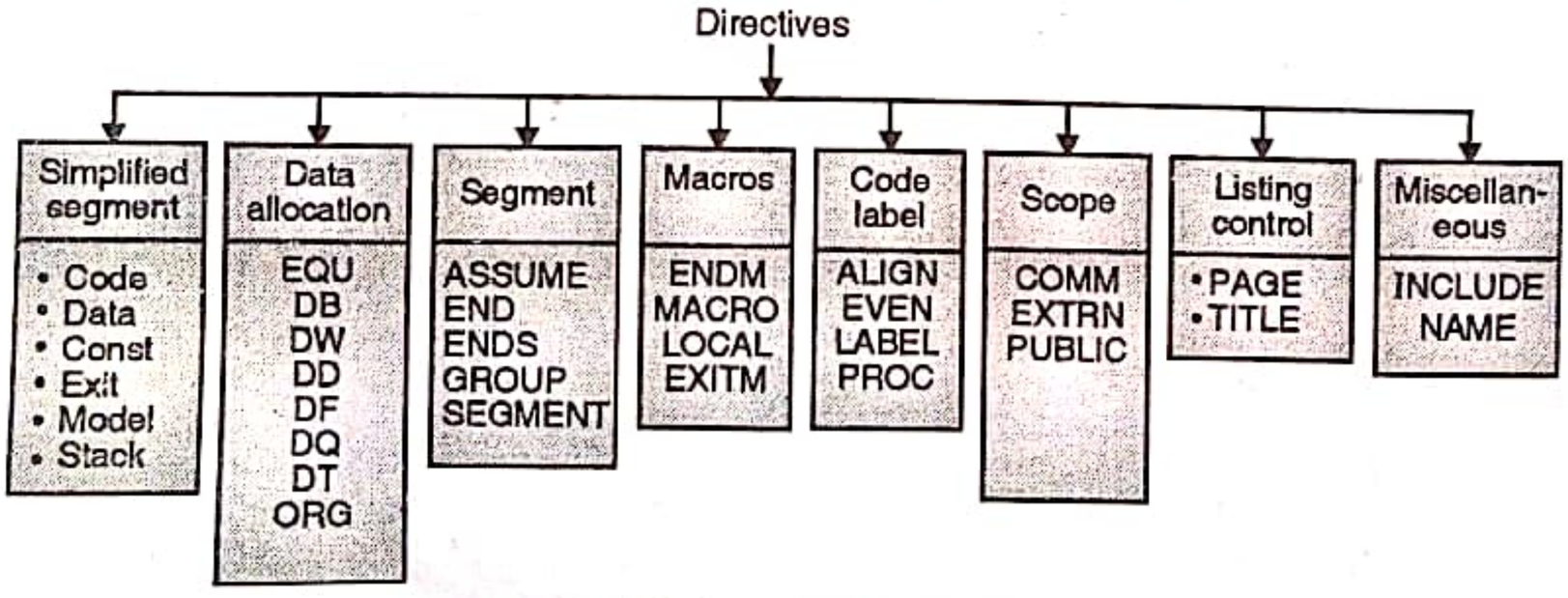


Fig. 5.1 : Assembly directives

1. .CODE

This assembler directive indicates the beginning of the code segment. Its format is as follows :

.CODE [name]

The name in this format is optional. For tiny, small and compact models the segment name is – TEXT always. The medium and large memory models use more than one code segments which can be distinguished by name.

2. .DATA

This directive indicates the beginning of the data segment.

3. EQU-Equate

It is used to give a name to some value or symbol in the program. Each time when the assembler finds that name in the program, it replaces that name with the value assigned to that r variable.

Its format is [name] EQU initial value.

e.g. : FACTORIAL EQU 05H.

This statement is written during the beginning of the program and whenever now FACTORIAL appears in an instruction or another directive, the assembler substitutes the value 5.

The advantage of using EQU in this manner is that if FACTORIAL is used several times in a program and the value has to be changed, all that has to change the EQU statement and reassemble the program. The assembler with automatically put the new value each time it finds the name FACTORIAL.

4. Define Byte [DB]

This directive defines the byte type variable. It is also usefui to set one or more storage locations aside.

The format of this directive is as follows :

[name] DB initial value

The initial value can be a numerical value (8 – bit long) or more than one 8 bit numeric values. It can be a constant expression, or a string constant or even a question mark. The initial value can be a signed or unsigned number. Its range is from – 128 to + 127 if unsigned and 0 to 255 if it is unsigned.

5. ASSUME

The directive is used for telling the assembler the name of the logical segment which should be used. The format of the assume directive is as follows,

ASSUME segment register : segment-name :

The segment register can be CS, DS, SS and ES.

The example of Assume directive is as follows,

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

ASSUME statement can assign upto 4 segment registers in any sequence. In this example, DS : Data means that the assembler is to associate the name of data segment with DS register. Similarly CS : Code tells the assembler to associate the name of code segment with CS register and so on.

6. END

This is placed at the end of a source and it acts as the last statement of a program. This is because the END directive terminates the entire program. The assembler will neglect any statement after an END directive. The format of END directive is as follows :

END

7. MACRO AND ENDM

The macros in the programs can be defined by MACRO directive. The ENDM directive is used along with the Macro directive. ENDM defines the end of macro.

Its format is

DISP MACRO

; Statements inside the Macro

ENDM

8. ALIGN

This directive will tell the assembler to align the next instruction on an address which corresponds to the given value. Such an alignment will allow the processor to access words and double words.

ALIGN number

This number should be 2, 4, 8, 16 .... i.e. it should be a power of 2.

The example of Align directive are ALIGN 2 and ALIGN 4. ALIGN 2 is used for starting the data segment on a word boundary whereas ALIGN 4 will start the data segment on a double boundary word.

9. EVEN (Align on Even Memory Address)

It tells the assembler to increment its location counter if required, so that the next defined data item is aligned on an even storage boundary. The 8086 can read a word from memory in one bus cycle if the word is at an even address. If the word starts at an odd address, the microprocessor must do two read cycles to get 2 bytes of the word. In the first cycle it will read the LSB & in the second it will read MSB. Therefore, a series of even words can be read more quickly if they are at an even address.

e.g. : EVEN TABLE DB 10 DUP (O) ; It declares an array named TABLE of 10 bytes which are starting from an even address.

10. EXTRN

It indicates that the names or labels that follow the EXTRN directive are in some other assembly module.

e.g. : EXTRN DISP : FAR.

The statement tells the assembler that DISP is a label of type far in another assembly module. To call a procedure that is in a program module assembled at a different time from that which contains the CALL instruction, the assembler has to be told that the procedure is external. The assembler will then put information in the object code file so that linker can connect the two modules together. The names or labels that are external in one module must be declared public with the PUBLIC directive in the module where they are defined.

Its format is

e.g. : Procedure_HereSegment

EXTRN FACT	:	FAR
EXTRN SUM	:	NEAR.

Procedure_HereEnds.

11. PAGE

This directive is used to specify the maximum number of lines on a page and the maximum number of characters on a line.

The format of this directive is as follows :

PAGE [length], [width]

- [length] → Number of lines on a page.
- [width] → Number of characters on a line.

The example is PAGE 55, 102 which shows that the, there are 55 lines per page and 102 characters per line. The number of lines per page typically range from 10 to 255 and the number of characters per line will range from 60 to 132.

12. TITLE

It helps the user for controlling the format of listing of an assembled program. It is used to give a title to program and print the title on the second line of each page of the program. The maximum number of characters allowed as a title is 60.

The format of this assembler directive is as follows,

TITLE Text

13. INCLUDE

This directive is used to tell the assembler to insert a block of source code from the named file into the current source module. This shortens the source code.

Its format is

INCLUDE path : file name.

14. NAME

This directive assigns a specific name to each assembly module when programs consisting of several modules are written.

Q. 2 Write an assembly language program to exchange the blocks of 1KB located at 01000H and 02000H using string instructions. May 12, Dec. 14

Ans. :

Explanation

The source block is at address 1000 H and destination block is at address 2000 H. Let the number of bytes in the block to be transferred be 10. Initialize this as count in CX register. Here first copy the 1KB block starting at location 01000H to another place at 0300H onwards. Then the block from 02000H onwards is transferred to the first block i.e. starting from 01000H. Finally the block copied in location 03000H onwards is transferred to the locations 02000H onwards.

Program

Instruction
MOV AX,0000H
MOV DS,AX
MOV ES,AX
MOV SI,1000H
MOV DI,3000H
MOV CX,0400H
CLD
REP MOVSB
MOV SI,2000H
MOV DI,1000H
MOV CX,0400H
CLD
REP MOVSB
MOV SI,3000H
MOV DI,2000H
MOV CX,0400H
CLD
REP MOVSB

Q. 3 Write a program to check if the given string is palindrome. May 14

Ans. :

Label	Instruction
	.model small
	.data
	str db "NITIN"
	str1 dB 5 dup (?)
	pal dB 0

Label	Instruction
	count dW 5
	. code
start :	MOV AX, @data
	MOV DS, AX
	MOV ES, AX
	MOV CX, count
	LEA SI, str
	LEA DI, str1
	ADD DI, CX
	DEC DI
back :	CLD
	LODSB
	STD
	STOSB
	LOOP back
	LEA SI, str
	LEA DI, str1
	MOV CX, count
	REPE CMPSB
	JNZ over
	MOV pal, 01H
over :	MOV AH, 4CH
	INT 21H
	end start

Q. 4 Write an assembly language program for 8086 to transfer the block of 1 KB located at 0100H to 0200H using string instructions. **Dec. 12**

Ans. :

Program :

```
MOV AX, 0000H
MOV DS, AX
MOV ES, AX
MOV SI, 0100H
MOV DI, 0000H
MOV CX, 0400H
CLD
REP MOVSB
```

Chapter 6 : Stacks and Subroutines

Q. 1 Explain procedures with example.

Ans. : Procedures

Whenever need to use a group of instructions several times throughout a program there are two ways avoid having to write the group of instructions each time to use them.

One way is to write the group of instructions as a separate procedure. Then CALL the procedure whenever it executes that group of instructions. For calling the procedure, the return address has to be stored back on to the stack. Another way is to write the group of instructions as a Macro.

The basic requirements which must be satisfied while calling a procedure are :

(1) A procedure call must save the address of the next instruction so that the return will be able to branch back to the proper place in the calling program.

(2) The registers used by the procedure need to be stored before their contents are changed and then restored just before the procedure is exited.

(3) A procedure must have a means of communicating or sharing data with the routine that calls it and other procedures.

The first requirement is met by the CALL and RET branch instructions. The second requirement is met by saving the register and flags by PUSH instruction and retrieving them back by POP instruction. The procedure are delimited within the source code by the statement form.

```
[procedure name]    PROC [attribute] ;    at beginning of procedure
                    :
                    :
[procedure name]    ENDP                  ;    at the end
```

The procedure name is the identifier used for calling the procedure and the attribute is NEAR or FAR. If the procedure is within the same code segment where the main program is stored then it is a NEAR procedure and the attribute NEAR is used for calling it.

If the procedure is in some other segment than where the main program is stored then it is a FAR procedure. For a near procedure CALL instruction pushes only the IP contents on stack as contents of CS register remain unchanged for main program and procedure, whereas for a far procedure the CALL instruction pushes the contents of CS and IP both on the stack.

Q. 2 Write short note on macros with example.

Ans. : Macros

When the repeated group of instructions is too short not appropriate to be written as a procedure, use a macro. A macro is a group of instructions that perform a task. Each time call the macro in our program the assembler will insert the group of defined instructions in place of "call". The macros are useful for following purposes.

1. To simplify and reduce the amount of repetitive coding.
2. To reduce the errors caused by repetitive coding.
3. To make the assembly language program more readable.
4. A macro executes faster because, there is no need of call and return.

The basic **format** of a Macro is

```
macro name MACRO      ; Define macro
                      ; Body of macro
              ENDM; End macro.
```

The MACRO directive on the first line tells the assembler that the instructions that follow upto ENDM are a part of macro definition. The ENDM directive ends the macro definition. The instructions between MACRO and ENDM comprise the body of macro definition.

```
e.g. : INIT MACRO              ;  define MACRO.
       MOV  AX, @ Data  }      ;
       MOV  DS, AX      }      ;  Body of MACRO
       MOV  ES, AX      }      ;
       ENDM                    ;  END MACRO.
```

The assembler places the macro instructions in the program when it is invoked, this is called as Macro expansion.

Q. 3 Compare Procedure and Macro.

Ans. : Comparison 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.
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. 4 Explain software delay and hardware delay.

Ans. :

1. Software delay

This delay is introduced using the instruction time. The microprocessor requires the number of T states while executing an instruction. By repeating the instructions for N number of times, the delay between two events can be easily obtained.

$$\text{Time required for 1 T state} = \frac{1}{\text{Operating frequency}}$$

Example :

To implement timer delay using an 8 bit counter.

First let us, write the program. Assume 8086 clock frequency to be 5 MHz.

```
. model small
. stack 100
. code                                          T states required
    MOV AL, count    ;  Load counter              4
Loop 1 : DEC AL      ;  Decrement counter         3
    JNZ Loop1        ;  Repeat till counter =     0.16
                                (When condition true)
    end              ;  end program.              4
                                (When condition false)
```

Total delay time = $4 + \text{count} \times 3 + [\text{count} - 1] \times 16 + 4$

MOV DEC AL JNZ Loop 1

MOV AL, count is executed only once so T states = 4.

DEC AL is executed count times so T states = count.

JNZ Loop 1 is executed count – 1 times

It will not satisfy when count = 0,

so T states = (count – 1) × 16 + 4

If count = 4

Timer delay = 4 + 4 × 3 + 3 × 16 + 4

= 4 + 12 + 48 + 4

= 68 T states.

Operating frequency of 8086 = 5 MHz.

∴ Time required for 1 T state = $\frac{1}{5\text{ MHz}} = 0.2\ \mu s.$

∴ Total Time Delay = 68 × 0.2

Total Time Delay = 13.6 μs.

2. Hardware Delay

In this, one can use 555 timer or 8253 programmable interval timer which will interrupt 8086 microprocessor, after a specified time duration.

Q. 5 Write short note on Mixed language programming. Dec. 15

Ans. : Mixed language programming

'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.

It is true that the time needed to write a program in assembly language is much more than the time taken in Higher Level Languages like C. However, there are special cases where a function is coded in assembly language to reduce execution time.

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.

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 :

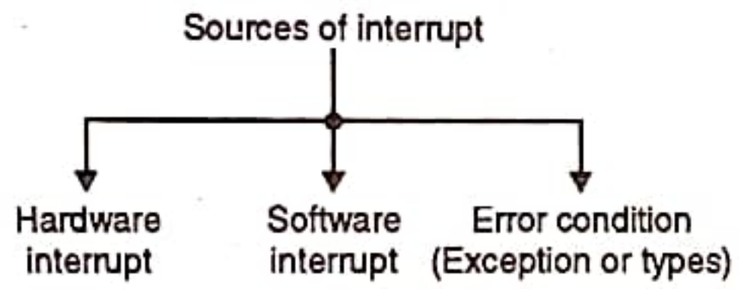
1. Increase the speed and efficiency of the routine.
2. Perform Machine specific functions not available in Microsoft C or in Turbo C.
3. Use third party routines.

Chapter 7 : 8086 Interrupt Structure

Q. 1 Explain the Interrupt structure of the 8086.

May 11, May 17

Ans. : In 8086, we have three sources of interrupt.



1. Hardware Interrupt

In this type of interrupt, physical pins are provided in the chip. In 8086 we have two pins :

(i) NMI (Non maskable interrupt). (ii) INTR.

To interrupt the processor we have to apply signal to these pins. As name suggests, 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)

The 8086 supports division, multiplication, addition etc. Suppose by mistake if user asks microprocessor to divide any number by ZERO, then we 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"**. (Possibly you may have come across this error). Instead of showing the answer as **'∞ (infinity)"**. Thus, internally generated errors produces an interrupt for microprocessor, normally referred as **"TYPE"** by Intel engineer and referred as **"Exception"** by motorola engineer. Thus we conclude that 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. 7.1 shows interrupt sources for 8086. As shown in Fig. 7.1 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.

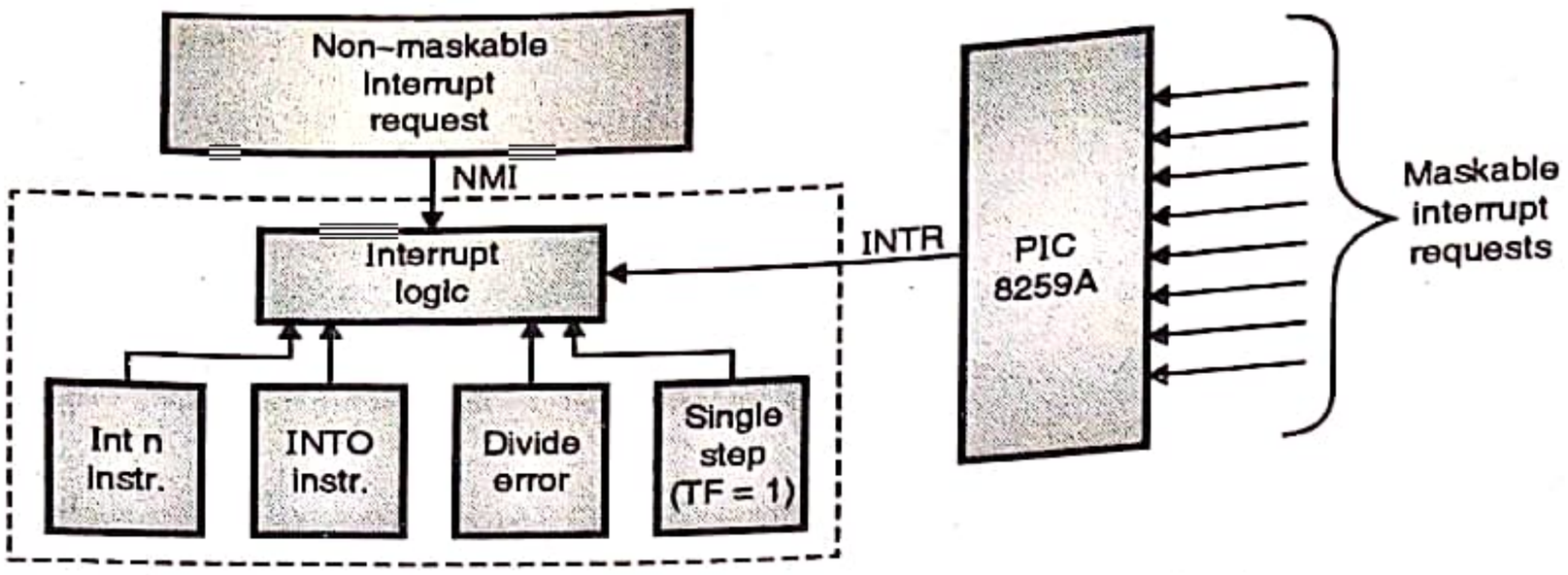


Fig. 7.1 : Interrupt sources (INTO-Interrupt on Overflow)

Q. 2 What is an ISR ? How does 8086 acknowledge an interrupt ? Draw flowchart for interrupt processing sequence.

Ans. :

At the end of each instruction cycle, the 8086 checks to see if any interrupts have been requested. Therefore whenever interrupt occurs, it won't be immediately checked by microprocessor.

Servicing of Interrupts by 8086 Microprocessor

Microprocessor first completes, execution of current instruction and then checks for an interrupt. Now the main important point is, how 8086 acknowledges it. The same has been graphically presented in flow chart (Refer Fig. 7.2).

(1) First, microprocessor will complete execution of current instruction.

(2) It checks for any internal interrupt, suppose the same is not present.

(3) Then it checks for NMI i.e. hardware interrupt.

(4) If NMI is not present, it will check for INTR.

(5) In absence of INTR, it will continue checking for TF (Trap, single step) flag.

(6) If TF is not equal to 1, it will switch over to next instruction.

(7) If you observe flow chart, you conclude that except INTR, all interrupt comes to point of pushing flags. Therefore first we will take path of INTR and after that next discussion is common to all interrupts.

(8) Suppose microprocessor finds that INTR is present, then it checks for IF (interrupt enable flag), If it is 0, it will not service INTR and return back to check TF flag.

(9) Suppose IF = 1, then it will execute **"interrupt acknowledge"** cycle. In this cycle it captures **"TYPE CODE"**.

(10) After capturing the **"type code"**, the next sequence will be executed which is common to all interrupt.

(a) Push flag register.

(b) Temp = TF (Save present status of TF).

(c) Clear IF and TF. This disables INTR input and single step function.

(d) Push CS and IP (OLD CS : OLD IP).

(e) CALL INTERRUPT service routine.

This CALL is equivalent of an intersegment indirect called instruction. By calling this routine, microprocessor receives NEW CS and NEW IP value from vector table (also referred as Interrupt Pointer Table).

(f) These NEW CS and NEW IP values, will be loaded into code segment register and instruction pointer, respectively.

(g) Before transferring control to NEW CS : NEW IP again checks for NMI. If yes, it will jump to point (a), else check for TEMP. In TEMP we had stored TF status. So if TF = 1, i.e. single step is activated, then microprocessor jumps to point (a).

(h) But if TEMP = TF = 0, it will execute USER Interrupt procedure, i.e. ISR written by user.

(i) At the end of ISR, user will write IRET i.e. return from an interrupt.

(j) In response to IRET, microprocessor will pop up CS and IP (normally referred as OLD CS and OLD IP).

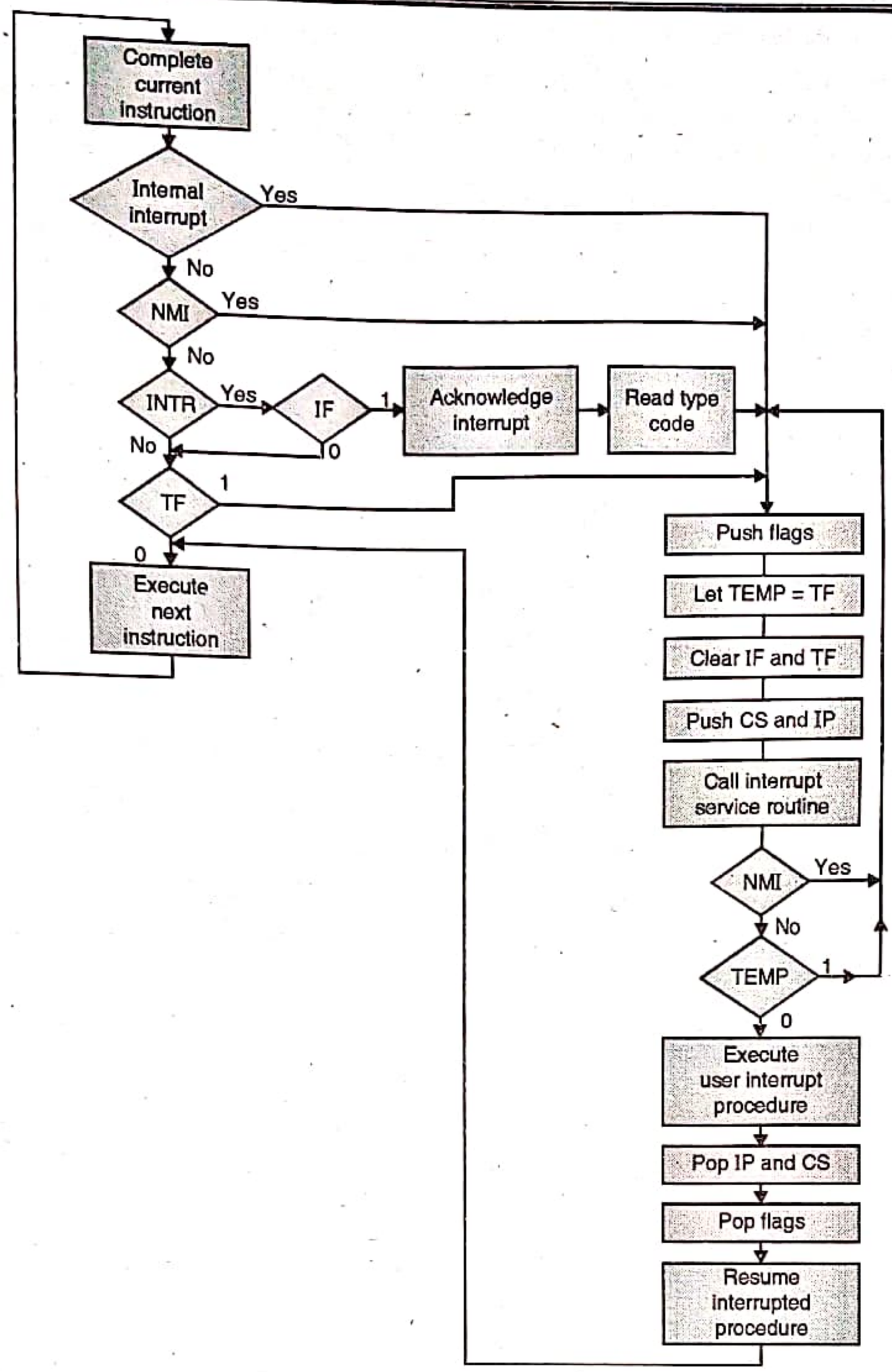


Fig. 7.2 : Interrupt processing sequence/ priority sequence

It will also pop flag status and will resume program execution from the point where it was interrupted. To resume interrupted procedure microprocessor simply loads OLD CS and OLD IP value to CS register and IP register respectively. The full process is diagrammatically presented in Fig. 7.3.

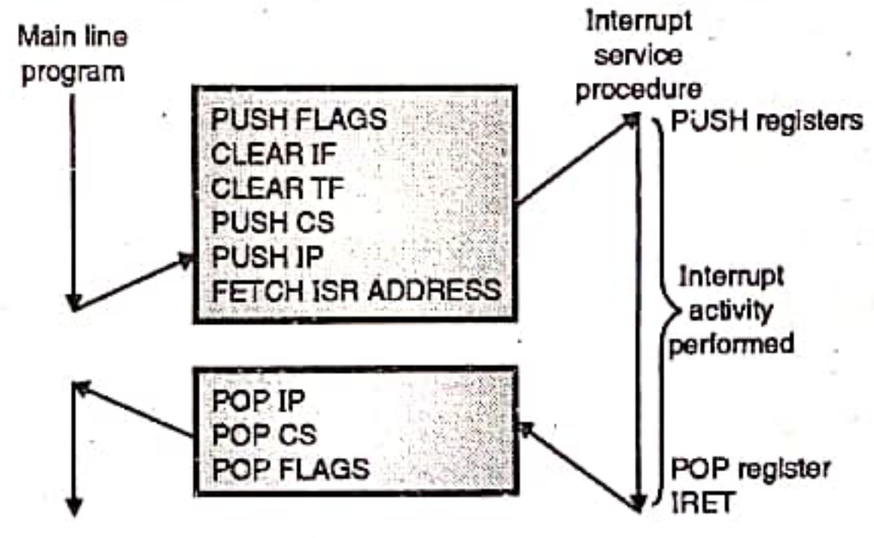


Fig. 7.3 : 8086 interrupt response

Q. 3 Explain Interrupt Vector Table (IVT) with diagram.

Ans. : Interrupt Vector Table

The interrupt vector (or interrupt pointer) table is the link between an interrupt type code and the procedure that has been designated to service interrupts associated with that code. 8086 supports total 256 types i.e. 00H to FFH. For each type it has to reserve four bytes i.e. double word. This double word pointer contains the address of the procedure that is to service interrupts of that type. The higher addressed word of the pointer contains the base address of the segment containing the procedure.

This base address of the segment is normally referred as NEW CS. The lower addressed word contains the procedure's offset from the beginning of the segment. This offset we normally refer as NEW IP. Thus NEW CS : NEW IP provides NEW physical address from where user ISR routine will start.

As for each type, four bytes (2 for NEW CS and 2 for NEW IP) are required, therefore interrupt pointer table occupies upto the first 1k bytes (i.e. 256 × 4 = 1024 bytes), of low memory. Thus 00000 H to 003FF H, these locations of 8086 microprocessor is reserved for interrupt vector table. The microprocessor receives NEW CS and NEW IP value from vector table, after calling internal service routine.

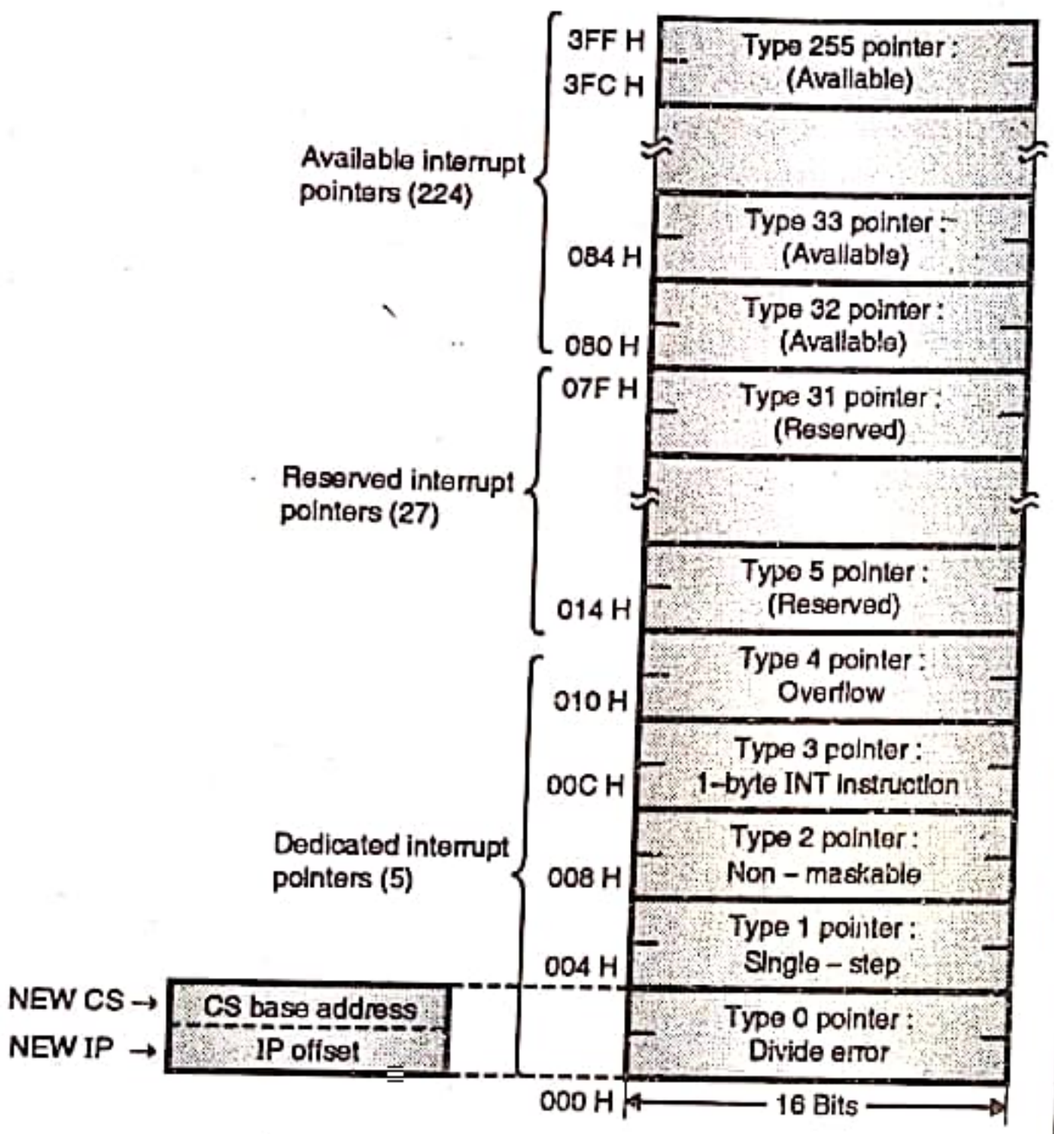


Fig. 7.4 : Interrupt vector table

We want to know that, what activity is performed by this routine to get NEW CS and NEW IP. We know that for each **"type"** we have 4 locations reserved. So call routine, will simply sense the type number, multiply the same with 4 and will get double word pointer from that location. Suppose for example type code is **'3'**. Then $3 \times 4 = (12)_{10} = 0C$ H.

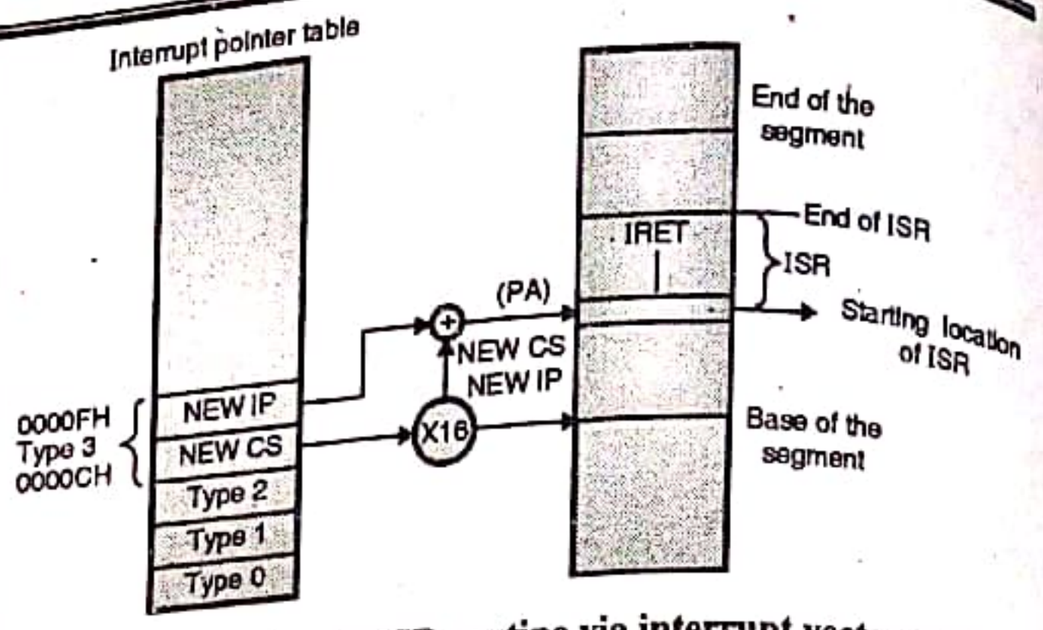


Fig. 7.5 : Pointing to ISR routine via interrupt vector table

Then 0CH/0DH location will provide OFFSET IP (NEW IP) and 0EH/0FH will provide base address of the segment (NEW CS).

Interrupt vector table is divided into three groups :

(1) Dedicated interrupt pointers (Type 0/1/2/3/4)

(2) Reserved interrupt pointers (Type 5 to Type 31)

(3) Available interrupt pointers (Type 32 to Type 225).

Type 0 to 4 are dedicated interrupt pointers by Intel for divide by zero error, single step, NMI, 1-byte INT instruction and overflow. Interrupt types from 5 to 31 are reserved by Intel for use in more complex microprocessors, such as 80286, 80386, 80486. Finally the types from 32 to 255 i.e. total 224, are available to user to use for either hardware or software interrupt.

Q. 4 How does 8086 decide the priority of interrupts ?

Ans. :

Table 7.1

Interrupt	Priority
NMI	Highest
Divide Error, INT n, INTO	
INTR	
Single Step	Lowest

When considering the precedence of interrupts for multiple simultaneous interrupts, the following guideline apply :

1. INTR is the only maskable interrupt and if detected simultaneously with other interrupts, resetting of IF by the other interrupts will mask INTR. This causes the INTR to be the lowest priority interrupt serviced after all other interrupts unless the other interrupt service routine enable interrupts.
2. Of the non-maskable interrupts (NMI, single step and software generated), in general, NMI has highest priority followed by software, followed by single step software interrupt. This implies following three cases :

CASE 1 : Simultaneous NMI and single step will cause NMI routine to be followed by single step.

CASE 2 : Simultaneous software trap and single step trap will cause the software interrupt service to be followed by single step.

CASE 3 : Simultaneous NMI and software trap will cause NMI routine to be executed followed by the software interrupt service routine.

Chapter 8 : IC 8259 Programmable Interrupt Controller (PIC)

Q. 1 Write short note on 8259-PIC. May 16

Ans. :

The block diagram of 8259 is as shown in Fig. 8.1. It contains following blocks :

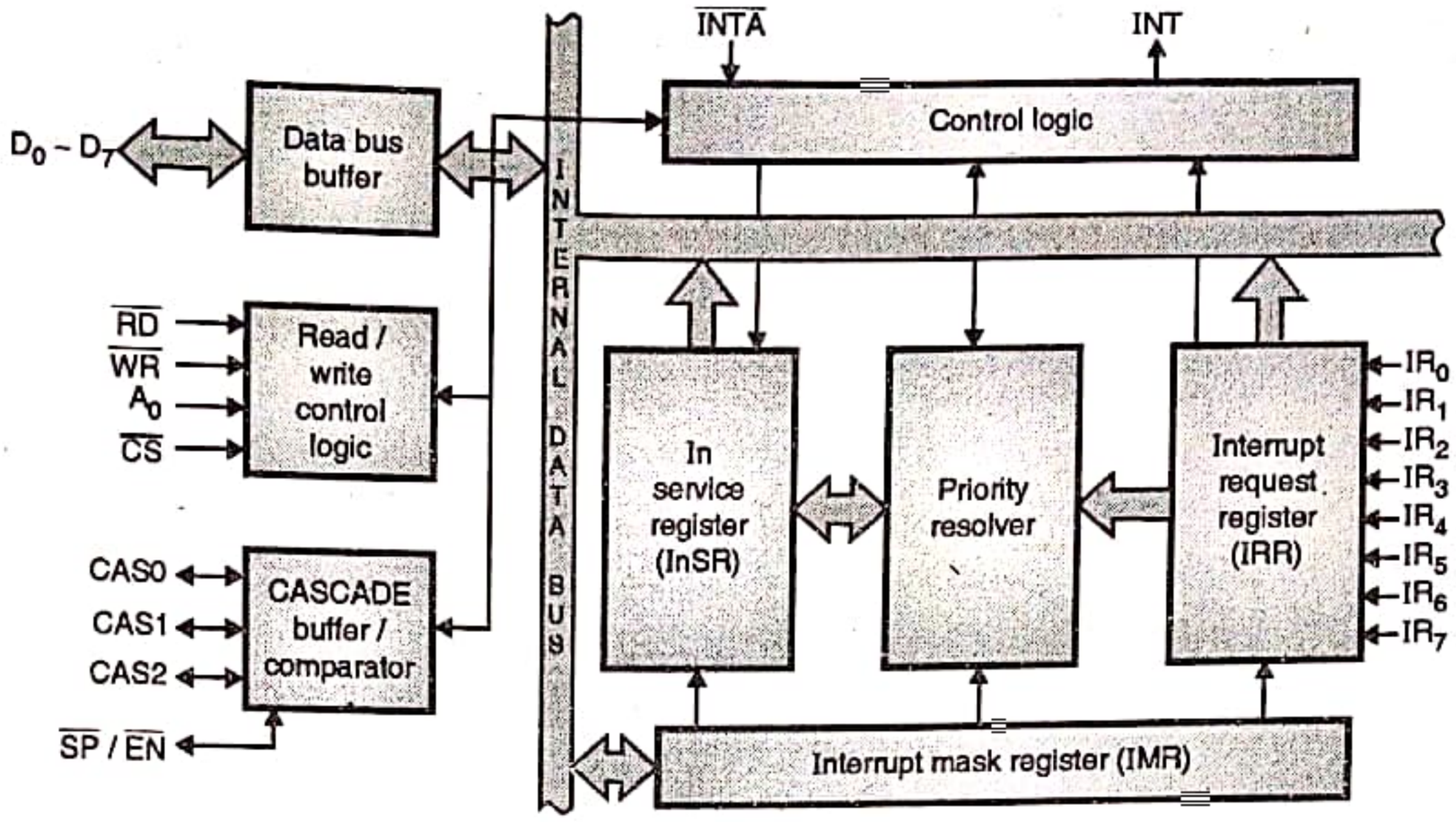


Fig. 8.1 : Functional block diagram of 8259

(1) Data bus buffer

It is used to transfer data between microprocessor and internal bus.

(2) Read/Write control logic

It sets the direction of data bus buffer. It controls all internal read / write operations. It contains initialization and operation command registers.

(3) Cascaded buffer and comparator

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

(4) Control logic

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

(5) Interrupt Request Register (IRR)

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

(6) In Service Register (InSR)

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

(7) Priority resolver

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

(8) Interrupt Mask Register (IMR)

It is a programmable register. It is used to Mask unwanted interrupt request, by writing appropriate command word. The

microprocessor can read contents of this register without issuing any command word.

Q. 2 Explain the fully nested mode of PIC 8259. Dec. 11

Ans. :

Fully Nested Mode (FNM)

After initialization, the 8259A operates in fully nested mode i.e. it is default priority mode. It continues to operate in this mode until the mode is changed through OCWs. It is also called as default mode. In this mode, IR_0 has highest priority and IR_7 has lowest priority. When the interrupt is acknowledged, it sets the corresponding bit of ISR.

This bit will prevent all interrupts of the same or lower level, however it will accept higher priority interrupt requests. The bit in the InSR will remain set until an EOI (End of Interrupt) command is issued by the microprocessor at the end of ISR (Interrupt Service Routine). If the AEOI (Automatic End of Interrupt) bit is set, the bit in the InSR resets at the trailing edge of last $\overline{\text{INTA}}$.

End Of Interrupt (EOI)

(1) The InSR bit can be reset by on EOI command that is issued by the microprocessor before existing from the interrupt routine.

(2) In the FNM, the highest level in the InSR would correspond to the last interrupt that is acknowledged and serviced in such a case, a non-specific EOI command can be issued

(3) If the fully nested mode is not used, the 8259 PIC may not be able to determine the last interrupt that is acknowledged. In such a case a specific EOI command needs to be issued.

(4) In the cascade mode, the EOI command should be issued twice once for master and once for salve.

(A) Nonspecific EOI command

This command informs the 8259A that the current interrupt service routine has been completed. It resets current bit (highest priority bit) of the InSR. This command is independent of the interrupt level and is thus called a nonspecific EOI. It must be used in fully nested mode.

(B) Specific EOI command

If the fully nested mode is not used, the 8259A may not be able to tell which interrupt was just acknowledged. In such a case a specific EOI command must be issued. This command resets a bit of InSR, which is specified by $L_2L_1L_0$.

Automatic End Of Interrupt (AEOI)

In this mode the 8259 will perform a non-specific EOI on its own and on the trailing edge second $\overline{\text{INTA}}$ pulse. It can be used only for master and not for salve.

Q. 3 Explain the initialization Command Words (ICWs) and Operational Command Words (OCWs) of the 8259 PIC ? May 11

Ans. :

The 8259 can be programmed through a sequence of simple I/O operations. It accepts two types of command words. They are :

1 Initialization Command Word 1 (ICW1)

It is used to program the basic operation of 8259A. To write this command word into ICW1 register A_0 pin should be at logic '0'. After accepting this command the 8259A performs the following internal operations :

(1) It resets edge sense circuit, hence an interrupt request must make a low to high transition to trigger IR input after initialization.

(2) It clears interrupt mask register hence all interrupt are unmasked.

(3) It assigns lowest priority to IR_7 and highest priority to IR_0.

(4) It sets slave identification number of slave 7 (1,1,1), if D_1 bit of ICW1 is '0'.

(5) It clears special mode and sets the status read to IRR, i.e. microprocessor can read IRR without issuing a special command word.

(6) If D_0 bit of ICW1 is reset, then it clears all functions associated with ICW4.

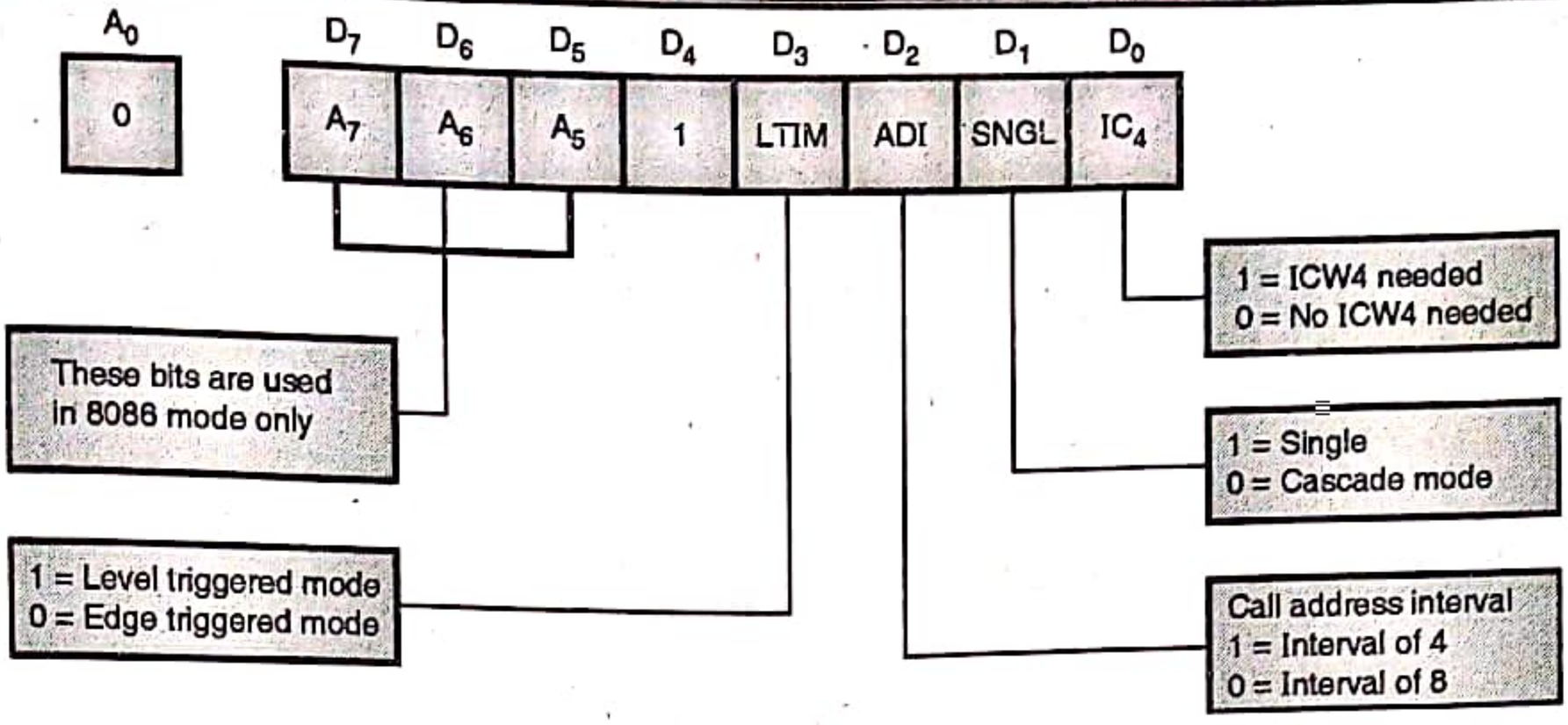


Fig. 8.2 : Initialization Command Word 1 (ICW1)

2. Initialization Command Word 2 (ICW2)

The ICW2 is used to program 8 bit vector number of interrupt type.

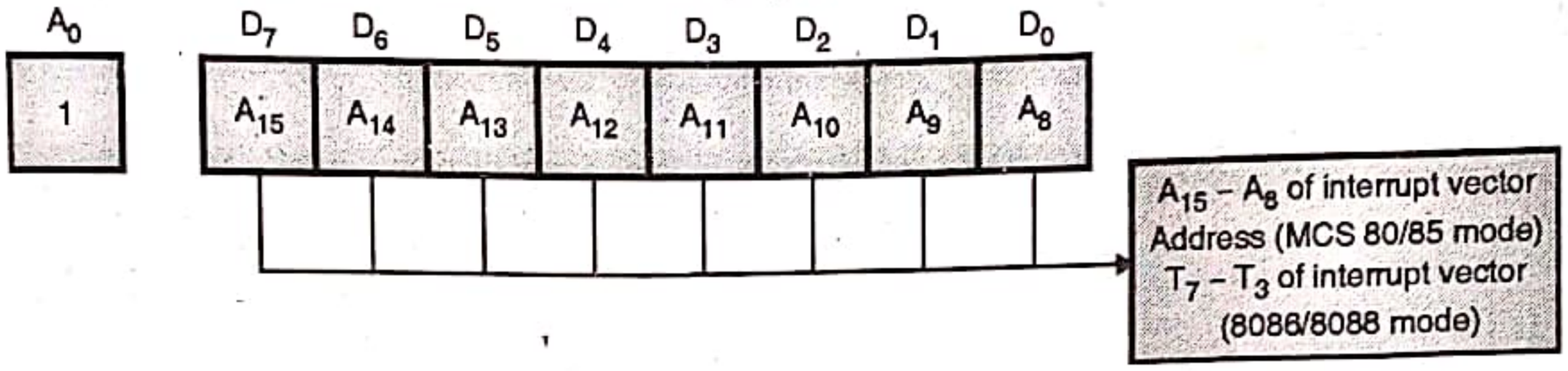


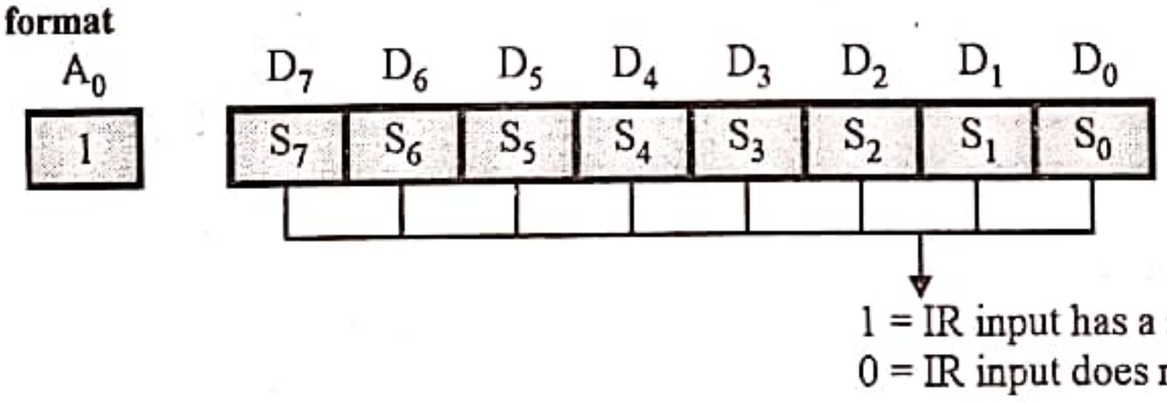
Fig. 8.3 : Initialization Command Word 2 (ICW2)

A write command issued after ICW1 with $A_0 = 1$ is considered as ICW2. The ICW2 format is as shown in Fig. 8.3. In 8085 mode, ICW2 bits are used to program A_8 to A_{15} address bits of ISR address. In 8086 mode, D_3 to D_7 bits are used to program T_3 to T_7 bits of 8 bit vector number. The lower bits T_0 to T_2 are provided by 8259 depending on which interrupt input is activated.

3. Initialization Command Word 3 (ICW3)

It is used in cascaded mode only. There are two types of ICW3's viz master ICW3 and slave ICW3. The master ICW3 is used to specify whether it has a slave 8259 connected to its interrupt request input. The slave ICW3 is used to assign a slave identification number. Identification number is used to tell slave 8259 on which IR input it is connected to master. A write command is issued after ICW1 and ICW2 and multiple 8259 system with $A_0 = 1$ is considered as ICW3. The ICW3 format is as shown in Fig. 8.4.

(ICW3) Master format



(ICW3) Slave format

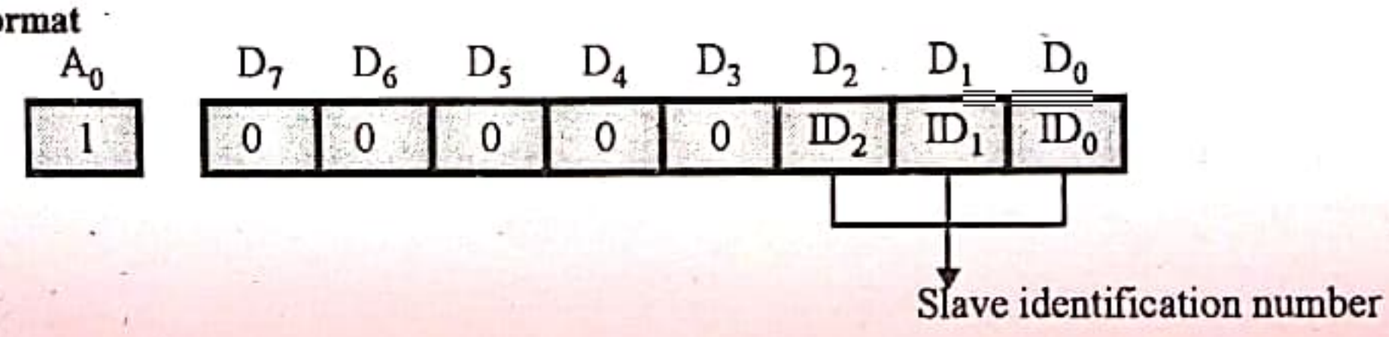


Fig. 8.4 (Contd...)

ID_2	ID_1	ID_0	Slave PIC
0	0	0	Slave on IR0
0	0	1	Slave on IR1
0	1	0	Slave on IR2
0	1	1	Slave on IR3
1	0	0	Slave on IR4
1	0	1	Slave on IR5
1	1	0	Slave on IR6
1	1	1	Slave on IR7

Fig. 8.4 : (ICW3) Master/slave format

4. Initialization Command Word 4 (ICW4)

The ICW4 is used to initialize the 8259A in the following modes :

1) Special fully nested mode.
2) Buffered mode.
3) Auto EOI mode.
4) 8086/8088 mode.

A write command issued after ICW3 and ICW4 bit needed bit set in ICW1 with $A_0 = 1$ is considered as ICW4. The ICW4 format is as shown in Fig. 8.5.

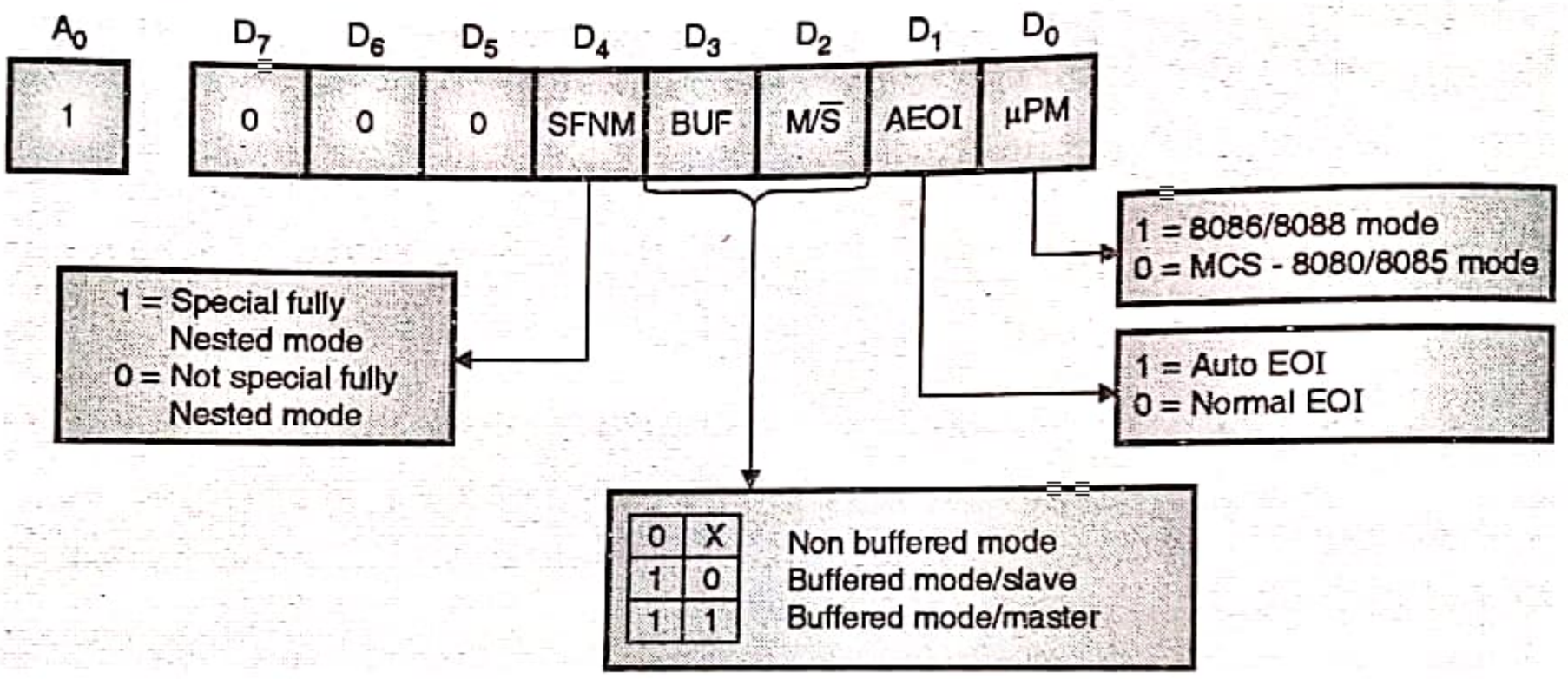


Fig. 8.5 : Initialization Command Word 4 (ICW4)

5. Operation Command Word 1 (OCW 1)

The OCW1 is used to mask unwanted interrupt request inputs (IR inputs). A write command issued after initialization with $A_0 = 1$ is considered as OCW1. The OCW1 format is as shown in Fig. 8.6.

OCW1 Format

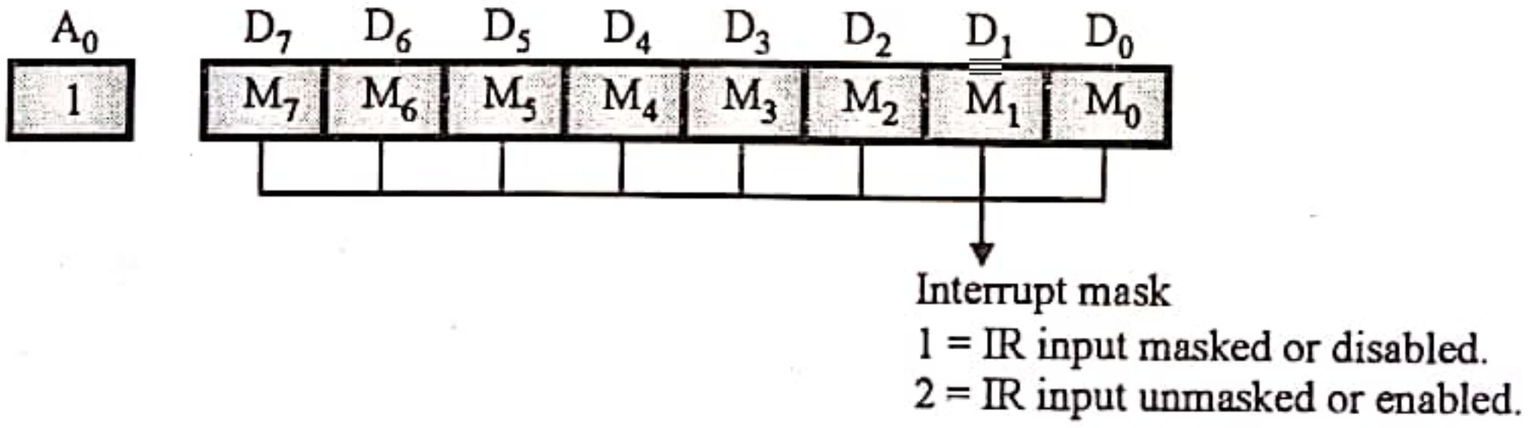
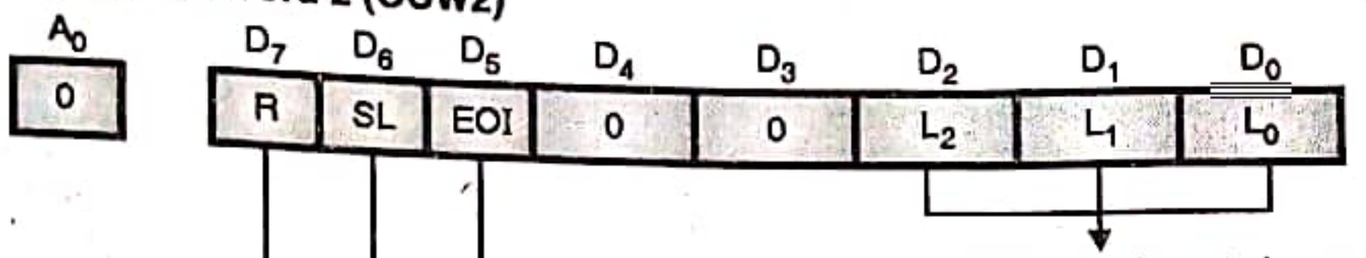


Fig. 8.6 : OCW1 format

6. Operation Command Word 2 (OCW2)



End of interrupt	0	0	1	Non specific EOI command
	0	1	1	Specific EOI command. The bit of ISR to be cleared is specified by $L_2 L_1 L_0$
Non Specific rotation	1	0	1	Rotate priorities on non specific EOI command
Automatic rotation of enable / disable	1	0	0	Rotate priorities in auto EOI mode set command
	0	0	0	Rotate priorities in auto EOI mode clear command
Specific rotation of priorities	1	1	1	Rotate priorities on specific EOI command. The IR input, to be assigned at lowest priority is specified by $L_2 L_1 L_0$
	1	1	0	Set priority command. The IR input to be assigned lowest priority is specified by $L_2 L_1 L_0$
	0	1	0	No operation.

L_2	L_1	L_0	Interrupt
0	0	0	IR_0
0	0	1	IR_1
0	1	0	IR_2
0	1	1	IR_3
1	0	0	IR_4
1	0	1	IR_5
1	1	0	IR_6
1	1	1	IR_7

Fig. 8.7 : OCW2 format

The OCW2 is used to program EOI (End Of Interrupt), rotate priorities and combination of both. A write command issued with $A_0 = 0$ and $D_4D_3 = 00$ is considered as OCW2. The OCW2 format is as shown in Fig. 8.7.

7. Operational Command Word 3 (OCW3)

The OCW3 is used to program

1. Special mask mode.
2. Polled mode and
3. Read IRR and InSR. A write command issued with $A_0 = 0$ and $D_4D_3 = 01$ is considered as OCW3. The OCW3 format is as shown in Fig. 8.8.

OCW3 Format

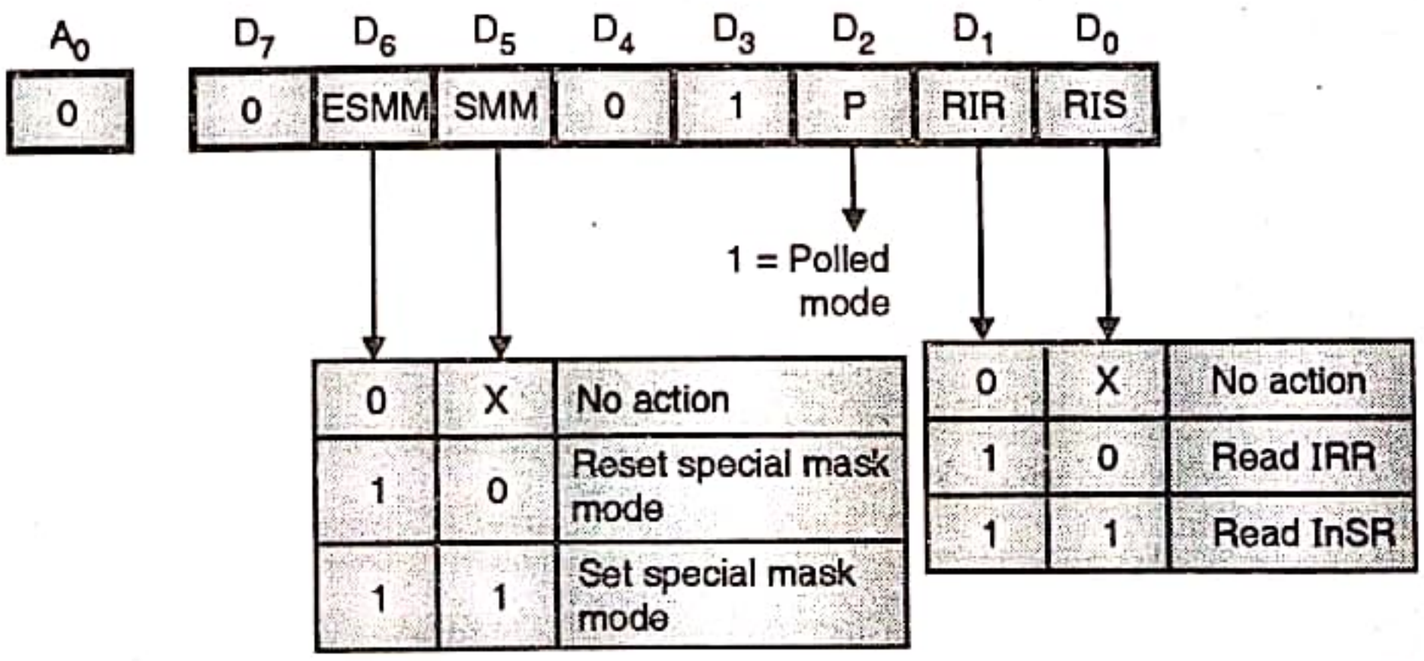


Fig. 8.8 : OCW3 format

Q. 4 Interface three 8259s with 8086 in minimum mode and explain its functionality in fully nested mode. Dec. 17

Ans. :

For 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. 8.9 shows cascaded 8259 interfaced with 8086 in maximum mode.

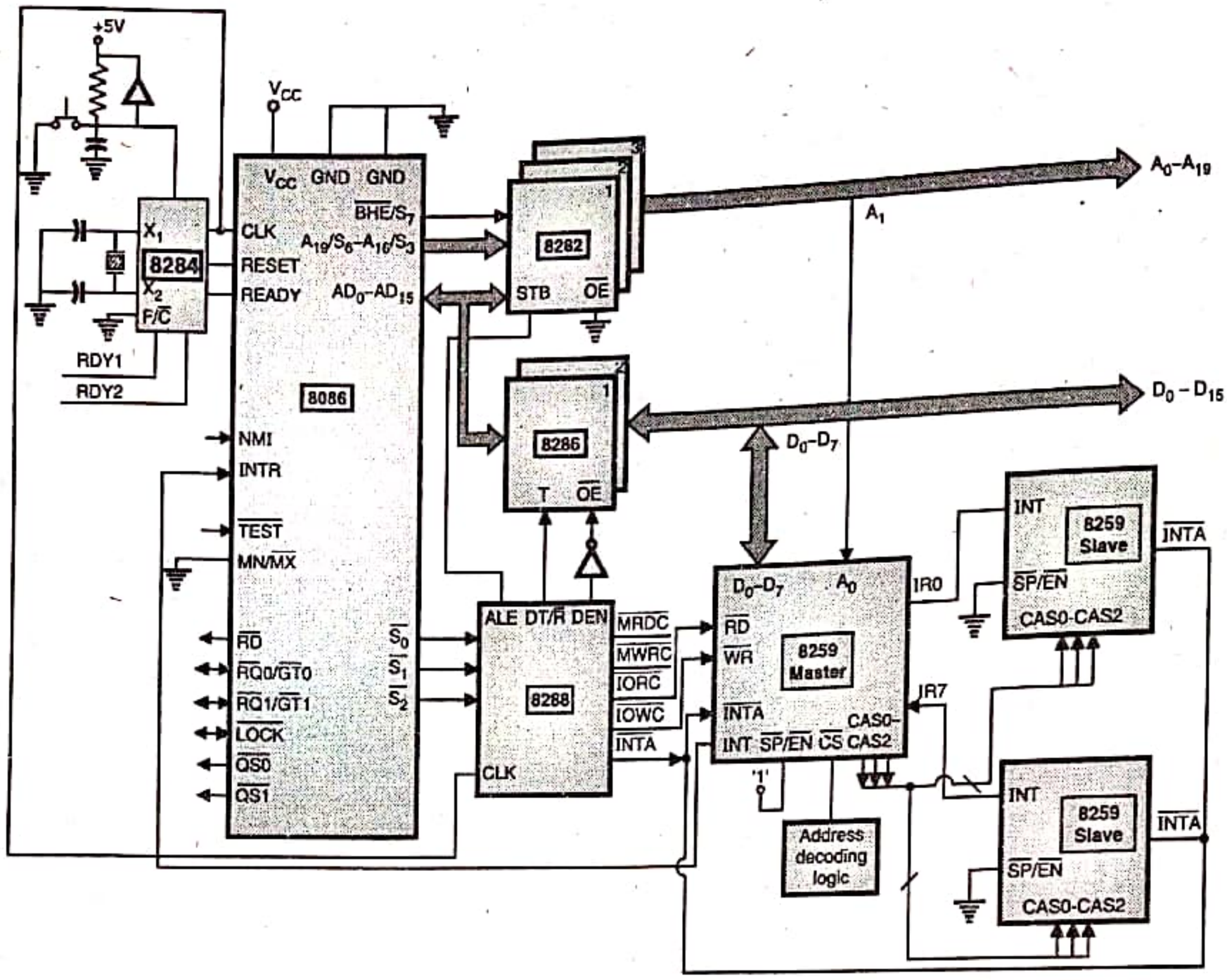


Fig. 8.9 : Interfacing 8259 with 8086 (8259 – cascaded, 8086 – maximum mode)

Chapter 9 : Multiprocessor Systems

Q. 1 Write short note on 8087 Math coprocessor. Dec. 15

Ans. :

1. Intel 8087 is a processor with architecture and instruction set optimized for performing complicated arithmetic operations.
2. An 8087 is along with the host microprocessor 8086, rather than serving as the main processor itself. Therefore, it is referred to as the coprocessor.
3. An 8087 instruction may perform a given mathematical computation 100 times faster than the equivalent sequence of 8086 instructions.
4. 8087 is an actual processor with its own, specialized instruction set. Instructions for 8087 are written in the program as needed, interspersed with 8086 instructions.
5. As the 8086 fetches instruction bytes from the memory and puts them in its queue, the 8087 also reads these instruction bytes and puts them in its internal queue. The 8087 decodes each instruction that comes into its queue.
6. When 8087 decodes an instruction from its queue and finds that it is an 8086 instruction, the 8087 simply treats the instruction as NOP. When the 8086 decodes an instruction from its queue and finds that it is an 8087 instruction, the 8086 simply treats the instruction as NOP or in some cases reads a data word from the memory for the 8087. Each processor decodes all instructions in the fetched instruction byte stream but executes only its own instructions.
7. All the 8087 instruction codes have 11011 (by the use of ESC prefix to that instruction) as the most significant bits of their first code byte, which helps the processors to differentiate the 8086 and the 8087 instructions.

Q. 2 Draw and explain interfacing of math coprocessor (8087) with 8086.

Ans. : Dec. 16

1. AD_{15} - AD_0 (Input/output lines) : (Address / Data)

These lines constitute the time multiplexed memory address and data bus. A_0 is analogous to the $\overline{BHE}$ for the lower byte of the data bus. These lines are active high. These line will be input lines for 8087 when CPU is having the control of the bus.

2. A_{19}/S_6 - A_{16}/S_3 (Input/output lines) : (Address / Status)

These lines are multiplexed address/status lines. Initially, address information will be available, after that status information will be available.

For 8087 controlled bus cycle.

S_6, S_4 and S_3 = Reserved (currently high)

S_5 = Always low

These lines will be monitored by 8087, when CPU is having the control of the bus.

3. $\overline{BHE}$ /S_7 - (Input/output line) : (Bus HIGH Enable)

This pin is used to enable data onto the most significant half of the data bus. i.e. D_8 to D_{15}.

$\overline{BHE}$ = 0 during read/write cycle when a byte is to be transferred on the high portion of the bus. S_7 information available during T_2, T_3, T_w and T_4.

4. $\overline{S}_2$,$\overline{S}_1$, $\overline{S}_0$ - (Input/output lines) : (status lines)

These lines are encoded as follows :

$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$	Operation
0	x	x	Unused
1	0	0	Unused
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

Status is driven during T_2 cycle, and is returned to passive status (1, 1, 1) during T_3 or during T_w.

5. $\overline{RQ0}$ / $\overline{GT0}$ (Input/output) : (Request/Grant)

This pin is used by 8087 to gain control of the local bus from the CPU for operand transfers or on behalf of another bus master. It must be connected to one of the two processor request/grant pins.

6. $\overline{RQ0}$ / $\overline{GT1}$ (Input/output) : (Request/Grant)

This request/grant pin is used by another local bus master to force the 8087 to request the local bus. If the 8087 is not in control of the bus, then it will make request through this line.

7. QS1, QS0 (Input) : (Queue status)

QS1, QS0 provide 8087 with status to allow tracking of the CPU instruction queue.

QS1	QS0	
0	0	No operation
0	1	First byte of opcode from queue
1	0	Empty the queue
1	1	Subsequent byte from the queue

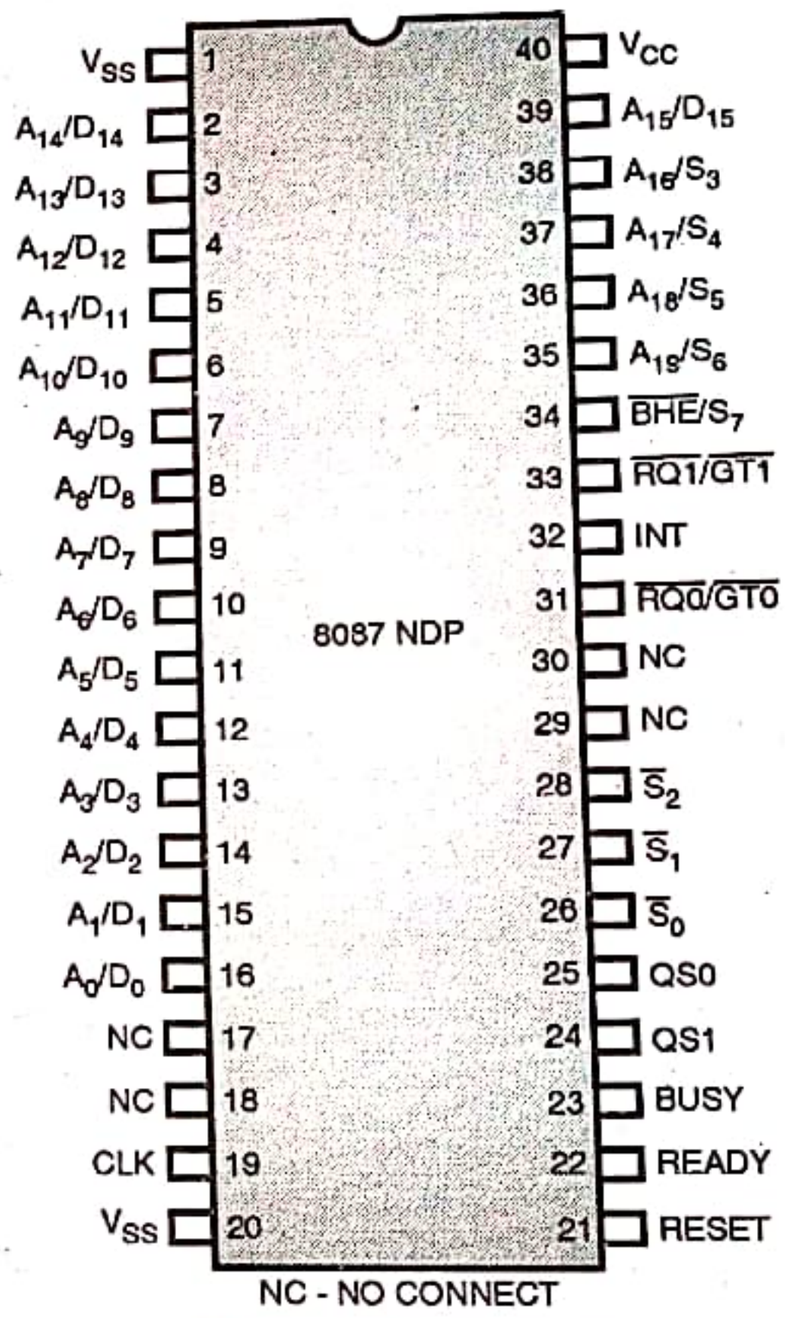


Fig. 9.1 : Pin configuration of NDP 8087

8. INT (Output) : (Interrupt)

This line is used to indicate that, an unmasked exception has occurred during numeric instruction execution, when 8087 interrupts are enabled. Normally this signal is routed through 8259 (PIC).

9. BUSY (Output) : (Busy)

This signal indicates that the 8087 NEU is executing numeric instruction. It is connected to the CPU's $\overline{TEST}$ pin to provide synchronization. Busy is active high.

10. Ready (Input) : (Ready)

Used for slower peripheral devices.

11. RESET (Input) : (Reset)

This causes processor to immediate terminate its present activity.

12. CLK (Input) : (clock)

Clock provides basic timing for processor and bus controller. It is asymmetrical with 33% duty cycle.

The Fig. 9.2 shows interfacing of 8086 with 8087.

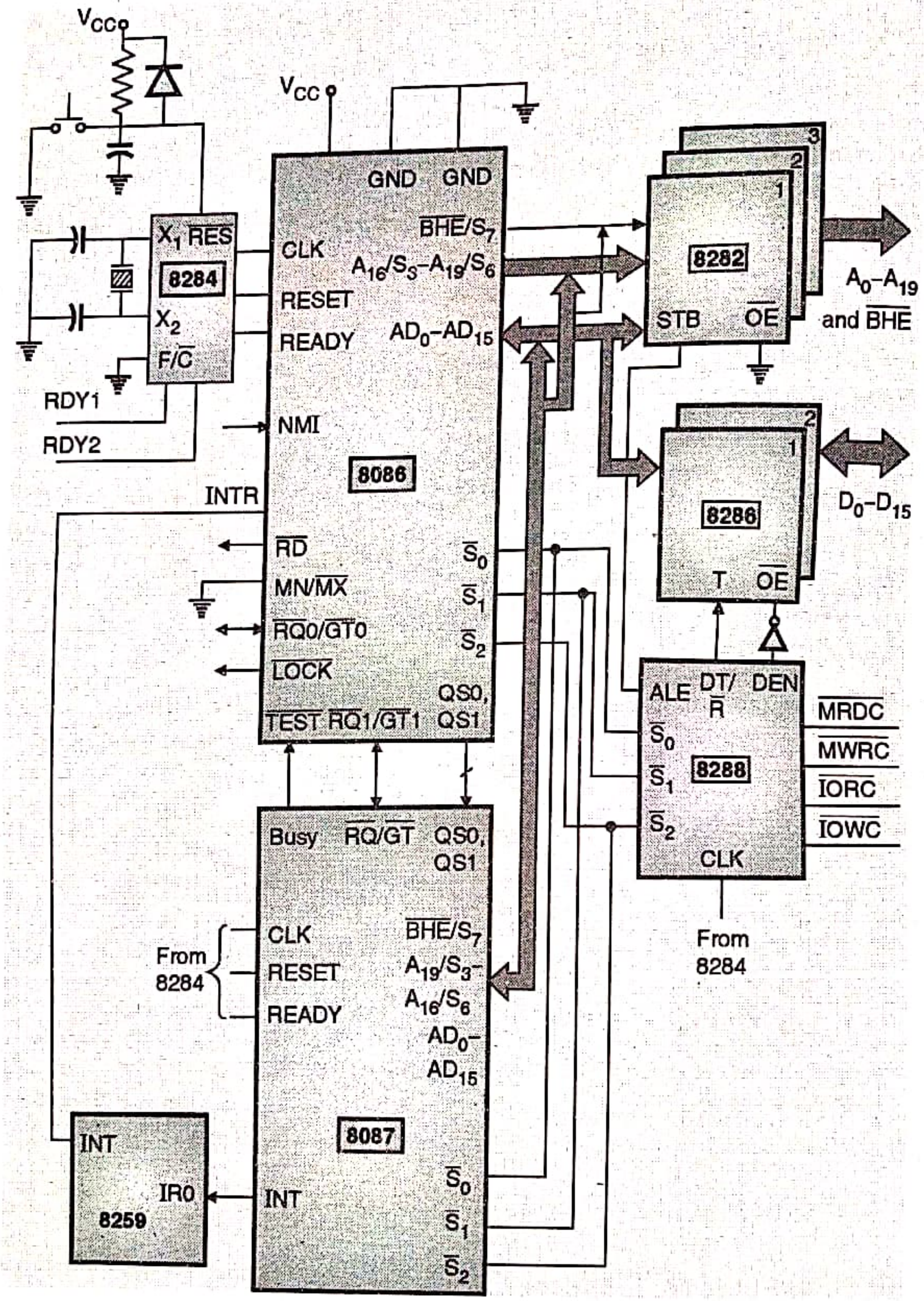


Fig. 9.2

Q. 3 Write short notes on 8089 I/O processor. May 16

Ans. : 8089 I/O processor

CLK, Reset and ready given to 8086/8088 as well as IOP 8089, $\overline{S_0}$ - $\overline{S_2}$ from 8086 given to 8089 as well as bus controller 8288. AD_0 - AD_{15} and A16/S3 to A19/S7 lines of 8086 given to 8282 latch for demultiplexing address and data bus. A_1 to A_{15} lines of 8086 also given to 8286 (data bus buffer). Output from 8282 is A_0 to A_{19} with $\overline{BHE}$ signal. A_1 to A_{15} lines given to address decoder for generating chip select for IOP. INT pin of IOP is given to 8259, for generating interrupt for 8086, whenever required.

$\overline{RQ}$ / $\overline{GT}$ of IOP 8089 connected to $\overline{RQ}$ / $\overline{GT}$ of microprocessor. When 8089 is directly connected to 8086/8088, the $\overline{RQ}$ / $\overline{GT}$ lines built into all these processors are used to arbitrate use of a local bus. An external processor sends a pulse to the CPU to request use of the bus. The CPU finishes its current bus cycle, if one is in progress, and sends a pulse to the processor to indicate that it has been granted the bus. When the external processor is finished with the bus, it sends a final pulse to the CPU, to indicate that it is releasing the bus.

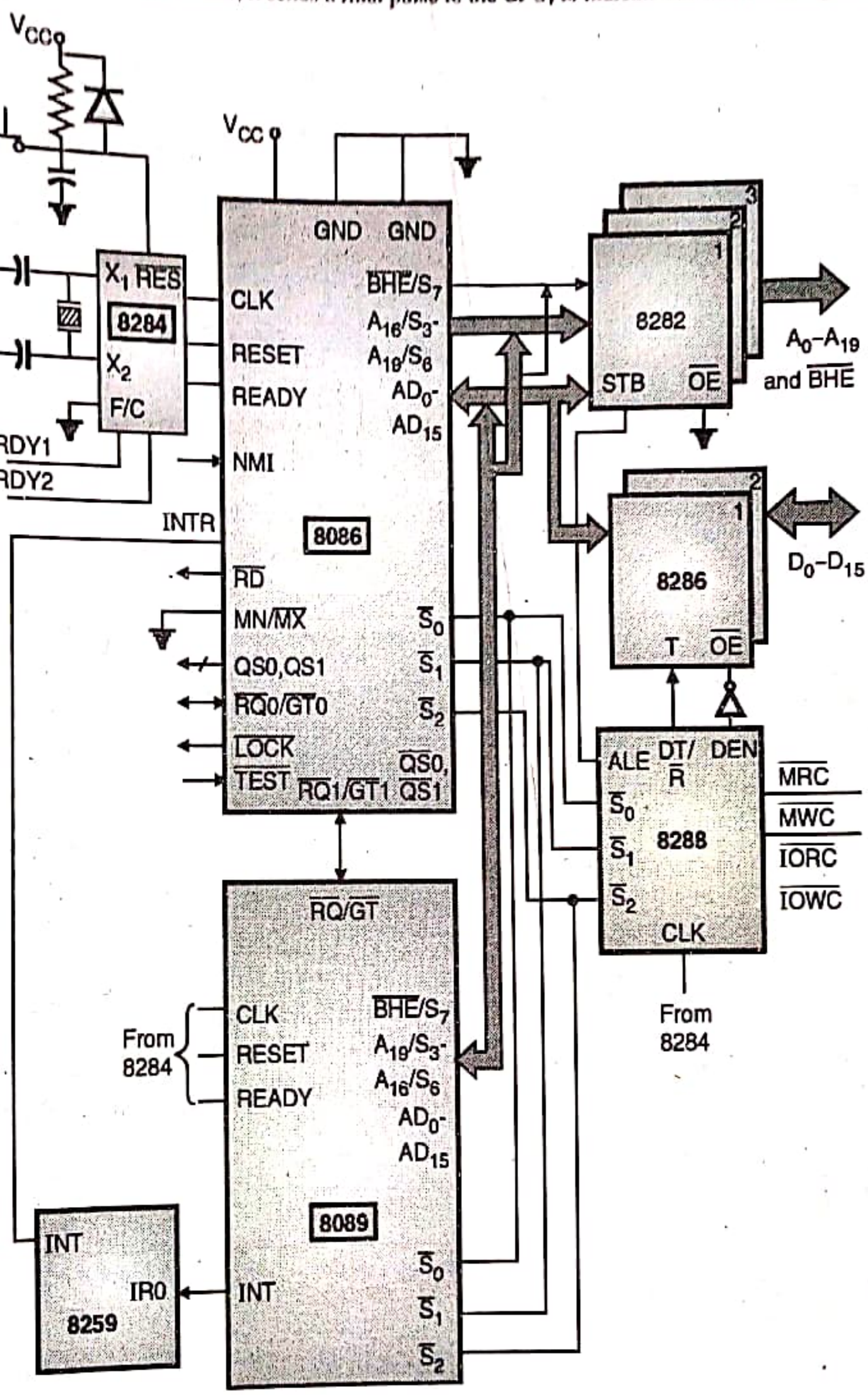


Fig. 9.3

(1) The host sets up message in memory and then wakes up the independent processor by sending a command to one of the independent processor's ports.

(2) The independent processor then accesses the shared memory to get the assigned task and executes the task in parallel with the host.

(3) After the task is completed, the external processor notifies its host of the completion by using either a status bit or an interrupt request.

(4) The message format, totally depends upon independent processor and the application. The message should specify which operation is to be performed, the input parameters and the addresses of the locations in which to store the result.

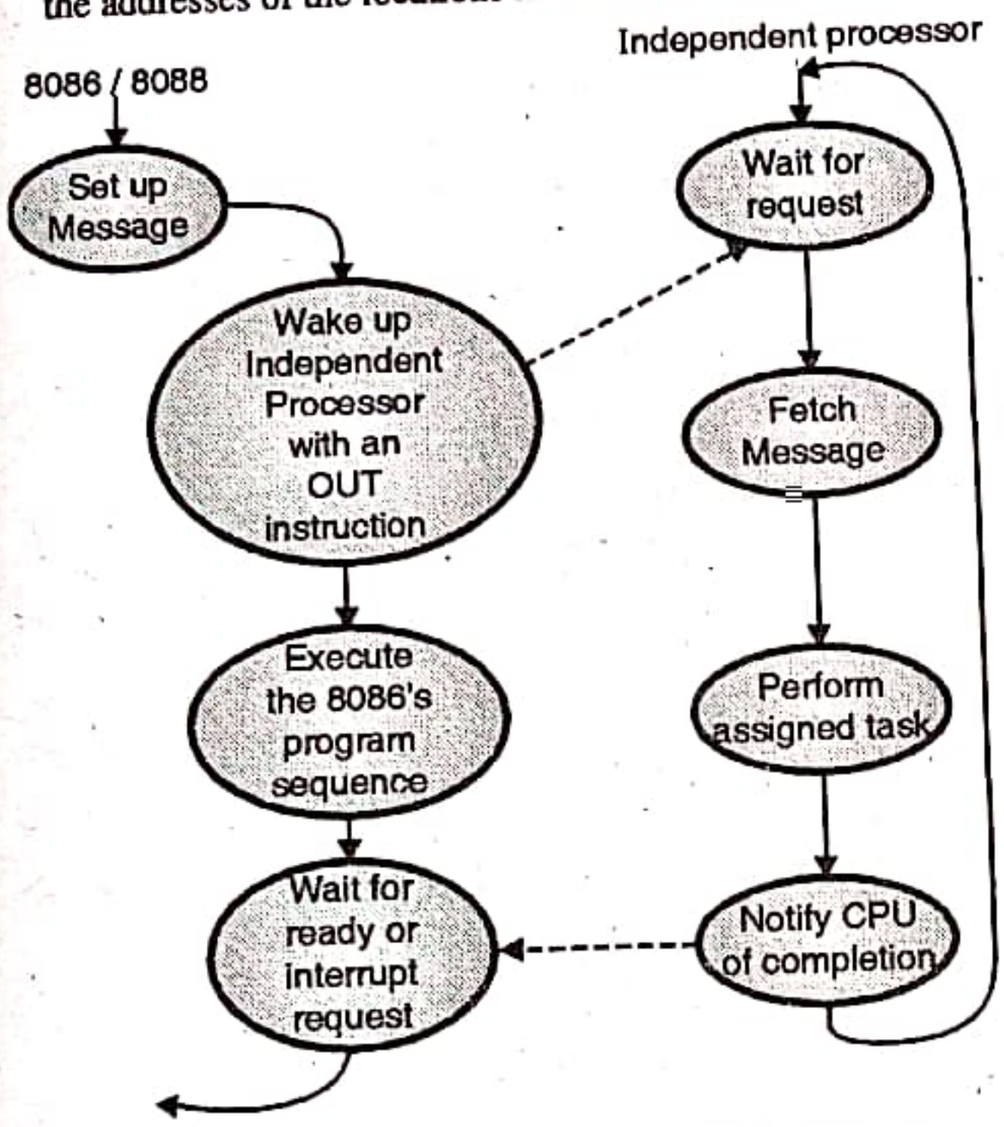


Fig. 9.4 : Interprocessor communication through shared memory

Q. 4 Explain the role of bus arbiter in loosely coupled systems. May 14

Ans. :

Each processor runs independently in a loosely coupled system. Also there is no direct connection between the processor. Interprocessor is achieved with the help of shared resources. If more than one processor grants access to the common resources at the same time bus contention occurs. In order to solve the problem of bus contention we use Bus arbitration. The different schemes of bus arbitration are :

1. Daisy chaining

Daisy chain method is characterized by its simplicity and low cost.

Mainly three buses are there :

(a) Bus request

(b) Bus grant

(c) Bus busy

Here Bus busy and Bus request is common. All masters use the same line for making bus requests. In response to bus request, the controller sends out a bus grant signal. Bus grant is activated if and only if bus busy signal is inactive. As shown in Fig. 9.5, bus grant serially propagates through each master until it gets first one master that has requested for the bus. Requested module receives bus grant signal, activates busy line, and gains control of the bus. Therefore any other module will not receive the grant signal.

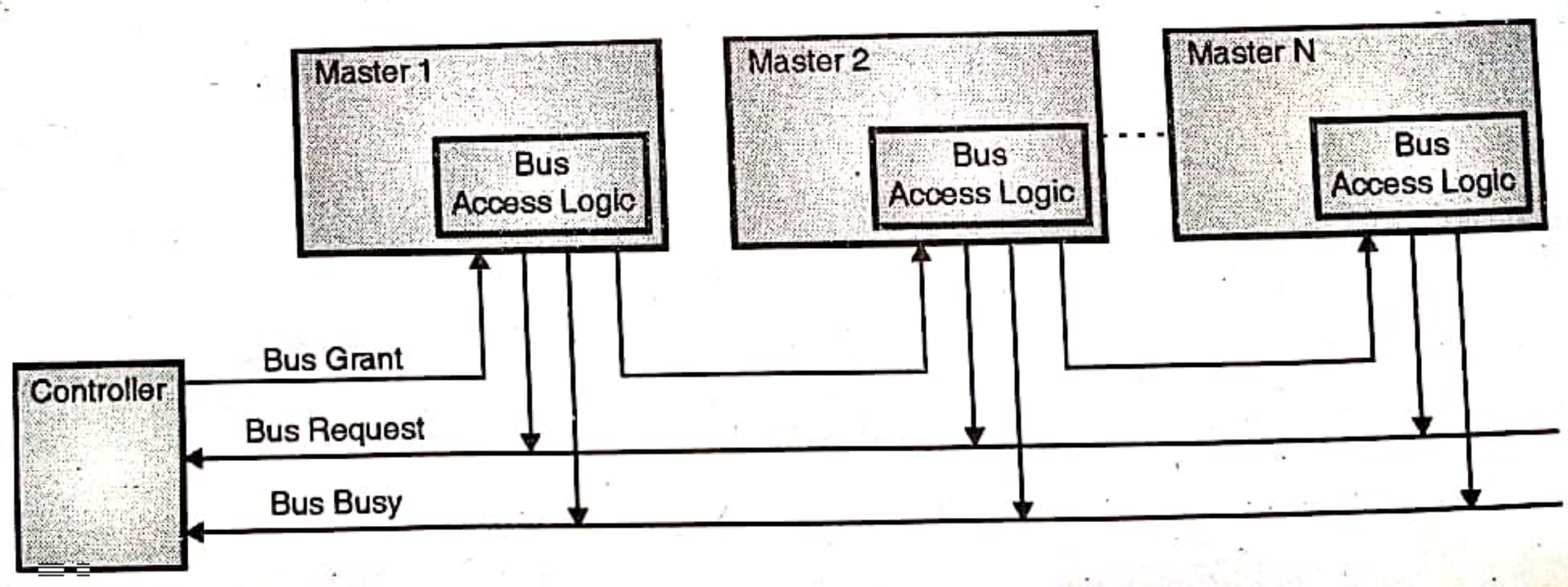


Fig. 9.5 : Daisy chaining

Advantages of daisy chaining

1. Compared to remaining two system, daisy chain requires less number of control lines. Number of control lines is independent of the number of modules in the system.
2. It is simple and cheap.

Disadvantages of daisy chaining

1. Propagation delay : If the number of master modules are more, the bus grant has to travel through all the master module and reach to last master module, which takes lot of time. The propagation delay depends upon number of modules. If user want fast system he/she has to limit number of modules.
2. If in between module fails because of some reason, the chain is broken. This particular aspect is not allowed in multiprocessing system.
3. The priority of master is fixed by its physical location.

2. Polling Method

Here bus request and bus busy lines are common to each module. In this scheme Address is assigned to each module.

When bus request is present, controller will generate sequence of module addresses. This address will be polled by each master module. If match occurs that particular module will activate busy line and begins to use bus.

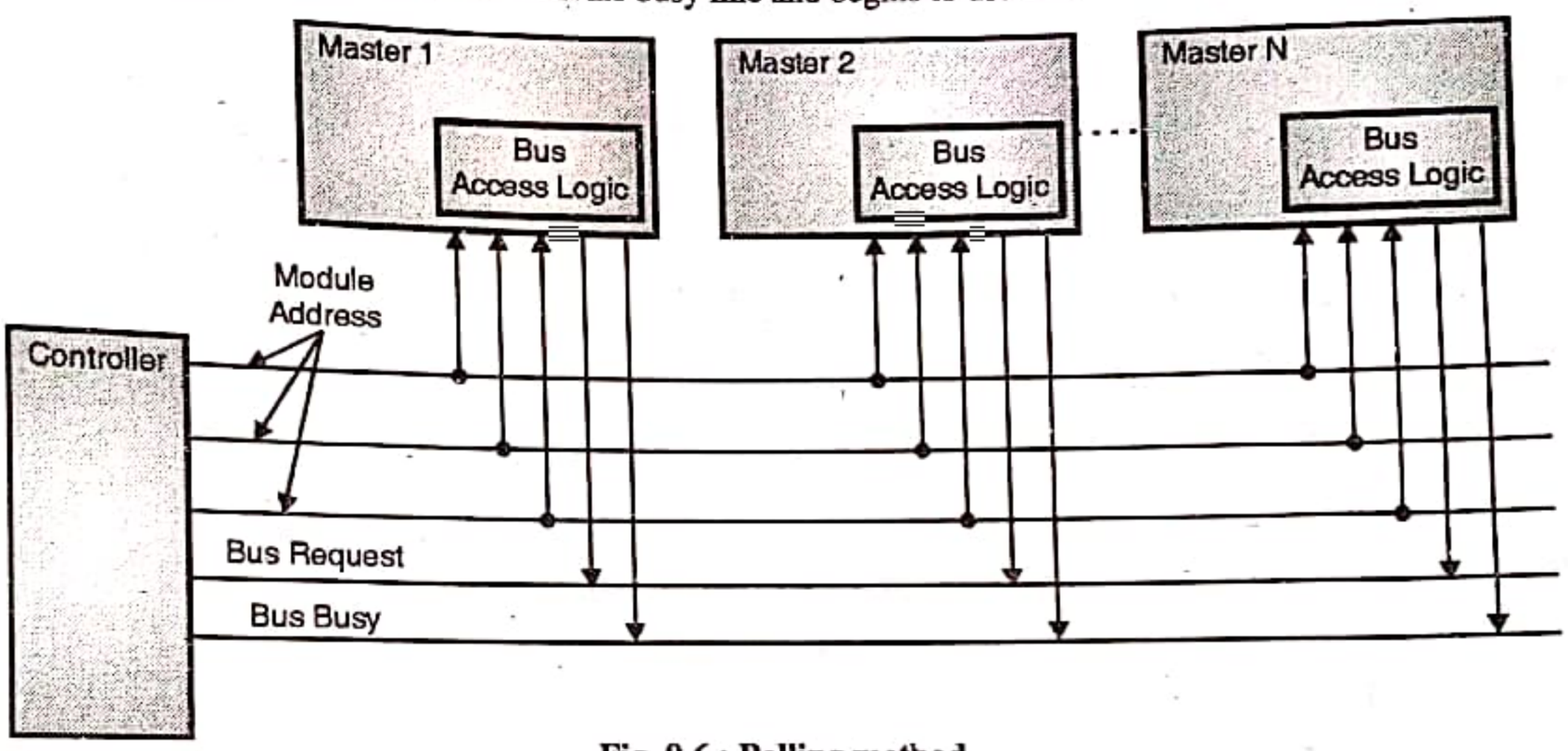


Fig. 9.6 : Polling method

Advantages of polling method

1. The priority can be changed by changing the polling sequence stored in the controller.
2. If one module fails the entire system does not fail.

Disadvantage of polling method

It requires decoding.

3. Independent Requesting

As shown in this scheme we have common bus busy signal. Bus grant signal and Bus request signals depends upon number of master modules. Each module has a separate pair of bus request and bus grant lines. At the same time each pair has been assigned priority. Internally controller has priority encoder, which selects the request with highest priority and returns corresponding bus grant signal.

Advantages of independent requesting

1. Arbitration is fast and is independent of the number of modules in the system.
2. In this scheme propagation delay is very less.

Disadvantage of independent requesting

The only disadvantage is number of bus request and bus grant lines.

Chapter 10 : Interfacing Memory to 8086

Q. 1 Design 8086 based minimum mode system for following requirements.

(a) 256 KB ROM using 32 KB × 8 devices.

(b) 512 KB RAM using 64 KB × 8 devices

(c) 2-NOs. 8-bit parallel port

(d) Support top 15 Interrupts. Dec. 13, Dec. 16

Ans. :

Step 1 : Total ROM required = 256 KB

Chip size available = 32 KB

∴ No of chips required = $\frac{256\text{ KB}}{32\text{ KB}} = 8$

∴ No of sets required = $\frac{8}{2} = 4$

Set 1 : Ending address = FFFFF H

Set size = chip size * 2 = 64 KB (0FFFFH)

Starting address = Ending address – set size = F0000H

	Even Bank	Odd Bank
Starting Address	F0000H	F0001H
Ending Address	FFFFEH	FFFFFH

Set 2 : Ending Address = Previous Starting – 1 = EFFFFH

Set size = 64 KB (0FFFFH)

Starting Address = EFFFFH – 0FFFFH = E0000H

	Even Bank	Odd Bank
Starting Address	E0000H	E0001H
Ending Address	EFFFEH	EFFFFH

Set 3 : Ending Address = DFFFFH

Set size = 64 KB (0FFFFH)

Starting address = DFFFFH – 0FFFFH = D0000H

	Even Bank	Odd Bank
Starting Address	D0000H	D0001H
Ending Address	DFFFEH	DFFFFH

Set 4 : Ending Address = CFFFFH

Set size = 64 KB (0FFFFH)

Starting Address = CFFFFH – 0FFFFH = C0000H

	Even Bank	Odd Bank
Starting Address	C0000H	C0001H
Ending Address	CFFFEH	CFFFFH

Step 2 : Total RAM required = 512 KB

Chip size available = 64 KB

∴ No. of chips required = $\frac{512\text{ KB}}{64\text{ KB}} = 8$

∴ No. of sets required = $\frac{8}{2} = 4$

Set 1 : Starting Address = 00000H

Set size = chip size * 2 = 128 KB (1FFFFH)

Ending Address = Starting Address + set size = 00000H + 1FFFFH = 1FFFFH

	Even bank	Odd bank
Starting Address	00000H	00001H
Ending Address	1FFFEH	1FFFFH

Set 2 :

Starting Address = Previous ending + 1 = 20000H

Set size = 128 KB (1FFFFH)

Ending address = 20000H + 1FFFFH = 3FFFFH

	Even bank	Odd bank
Starting Address	20000H	20001H
Ending Address	3FFFEH	3FFFFH

Set 3 :

Starting Address = 40000H

Set size = 128 KB (1FFFFH)

∴ Ending Address = 40000H + 1FFFFH = 5FFFFH

	Even bank	Odd bank
Starting Address	40000H	40001H
Ending Address	5FFFEH	5FFFFH

Set 4 :

Starting Address = 60000H

Set Size = 128 KB (1FFFFH)

∴ Ending Address = 60000H + 1FFFFH = 7FFFFH.

	Even bank	Odd bank
Starting Address	60000H	60001H
Ending Address	7FFFEH	7FFFFH

Step 3 : Memory Map

Table 10.1

			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 $\overline{y}_0 \cdot \overline{y}_1$	EB	SA = 00000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		EA = 1FFFEH					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	OB	SA = 00001H	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		EA = 1FFFFH					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM Set 2 $\overline{y}_2 \cdot \overline{y}_3$	EB	SA = 20000H	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		EA = 3FFFEH					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	OB	SA = 20001H	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		EA = 3FFFFH					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM Set 3 $\overline{y}_4 \cdot \overline{y}_5$	EB	SA = 40000H	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		EA = 5FFFEH					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	OB	SA = 40001H	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		EA = 5FFFFH					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM Set 4 $\overline{y}_6 \cdot \overline{y}_7$	EB	SA = 60000H	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		EA = 7FFFEH					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	OB	SA = 60001H	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		EA = 7FFFFH					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EPROM Set 4 $\overline{y}_{12}$	EB	SA = C0000H	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		EA = CFFFEH					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	OB	SA = C0001H	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		EA = CFFFFH					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EPROM Set 3 $\overline{y}_{13}$	EB	SA = D0000H	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	OB	EA = DFFFEH					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
		SA = D0001H	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		EA = DFFFFH					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EPROM Set 2 $\overline{y}_{14}$	EB	SA = E0000H	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		EA = EFFFEH					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	OB	SA = E0001H	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		EA = EFFFFH					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EPROM Set 1 $\overline{y}_{15}$	EB	SA = F0000H	1	1	1	1	0	0	0	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	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 = FFFFFH					1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Step 4 : *I/O Map*

For 2 nos., 8-bit parallel port we need one chip of 8255.

For 15 interrupts we need two 8259, one as a master and other as a slave.

		A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
8255	PA = 00H	0	0	0	0	0	0	0	0
Even bank	PB = 02H	0	0	0	0	0	0	1	0
A_7	PC = 04H	0	0	0	0	0	1	0	0
	CW = 06H	0	0	0	0	0	1	1	0
8259 Even bank	Addr 1 = 80 H	1	0	0	0	0	0	0	0
	Addr 2= 82H	1	0	0	0	0	0	1	0
Odd bank	Addr 1 = 81H	1	0	0	0	0	0	0	1
$\overline{A_7}$	Addr 2 = 83 H	1	0	0	0	0	0	1	1

Step 5 : Final Implementation :

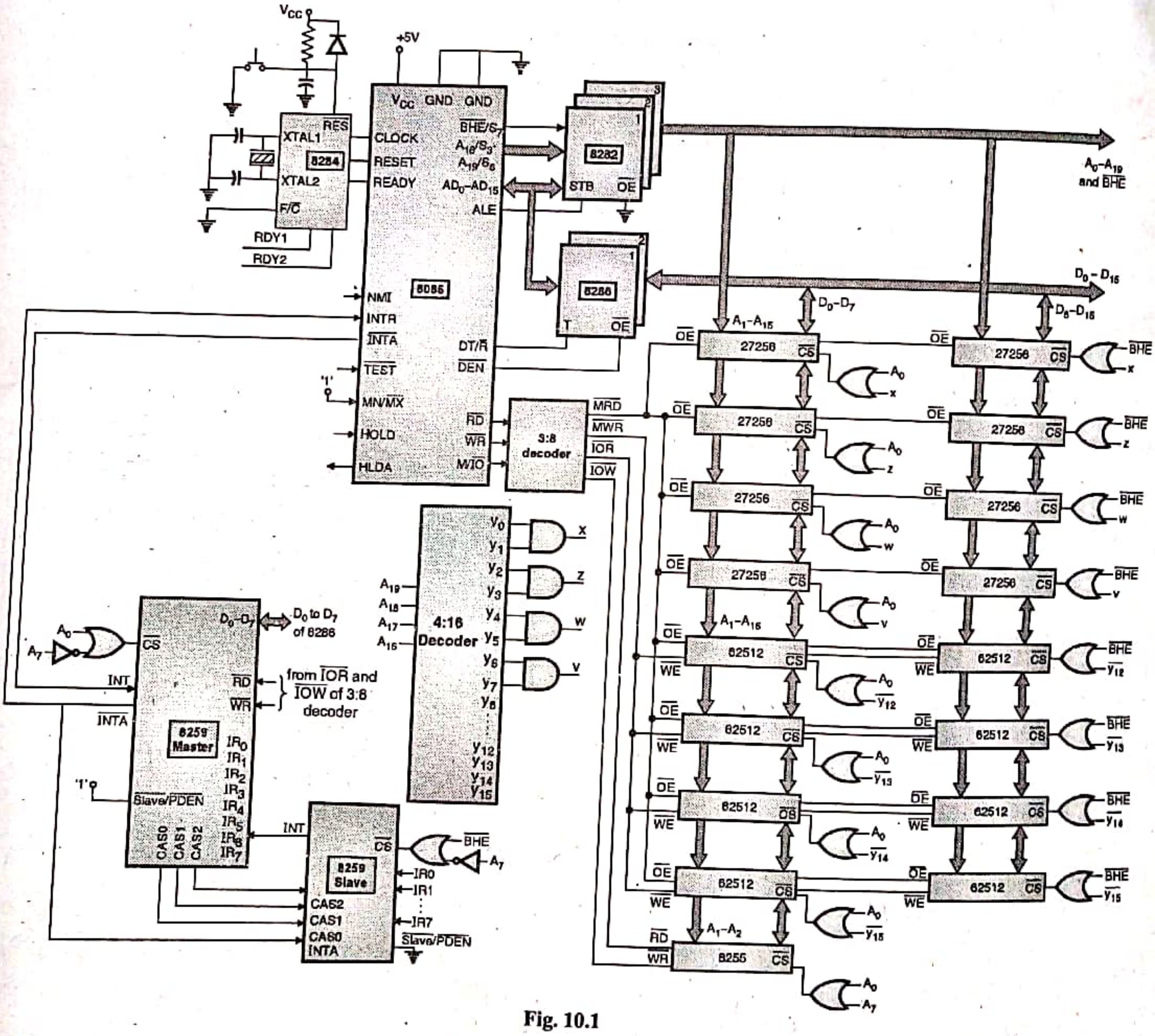


Fig. 10.1

Q. 2 Differentiate between the Memory mapped I/O and I/O mapped I/O.

Ans. : Dec. 11, May 12, Dec. 12

Difference between memory mapped I/O and I/O mapped I/O

	Memory mapped I/O	I/O mapped I/O
1.	A memory address is allotted to an I/O device.	An I/O address is given to an I/O device.
2.	Any memory related instruction will be able to access this I/O device. e.g. : MOV BX, [3500 H]	Only IN and OUT instruction can access such devices.
3.	$\overline{MRDC}$ or $\overline{MRD}$ and $\overline{MWR}$ or $\overline{MWRC}$ are given to $\overline{RD}$ and $\overline{WR}$ of I/O device	$\overline{IORC}$ / $\overline{IORD}$ and $\overline{IOWC}$/ $\overline{IOWR}$ are given to $\overline{RD}$ and $\overline{WR}$ of I/O device
4.	The data from I/O device can also be given to ALU and result given back to I/O device using single instruction e.g. : ADD [3500 H], CX	ALU operations are not possible directly on I/O data. They are to be first brought into accumulator.

Q. 3 Design 8086 based minimum mode system for following requirements:

I. **256 KB of RAM using 64 KB x 8-bit device**
II. **128 KB of RAM using 64 KB x 8-bit device**
III. **Three 8-bit parallel ports using 8255**
IV. **Support for 8 interrupts** May 16

Ans. :

Step 1 : Total EPROM required is equal to 128 KB

Chip size available = 64 KB

$\therefore$ Number of chips required $= \frac{128\ KB}{64\ KB} = 2$

$\therefore$ Number of sets required $= \frac{No \cdot of\ Chips}{No \cdot of\ Banks} = \frac{2}{2} = 1$

SET 1 : Ending address = FFFFFH

Set size = chip size × 2 = 64 KB × 2 = 128 KB

128 KB = 2^{17} = 0001 1111, 1111, 1111, 1111 = 1FFFF

Starting address = Ending address – SET size.

= FFFFFH – 1FFFFH = E0000H

		Even Bank	Odd Bank
ROM SET 1	Starting Address	E0000H	E0001H
	Ending Address	FFFFEH	FFFFFH

Step 2 : Total RAM required is 256 KB

Chip size available = 64 KB

$\therefore$ Number of chips required = $\frac{256}{64} = 4$

Number of sets required $= \frac{no \cdot of\ chips}{no \cdot of\ banks} = \frac{4}{2} = 2$

SET 1 : Starting address = 00000 H

Set size = Chip size × 2 = 64K × 2 = 128 KB

2^{17} = 128 KB $= \frac{0001}{1}\ \frac{1111}{F}\ \frac{1111}{F}\ \frac{1111}{F}\ \frac{1111}{F}$

Ending address = starting address + SET size.

= 00000H + 1FFFFH = 1FFFFH

SET 2 : Starting address = 20000H

Set size = chip size × 2 = 64 KB = 1FFFFH

Ending address = Starting address + set size

= 20000H + 1FFFFH = 3FFFFH

		Even bank	Odd bank
RAM SET 1	Starting Address	00000H	00001H
	Ending Address	1FFFEH	1FFFFH
RAM SET 2	Starting Address	20000H	20001H
	Ending Address	3FFFEH	3FFFFH

Step 3 : Memory Map

			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	EB	SA = 00000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		EA = 1FFFEH	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
$\overline{y_0}$	OB	SA = 00001H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		EA = 1FFFFH	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM Set-2	EB	SA = 20000H	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		EA = 3FFFEH	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
$\overline{y_1}$	OB	SA = 20001H	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		EA = 3FFFFH	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ROM Set-1	EB	SA = E0000H	1	1	1	0	0	0	0	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	1	1	0
$\overline{y_7}$	OB	SA = E0001H	1	1	1	0	0	0	0	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	1	1

Step 4 : I/O Map :

∴ We require 1 8255 chip to get 3, 8-bit port.

I/O Map

			A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
8255 Set 1	EB	PA = 00 H	0	0	0	0	0	0	0	0
		PB = 02 H	0	0	0	0	0	0	1	0
		PC = 04 H	0	0	0	0	0	1	0	0
		CW = 06 H	0	0	0	0	0	1	1	0

Step 5 : Final Implementation

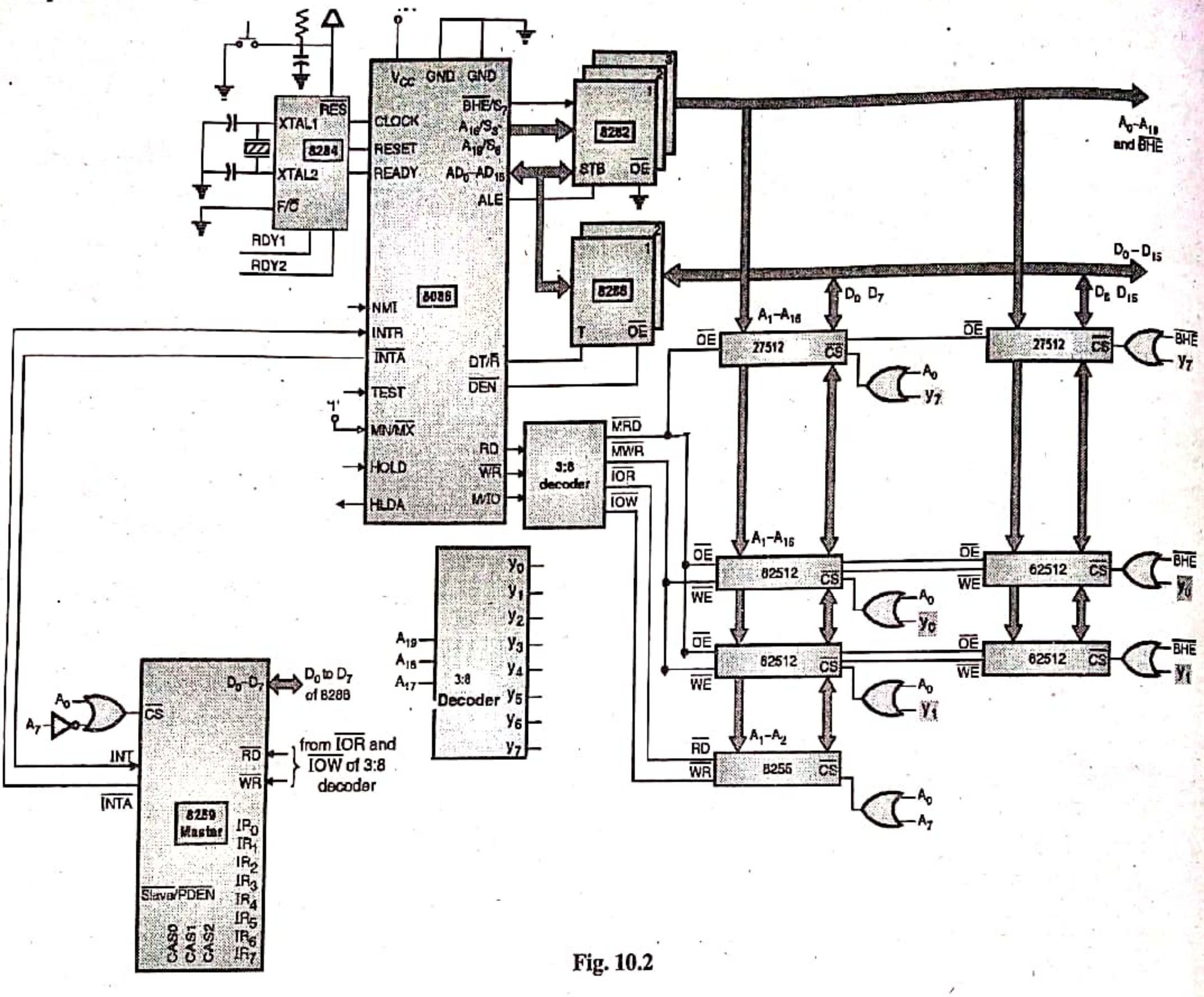


Fig. 10.2

Q. 4 Design a 8086 based microprocessor system with the following specifications :

A) 8086 is working at 8 MHz.

B) 32 KB EPROM using 16 KB devices.

C) 64 KB SRAM using 32 KB devices.

Explain the design and show memory map.

May 13, Dec. 15

Ans. : Total EPROM required = 32 KB.

Chip Size available = 16 KB.

∴ Total number of chips required = 2

∴ number of sets = $\frac{2}{2} = 1$

Step 1 : Ending address = FFFFFH

Set size = chip size ×2 = 32 KB ⇒ (07FFFH)

Starting address = FFFFFH – 07FFFH = F8000H

	Even bank	Odd Bank
Starting address	F8000 H	F8001 H
Ending address	FFFFE H	FFFFFH

Step 2 : Total RAM required = 64 KB.

Chip size available = 32 KB.

$\therefore$ number of chips = 2.

$\therefore$ number of sets = 1

SET 1 : Starting address = 00000H

Set size = 64 KB = 0FFFF H

Ending address = 00000H + 0FFFFH = 0FFFFH

	Even bank	Odd Bank
Starting address	00000 H	00001 H
Ending address	0FFFE H	0FFFF H

Step 3 : Memory Map :

Table 10.2

			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	EB	00000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\bar{y}_0 \cdot \bar{y}_1 \cdot$		0FFFEH	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	OB	00001H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		0FFFFH	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EPROM	EB	F8000H	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Set - 1 $\bar{y}_{31}$		FFFFEH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	OB	F8001H	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

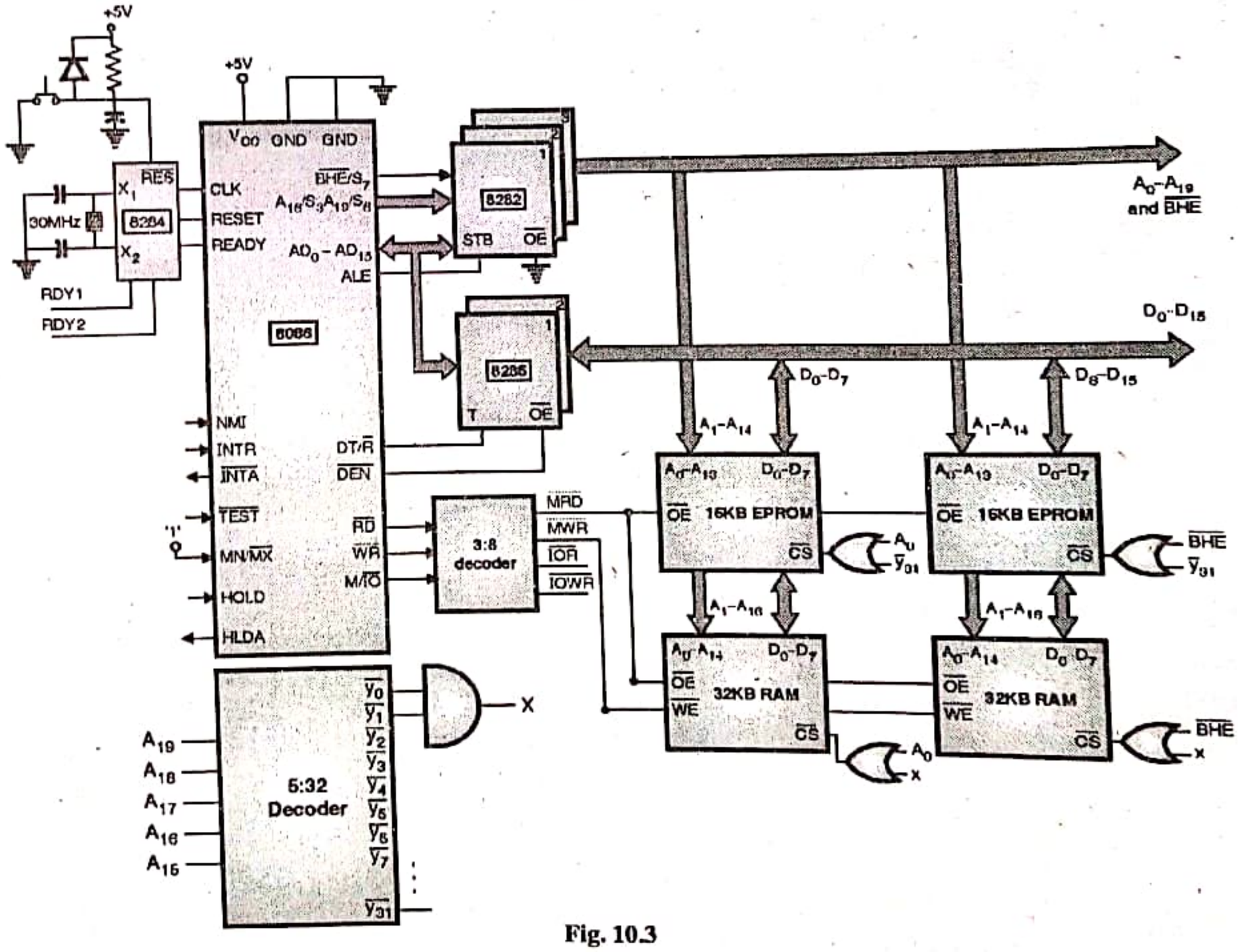


Fig. 10.3

Q. 5 Design 8086 based minimum mode system for following requirements :

I. 256 KB of RAM using 64 KB x 8-bit device

II. 128 KB of RAM using 64 KB x 8-bit device

III. Three 8-bit parallel ports using 8255

IV. Support for 8 interrupts May 16

Ans. :

Step 1 : Total EPROM required is equal to 128 KB

Chip size available = 64 KB

$\therefore$ Number of chips required = $\frac{129\ KB}{64\ KB} = 2$

$\therefore$ Number of sets required = $\frac{No \cdot of\ Chips}{No \cdot of\ Banks} = \frac{2}{2} = 1$

SET 1 : Ending address = FFFFFH

Set size = chip size × 2

= 64 KB × 2 = 128 KB

$128\ KB = 2^{17}$ = 0001 1111, 1111, 1111, 1111

= 1FFFF

Starting address = Ending address – SET size.

= FFFFFH – 1FFFFH

= E0000H

		Even Bank	Odd Bank
ROM SET 1	Starting Address	E0000H	E0001H
	Ending Address	FFFFEH	FFFFFH

Step 2 : Total RAM required is 256 KB

Chip size available = 64 KB

$\therefore$ Number of chips required = $\frac{256}{64} = 4$

Number of sets required = $\frac{no \cdot of\ chips}{no \cdot of\ banks} = \frac{4}{2} = 2$

SET 1 : Starting address = 00000 H

Set size = Chip size × 2 = 64K × 2 = 128 KB

2^{17} = 128 KB = $\frac{0001}{1}\ \frac{1111}{F}\ \frac{1111}{F}\ \frac{1111}{F}\ \frac{1111}{F}$

Ending address = starting address + SET size.

= 00000H + 1FFFFH = 1FFFFH

SET 2 : Starting address = 20000H

Set size = chip size × 2 = 64 KB = 1FFFFH

Ending address = Starting address + set size

= 20000H + 1FFFFH = 3FFFFH

		Even bank	Odd bank
RAM SET 1	Starting Address	00000H	00001H
	Ending Address	1FFFEH	1FFFFH
RAM SET 2	Starting Address	20000H	20001H
	Ending Address	3FFFEH	3FFFFH

Step 3 : Memory Map

			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 $\overline{y}_0$	EB	SA = 00000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		EA = 1FFFEH	0	0	0	1	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 = 1FFFFH	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM Set-2 $\overline{y}_1$	EB	SA = 20000H	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		EA = 3FFFEH	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	OB	SA = 20001H	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		EA = 3FFFFH	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ROM Set-1 $\overline{y}_7$	EB	SA = E0000H	1	1	1	0	0	0	0	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	1	1	0
	OB	SA = E0001H	1	1	1	0	0	0	0	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	1	1

Step 4 : I/O Map :

∴ We require 1 8255 chip to get 3, 8-bit port.

I/O Map

			A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
8255 Set 1	EB									
		PA = 00 H	0	0	0	0	0	0	0	0
		PB = 02 H	0	0	0	0	0	0	1	0
		PC = 04 H	0	0	0	0	0	1	0	0
		CW = 06 H	0	0	0	0	0	1	1	0

Step 5 : Final Implementation

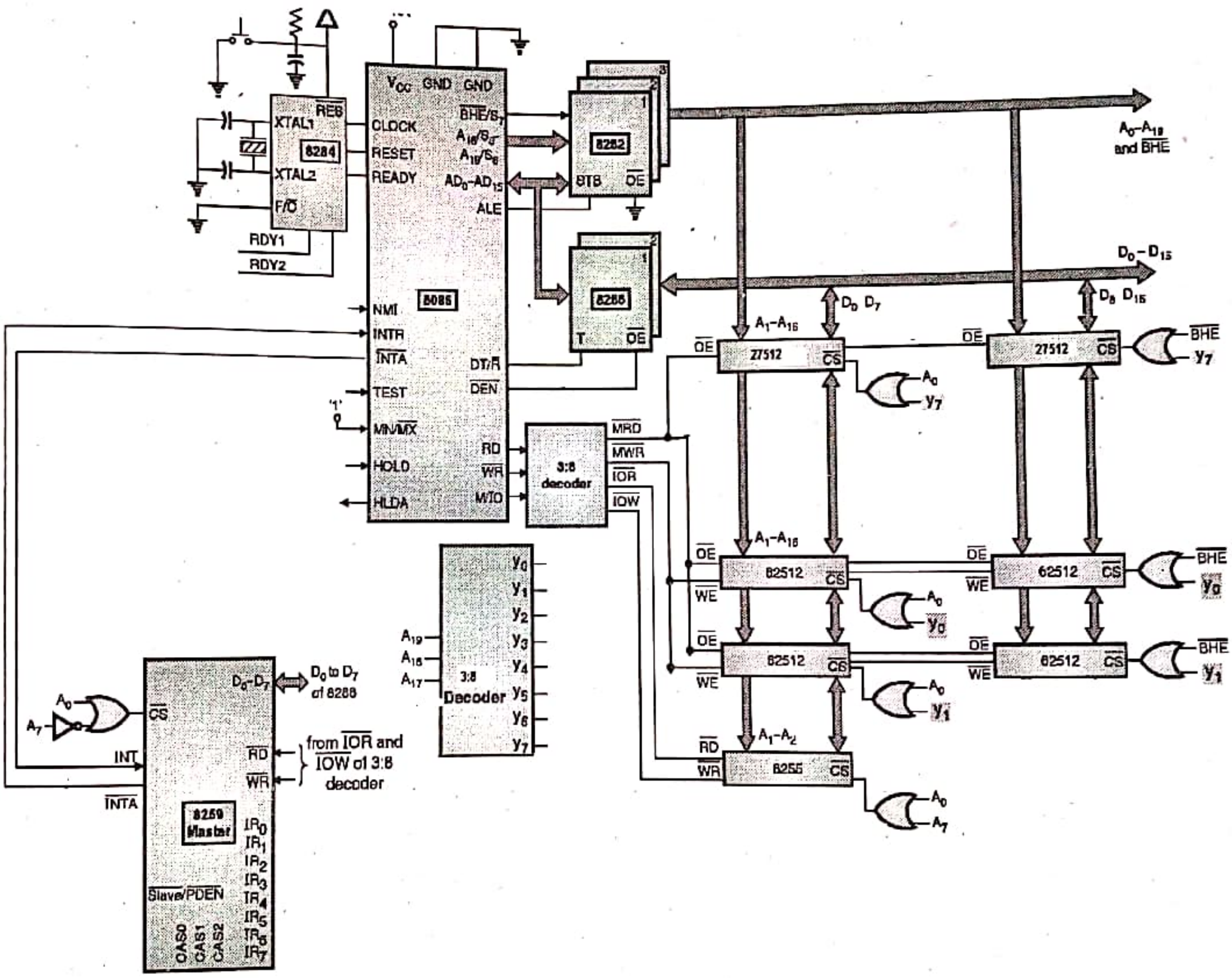


Fig. 10.4

Chapter 11 : 8255 Programmable Peripheral Interface

Q. 1 Drew and explain architecture of 8255.

Dec. 13, Dec. 15, May 16

Ans. :

The block diagram of 8255 is as shown in Fig. 11.1.

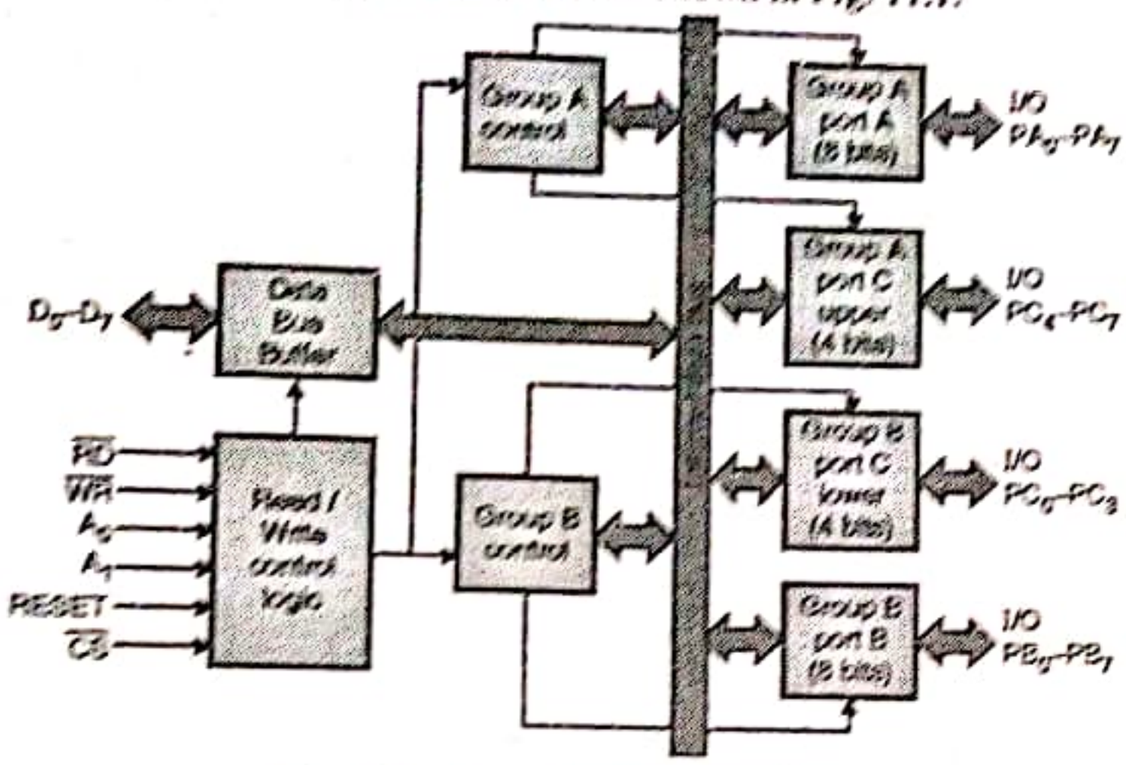


Fig. 11.1 : Block diagram of 8255

1. Data Bus Buffer

The 8 bit bi-directional, tristate data bus buffer is used to interface 8255 internal data bus with system data bus. The direction of data buffer is decided by read and write control signals. When read is activated, it transmits data to the system data bus. When write is activated, it receives data from system data bus.

2. Read / Write Control Logic

This block accepts inputs from system control bus and address bus and performs operations as shown in Fig. 11.1. The control signals are $\overline{RD}$ and $\overline{WR}$ and address signals used are A_0 and A_1 and $\overline{CS}$. The signals $\overline{RD}$ and $\overline{WR}$ are connected to $\overline{IOR}$, $\overline{IOW}$ or $\overline{MEMR}$, $\overline{MEMW}$. A_1 and A_2 of 8086 are directly connected to address lines A_0 and A_1 of 8255. $\overline{CS}$ is connected to address chip select decoder. The 8255 operation / selection is enabled/disabled by $\overline{CS}$ signal.

3. Group A and Group B Control

The 8255 I/O ports are divided into 2 sections Group A (GA) and Group B (GB). Group A consists of port A and port C upper. Group B consists of port B and port C lower. Each group is programmed through software. The GA and GB control block receives commands from the R/W control logic to accept bit pattern from CPU. GA control will control GA ports and GB control will control GB ports. The bit pattern given by CPU consists of information (i) To control the operation of GA and GB (ii) The mode in which they should be operated.

4. Port A and Port B

The port A and port B consists of 8 bit bi-directional data output latch/ buffer and 8 bit data input buffer. The function of port A and B is decided by control bit pattern available in GA and GB control. The function of ports A and B are also dependent on mode of operation.

5. Port C

The port C consists of 8 bit bi-directional data output latch/buffer and 8 bit data input buffer. It is divided into 2 sections, port C upper PC_U and port C lower PC_L These two sections can be programmed and used separately as a 4 bit I/O ports. The port C function is dependent on mode of operation.

It can be used as : (i) simple I/O (ii) handshake signals (iii) status signal inputs. For handshake signals and status signals it is used in co-ordination with port A and port B. The direct bit set/reset capability is provided by port C only.

Q. 2 Discuss control word format for Bit Set Reset (BSR) mode of 8255 PPI. **Dec. 17**

Ans. :

The BSR mode is a port C bit set/reset mode. The individual bit of port C can be set or reset by writing control word in the control register. The control word format of BSR mode is as shown in Fig. 11.2.

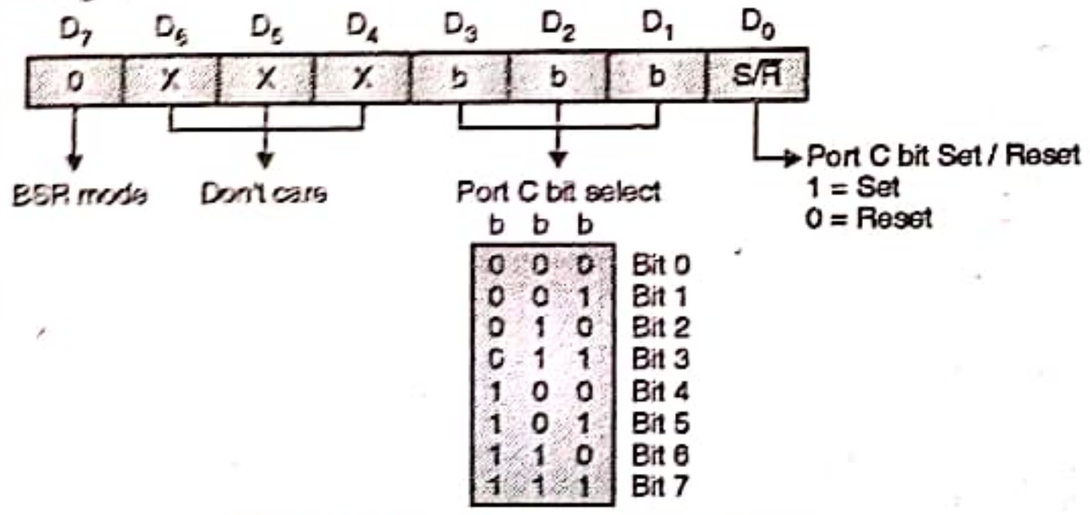


Fig. 11.2 : BSR control word format

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

Q. 3 What are the various modes of operation of 8255 PPI ? **May 12**

Ans. : Operation modes of 8255 PPI

1. Mode 0 (Simple Input / Output Mode)

In Mode 0, all ports i.e. port A, port B, port C provide simple input or output operation separately. The data is simply read from a

port or it is simply written to a port. In Mode 0 there is no restriction between function of ports. The port C, 2 sections port C upper and port C lower can be individually programmed as 4 bit ports.

(a) Mode 0 - Input mode

The 8255 is initialized in input mode by using control register. When CPU wants to read data from an input port, the CPU will first send address of port on address lines so $\overline{CS}$, A_0 and A_1 will select the appropriate port. After selecting a port CPU will send a control signal $\overline{RD}$ to read data from external peripheral through port and data bus. The timing waveform for Mode 0 input mode is as shown in Fig. 11.3.

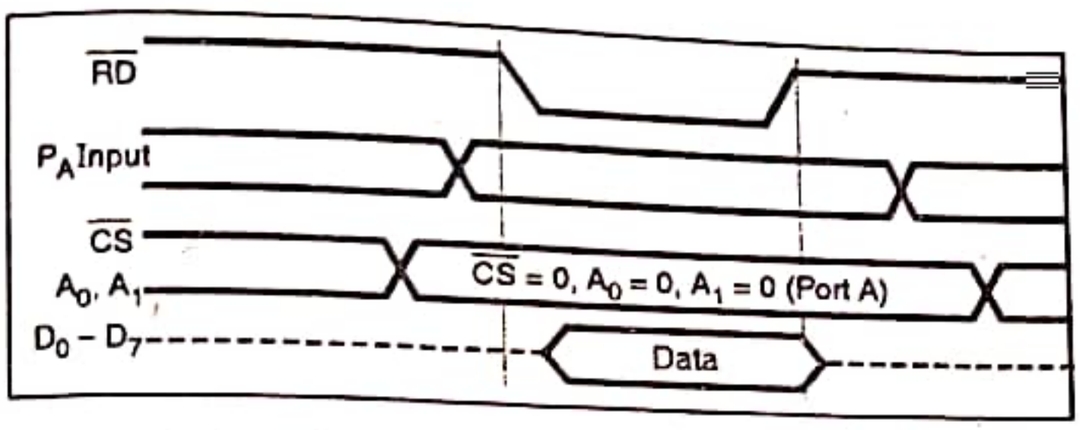


Fig. 11.3 : Timing waveforms for mode 0 input mode

(b) Mode 0 - Output mode

The 8255 is initialized in output mode by using control register. When CPU wants to send data to an output port, the CPU will first send address of port on address lines so $\overline{CS}$, A_0 and A_1 will select the appropriate port. After selecting a port CPU will send data and control signal $\overline{WR}$ to write data to port through data bus.

As the port is in output mode the contents will get latched in the port. The timing waveform for Mode 0 output mode is as shown in Fig. 11.4.

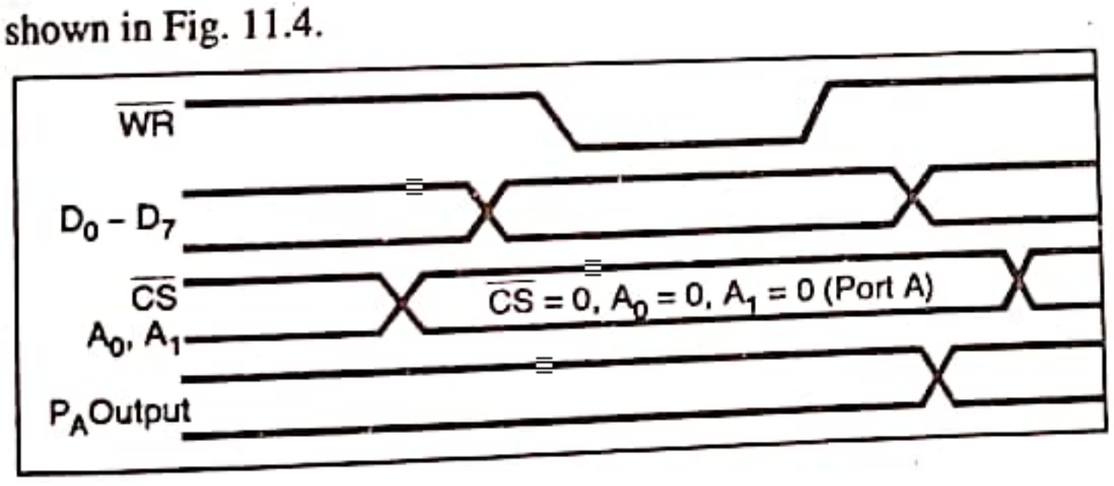


Fig. 11.4 : Timing waveforms for mode 0 output mode

2. Mode 1 (Strobed I/O)

Mode 1 - Input Mode

In Mode 1 input mode port A uses PC_3, PC_4, PC_5 and port B uses PC_0, PC_1, PC_2 signals as handshake signals. The port A, port B and handshake signals are interfaced with peripheral as shown in Fig. 11.5. The aim of interfacing is to transfer data from peripheral to CPU through 8255. The Fig. 11.5 shows interfacing I/P port to PA, similarly it can be done with PB.

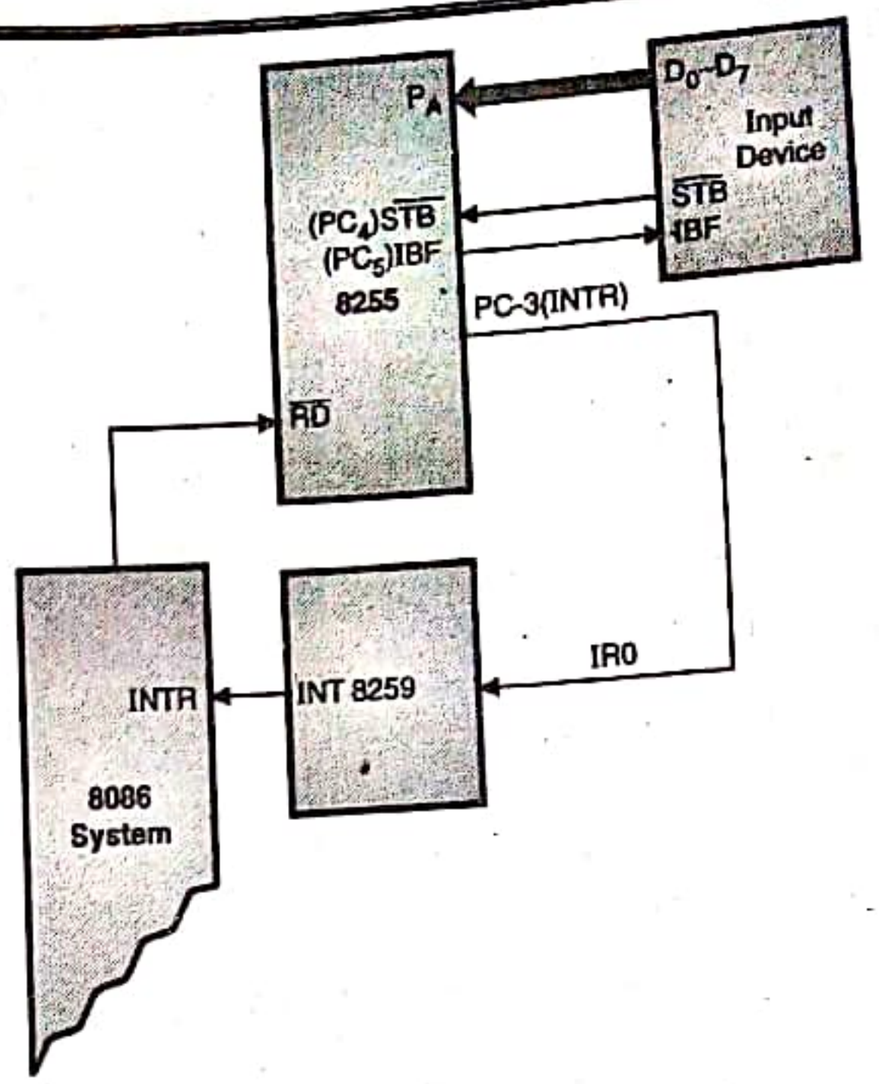


Fig. 11.5 : Mode 1 : Input mode interfacing

Mode 1 - Output Mode

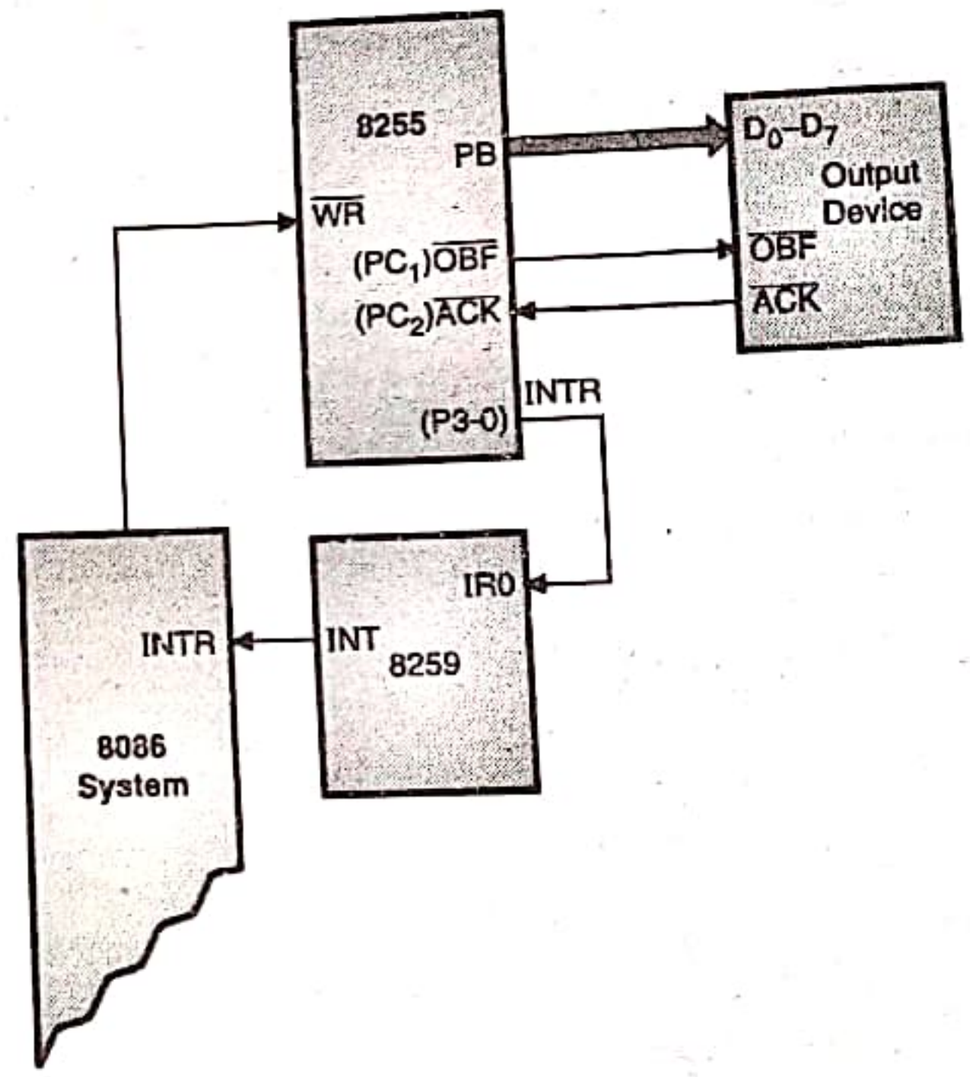


Fig. 11.6 : Mode 1 : Output mode interfacing

In Mode 1 output mode port A uses PC_3, PC_6 and PC_7 and port B uses PC_0, PC_1, PC_2 signals as handshake signals.

The port A, port B and handshake signals are interfaced with peripheral as shown in Fig. 11.7. The aim of interfacing is to transfer data from CPU to peripheral through 8255.

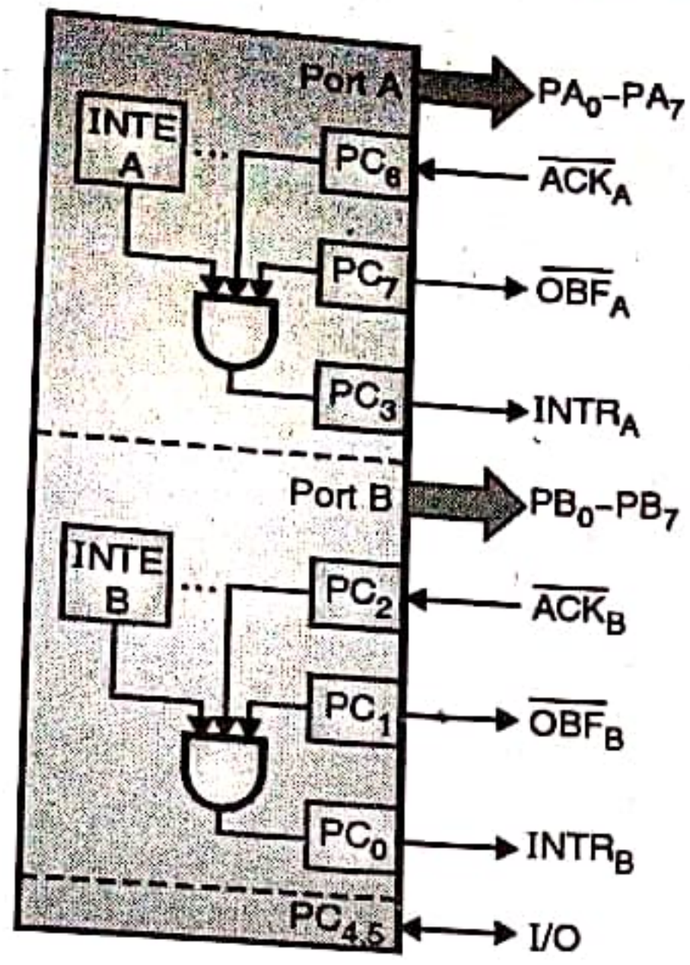


Fig. 11.7 : P_A, P_B and P_C in mode 1 output mode

Q. 4 Draw and explain the 4 × 4 keyboard interface using 8255. May 14

Ans. :

There are two software methods of identifying a closed key viz.

(1) Row-scanning technique and

(2) Line reversal technique e.g. Hexadecimal keyboard consists of 16 keys. The keys are arranged as 4 × 4 matrix, hence the number of input/output lines required to connect this keyboard are 4 + 4 = 8. Fig. 11.8 shows a connection of hexadecimal key board with 8255.

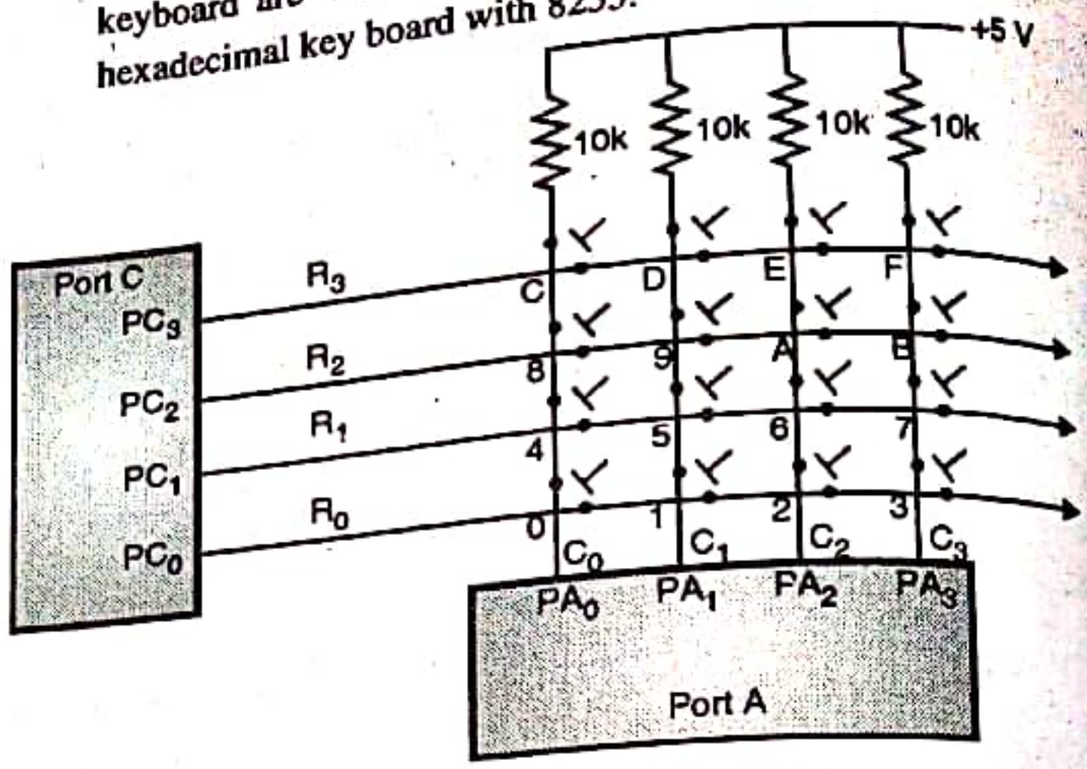


Fig. 11.8 : Keyboard matrix

Row Scanning Technique

The row lines are configured in output mode and column lines in input mode. The key closure at any row can be detected by outputting '0' on the corresponding port line and '1' on the rest of row port lines. The port A data is read and if the data contains all 1's, no key of the current row is pressed. The pressed key is determined by row number and column number.

Chapter 12 : 8253 / 8254 Programmable Interval Timer

Q. 1 List features of 8253. Dec. 17

Ans. :

The features of 8253 are as follows :

1. Three independent 16 bit down counters.
2. Counters can be programmed in 6 different programmable counter modes.
3. Counting facility in both binary or BCD number system.
4. Compatible with Intel and other microprocessors.
5. Single + 5V supply.
6. 24 pin dual in-line package.
8. It is completely TTL compatible.
9. It has a powerful command called READ BACK COMMAND which allows the user to check the count value, programmed mode and current mode and the current status of the counter (Only for 8254).
10. Operating frequency range; For 8253 - DC to 2.6 MHz; For 8254 - DC to 10 MHz.

Q. 2 Write short notes on 8253 programming interrupt timer. Dec. 13

Ans. :

The pin configuration of 8254 programmable interval timer is as shown in Fig. 12.1.

$D_7 - D_0$	I/O	data bus
CLK_N	I	Counter inputs
$GATE_N$	I	Counter gate inputs
OUT_N	O	Counter outputs
$\overline{RD}$	I	Read
$\overline{WR}$	I	Write
$\overline{CS}$	I	Chip select
$A_0 - A_1$	I	Counter select
V_{CC} / GND	I	+ 5 V supply/Ground

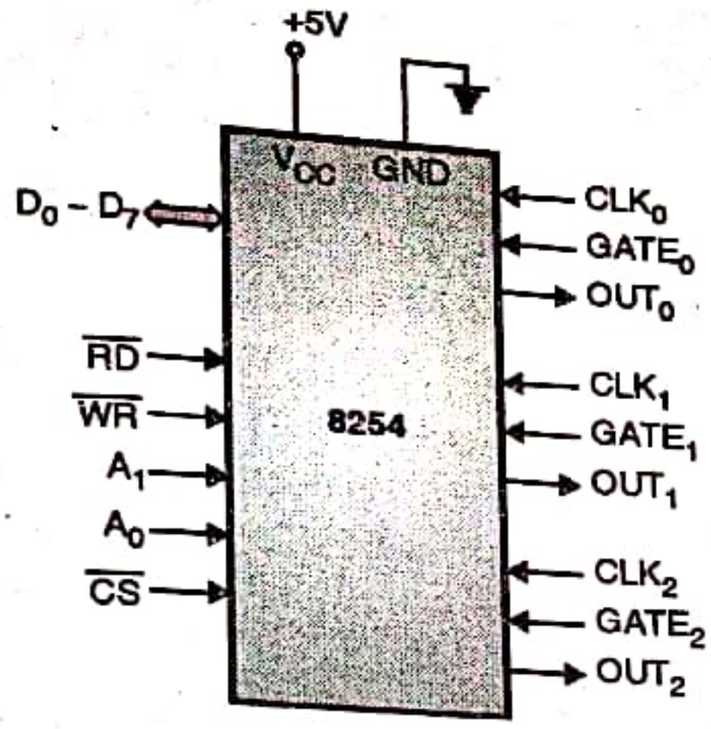


Fig. 12.1 : Pin diagram of 8254

Pin Description

Symbol	Name and function
1. D_0-D_7	**Data bus :** These are 8 bit bidirectional data bus lines, connected to the system data bus for data transfer between 8086 and 8254.
2. $\overline{CS}$	**Chip select :** This is an active low input signal, used to select the 8254 IC. If $\overline{CS}$ = 0 then 8254 will be active and take part in data transfer from/to 8086, otherwise 8254 will be in deactive state.
3. $\overline{RD}$	**Read :** This is an active low input signal, used in coordination with A_0 , A_1 to send data from appropriate counter to data lines $D_0 - D_7$.
4. $\overline{WR}$	**Write :** This is an active low input signal, used in coordination with A_0 , A_1 to load counters or to initialize counters.
5. A_0 - A_1	**Address lines :** These are input address lines used to distinguish different parts of 8254 such as Counter 0, Counter 1, Counter 2, control word register.

A_1	A_0	Selected part
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Control word register

6. CLK_{0-2}	**Clock input :** These are clock inputs to 3 independent counters. The pulses applied at these pins will be counted by respective counters.
7. $GATE_{0-2}$	**Gate : control :** These are active high, input signals used to allow external hardware to control the respective counter. The function of gate input is dependent on operating mode.
8. OUT_{0-2}	**Output :** These lines are active high, output lines. The output is dependent on operating mode.

Q. 3 Draw and explain block diagram of PIT8253. May 14, Dec. 16

Ans. : The block diagram of 8254 is as shown in Fig. 12.2.

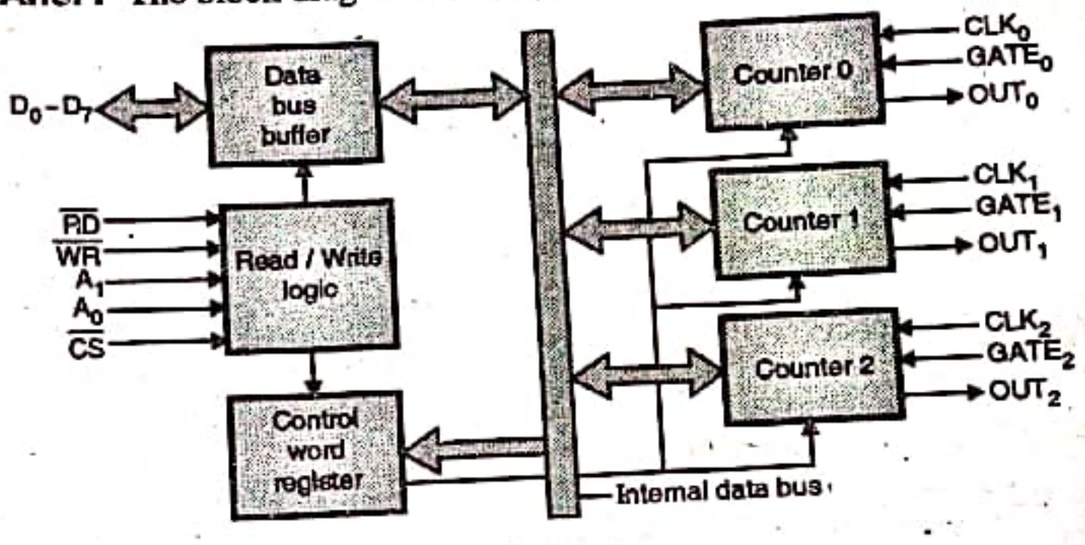


Fig. 12.2 : Functional block diagram of 8254

It includes data bus buffer, read/write logic, control word register and counters 0, 1, 2.

1) Data bus buffer

It is tristate, bi-directional 8 bit data bus buffer. It is used to interface 8254 data bus with system data bus. It is internally connected to internal data bus and its outer pins D_0 - D_7 are connected to system data bus. The direction of data buffer is decided by read and write control signals.

2) Read/Write logic

This block accepts inputs from system control bus and address bus. In I/O mapped I/O, the signals $\overline{RD}$ and $\overline{WR}$ are connected to $\overline{IOR}$ and $\overline{IOW}$. In memory mapped I/O, $\overline{RD}$ and $\overline{WR}$, are connected to $\overline{MEMR}$ and $\overline{MEMW}$. A_0 and A_1 are directly connected to address lines A_1 and A_2 of 8086. $\overline{CS}$ is connected to address decoder. The 8254 operation/selection is enabled/disabled by $\overline{CS}$ signal. A_0, A_1 selects a specific part $\overline{WR}$, $\overline{RD}$ decides writing data to 8254 or reading data from 8254. The control word registers and the counters are selected according to the signals on lines A_0 and A_1.

A_1	A_0	Selection
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	-1	Control word register

3) Control word register

This register of 8254 gets selected when $A_0 = 1$ and $A_1 = 1$. It is used to specify the BCD or binary counter to be used, its mode of operation and the data transfer to be used i.e. read or write the data bytes (LSB, MSB or both). If the CPU performs a write operation ,the data is stored in the control word register and is referred to as Control Word . It is used to define counter operation. The data can only be written into control word register, no read operation is allowed. Status information is available with the help of Read back Command.

4) Counters

There are three independent, 16 bit down counters. They can be programmed separately through control word register to decide mode of counter. Each counter is having 2 inputs viz. CLK and GATE. CLK is used as an input to counter and GATE is used to control the counter. The counters give output on OUT pin. The loaded count value in counter will be decremented by counter at each clock input pulse. The programmer can read counter without disturbing counter operation.

Q. 4 List operating modes of 8253. May 15.

Ans. : The programmable timer IC provides following modes of operations.

1. Mode 0 : Interrupt on Terminal count
2. Mode 1 : Programmable one shot / hardware triggerable one shot.
3. Mode 2 : Rate Generator / Pulse Generator
4. Mode 3 : Square wave Generator
5. Mode 4 : Software triggered Strobe
6. Mode 5 : Hardware Triggered Strobe

Q. 5 Explain Interfacing of 8253 / 8254 with 8086.

Ans. :

To interface 8254 or 8253 with 8086, we need to do the following connections.

1) Data lines $D_0 - D_7$ of 8253 are to be connected to data lines D_0 to D_7 (or D_8 to D_{15}) of 8086.
2) A_0 and A_1 of 8253 / 8254 are taken from A_1 and A_2 of 8086.
3) Control signals like $\overline{RD}$, $\overline{WR}$ and $\overline{CS}$ are to be connected.

Fig. 12.3 shows interfacing of 8253 / 8254 with 8086 in maximum mode. Similar connection can be done in minimum mode.

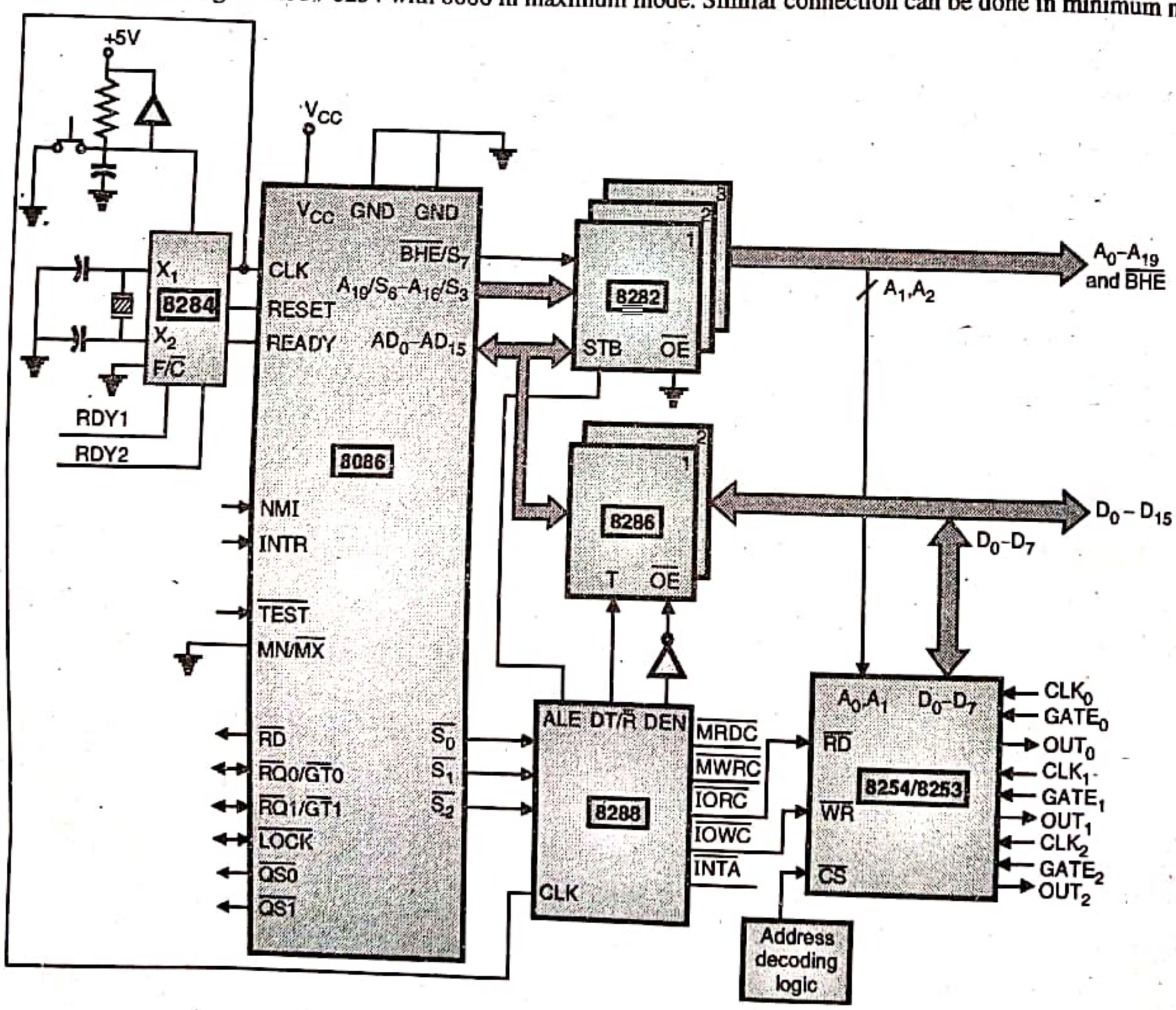


Fig. 12.3 : Interfacing 8253 / 8254 with 8086

Chapter 13 : 8257 DMAC

Q. 1 Write short note on DMA controller.

Ans. : DMA controller

In I/O data transfer, data is transferred, by using microprocessor. The microprocessor will read data from I/O device and then will write data to memory. In this case, there are two operations for single data transfer. If the data is less, then microprocessor will not waste its time; transferring data from I/O to memory or back. But suppose, data is huge, then the transfer rate from I/O to memory or back will slow down because of microprocessor intervention. In such case, to speed up the process of transferring the data, we can think, Can I/O have direct access to memory ?; and the answer is, yes. It can have Direct Memory Access (DMA), but under Supervision. The device which supervises, data transfer is named as DMA controller. Now let's have diagrammatic representation of the scheme, which depicts microprocessor, DMA controller, memory and I/O device.

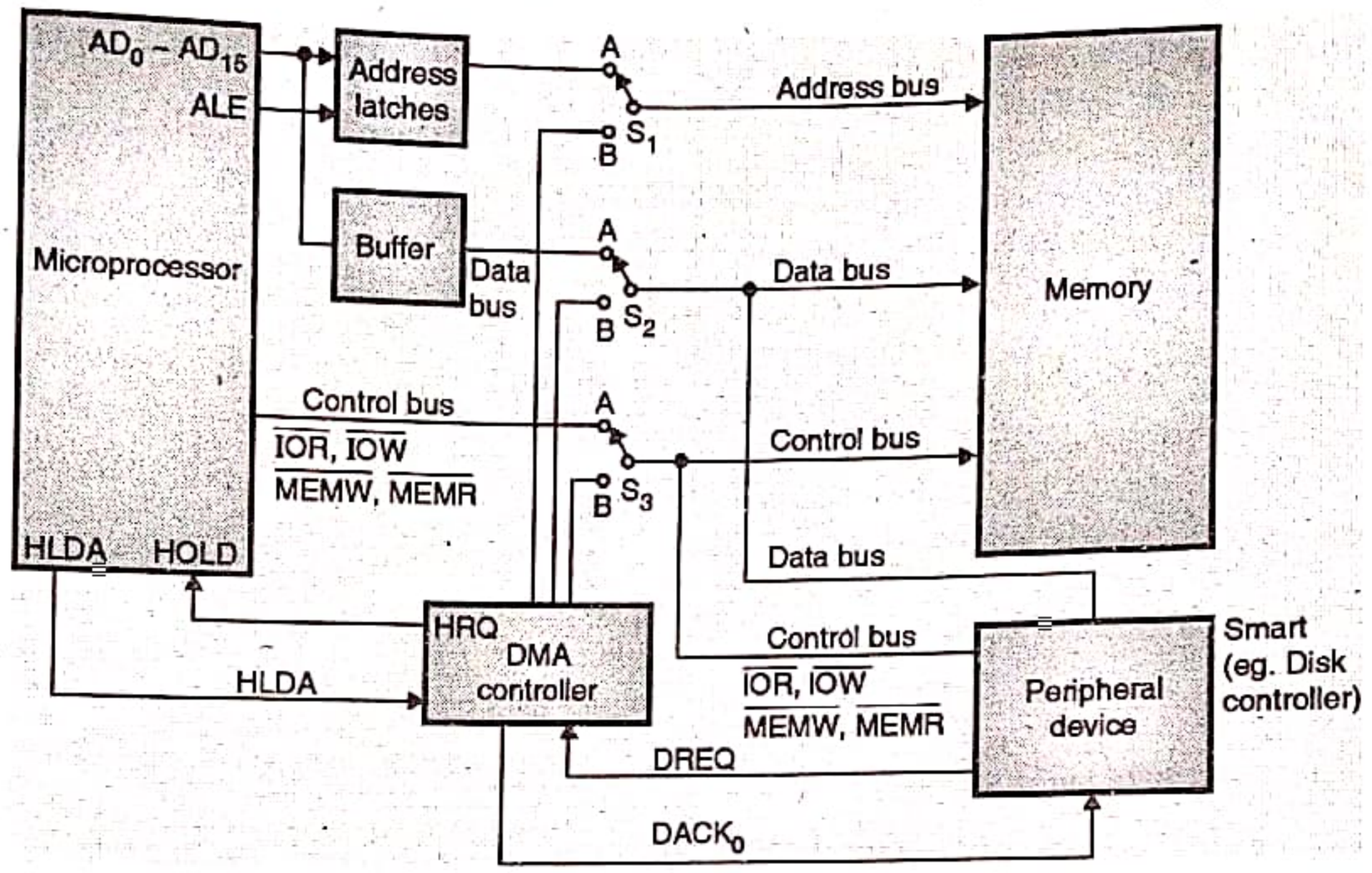


Fig. 13.1 : DMA controller interconnection scheme

1. Initially, switches S_1, S_2 and S_3 are at position A.
2. No direct access to memory; by I/O.
3. Microprocessor is MASTER of all three buses; address, data and control.
4. Microprocessor treats DMA controller, as I/O device ONLY.
5. Using IN/OUT instruction, you can program DMA controller chip, for various modes.
6. Whenever, peripheral device is ready to transfer data, DIRECTLY to memory, it will generate REQUEST (DRQ → DMA REQUEST), to DMA controller, asking for direct access.
7. In response to DRQ, DMA controller will activate, HRQ (HOLD request); connected to HOLD pin of microprocessor. By activating HOLD line, DMA controller request microprocessor, to HOLD for some time and allow him to become a master of all three buses.
8. Moment HOLD pin is HIGH, microprocessor will complete the present job and also active HLDA (HOLD acknowledge) signal; informing, DMA controller to become master.
9. Microprocessor tristates, all its buses, so total cutoff from memory and I/O device. Thus microprocessor relinquishes the buses and provides control to DMA controller.
10. Now DMA controller is master. It will position all three switches to position B.
11. DMA controller will also generate $\overline{\text{DACK}}$ (DMA acknowledge) signal to peripheral device; informing that, direct access is allowed.
12. Now it will generate address and control signal. Data will flow from memory to I/O or V.V.
13. After completing data transfer, DMA controller will deactivate HOLD line. It also positions, switches back to position A.

14. Now microprocessor will regain the control over the three buses.
15. Microprocessor will start executing instructions from main program. Till DMA is inactive or not master of the bus, is referred as DMA IDLE Cycle. When DMA controller gains the control, it is referred as DMA Active Cycle.

Q. 2 Discuss different data transfer modes of DMA controller 8257.

Ans. :

1. Burst or Block Transfer DMA

It is the fastest DMA mode. In this mode, two or more data bytes are transferred continuously. The microprocessor is disconnected from the system bus during DMA transfer i.e. the microprocessor cannot executes its own program during this transfer. N number of DMA cycles are added into the machine cycles of the microprocessor where N is number of bytes to be transferred.

In this mode, the DMA controller sends 'HOLD' signal to the microprocessor and waits for HLDA signal. After receiving HLDA signal, the DMA controller gains control of the system bus and executes a DMA cycle to transfer one byte. After transferring one byte, it increments memory address, decrements counter and transfers next byte. In this way, it transfers all data bytes between memory and I/O devices. After transferring all data bytes, the DMA controller disables 'HOLD' signal and enters into slave mode.

2. Cycle Steal or Single Byte Transfer DMA

In cycle steal transfer only one byte of data is transferred at a time. This type of DMA is slower than burst DMA. In this mode, only one DMA cycle is added between two machine cycles of the microprocessor, hence the instruction execution speed of the microprocessor is reduced slightly. In this mode the DMA controller sends 'HOLD' signal to the microprocessor and waits for HLDA signal. After receiving HLDA signal, the DMA controller gains control of the system bus and executes only one DMA cycle.

After transferring one byte, it disables 'HOLD' signal and enters into slave mode. The microprocessor then gains control of the system bus and executes next machine cycle. If the count is not zero and next data is available then the DMA controller sends 'HOLD' signal to the microprocessor and transfers next byte of data block.

3. Transparent or Hidden DMA Transfer

The microprocessor executes some states during which it floats the address and data buses. During these states, the microprocessor is isolated from the system bus. The DMA controller transfers data between memory and I/O devices during these states. This operation is transparent to microprocessor.

This is the slowest DMA transfer, In this mode, the instruction execution speed of microprocessor is not reduced. But, the transparent DMA requires logic to detect the states when the microprocessor is floating the buses. Now, we will study DMA controller chip 8237.

Q. 3 Write short note on Pin Configuration of 8257.

Ans. :

Fig. 13.2 : Pin diagram of 8257

Symbol	Description
$D_0 - D_7$	These are bi-directional, tristate, multiplexed data ($D_0 - D_7$) / address ($A_8 - A_{15}$) lines. In slave mode, these lines function as bi-directional data lines and transfer information (Address, count value, control word and status word) between microprocessor and 8257's registers. In master mode, these lines function as address output lines $A_8 - A_{15}$ (higher byte of memory address).
$\overline{IOR}$	It is an active low, tristate, bi-directional control signal. In slave mode, it functions as an input line. It is generated by microprocessor to read contents of 8257 registers. In master mode, it functions as an output line. It is generated by 8257 during DMA write cycle.

Symbol	Description
$\overline{IOW}$	It is an active low tristate, bi-directional control signal. In slave mode it functions as an input line. It is generated by microprocessor to write data into 8257 registers. In master mode, it functions as an output line. It is generated by 8257 during DMA read cycle.
CLK	It is a clock input line. It is connected to single phase, 50% duty cycle, external TTL clock generator.
RESET	It is a reset input line. It is connected to RESET OUT pin of microprocessor. This signal clears mode set register and status register, and forces 8257 into slave mode.
$A_0 - A_3$	These are tristate, buffered, bi-directional address lines. In the slave mode, these lines are used as address inputs and are internally decoded to access one of the registers. In master mode, these lines function as address output lines. The 8257 places $A_0 - A_3$ bits of memory address on these lines.
$\overline{CS}$	It is an active low, chip select input line. In slave mode, this signal is generated by address decoder to select the 8257 chip. In master mode it is ignored.
$A_4 - A_7$	These are tristate, buffered, address output lines. In slave mode these lines are tristated. In the master mode, the 8257 places $A_4 - A_7$ bits of memory address on these lines.
READY	It is an asynchronous input line. In master mode it is used to interface slow devices. When 8257 finds READY low, it adds a wait state. In slave mode, it is ignored.
HRQ	It is a hold request output line. It is connected to the HOLD input of the microprocessor.
HLDA	It is a HOLD acknowledge input line. It is connected to the HLDA output of the microprocessor. In response to this signal, the 8257 gains control of the system bus.

Symbol	Description
$\overline{MEMR}$	It is an active low, tristate, output control signal. It is tristated in slave mode. In master mode, it is activated during DMA read cycle.
$\overline{MEMW}$	It is an active low, tristate, output control signal. It is tristated in slave mode. In master mode, it is activated during DMA write cycle.
AEN	Address enable : It is a control output signal. It goes low in slave mode and high in master mode. It is used to : (1) Isolate CPU address, data and control lines and non DMA devices from the respective systemlines. (2) Disconnect data lines of 8257 from the system data bus and connect these lines to the high order system addresslines ($A_8 - A_{15}$). (3) Isolate DMA I/O devices from the system address bus.
ADSTB	Address strobe : It is an output control signal. In master mode it is used to latch higher byte of memory address.
TC	Terminal count : It is an output status signal. It is activated in master mode only. The high level on this line indicates the selected peripheral that, the present DMA cycle is the last cycle for block transfer. This signal is activated when the contents of TC register of the selected channel are equal to zero.
MARK	It is a modulo 128 MARK output line. It is activated in master mode only. It goes high after transferring every 128 bytes of data block. If the length of block is less than 128 then this signal is not activated.
$DRQ_0 - DRQ_3$	DMA request : These are asynchronous DMA request input lines. These signals are generated by peripherals.
$\overline{DACK_0} - \overline{DACK_3}$	DMA acknowledge : These are active low DMA acknowledge output lines. The low level on this line informs the peripheral that, it has been selected for a DMA cycle. It is used as a chip select input for a DMA I/O device in master mode.

Q. 4 Explain command words of 8257.

Ans. :

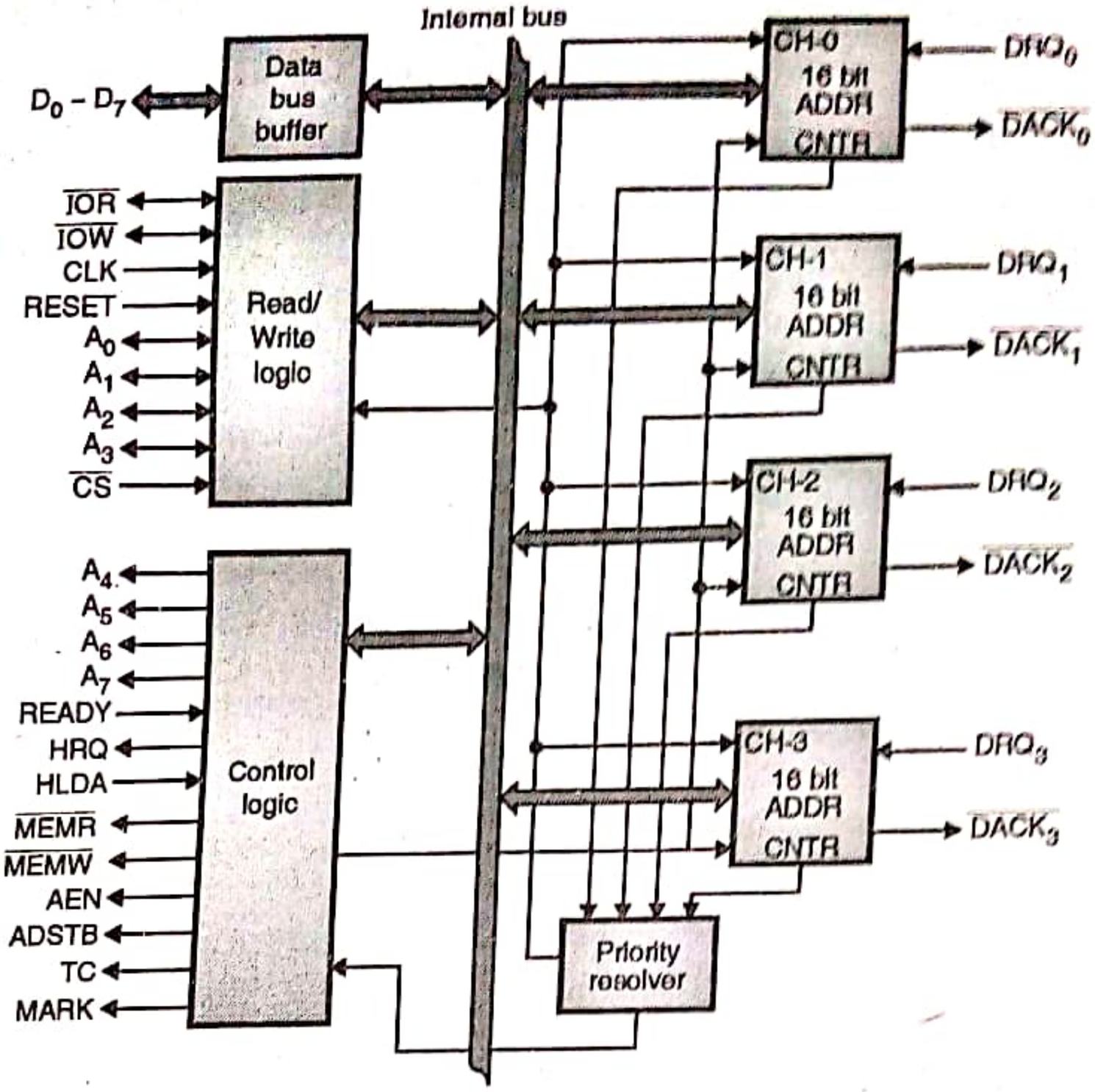


Fig. 13.3 : 8257 functional block diagram

The block diagram of 8257 is as shown in Fig. 13.3. It contains following blocks :

(1) Data bus buffer

It is a tristate, bi-directional buffer. In slave mode, it transfers data between microprocessor and internal bus. The direction of data bus buffer is set by read/write control logic. In master mode, it outputs memory address.

(2) Read/write control logic

In slave mode, it accepts address bits and controls signals from the microprocessor. In master mode, it generates address bits and control signals. It controls all internal read/write operations. It contains F/L flip-flop.

(3) Control logic block

It contains control logic, mode set register and status register. Control logic controls the sequence of DMA operations during all DMA cycles in master mode. It generates address and control signals. It increments 16 bit address and decrements 14 bit count register. It activates a HRQ signal on channel DMA request. It is disabled in slave mode.

(4) Priority resolver

It contains priority resolving logic circuit that resolves the priority of each channel. It can be initialized either in rotating or fixed priority mode.

(5) DMA channels

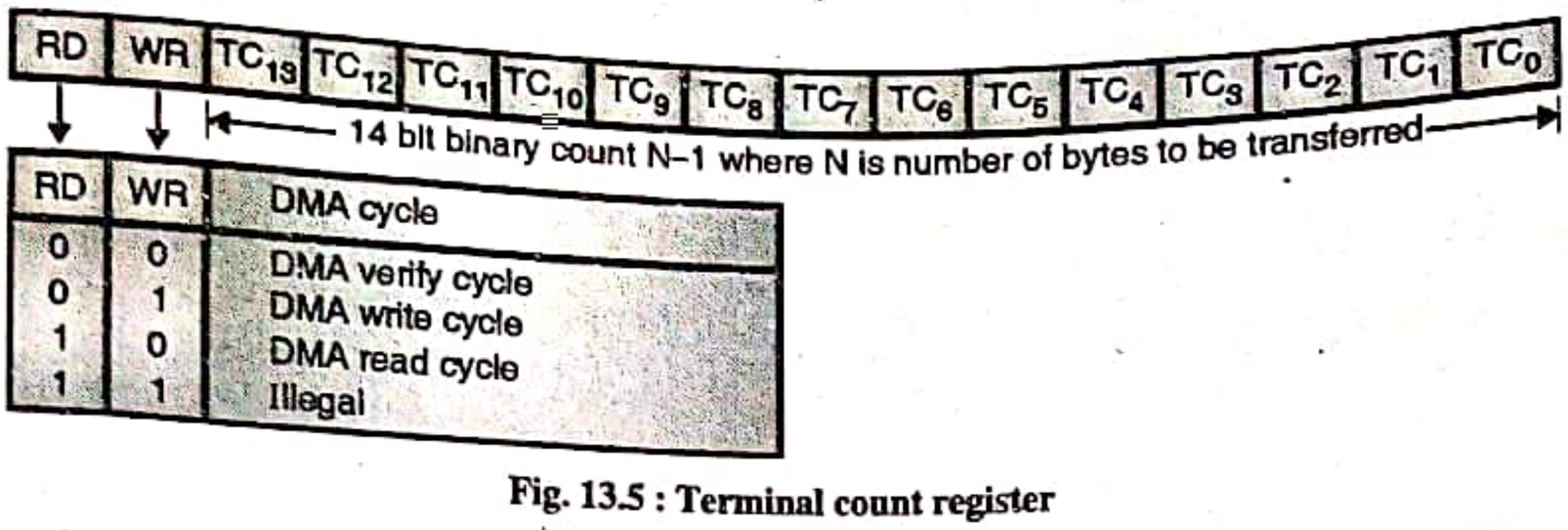
8257 provides four separate channels. Each channel contains two 16 bit registers.

(a) **DMA address register :** It is a 16 bit register. It is used to hold the starting address of memory. This register is incremented after each DMA cycle. If the address register is read in the middle of a DMA operation, it provides the address of next memory location. The format of address register is as shown in Fig. 13.4. A memory address must be programmed before the channel is enabled.

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

Fig. 13.4 : DMA address register

(b) Terminal count register :



RD	WR	DMA cycle
0	0	DMA verify cycle
0	1	DMA write cycle
1	0	DMA read cycle
1	1	Illegal

Fig. 13.5 : Terminal count register

It is a 16 bit register divided into two fields, 14 bit count and cycle control bits as shown in Fig. 13.5. The $TC_0 - TC_{13}$ bits indicate number of (DMA cycles bytes to be transferred) – 1. So to transfer N bytes this value should be N – 1. The 14 bit count value is decremented after each DMA cycle. RD and WR bits indicate type of DMA cycle or direction of data transfer. The count of DMA cycles and type of DMA cycle must be programmed before channel is enabled.

Q. 4 Write short note on DMA controller modes.

Ans. :

1. Rotating Priority Mode

If the RP bit of mode set register is set then the 8257 operates in rotating priority mode. After each DMA cycle, the priority of each channel changes. Hence all channels will get equal opportunity, if they are enabled and their DMA requests exist. Initially CH-0 gains highest priority while CH-3 gains lowest priority. The channel which has just been serviced will get the lowest priority after the DMA cycle and other channels move up to the next higher priority levels. The rotating pattern of channels is as shown in Fig. 13.6.

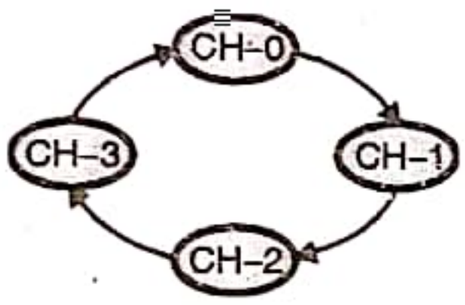


Fig. 13.6

Fig. 13.7

2. Fixed Priority Mode

If the RP bit of mode set register is reset then 8257 operates in fixed priority mode. In fixed priority mode, channel 0 has highest priority and channel 3 has lowest priority. The priority is resolved during state 4 of each DMA cycle.

3. Extended Write Mode

If the EW bit of mode set register is set, then 8257 generates advanced or extended write control signals (IOW & MEMW). i.e. the write control signals will go LOW, one clock cycle earlier, as shown in Fig. 13.7. This mode is used to interface slower devices to the system. If the memory device or I/O device connected is slower, then for synchronization READY signal is used. In this method the write signal is delayed by adding wait states into a DMA cycle. This reduces the speed of transfer. But in extended write mode, the write signal is extended earlier without adding states. i.e. the set up time of write input signal of an I/O device or memory is increased in extended write mode without reducing the speed of transfer. This signal allows more time to external logic for deciding if additional wait states are needed.

4. TC Stop Mode

If the TC stop bit in mode set register is set, then 8257 disables the channel whose TC is reached. Thus it stops further DMA operations on that channel. If the TC stop bit is reset, then the TC have no effect on channel, corresponding channel must be disabled by the microcomputer system through software. The TC stop bit option should be common for all channels.

5. Autoload Mode

If AL bit of mode set register is set, the 8257 operates in autoload mode. In this mode the data is transferred by channel 2 only i.e. other channels are not used for data transfer. It can be used for repeat block or block chaining operations.

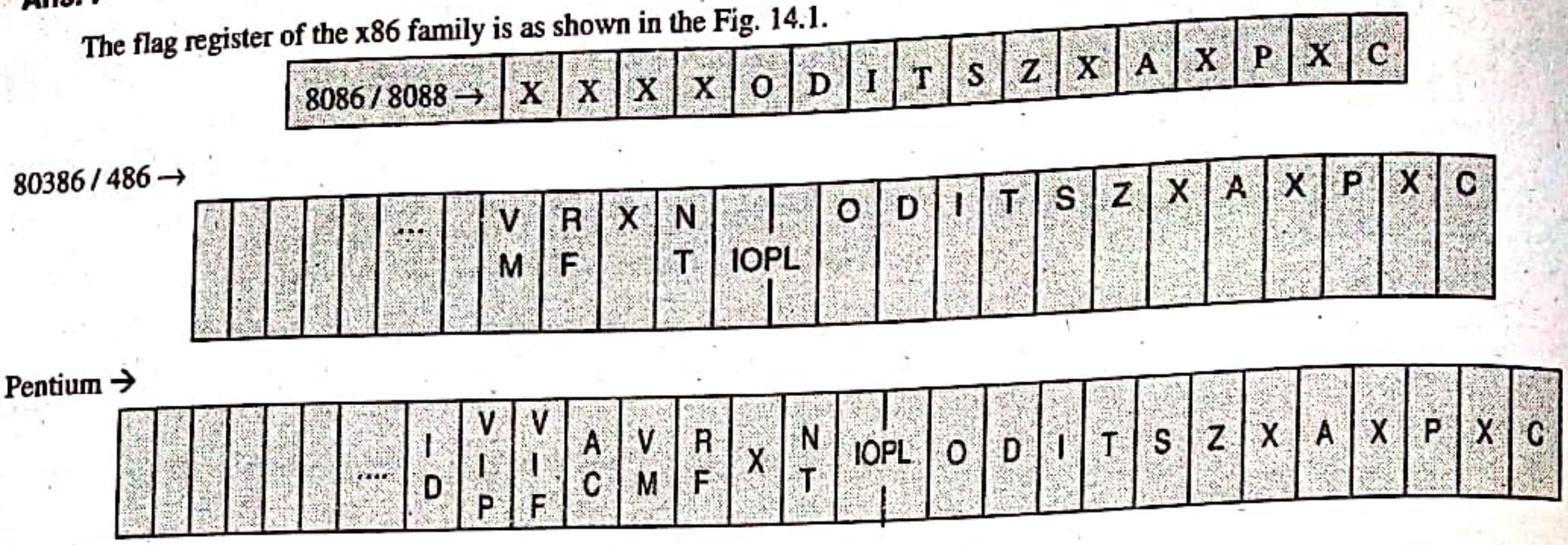
Chapter 14 : Intel 80386DX Processor

Dec 17

Q. 1 Draw and explain EFLAG register format of 80386 DX.

Ans. :

The flag register of the x86 family is as shown in the Fig. 14.1.



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**

Fig. 14.1 : Flag register

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 and 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 can access things accordingly. Protection mode will be studied in the further sections of this chapter.
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. 2 Write short note on control registers of 80386 DX. May 17

Ans. :

1. There are five control registers (CR0, CR1, CR2, CR3, and CR4).
2. Only four of them are used by the current implementation; register CR1 is reserved for future use.

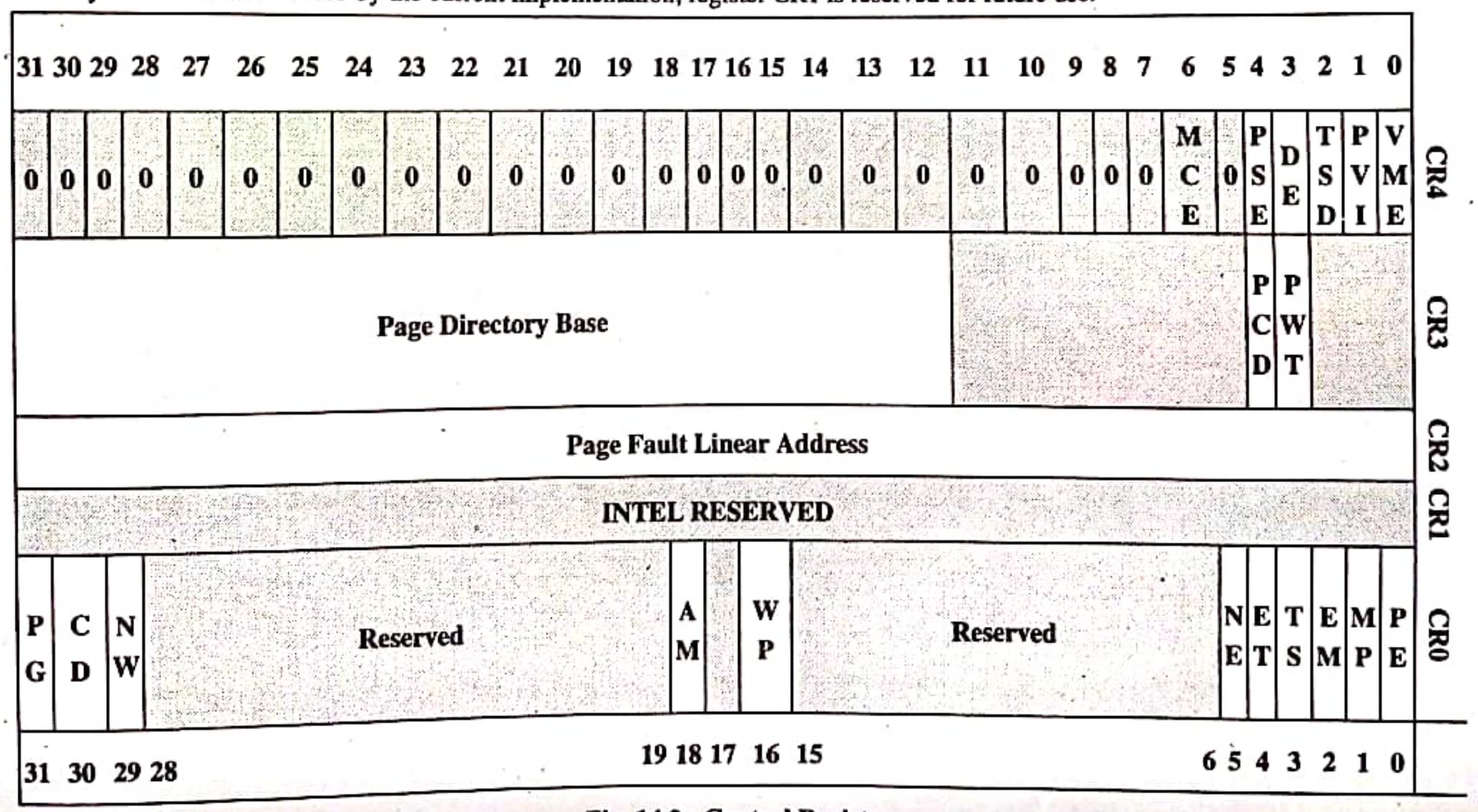


Fig. 14.2 : Control Registers

CR0

The lower 16-bits of CR0 are also called as MSW (Machine Status Word), for compatibility with 80286. The CR0 register contains system control flags, which control modes of operation or indicate state of processor.

1. PG – Paging Enable

When PG = 1, paging is enabled. When PG = 0, paging is disabled.

2. CD-Cache Disable

The CD bit is used to enable or disable the on-chip cache (This bit is not present in 80386; it is present in Pentium processor). When CD = 1, the cache line fill on cache misses are disabled. CD has no effect on cache hits.

3. NW-Not Write-through

It enables on-chip cache to use write-through or write-back policy. When NW = 0, L1 uses write back policy else it uses write through policy (This bit is not present in 80386; it is present in Pentium processor).

4. AM-Alignment Mask

The AM bit allows alignment checking when AM = 1 and disables alignment checking when AM = 0 (This bit is not present in 80386; it is present in Pentium processor).

5. WP-Write Protect

When WP = 1, processor offers write protection to user-level pages against supervisor-level write operations. When WP = 0, read-only user-level pages can be written by a supervisor process (This bit is not present in 80386; it is present in Pentium processor).

6. NE-Numeric Error

When NE = 1, it enables the standard mechanism for reporting floating-point numeric errors by the on-chip FPU (This bit is not present in 80386; it is present in Pentium processor).

7. ET-Extension Type

The ET bit indicates if 80387 is connected or a previous math coprocessor is connected.

8. TS-Task Switched

The processor sets this bit whenever a task switch operation is performed.

9. EM-Emulation

When EM = 1, it causes INT 11 on a floating point operation where the floating point instruction may be emulated, when the processor does not have a floating-point unit.

10. MP-Monitor coprocessor

On the i286 and i386 processors, the MP bit controls the function of the WAIT instruction, which makes the processor to wait for the completion of the task by math coprocessor. When it is '1', it indicates a coprocessor is connected and on wait instruction the host processor has to wait for the TEST# pin to be enabled.

11. PE-Protection Enable

When the processor enters into the protected mode it makes PE = 1, and the protection mechanism is enabled. When PE = 0, the processor operates in real (fast 8086) mode.

CR1 : It is reserved by Intel for future use.

CR2 : The CR2 register holds the 32-bit linear address that caused the last page fault detected.

CR3 : The CR3 is also known as the page directory base register (PDBR).

1. PCD - Page-level Cache Disable

When PCD = 1, the on-chip cache is disabled for the particular page. When PCD = 0, on-chip cache is enabled, provided it is not disabled by other means (This bit is not present in 80386; it is present in Pentium processor).

2. PWT - Page-level Write Transplant

The PWT bit can be used to control the write policy of an external second-level cache. When PWT = 1, it allows a write-through policy for the external cache. If PWT = 0, a write-back policy for the external cache is implemented (This bit is not present in 80386; it is present in Pentium processor).

CR4

(This register is not present in 80386; it is present in Pentium processor). Register CR4, new on Pentium, contains bits that enable certain architectural extensions.

1. MCE-Machine Check Enable

MCE = 1, enables the machine check exception. These machine check exceptions are used to measure the performance of the processor. This bit enables or disables the use of RDMSR and WRMSR instructions.

2. PSE-Page Size Extension

The page size for virtual memory implementation is 4KB when PSE = 0, while the page size is 4MB when PSE=1.

3. DE-Debugging Extensions

When DE = 1, I/O breakpoints programmed in the debug registers are enabled.

4. TSD-Time Stamp Disable

When TSD = 1, the read from time stamp counter using RDTSC a privileged instruction is enabled.

5. PVI-Protected-mode Virtual Interrupts

When PVI = 1, support for virtual interrupt flag in protected mode is enabled.

6. VME-Virtual-8086 Mode Extensions

When VME = 1, support for a virtual interrupt flag in virtual-8086 mode is enabled. This feature may improve performance in this mode.

Q. 3 Explain V86 mode of 80386DX. May 16

Ans. : V86 mode of 80386DX

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. 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 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, 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.

Q. 4 What is GDT? Explain structure of GDT.

Ans. : GDT

Global memory location is one which is accessible to all the tasks. The Global Descriptor Table (GDT) is a common table accessible to all the tasks. The GDT is selected by a 48-bit register called as GDT Register (GDTR). GDTR has the following two fields :

1. 32-bit base address that provides the starting address of the descriptor table
2. 16-bit Limit address that when added to the 32-bit base address gives the last address of the GDT.

Hence the maximum value of the 16-bit limit address can be 64 KB which means that the maximum size of the GDT can be 64 KB. Each descriptor, is of 8 bytes. Hence the total number of descriptors in GDT is 8K (64 KB / 8 B). 13-bit selector field of the segment register selects one of the descriptor which gives the base address, limit address and the ARB of the corresponding segment. The base address of the segment when added with the offset address provided in the instruction generates the physical address. The offset address must not be more than the limit address or else it will generate a exception interrupt 5 i.e. bound exception. This access of global memory or segment translation mechanism for global data is as shown in Fig. 14.3.

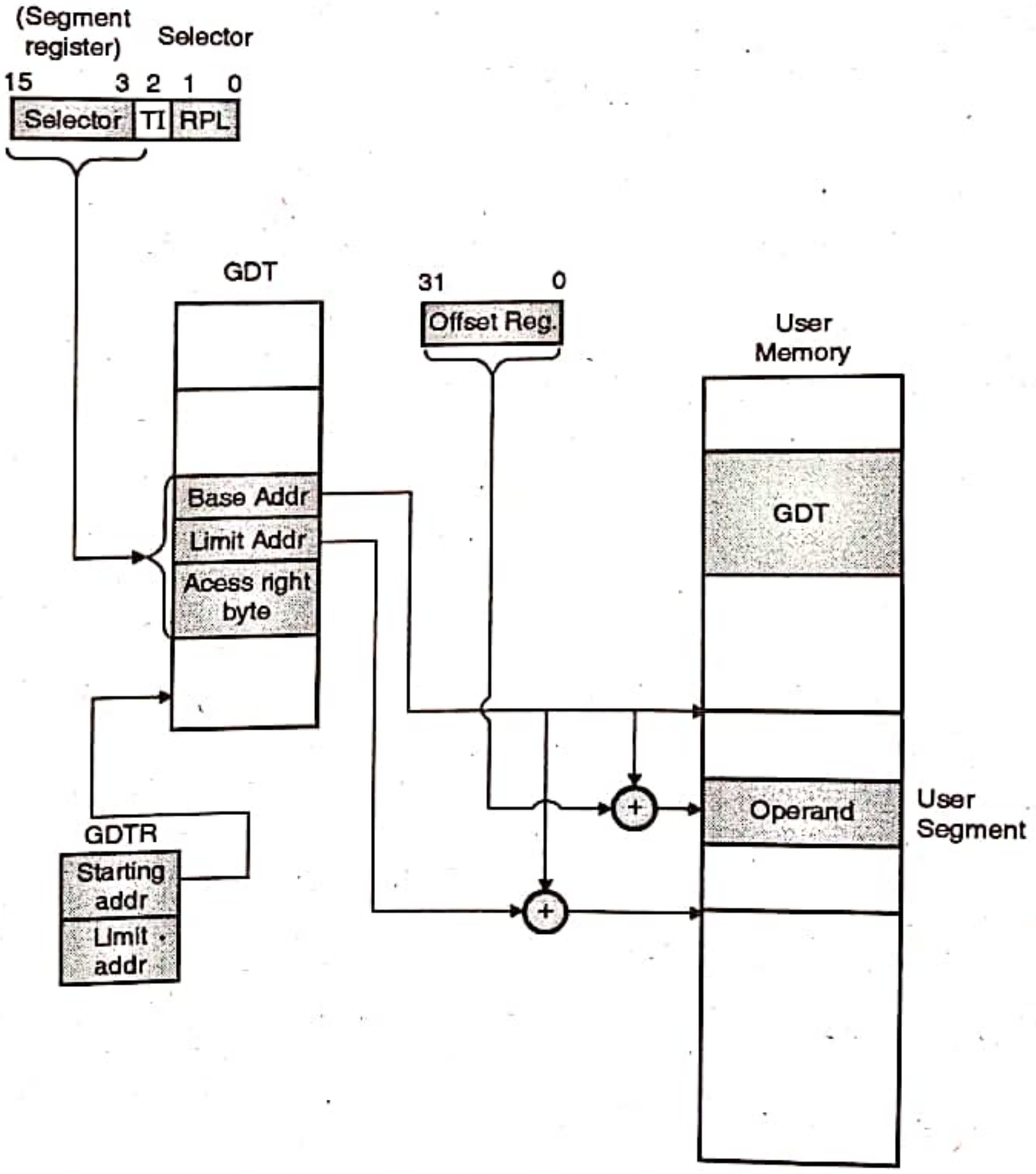


Fig. 14.3 : Segment translation mechanism in protected mode for global memory access

Q. 5 Differentiate between real mode and protected mode. Dec. 14, May 15

Ans. :

Parameter	Real mode	Protected Mode
General	Real mode is also called real address mode. It is the default operating mode on Reset.	When a processor that supports x86 protected mode is powered on, it begins executing instructions in real mode, in order to maintain backwards compatibility with earlier x86 processors. Protected mode may only be entered after the system software sets up several descriptor tables and enables the Protection Enable (PE) bit in the Control Register 0 (CR0)
Use	It is the default operating mode on Reset. Its main function is to initialize 80386 for protected mode operation.	It allows system software to utilize features such as virtual memory, segmentation paging, multi-tasking, protection and other features designed to increase an operating system's control over application software
Access	In the real mode 80386 can access all the registers.	In the protected mode all general purpose registers, control registers, debug registers, test registers, Segment selectors, segment descriptors can be accessed.
Memory addressing	In the real mode 80386 can directly address upto 1MB of memory.	In the protected mode 80386 can access $2^{32} = 4$ GB of memory with 32 bit addressing.
Entering the mode	The 80386 begins its execution in real address mode on power up or reset.	For protected mode operation the PE bit of CR0 register must be set. It is essential to maintain IDT, GDT or LDT.
Leaving the mode	To leave the real mode and enter protected mode the PE bit of CR0 must be set.	Whenever processor wants to return to real mode the user can clear the PE bit in CR0.
Addressing	**Refer Fig. 14.4**	**Refer Fig. 14.5**

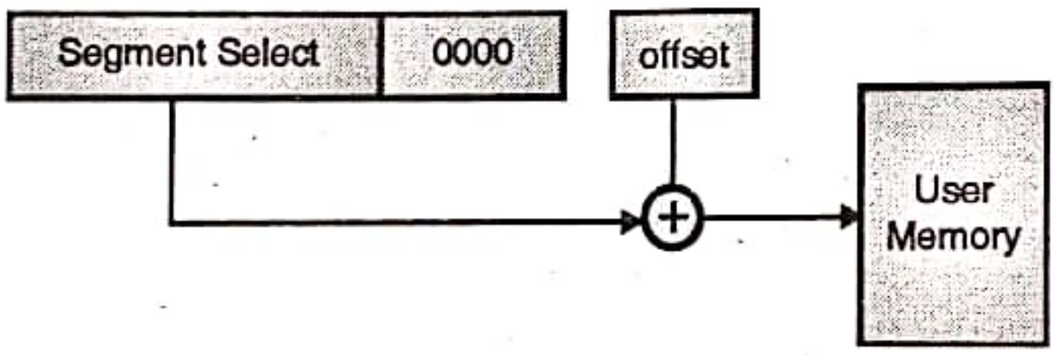


Fig. 14.4

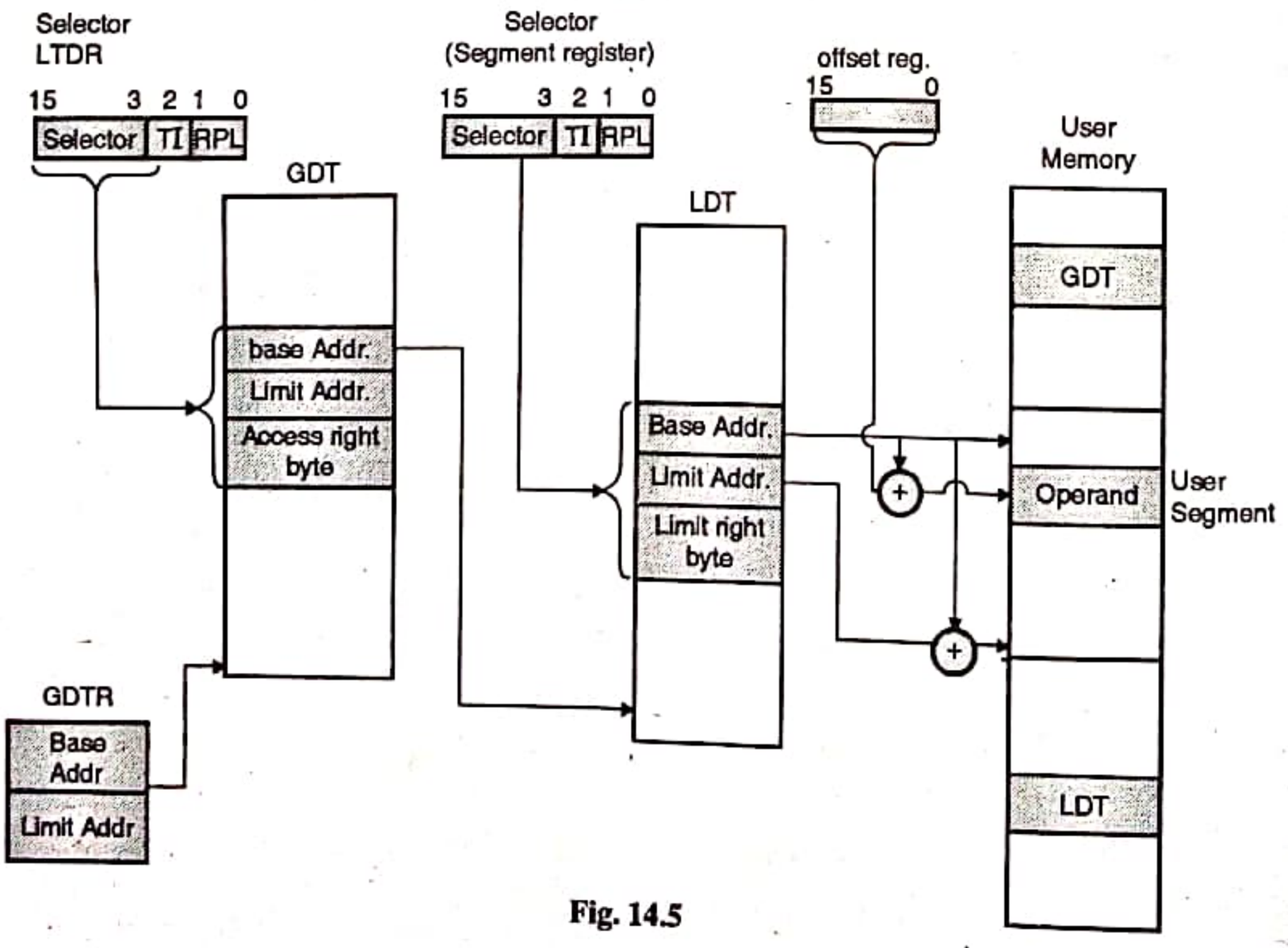


Fig. 14.5

Chapter 15 : Intel P5 Microarchitecture

Q. 1 Write down features of Pentium processor.

Ans. : May 15

Features of Pentium processor

1. Wider Data bus

64 bit data bus, that permits 8-bytes or 4-words to be transferred in a single bus cycle. Hence faster cache line fills into its internal caches. Supports 64-bit addressing scheme.

2. Dedicated Instruction Cache

8KB instruction cache that feeds its two integer execution units and a floating point unit via a dual instruction pipeline.

Instruction cache is read-only cache and is organized as a 2-way set associative cache with line size of 32 bytes.

3. Dedicated Data Cache

Data Cache is an 8-KB cache that serves all three execution units. The data cache is a write back cache and like code cache is a 2-way set associative with line size of 32 bytes.

External pins are implemented to control the write back feature, thus ensuring cache coherency with other caches and main memory.

4. Parallel Integer Execution

The instruction pipeline includes two parallel paths called U pipeline and the V pipeline.

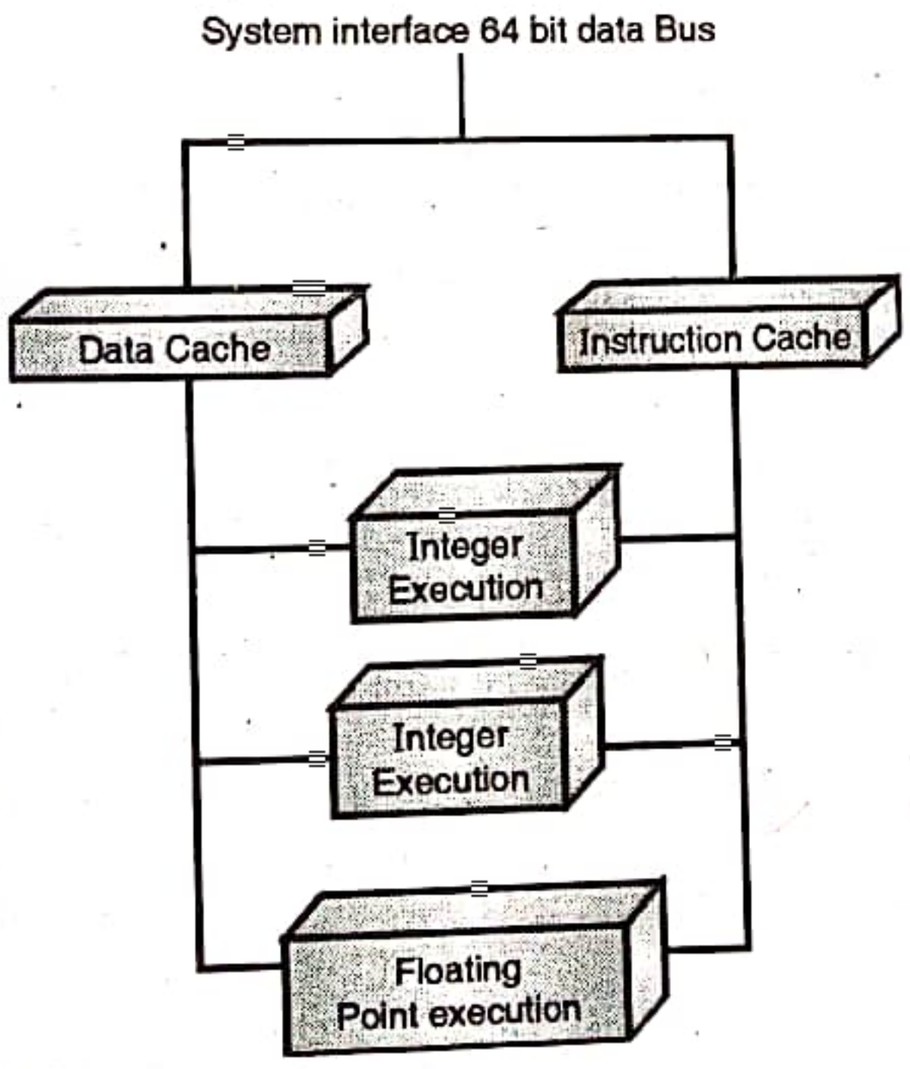


Fig. 15.1

The dual instruction pipeline and execution unit allows two instructions to be decoded and executed simultaneously, permitting two instructions to complete executing in a single processor clock, hence its superscalar performance.

5. Floating-Point Unit

The floating point unit employs a new design that provides substantial performance increases over the i486 design. It gains better results mainly due to pipelined execution of floating point instructions.

6. Branch Prediction

Pentium has a branch target buffer and branch prediction algorithm. Branch target buffer keeps the track of execution history to determine branch is likely to occur or not. If Branching is predicted, the Prefetcher fetches the instruction from predicted target instead to be in sequential fashion as in case of earlier x86 processor. That means execution pipelines do not always stall the processor when branching occurs.

7. New Instructions

There are new instructions added in Pentium processor. e.g. CPUID, RDMSR, WRMSR

8. Speed

60MHz, 66MHz, 75MHz, 90MHz, 100MHz, 120MHz, 33MHz, 150MHz, 166MHz, and 200MHz.

Q. 2 Draw and explain pentium processor architecture.

Ans. :

The functional units of Pentium processor are :

(A) Bus Unit

It provides a physical interface between the Pentium processor and the rest of the system.

It consists of the following units:

1. Address Drivers and Receivers: 32 address driver circuits.
2. Write Buffers

2 write buffers, of 64 bit each one for each of the two internal execution units, hence increasing performance when both have write operation. Each can hold a single write operation that misses the internal cache. If the bus unit is busy running a cycle, the write data can be stored into it and the execution unit can proceed with next instruction.

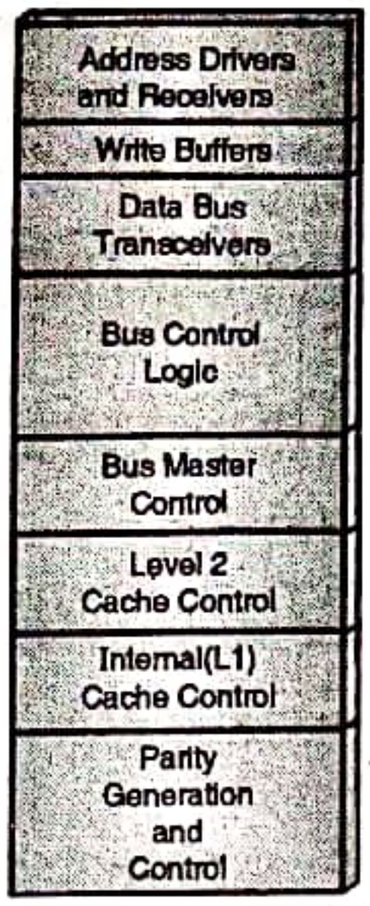


Fig. 15.2

3. **Data Bus Transceivers :** It consists of bidirectional tristate buffers for data.

4. **Bus Master Control :** In Multiprocessor system, request from the Bus Arbiter is handled by this unit.

5. **Bus Control Logic :** This unit generates control signals for the system bus.

6. **Level Two (L2) Cache Control :** To check whether L2 cache is present or not; if present it uses write back or write through policy

7. **Internal Cache control :** To implement Line fill, snooping etc.

8. **Parity Generation and Control :** To assure error free transmission of data.

(B) Data Cache

8 KB write back cache, organized as 2-way set associative with 32 byte lines. It is triple ported to allow simultaneous access from each of the pipelines and snooping. Keeps copy of most frequently used data by 2 integer pipelines and FPU.

(C) Code Cache

8 KB write back cache, organized as 2-way set associative with 32 byte lines. It is triple ported to allow split line access from Prefetcher and snooping. Keeps copy of most frequently used instruction by 2 integer pipelines and FPU.

(D) Prefetcher

Instructions are requested from code cache by the Prefetcher If the requested line is not in cache, a burst cycle is run to external memory to perform a cache line fill.

(E) Prefetch Buffers

Four Prefetch buffers as 2 independent pairs. i.e. 64 B each When instruction is prefetched it is placed into one set of Prefetch buffers, while the other pair is idle. When BTB (Branch Target Buffer) predicts branch operation, it requests the predicted target from cache which is placed in the second pair of buffers that was previously idle and this new pair of buffer will be used until another branch is predicted by BTB.

(F) Instruction Decode Unit

During, Decode 1, the instruction is decoded in both pipelines to determine whether the 2 instruction can be paired. If pairable, the two instructions are sent together to D2 and during D2, the memory resident operands are found.

(G) Control Unit

Consists of Microcode Sequencer and Microcode Control ROM.

(H) ALU

The ALU for 'U' pipeline can complete an instruction prior to the ALU in V pipeline, but the ALU for 'V' pipeline cannot complete an instruction prior to the ALU in 'U' pipeline.

(I) Paging Unit

If Paging is enabled, the Paging Unit translates the linear address from address generator to a physical address. Two Translation look aside Buffers (TLB) are implemented, one for each code and data cache.

(J) Floating Point Unit (FPU)

It can accept up to 2, floating point operations per clock if one of the instruction is an exchange instruction. The FPU uses an 8 stage pipeline. FP instruction cannot be paired with integer instruction but some pairing of FP instruction is possible. Pairing of FP instruction is possible with XCHG instruction of FPU. 3 operations can be done simultaneously for e.g. 3, FADD instructions

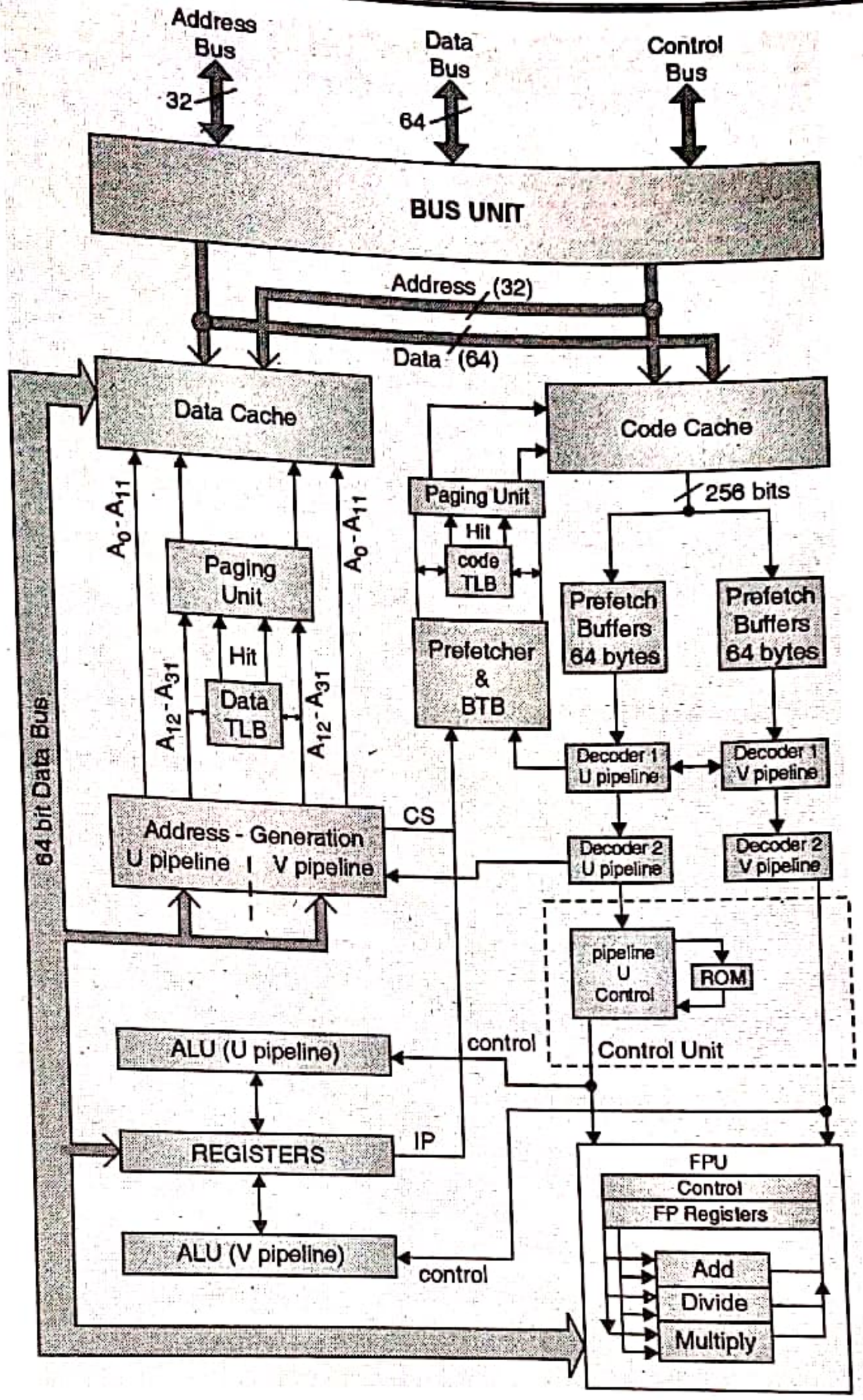


Fig. 15.3

Q. 3 Draw and explain the state diagram of MESI transitions that occur within the Pentium's data Cache memory. May 13

Ans. :

State diagram of MESI transitions

1. Internal snoop hit is to be considered in case when the data required by the processor is there in code cache or vice versa. In such cases the modified line is written back to the main memory and then invalidated, while a non modified line is simply invalidated.
2. Flush# signal indicates the L1 cache to clear the entire cache. WBINVD indicates to clear a particular line in the L1 cache. In case of a Flush# signal or WBINVD, also the modified line is first written back to the main memory and then invalidated while non modified line is simply invalidated.
3. In case of a INVD signal that simply indicates to invalidate a line without even writing it back for a modified line also, the processor simply invalidates the corresponding line.

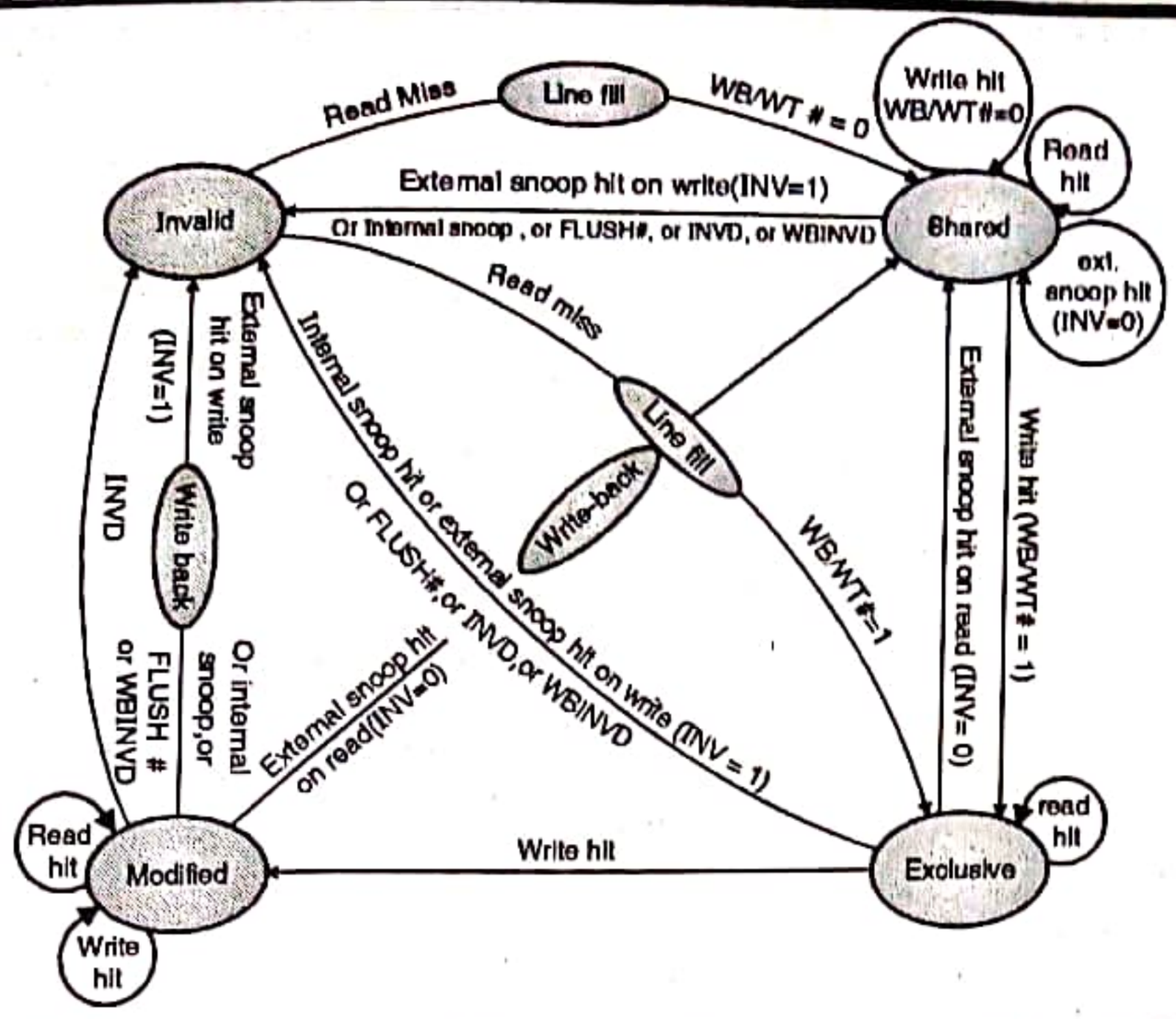


Fig. 15.4 : State diagram of MESI transition states as in Pentium's Data cache

Q. 4 Explain, with neat diagram, cache memory organization is supported by Pentium processor. May 17

Ans. :

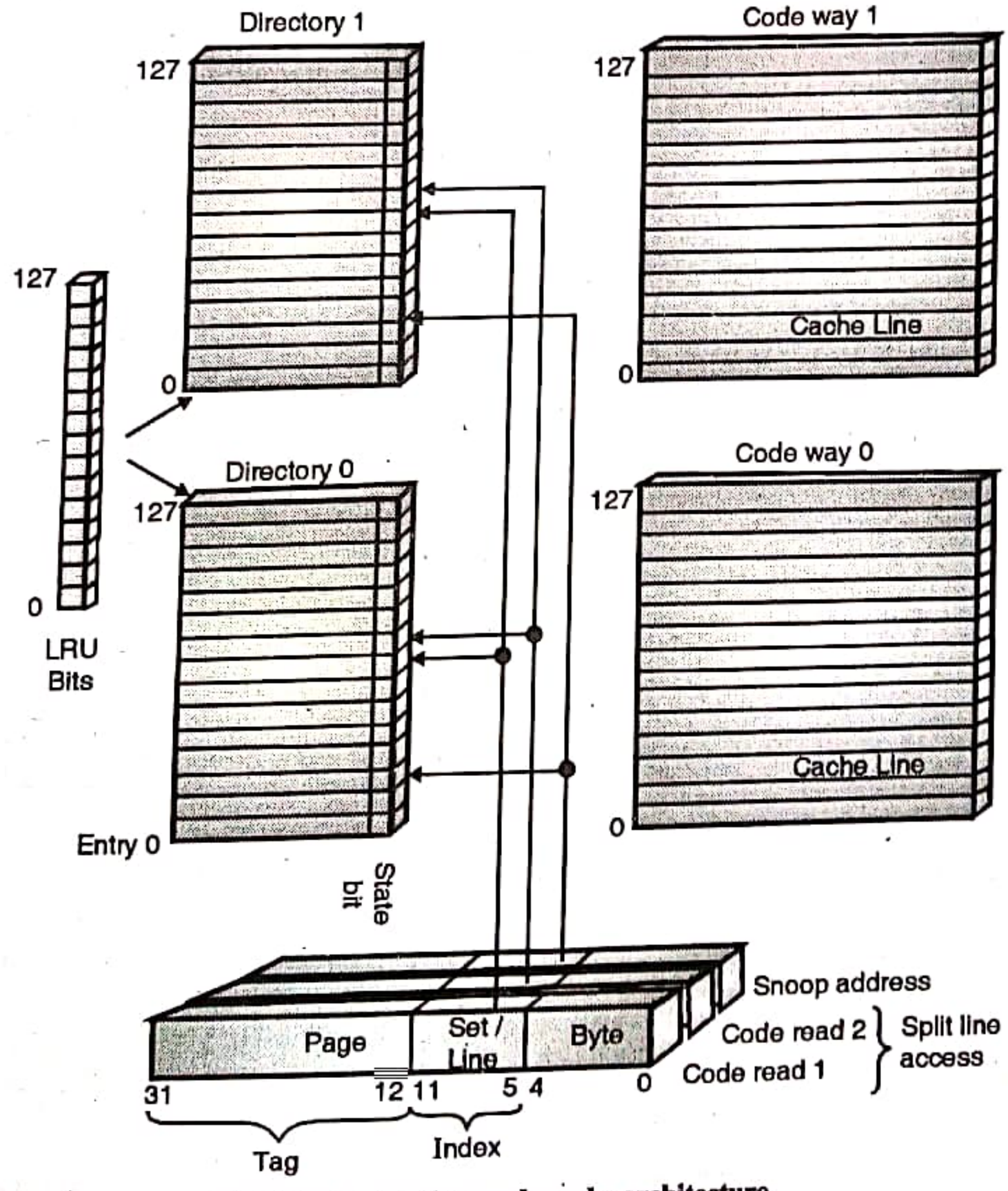


Fig. 15.5(a) : Pentium code cache architecture

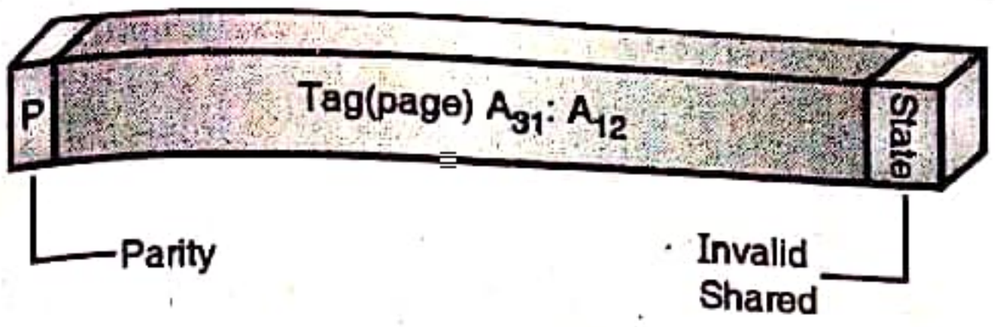


Fig. 15.5(b) : Code cache directory entry

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. 15.5.

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.
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. 15.6.

Fig. 15.6 : 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.

Q. 5 Discuss data cache organization of Pentium. Dec. 17

Ans. :

All accesses by the execution units for data are routed through the data cache.

1. Assume that two memory read instructions that are being executed simultaneously in the two pipelines are as follows:

```
MOV AX, [1056]      ; Read in U pipeline
MOV BX, [1086]      ; Read in V pipeline
```

2. Assume the Pentium is in protected mode, the data segment starts at memory location 00200000H, and paging is turned off. The instruction in U pipeline says read two bytes from locations 00201056H and 00201057H and places them in AX register, while the instruction in V pipeline causes two bytes to be read from locations 00201086H and 00201087H and placed in register BX.
3. Cache controllers view main memory as being divided into pages equal in size to the cache ways. The Pentium processor has an 8KB, two-way set associative data cache i.e. each cache way is 4KB is size. This means that the total 4GB of memory address space is viewed as 1M pages, each of which is 4KB in size.
4. Each memory page is viewed by the internal cache controller as having 128 lines, each containing 32 bytes of data.
5. The Pentium processor's data cache is banked on four byte (doubleword) boundaries to permit simultaneous doubleword accesses for both the U- and V-pipelines. Since the ALU are of 32-bit, they require 32-bit (doubleword) at a time.
6. This permits simultaneous access by the two ALUs if the two accesses do not target the same bank in a single clock cycle. Otherwise two separate cycles are required.
7. Each memory address sent to the internal cache controller is examined to determine the page, line and doubleword that contain the target memory location by the division of the address as shown in Fig. 15.7.

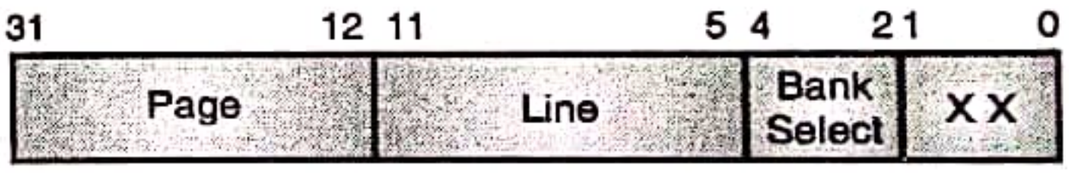


Fig. 15.7 : Address as viewed by L1 data cache

Q. 6 Write the instruction issue algorithm used in Pentium. May 15, Dec. 17

Ans. :

The instructions of Pentium processor are pairable if the following rules are followed.

1. 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.

2. Only simple instructions can be executed in 'V' pipeline while all other instructions are executed in 'U' pipeline
3. 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
4. Both pipelines are supplied by a steady stream of instructions by the prefetcher, which in turn is supplied by the code cache.
5. 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 :

   i. Both instructions are simple

   ii. Instruction must not have register contention
6. 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.
7. 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.

Q. 7 Compare Pentium II and Pentium III processors. Dec. 16

Ans. :

Comparision of Pentium II and Pentium III processors

Sr. No.	Processors → Features ↓	Pentium II	Pentium III
1	Processor Size	32 bits	32 bits
2	Speed	233 to 300 MHz (later upto 450 MHz)	450 to 500 MHz (later upto 1.4 GHz)
3	MIPS	-	-
4	Address bus size	32	32
5	Data bus size	64	64
6	No. of transistors	7.5 million	9.5 to 28 million
7	Addressable memory	4 GB	4 GB
8	Virtual memory	64 TB	64 TB
9	L1 Cache	32KB Spilt	32KB Spilt
10	L2 Cache	512 KB	256 KB ATC (or 512 KB)
11	L3 Cache	-	-
12	On chip FPU	Yes	Yes
13	Superscalar	Yes	Yes

Sr. No.	Processors → Features ↓	Pentium II	Pentium III
14	MMX instruction set	SSE1	SSE2
15	Hyper threading support	No	No
16	No. of cores	1	1
17	Fabrication process	0.35 µm	0.18 µm
18	Generation	P6	P6
19	SMP (multiprocessor) support	No	No
20	Integer pipeline stages	14	14
21	No. of integer pipelines	2	2
22	Floating point pipeline stages	8	11
23	No. of floating point pipeline stages	1	1
24	Overclocking feature	No	No

Q. 6 Write short notes on Comparison between i5 and i7. May 16

Ans. :

Comparision of i5 and i7 processors

Sr. No.	Feature	i5	i7
1.	Number of cores	4 for Desktop 2 for Laptop	4 or 6 for Desktop 2 or 4 for Laptop
2.	Processing threads	8 threads for desktop 4 threads for Laptop	8 or 12 threads for desktop 4 or 8 threads for laptop
3.	Maximum base clock frequency	3.4GHz	3.2GHz
4.	Maximum turbo boost frequency	3.8GHz	3.8GHz
5.	Maximum smart cache size	6MB	12MB
6.	Intel turbo boost 2.0	Present	Present
7.	Intel Hyperthreading	Present only in Laptop processors	Present
8.	Best Desktop processor	Intel Core i5-2550K (3.4GHz, 6MB)	Intel Core i7-3930 (3.2GHz, 12MB)
9.	Best Mobile (Laptop) processor	Intel Core i5-2540M (2.6GHz, 3MB)	Intel Core i7-2860 (2.5GHz, 8MB)

❑❑❑