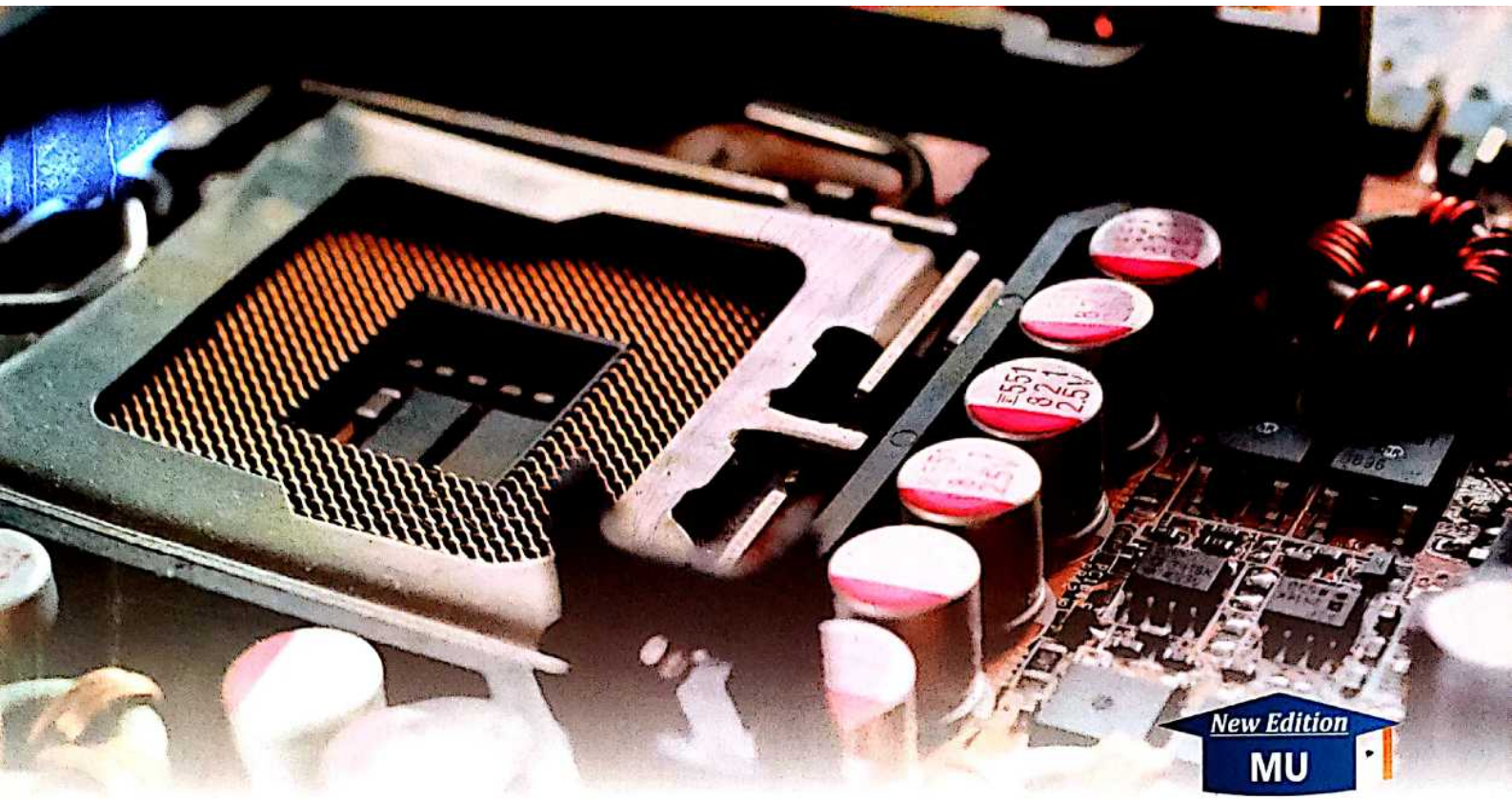


Strictly as per the New Revised Syllabus (REV- 2019 'C' Scheme) of Mumbai University w.e.f. academic year 2020-21 (As per Choice Based Credit and Grading System)

Microprocessor

(Code : CSC405)

Semester IV

Computer Engineering / Computer Science and Engineering / Artificial Intelligence and Data Science / Machine Learning / Cyber Security / Internet of Things (IoT) /Data Engineering / Data Science / Internet of Things and Cyber Security Including Block Chain Technology

Harish G. Narula **Khushboo Shah**

Includes :

- Solved Latest University Question Papers.
- Lab Manual.

TechKnowledge Publications

Syllabus

Mumbai University
Second Year of Computer Engineering (2019 Course)

MICROPROCESSOR (CSC405)

Course Code	Course Name	Credits
CSC405	Microprocessor	3

Prerequisite Courses : *Digital Logic and Computer Architecture*

Course Objectives :

1. *To equip students with the fundamental knowledge and basic technical competence in thefield of Microprocessors.*
2. *To emphasize on instruction set and logic to build assembly language programs.*
3. *To prepare students for higher processor architectures and embedded systems*

Course Outcomes : *On successful completion of course, learner will be able to:*

1. *Describe core concepts of 8086 microprocessor.*
2. *Interpret the instructions of 8086 and write assembly and Mixed language programs.*
3. *Identify the specifications of peripheral chip.*
4. *Design 8086 based system using memory and peripheral chips.*
5. *Appraise the architecture of advanced processors*
6. *Understand hyperthreading technology*

COURSE CONTENTS

Module		Detailed Contents	Hours
1		**The Intel Microprocessors 8086 Architecture**	8
	1.1	8086CPU Architecture,	
	1.2	Programmer's Model	
	1.3	Functional Pin Diagram	
	1.4	Memory Segmentation	
	1.5	Banking in 8086	
	1.6	Demultiplexing of Address/Data bus	
	1.7	Functioning of 8086 in Minimum mode and Maximum mode	
	1.8	Timing diagrams for Read and Write operations in minimum and maximum mode	
	1.9	Interrupt structure and its servicing **(Refer Chapters 1, 2, 3 and 7)**	
2		**Instruction Set and Programming**	6
	2.1	Addressing Modes	
	2.2	Instruction set-Data Transfer Instructions, String Instructions, Logical Instructions, Arithmetic Instructions, Transfer of Control Instructions, Processor Control Instructions	
	2.3	Assembler Directives and Assembly Language Programming, Macros, Procedures **(Refer Chapters 4, 5, and 6)**	
3		**Memory and Peripherals interfacing**	8
	3.1	Memory Interfacing - RAM and ROM Decoding Techniques – Partial and Absolute	
	3.2	8255-PPI-Block diagram, CWR, operating modes, interfacing with 8086.	
	3.3	8257-DMAC-Block diagram, DMA operations and transfer modes.	
	3.4	Programmable Interrupt Controller 8259-Block Diagram, Interfacing the 8259 in single and cascaded mode. **(Refer Chapters 8, 9, 10 and 11)**	
4		**Intel 80386DX Processor**	7
	4.1	Architecture of 80386 microprocessor	
	4.2	80386 registers–General purpose Registers, EFLAGS and Control registers	
	4.3	Real mode, Protected mode, virtual 8086 mode	
	4.4	80386 memory management in Protected Mode – Descriptors and selectors, descriptor tables, the memory paging mechanism **(Refer Chapter 12)**	
5		**Pentium Processor**	6
	5.1	Pentium Architecture	
	5.2	Superscalar Operation,	
	5.3	Integer &Floating-Point Pipeline Stages,	
	5.4	Branch Prediction Logic,	
	5.5	Cache Organization and	
	5.6	MESI protocol **(Refer Chapter 13)**	
6		**Pentium 4**	4
	6.1	Comparative study of 8086, 80386, Pentium I, Pentium II and Pentium III	
	6.2	Pentium 4: Net burst micro architecture.	
	6.3	Instruction translation look aside buffer and branch prediction	
	6.4	Hyper threading technology and its use in Pentium 4 **(Refer Chapter 14)**	

MICROPROCESSOR LABORATORY (CSL404)

Lab Code	Lab Name	Credits
CSL404	Microprocessor Lab	1

Prerequisite : *Basic knowledge digital integrated circuits*

Lab Objectives :

- *To emphasize on use of Assembly language program.*
- *To prepare students for advanced subjects like embedded system and IOT.*

Lab Outcomes : *At the end of the course, the students will be able to*

- *Use appropriate instructions to program microprocessor to perform various task*
- *Develop the program in assembly/ mixed language for Intel 8086 processor*
- *Demonstrate the execution and debugging of assembly/ mixed language program*

Suggested List of Laboratory Experiments/Assignments(any 10)

Sr. No.	Title of Experiments
1	Use of programming tools (Debug/TASM/MASM/8086kit) to perform basic arithmetic operations on 8-bit/16-bit data
2	Code conversion (Hex to BCD and BCD to Hex)/ (ASCII to BCD and BCD to ASCII)
3	Assembly programming for 16-bit addition, subtraction, multiplication and division (menu based)
4	Assembly program based on string instructions (overlapping/non-overlapping block transfer/ string search/ string length)
5	Assembly program to display the contents of the flag register.
6	Any Mixed Language programs.
7	Assembly program to find the GCD/ LCM of two numbers
8	Assembly program to sort numbers in ascending/ descending order
9	Any program using INT 10H
10	Assembly program to find minimum/ maximum number from a given array.
11	Assembly Program to display a message in different color with blinking
12	Assembly program using procedure.
13	Assembly program using macro.
14	Program and interfacing using 8255.
15	Program and interfacing of ADC/ DAC/ Stepper motor.

❑❑❑

MODULE 1

MODULE 2

MODULE 1

MODULE 3

MODULE 4

1 Fundamentals of Microprocessors

MODULE 1

1.1 Basics of Digital Systems

1.1.1 Analog Signal

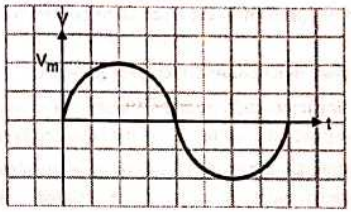


Fig. 1.1.1 : Analog signal

- A signal that continuously varies with respect to time is called as an analog signal. It is represented by an expression like $v(t) = V_m \sin \omega t$ as shown in Fig. 1.1.1.
- For such a signal, to indicate the value at any given instant, we require decimal numbers which may be fractional also i.e. digits 0 to 9.

1.1.2 Digital Signal

- A signal that varies discretely with respect to time is called as a digital signal. It requires only two levels for representation. i.e. 0 or 1. These levels are normally given as 0V and 5V respectively. If there is a slight variation in the voltage level i.e. 5V reduced to 4.7V, it is obvious that there is no option for 4.7V, hence it has to be 5V or logic '1'. Hence, there is less scope for any errors when it is to be transmitted or transferred from one device to another.
- It normally uses binary number system ('0' or '1'). '0' represents logic low (0V) and '1' represents logic high (5V). Fig. 1.1.2 shows a digital signal. Hexadecimal number system or octal number system is derived from binary number system by grouping 3 or 4 bits, to make it convenient for programmers.

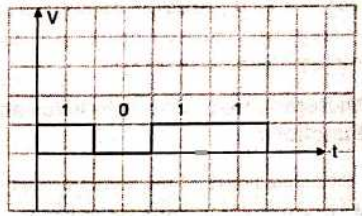


Fig. 1.1.2 : Digital signal

1.2 Basics of Microprocessors

- When we hear the word microprocessor, what comes to our mind is a small (i.e. micro) IC (integrated circuit) that can process data i.e. perform arithmetic and logical operations.

1.2.1 Some Basic Terms Used in Microprocessors

1. **Opcode :** A binary code, that indicates the operation to be performed is called as an Opcode.
2. **Operands :** The data on which the operation is to be performed (as well as the result of an operation) are termed as operands.
3. **Instruction :** The combination of opcode and operands, that can be used to instruct a system, is called as an instruction.
4. **Instruction set :** A list of all the instructions that can be issued to a system, is called as instruction set of that system.
5. **Program/subroutine/routine :** A set of instructions written in a particular sequence, so as to implement a given task is called as a program. A subroutine in assembly language refers to function as of High level languages like C/C++. The term routine is used in a special case called as Interrupt service routine (ISR).
6. **Bus :** A group of lines, pins or signals having common function is termed as bus.

   The functional grouping of signals results in

   (a) Data bus i.e. signals used to carry data

(b) Address bus i.e. signals to address or select a memory or I/O location.

(c) Control bus are signals to issue and receive control operations.

7. **Register :** It is a flip-flop based circuit used to store the data or as the name says, register the data into it. It is as good as a PIPO (parallel in parallel out) register. These are present inside the processor to store data temporarily. These are few in count.

1.2.2 Microprocessor Characteristics and Functions

The computing power of the microprocessor can be determined by certain characteristics listed below:

1. **The processor size or number of bits :** For example, processors are 8-bit, 16-bit, 32-bit, 64-bit etc.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 on the number of instructions supported by a processor. It also depends on the different types of data supported for execution by an instruction like binary, BCD, ASCII etc.
3. **Clock frequency :** The different operations that a microprocessor does on the external bus are memory read, memory write, input read and output write. The internal operations involve transfer of data between storage, arithmetic operations, logical operations etc. All these operations are synchronized with the clock. The frequency of this clock decides the speed of the processor.
4. **Width of the address bus :** The address bus width decide the number memory locations and I/O locations it can access. For a microprocessor, access refers to reading and writing a location.
5. **Interrupt capability :** The number of hardware and software interrupts for a processor depends on the number of interrupt pins and number of interrupt instructions respectively. Different types of interrupts like vectored/non-vectored, maskable/non-maskable have their importance, we will study the same later in this chapter.

1.2.2(A) Functions of a Microprocessor

University Question

Q. Explain Basic functions of microprocessor. **MU - May 14, 5 Marks**

Microprocessor is an IC that can fetch, decode and execute an instruction. Thus microprocessor is a semiconductor chip, which can fetch or get the instruction from the memory. The instruction fetched is in binary form and hence, it decodes the instruction. After decoding, the operation as specified in the instruction is executed by the processor.

The basic functions of a microprocessor are :

1. Fetching instruction from the memory location pointed by the program counter. (Fetching means to bring)
2. Decoding the instruction fetched. Since the instruction is coded in binary form, it is to be decoded to find out the operation to be performed.
3. Executing the instruction. Once the instruction is decoded, the required operation is to be performed. The various operations performed by the processor are :
   (a) Transferring a data from one location to another in the system. This could be transferring of data from memory to the processor, memory to the Input/Output device, from processor to memory etc.
   (b) Performing different arithmetic operations like addition, subtraction, increment, decrement etc.
   (c) Performing various logical operations like AND, OR, EXOR etc.
   (d) Performing branching operations

There are instructions required to perform the above functions. We will study these instructions in Chapter 2.

1.2.3 Evolution of Microprocessors

Q. Write a short note on evolution of microprocessors. Give one examples of each generation. **(3 Marks)**

Intel began with developing the first microprocessor of 4-bit i.e. 4004. This was the first generation of microprocessors. Few processors in each generation are listed below.

1.2.3(A) Generations of Microprocessors

1st Generation (1971 to 1973)

- The first ever microprocessor 4004 was designed and manufactured by INTEL in 1971. It operated at a speed of 108KHz and was a 4-bit processor.
- There were other 4-bit processors manufactured in this generation by other companies like Rockwell international, National Semiconductors etc. INTEL also developed an 8-bit microprocessor named 8008 in this generation.

2nd Generation (1973 to 1977)

- For the first time a general purpose processor 8080 was developed by INTEL.
- This followed by many other processors like Zilogs-Z80 processor, Motorola processors 6800 and 6801, INTEL processor 8085, etc.

3rd Generation (1978 to 1980)

During this generation, microprocessors were developed to be used in computers. INTEL developed a 16-bit microprocessor 8086 followed by its advanced versions namely 80186, 80286 etc. Motorola also developed its processors 68000 and 68010 in this generation

4th Generation (1981 to 1995)

This was the generation of 32-bit processors. INTEL developed its 32-bit processors like 80386 followed by 80486.

Motorola in this period also developed 32-bit processors like 68020 and 68030. Besides these HP also developed a 32-bit processor called HP-32

5th Generation (1995 till today)

- Finally, there were 64-bit microprocessors in the market. Although not all attributes of these processors are 64-bit, but certain attributes like the data bus was 64-bit. It includes INTEL Pentium processor.
- There were also multi core processors in this generation like dual core and core 2 duo, developed by INTEL in this generation.
- There are 6th, 7th etc. generation processors also available today in markets for processors like INTEL i5, and i7.

1.2.4 Block Diagram of a Microcomputer

Q. Draw a neat simplified block diagram of CPU architecture of micro-computer. **(5 Marks)**

A computer or a microcomputer mainly consists of three components, namely

1. Microprocessor or the CPU
2. Memory
3. Input/Output Devices

- Besides the above three, it also consists of busses that connect the three components of the microcomputer.

- Fig. 1.2.1 below show the block diagram of a microcomputer.

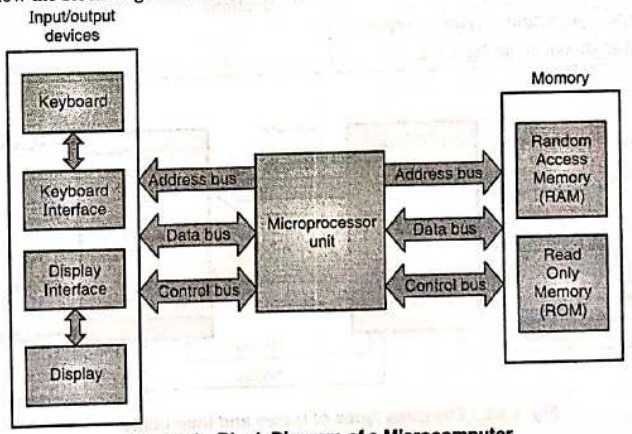


Fig. 1.2.1 : Block Diagram of a Microcomputer

1. Microprocessor Unit

- The microprocessor unit or the microprocessor contains arithmetic and logical unit (ALU) as well as a control unit (CU).
- The ALU as the name says is responsible for all the arithmetic and logical operations done by the processor.
- The CU controls not only the components inside the microprocessor unit, but also outside the microprocessor in the system. It produces control signals to be issued to the memory as well as the Input/Output (I/O) devices.

2. Memory

- It mainly contains Random Access Memory (RAM) and Read Only Memory (ROM) as shown in the Fig. 1.2.1.
- RAM is normally used to store temporary data as RAM is volatile i.e. the data is lost as soon as the supply goes off. ROM is used to store permanent data as it retains data even after the system is switched off. Hard disk, CD are various examples of ROM

3. I/O devices

- Each input device or output device normally works at a very slower speed than the microprocessor. Also each input or output device follows a different method of data transfer.
- Hence, for each input or output device a separate interface is required as shown in the Fig. 1.2.1.
- Keyboard interface to interface the keyboard to microprocessor unit and similarly display interface to interface display to the microprocessor unit.

4. Buses

- To connect the different components of a microcomputer, we need various connections.
- These connections having common functions when combined together form a bus. Thus we have three busses namely data bus, address bus and the control bus.

1.2.5 Buses and Memory Accessing

Q. Explain the terms in a micro-computer :
(i) Address bus (ii) Data bus
(iii) Control Bus **(3 Marks)**

1.2.5(A) Address Bus

- A set of lines that are used to select a memory or I/O location forms a bus (group of lines) called as the **address bus.**
- The address lines are used to select the location of memory or input/output device to be read or to be written on.
- The number of locations accessible by a processor depends on the number of address lines. Thus if there are 'n' lines, the processor can access 2^n memory locations.

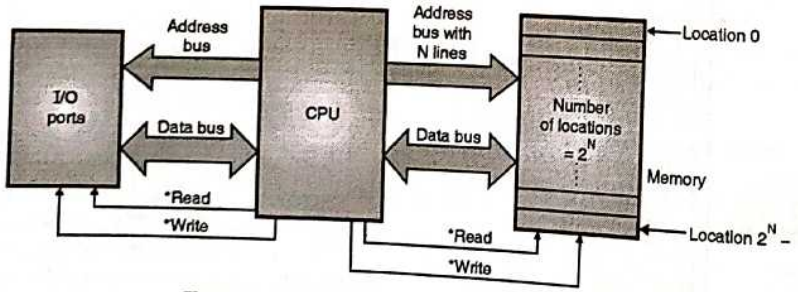


Fig. 1.2.2 : The three types of buses and their utility

- As processor selects the location it wants to access, the address bus is a unidirectional bus (indicated by the arrow).

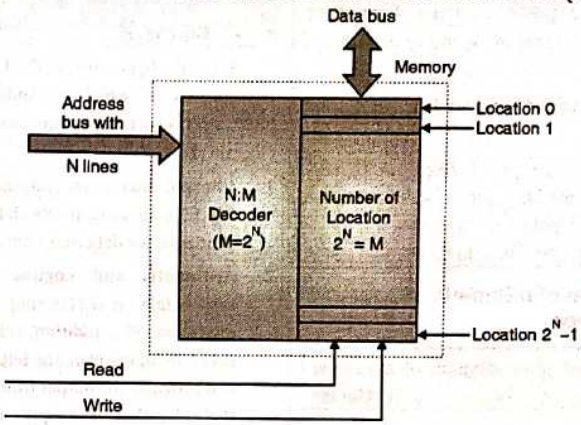


Fig. 1.2.3 : Structure of Memory

- If the number of address lines, N = 16 then it can address 2^{16} = 65,536 or 64K memory locations.

$2^0 = 1$	$2^1 = 2$
$2^2 = 4$	$2^3 = 8$
$2^4 = 16$	$2^5 = 32$
$2^6 = 64$	$2^7 = 128$
$2^8 = 256$	$2^9 = 512$
$2^{10} = (1024)$ 1 K (1 Kilo)	$2^{20} = 1\text{ K} * 1\text{ K} = 1$ M (1 Mega)
$2^{30} = 1$ G (1 Giga)	$2^{40} = 1$ T (1 Tera)

Ex. 1.2.1 : A processor has 24 address lines, how many memory locations it can access.

Soln. :

$$2^{24} = 2^4 \times 2^{20} = 16 \times 1\text{ M}$$
$$= 16\text{ MB}$$

Ex. 1.2.2 : A processor has 4 GB memory, how many address lines are required to access this memory.

Soln. :

$$4\text{ GB} = 4 \times 1\text{ G} = 2^2 \times 2^{30}$$
$$= 2^{32}$$

Hence, 32 address lines are required.

1.2.5(B) Data Bus

- These lines are used to carry data between memory, processor and I/O devices.
- The size of data bus decides the number of bits that the processor can access simultaneously from memory or I/O device.
- The address lines select a location to be accessed (read/write), the data read or to be written is available on data bus.

1.2.5(C) Control Bus

- Since the same address lines are used for memory as well as I/O, there has to be some signalling that indicates the address is to access memory or I/O.
- Similarly, since same data bus is used to read or write data, some control signalling is required to indicate the operation to be carried out is read or write.
- Such signals are called as control signals. Some of the control bus signals are as follows :
  1. Memory read 2. Memory write
  3. I/O read 4. I/O write

Combined utilization of the three buses

- In practice the CPU needs to make use of all the buses in order to perform a desired operation.
- For example if a byte (8 bits) is to be read from the memory location 18F8H, then the sequence of operations followed by the CPU is as follows :

– Task : Read data from memory location 18F8H

Step 1 : CPU sends out "18F8H" i.e. the address of the desired memory location, on the 16-bit address bus.

Step 2 : CPU sends the memory read signal on the control bus.

Step 3 : The memory read signal enables the memory device to put the data stored in the location 18F8H onto the data bus. This data travels on the data bus from memory to CPU.

1.2.6 Block Diagram of a Generic Microprocessor

Q. Draw neat labeled block diagram of a generic microprocessor. **(3 Marks)**

Q. Write the function of following unit in the microprocessor 8085 : Accumulator, Program Counter **(1 Mark)**

Fig. 1.2.4 below shows the block diagram of a generic microprocessor (which is similar to the architecture of 8080 microprocessor). The various units in this block diagram are :

1. **Internal bus :** The internal bus as shown in the Fig. 1.2.4 above connects all the components inside the processor for data communication between them.
2. **Arithmetic and Logical Unit (ALU) :** ALU is responsible for performing all Arithmetic and Logical operations like addition, subtraction, AND, OR etc. It gets two inputs from the left side (as shown in the fig) and provides the output from right. It also gives output to flag flip-flops. The inputs are taken from the register called accumulator and from the internal bus.

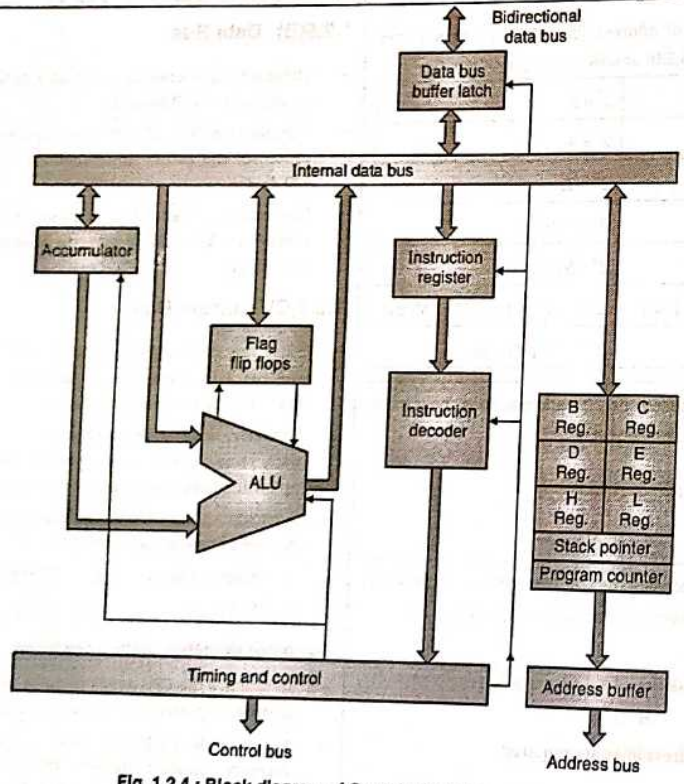


Fig. 1.2.4 : Block diagram of Generic Microprocessor

3. **Flag Flip flops :** There are special flip-flops that give the status of the result of ALU operation. Many times it is not required to know the exact result, but only its status is required, in such cases these flip-flops are useful. For example, when we compare two numbers, the processor performs subtraction operation. Now, to know which number is greater, we need not know the exact difference, instead only the sign of the result (positive or negative) is enough for us. Similarly whether the result is zero or non-zero will tell if the two numbers are equal or not.
4. **Accumulator :** It is a general purpose register to store temporary data. It provides one input to the ALU for its operation and also stores (or accumulates) the result given by ALU.
5. **Timing and Control Unit :** It provides timing signal for synchronization of all the components inside and outside the processor and also provides the control signals. As seen in the Fig. 1.2.4, it provides control signals to all units inside the processor as well as on external control bus.
6. **Instruction register and Instruction decoder :** The Instruction register holds the opcode of the instruction in execution. The opcode is decoded by the instruction decoder, for the processor to understand the operation to be performed. The decoder then indicates the timing and control unit, as to what operation is to be done, which in turn performs the said operation by issuing control signals in that accordance.
7. **Program Counter :** This is a register that points to the next instruction to be executed. Hence, whenever the processor executes a particular instruction, the address of next instruction to be executed is given by program counter
8. **Stack pointer :** It is a pointer that points to the top of stack. Stack is used to store the data of incomplete operations to be completed in near future. Hence, if the processor is performing a given task, and something more important is to be done in between; the processor will put the current task data on the stack memory and start the new important task. Once the important task is completed, the processor will be able to resume the incomplete task by taking the data kept on stack. We will understand more about stack by reading the instructions related to stack in next chapter.
9. **Register pair HL :** Memory contains three things namely, program, stack and data. Program counter is used to point to the program, stack pointer for stack and similarly register pair HL for data. Thus, HL pair is used to point to the data to be accessed by an instruction.
10. **General purpose registers (B,C,D and E) :** These registers as the name says are general purpose registers. Hence the programmer can use them for whatever purpose required in their programs.
11. **Data bus buffer latch :** This is used to connect the internal bus of the processor to its external bus. If the data is to be communicated within the processor, internal data bus is used. But if the data is to be communicated between memory and processor or memory and I/O device the data has to be on external data bus communicated with processor through this buffer.
12. **Address buffer :** Whenever the processor has to access an external location (memory or I/O), it gives the address on the address bus through this buffer.

1.2.7 Generic ALU (Arithmetic and Logic Unit)

Q. Explain the working /organization/ functions of ALU with simple block diagram. **(4 Marks)**

– This unit is used to perform arithmetic operations (like add, subtract, multiply, divide etc). and logical operations like (AND, OR, EX-OR etc).

– It consists of two inputs operands(4 bits in case of the ALU in Fig. 1.2.5) 'A' and 'B' and one output operand i.e. 'F'. 'S' are the select lines to select the operation (OPCODE). Besides these, there is also an input called as mode select for selecting Arithmetic or Logical operations.

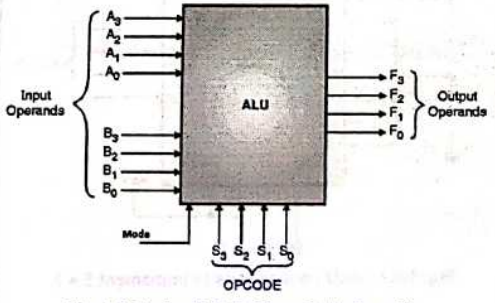


Fig. 1.2.5 : An ALU (Arithmetic logic unit)

An ALU comprises of various blocks as shown in the Fig. 1.2.6 below :

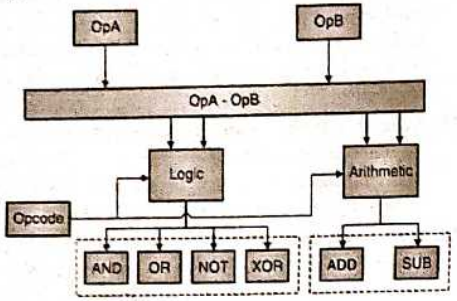


Fig. 1.2.6 : Block diagram of Generic ALU

- As shown in the figure, it comprises two registers OpA and OpB, to store the two operands to operate on. It has an input called as Opcode, which indicates the operation to be performed. There are two units
  1. Logic to perform various logical operations like AND, OR, NOT and EXOR
  2. Arithmetic to perform various operations like addition, subtraction etc.
- Now in the Fig. 1.2.6, to give an instruction to the system (ALU), the programmer has to give OPCODE and OPERAND i.e. the operation to be performed is to be given on the select lines while the data to be operated on is to be given on input operand lines.
- For e.g. if the operation is 5 + 7 i.e. $(0\ 1\ 0\ 1)_2$ + $(0\ 1\ 1\ 1)_2$ the data lines of A are to be $(0\ 1\ 0\ 1)_2$, while that of B are to be $(0\ 1\ 1\ 1)_2$.

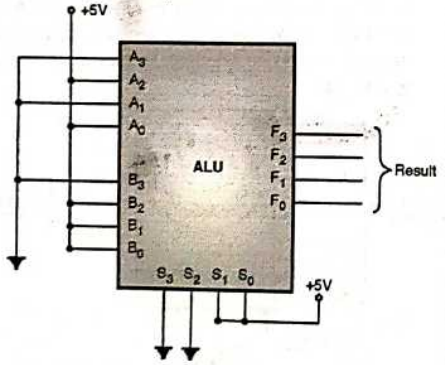


Fig. 1.2.7 : ALU connections to implement 5 + 7

- A binary code for addition is to be given on select lines. Suppose $(0\ 0\ 1\ 1)_2$ refers to add operation for this ALU. So the connections are to be made by the programmer as shown in Fig. 1.2.7.
- To do these connections, the programmer may take few seconds, while the IC can perform the operation in microseconds.
- Hence, the time in which the ALU could have performed few million operations, it can perform only one operation.
- The reason for this delay is the slow speed of the programmer to enter the instruction.
- Hence if the instructions were stored in a memory and the processor accesses them, it would be much faster.
- Thus one major additional aspect of a microprocessor over and ALU is that it can take instructions stored in memory, called as stored program concept.

1.3 8085 Microprocessor

- 8085 is an 8-bit microprocessor i.e. it can perform 8-bit operations simultaneously.
- It also has 8-bit registers, internal and external data bus. It was not used in making of computers but was mainly used in making of systems automated i.e. automatic control of systems.

1.3.1 Features of 8085 Microprocessor

Q. Write the features of 8085 microprocessor. (4 Marks)

The features of a processor can be divided into three broad groups viz. basic features, special features and miscellaneous features.

1.3.1(A) Basic Features of 8085

1. 8085 is a 8-bit microprocessor. This implies that :
   (a) It has 8-bit ALU that can perform 8-bit operations simultaneously.
   (b) It has 8-bit internal data bus and registers.
   (c) It has 8-bit external data bus.
2. It has three versions based on frequency of operation.
   (a) 8085→ 3 MHz (b) 8085-2 → 5 MHz
   (c) 8085-1 → 6 MHz

3. 8085 has 16-bit address bus to access memory, hence it can access $2^{16} = 2^6 \times 2^{10} = 64 \times 1K = 64$ KB memory locations.
4. It has 8-bit address bus to access I/O locations, hence it can access $2^8 = 256$ I/O locations.

1.3.1(B) Special Features of 8085

1. 8085 was the first processor that required only single +5V power supply.
2. 8085 has a full duplex serial port with 2 pins to receive and transmit serial data.
3. It has an on-chip clock generator. Hence there is no need for external clock generators, instead only a crystal is to be connected to 8085
4. It is an accumulator based architecture i.e. for any ALU operation, accumulator provides one of the input as well as stores the result

1.3.1(C) Miscellaneous Features of 8085

1. 8085 has 5 hardware interrupts and 8 software interrupts. Software interrupts are vectored. Out of the 5 hardware interrupts four are vectored, while one is non-vectored. Also four of the hardware interrupts are maskable and one is non-maskable interrupt. We will understand about these terms in Section 1.3.4.
2. 8085 has following registers
   (a) 8-bit accumulator
   (b) Six 8-bit general purpose registers named as B, C, D, E, H and L.
   (c) Flag register
   (d) 16-bit program counter and
   (e) 16-bit stack pointer
3. 8085 has a powerful instruction set that can do various arithmetic operations like 8-bit addition, 16-bit addition, 8-bit subtraction, increment and decrement of 8 bit as well as 16 bit. Also it can perform logical operations like AND, OR, EXOR and NOT.
4. 8085 can perform operations on bit, byte and some word (16-bit) data. It can work with binary as well as decimal data.

1.4 Advantages of Microprocessor

Q. Explain advantages of microprocessor. (5 Marks)

The advantages of microprocessors are :

(i) Microprocessors can be programmed to perform the tasks as an when required.
(ii) They do not require any special maintenance.
(iii) They can access (read or write) the memory as well as I/O devices.
(iv) They are cheaper and small in size.
(v) They have a versatile application and are required in almost all modern electronic devices.
(vi) They can process data.
(vii) They have good reliability.

1.5 Disadvantages of Microprocessor

Q. Explain disadvantages of microprocessor. (5 Marks)

The disadvantages of microprocessor are :

(1) Microprocessor is a general purpose entity. It has many attributes that may not be required for a particular application. But it does require the support of external memory and I/O controllers. To interface any I/O device to a microprocessor, we need to interface it through an I/O controller. For example, to interface a simple LED, we need an 8255 interfaced to microprocessor.
(2) Also memory is required externally as the microprocessor doesn't have any internal memory.Any application will require memory for storage of program and data.
These additional requirements of a microprocessor based system, increases the size of microprocessor.

1.6 Applications of Microprocessor

Q. Explain applications of microprocessor. (5 Marks)

- The microprocessor has its main use found in computers. But besides that, there are many places where a processor is used to control the system.

- Many systems like microwave oven, washing machine, air conditioner etc. have microprocessors in them to control their operations. Such systems, wherein a microprocessor is embedded into it to control the system, are called as embedded systems.
- Even mobile phones, cars etc. have microprocessors in them to control various operations.
- Thus microprocessors and microcontrollers have a wide range of applications besides computers. Let us list few of these applications:
  1. Many home appliances like Microwave oven, washing machine, Television (TV), Air Conditioner (AC) etc. have microprocessors in them to control their operation
  2. Most of the electronic gadgets like mobile phones, cameras, tablets, health band etc. have microprocessors.
  3. Electronic weighing scale, digital thermometer and many such instruments have microprocessors embedded in them.
  4. It has a wide application in automation systems like home automation and home security system.
  5. Microprocessors also have their application in Robotics, automated car etc.
  6. Medical instruments like ECG, lung function test, X-Ray machine etc. also use microprocessors.

1.7 Comparison between 8085, 8086, 8088, 80186, 80286, 80386 and 80486 Microprocessors

Processor	Address Lines	Data Lines	Memory
8085	16 bit	8 bit	64KB
8086	20 bit	16 bit	1 MB
8088	20 bit	8 bit	1 MB
80186	20 bit	16 bit	1 MB
80286	24 bit	16 bit	16 MB
80386	32 bit	32 bit	4 GB
80486	32 bit	32 bit	4 GB

1.8 Exam Pack (Review and University Questions)

Q. 1 Explain Basic functions of microprocessor. **(Refer Section 1.2.2(A))** **(May 14, 5 Marks)**

Q. 2 Write a short note on evolution of microprocessors. Give one examples of each generation. **(Refer Section 1.2.3)** **(3 Marks)**

Q. 3 Draw a neat simplified block diagram of CPU architecture of micro-computer. **(Refer Section 1.2.4)** **(3 Marks)**

Q. 4 Explain the terms in a micro-computer :

(i) Address bus (ii) Data bus

(iii) Control Bus **(Refer Section 1.2.5)** **(3 Marks)**

Q. 5 Draw neat labeled block diagram of a generic microprocessor. **(Refer Section 1.2.6)** **(3 Marks)**

Q. 6 Explain the working /organization/ functions of ALU with simple block diagram. **(Refer Section 1.2.7)** **(4 Marks)**

Q. 7 Write the features of 8085 microprocessor. **(Refer Section 1.3.1)** **(3 Marks)**

Q. 8 Write the function of following unit in the microprocessor 8085 : Accumulator , Program Counter **(Refer Section 1.2.6)** **(1 Mark)**

Q. 9 Explain advantages of microprocessor. **(Refer Section 1.4)** **(5 Mark)**

Q. 10 Explain Disadvantages of microprocessor. **(Refer Section 1.5)** **(5 Mark)**

Q. 11 What are the applications of Microprocessors. **(Refer Section 1.6)** **(5 Mark)**

□□□

MODULE 1

2 The Intel 8086 Architecture

2.1 Introduction

Initially microprocessors were 4-bit, then 8-bit and 8086 was the first 16-bit processor. Also 8086 was the first microprocessor manufactured to be used in computers. Until this microprocessors were mainly used in embedded systems.

In this chapter we will see the features of 8086, its architecture and basic components required to help us doing better programming as well as designing systems using 8086. Remember, this is a very important chapter as it covers the basics of 8086. Without these basics, programming as well as designing will be almost impossible

2.2 Features of 8086

The features of a processor can be divided into three broad groups viz. basic features, special features and miscellaneous features.

2.2.1 Basic Features of 8086

1. Processor size
2. Speed of processor
3. Address bus size for memory
4. Address bus size for I/O

(1) It is a 16-bit processor. This implies that

(a) It has a 16-bit ALU that can perform 16-bit operation simultaneously.

(b) It has 16-bit registers and internal data bus.

(c) It has 16-bit external data bus.

(2) It has three versions based on the basis of frequency of operation.

(i) 8086 → 5 MHz.

(ii) 8086-2 → 8 MHz.

(iii) 8086-1 → 10 MHz.

(3) 8086 has 20-bit address lines to access memory, hence it can access,

2^{20} = 1 MB memory locations

(4) It has 16-bit address lines to access I/O devices, hence it can access,

$2^{16} = 2^6 \times 2^{10} = 64 \times 1\,K$

= 64 K I/O locations

2.2.2 Special Features

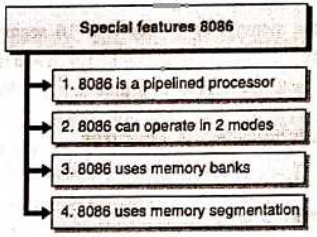


Fig. 2.2.1 : Special Features of 8086

1) 8086 is a pipelined processor

- It uses a two stage pipelining i.e. **Fetch stage** that pre-fetches up to 6 bytes of instructions stores them in the queue and **Execute stage** that executes these instructions.
- Pipelining improves the performance of the processor i.e. the operations are faster.

2) 8086 can operate in 2 modes

a. **Minimum mode :** A system with only 1 processor i.e. 8086.

b. **Maximum mode :** A system with 8086 and other processors like 8087-(Math Co-processor), 8089-(IO processor) or multiple 8086 processors

3) 8086 uses memory banks

- The 8086 uses a memory banking system i.e. the entire data is not stored sequentially in a single memory of 1 MB but the memory is divided into two banks of 512KB each.

- The banks are called Lower bank (or even bank, because it stores the data bytes at even locations i.e. 0, 2, 4....) and Higher Bank (or odd bank, because it stores the data bytes at odd locations i.e. 1, 3, 5...).
- The benefit of this is that 16-bit data can be accessed in a single access even though the memory chip can store only 8-bit at a location

4) 8086 uses memory segmentation

- A 16-bit address in an instruction or a 16-bit address in a register can access a memory location, although 8086 has 20 address lines. This is made possible using the concept of Segmentation that divides the memory into logical components.
- Here the memory is divided into 16 segments of a capacity of 2^{16} (= 65536 B = 64 KB) each and is used as: Code, Stack, Data and Extra Segment.

2.2.3 Miscellaneous Features

1. Interrupts
2. Registers
3. Instruction set
4. Data size for ALU
   1. **It has 256 vectored interrupts :** There are also non-vectored interrupts in 8086, but they are routed to one of these interrupts.
   2. It has 14, 16-bit registers.
   3. It has a powerful instruction set, that supports MULTIPLY AND DIVIDE operations also. (These operations were not possible in the processors earlier to 8086).
   4. 8086 can perform operations on bit, byte (8-bit), word (16-bit) or a string (block of data) types of data.

2.3 8086 Internal Architecture

University Question

Q. Explain architecture of 8086.

MU - Dec. 13, May 14, 10 Marks

- Before talking about programming, we need to discuss the special features and internal architecture of Intel 8086.
- Fig. 2.3.1 shows the block schematic of the internal structure of Intel 8086.

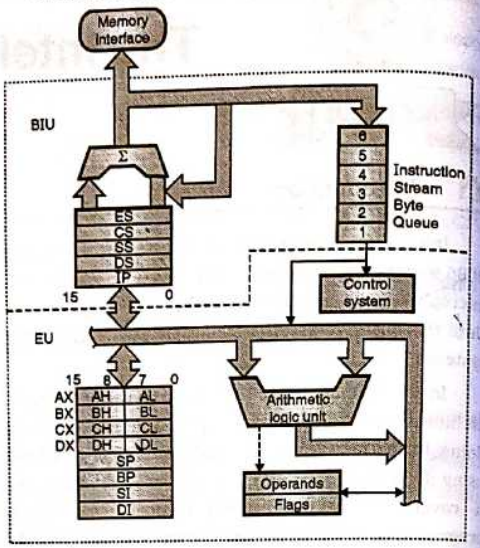


Fig. 2.3.1 : 8086 flag register format

- As shown in Fig. 2.3.1, the 8086 CPU is divided into two sections namely:
  1. The bus interfacing unit (BIU)
  2. The execution unit (EU).
- Bus Interface Unit (BIU) is mainly responsible for external accesses i.e. read or writing from memory or I/O devices.
- Hence it is called the external world interface of the processor.
- It works in synchronous with machine cycles. In the subsequent chapter we will see machine cycles.
- Execution Unit (EU) is the main processing section of the processor. It is responsible for doing all calculations, arithmetic and logical operations. It also controls the different operations in the processor.
- EU takes care of performing operations on the data.
- EU is called as the execution heart of the processor.
- It works in synchronous with t-states. We will also see the concept of t-states in subsequent chapter.

2.3.1 The Execution Unit (EU)

The execution unit consists of the following:

1. Arithmetic Logic Unit (ALU)
2. Flag Register.
3. General Purpose Registers.
4. Control Unit – Need explanation
5. Decoder – Need explanation
6. Pointer and Index Registers

2.3.1(A) Arithmetic Logic Unit (ALU)

- 8086 is a 16-bit processor and hence has a 16-bit ALU that can perform 16-bit operations simultaneously.
- ALU performs all arithmetic logic operations and also updates the flag register discussed in the next sub-section.
- It also executed conditional instructions that check the value of flag register and accordingly take decisions.

2.3.1(B) Flag Register

University Question

Q. Write short note on : State use of control flags of 8086. **MU - May 15, 5 Marks**

- It is a set of flip-flops that give the presence or absence of a particular attribute. The flip-slops together make a 16-bit register.
- Although there are only 9 flip-flops, but they are arranged in such a form so as to make a 16-bit register. Hence, 7 are unused while only 9 are used.
- The name "flag" is derived from the concept that there are only two possibilities, presence or absence similar to the green and red flag shown to a train to indicate the train either to go or to halt.
- The nine flags are divided into two groups i.e. Control flags and Status flags. The control flags as the name states controls certain operation of the processor while the status flags give the status of the result obtained by ALU.

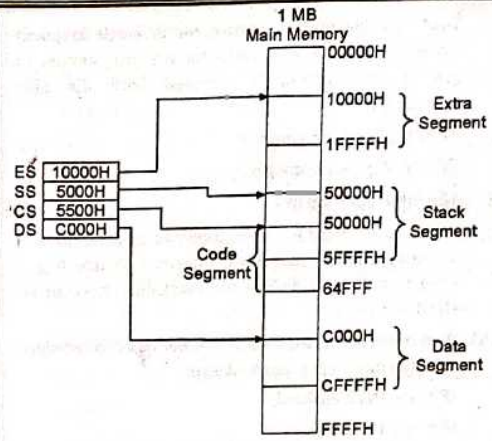


Fig. 2.3.2 : 8086 flag register format

Control flags

- As discussed, out of the nine flags of 8086, three are control flags that are used to control certain operation of the processor.
- The three control flags are :

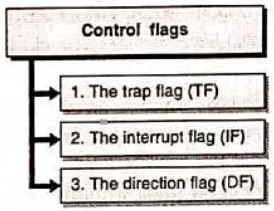


Fig. 2.3.3 : Control Flags

1. The Trap Flag (TF)

(i) This flag is used for debugging a program. To see the stepwise execution of a program, this flag should be set to '1'.

When this flag is set to '1', the processor executes, one instruction at a time and after that, the user/programmer can check the status and contents of various registers and flags.

This will help the user to detect the error in the program or help the programmer debug the program. "Bug" means a problem in program, "Debug" refers to removal of the bug.

Thus this flag puts the processor in single stepping mode i.e. the processor waits for the programmer to give a confirmation to proceed with the next instruction.

TF = '1' – Single stepping on,

TF = '0' –Single stepping off

2. The Interrupt Flag (IF)

(i) As discussed in the features, 8086 has maskable as well as non-maskable interrupts. Interrupt Enable flag is used to enable or disable the maskable interrupt i.e. INTR.

(ii) If an interrupt is disabled, it will not be acknowledged until the interrupt is enabled again.

IF = '1' – INTR enabled,

IF = '0' - INTR disabled

3. The Direction Flag (DF)

(i) 8086 supports string instructions. These are a special type of instructions that can work on an entire array of data. Since single instruction can do the operation on an entire array of data, the time required to fetch and decode the instruction again and again (as done in the processors without string instructions) will reduce drastically and hence the execution of such tasks can be done at a faster speed.

(ii) These instructions require a set of special pointers to point to source and destination data blocks called as Source Index (SI) and Destination Index (DI). For example one of the string instruction called as MOVS, transfers a block of data from one place in memory to another, while the traditional method requires multiple instructions along with a loop. In such instructions, SI points to source block, while DI points to the destination block.

(iii) Direction Flag (DF) controls auto-increment or auto-decrement of these pointers. Thus if we start the operation from the top of the block, we will keep direction flag to '0', so that the pointers are in auto-increment mode; while if we start from the last location of the array, then we will keep the direction flag reset to '0', so that the pointers are in auto-decrement mode.

DF = '1' - Up , DF = '0' - Down

Status flags

- Thus we have discussed three control flags, the remaining six are called as status flags or conditional flags.
- The names of the conditional flags indicate what conditions affect them. For example the Carry Flag (CF) is set to 1 if addition of two numbers produces a carry output.
- Status flags give the status of ALU operation. Hence whenever ALU performs an operation, the status flags are altered automatically to give the status of the result. Let us understand what status is given by these flag bits.
- Some of the 8086 instructions check the status of the conditional flags, before execution.
- The six conditional flags are :

Fig. 2.3.4 : Conditional flags

1. The Parity Flag (PF)

(i) Parity refers to number of '1's in a data. The parity flag indicates whether the number of '1's are even or odd in the lower byte of the result of ALU operation.

PF = '1' – indicates that the lower byte has an even parity

PF = '0' - indicates that the lower byte has odd parity

2. The Zero Flag (ZF)

This flag indicates whether the result of ALU is zero or non-zero.

ZF = '1' - indicates that the Result is zero ,

ZF = '0' - indicates that the result is non-zero

3. The Sign Flag (SF)

(i) This flag is set, when MSB (most significant bit) of the result is 1.

(ii) In other words it copies the MSB of the result.

(iii) Since negative binary numbers are represented (in the 8086 CPU) using standard two's complement notation, the MSB (copied in sign flag) indicates the number is positive or negative.

(iv) SF indicates sign of the result only in case of signed operation.

(v) In case of unsigned operation, sign bit has no significance.

SF = '1' - MSB is 1 (negative for signed operations),

SF = '0' - MSB is 0 (positive for signed operations)

4. The Auxiliary Carry Flag (AF)

(i) It is a carry from lower digit to upper digit i.e. it holds the carry generated by the lower nibble (4-bit) to the upper nibble.

(ii) It is used for BCD operations. We will understand the use better in later chapters when we go through these instructions

AF ='1' - carry out from bit 3 on addition or borrow into bit 3 on subtraction.

AF = '0' - no carry out from bit 3 on addition or borrow into bit 3 on subtraction..

5. The Carry Flag (CF)

(i) It can also be called as a final carry.

(ii) This flag holds the carry generated from the MSB of the ALU operation (8 or 16 bit). In case of subtraction, it holds the burrow required by the MSB.

(iii) It is used when we need to perform multi-word operations. For example, for 32-bit addition, we will first add the lower 16-bit data and then the upper 16-bit data considering the carry generated while adding lower 16-bit. We cannot perform 32-bit addition simultaneously as 8086 is a 16-bit processor.

CF = '1' –indicates that there is a carry generated by MSB,

CF = '0' - no carry out from MSB

6. The Overflow Flag (OF)

(i) It indicates an overflow from the magnitude to the sign bit of result.

(ii) It indicates that the result is outside the range (maximum or minimum representable number) in case of signed operations. In case of unsigned operation, this bit is insignificant.

(iii) Thus if this bit is set to '1' for a signed operation, it indicates that the result bits are to be ignored as the result is outside the range.

OF = '1' - indicates that an overflow occurred in case it is signed operation,

OF = '0' - indicates that no overflow occurred.

Hence we can say for the following bit structure of data, the flags will be affected as indicated in the Table 2.3.1

Table 2.3.1

Flags name	8-bit operation	16-bit operation
Auxiliary carry	Carry from D_3 to D_4	Carry from D_3 to D_4
Carry	Carry from D_7 to D_8	Carry from D_{15} to D_{16}
Overflow	Carry from D_6 to D_7	Carry from D_{14} to D_{15}
Sign	Copy of D_7	Copy of D_{15}

2.3.1(C) General Purpose Registers

- Another important component of the EU is the general purpose registers. As the name says, these registers can be used for general purposes.
- AX, BX, CX and DX are 16-bit general purpose registers while, AH, AL, BH, BL, CH, CL, DH and DL are 8-bit general purpose registers. Infact, AH and AL combined together forms AX as seen in the Fig. 2.4.5. Thus if AX has a data 3C45H, then the data 3C is actually in register AH and 45H in register AL.

- Similarly other 16-bit registers also have their 8-bits components.32-bit Registers - DX : AX together can be used for 32-bit operand.
- But besides being used for general purposes, they also have certain special functions.
- All the operands for ALU operation, will required to be stored in these registers or memory.
- These registers can be used for general purpose computing when their other specialized functions do not interfere.

Special functions of general purpose registers

Although the first four registers AX, BX, CX, and DX are called as "general purpose", each of them is designed to play a particular role in common use:

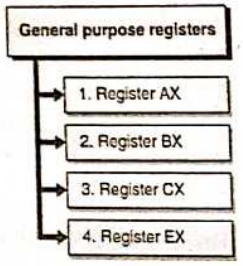


Fig. 2.3.5 : General purpose registers

1. Register AX

AX is the "16-bit accumulator" while AL is "8-bit accumulator"

Accumulator has the following special functions :

(i) Some of the operations, such as Multiplication and Division, require that one of the operands be in the accumulator and also the result is stored in accumulator.

Some other operations, such as Addition and Subtraction, may be applied to any of the registers (that is, any of the eight general- and special-purpose registers) but are more efficient when working with the accumulator.

(ii) It works as a via register for I/O accesses i.e. a data is routed through accumulator for the communication of the processor and I/O devices.

For OUT instruction the data in accumulator (AL for 8-bit data and AX for16-bit data) can only be given to the output device

For IN instruction the data taken from the input device can be taken only in accumulator (AL for 8-bit data and AX for 16-bit data)

(iii) It also works as a via register for string instructions. Whenever a data is to be brought from memory or given to memory in case of string operations it is routed through accumulator only.

2. Register BX

BX is the "base" register;

(i) It is used as a pointer register for indirect addressing mode

(ii) For example, the instruction MOV [BX], AX, will store the contents of 16-bit accumulator AX into the memory location pointed by BX.

3. Register CX

CX *is* the "count" register. It works as a default counter register for three instructions viz :

(i) The looping instructions (LOOP, LOOPE, and LOOPNE), to indicate the number of iterations

(ii) The shift and rotate instructions (RCL, RCR, ROL, ROR, SHL, SHR, and SAR), to indicate number of shifts or rotations (Here only CL is used and not entire CX)

(iii) The string instructions (with the prefixes REP, REPE, and REPNE) to indicate the size of the string block.

4. Register DX

DX is the "data" register

(i) It is used along with AX for the 16-bit MUL and DIV operations, to store the 32-bit product and quotient respectively.

(ii) It also holds the port number for the IN and OUT instructions. For 16-bit address accesses of I/O ports only DX can be used as a pointer.

Summary of Implicit use of General Purpose Registers

Registers	Operations
AX	16-bit accumulator used for 16-bit multiplication and division, I/O and String operations
AL	8-bit accumulator used for 8-bit multiplication and division, I/O and String operations
BX	Stores base address

Registers	Operations
CX	Counter for String operations and Loop instructions
CL	Counter for Variable shift and rotate.
DX	Used for 16-bit multiplication and division. Also used as indirect pointer for I/O accesses

2.3.1(D) Pointer and Index Register

Besides the general purpose registers, there are also certain registers that have a specific task of being used as pointers to memory locations. Memory mainly consists of data, program and stack. Thus, these pointers are to access either of these sections of the memory. These registers are:

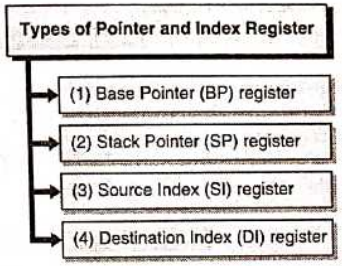


Fig. 2.3.6 : Types of Pointer and Index Registers

Pointer registers provide the offset address in a segment, while the starting address of a segment is given by segment register. We will understand more about this in Section 2.3.2(B) in Segmentation

(1) Base Pointer (BP) register and Stack Pointer (SP) register

- These are 16-bit pointer register to point in the stack section or stack segment of the memory.
- Stack memory has a special use of passing parameters and returning values amongst the functions or procedures
- Stack pointer is used to access the stack memory in sequential manner while Base pointer is used for random access of stack memory

(2) Source Index (SI) register and Destination Index (DI) register

- These registers are used to hold the offset address for data memory or the data segment.
- For string instructions, SI is used to point to the source block while DI is used to point to the destination block.
- 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.

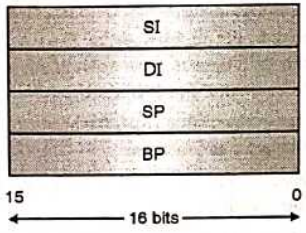


Fig. 2.3.7 : Pointer register of EU

2.3.1(E) Control Unit and Decoder

The decoder takes the instruction from the prefetch queue and after decoding indicates the operation to be performed to the control unit. The control unit sends control signals to all the units inside and outside the processor. These signals are also called as micro-instructions. These micro-instructions are properly sequenced so as to get the instruction executed.

2.3.2 The Bus Interfacing Unit (BIU)

The BIU consists of the following main components :

1. Instruction queue
2. Segment registers
3. An instruction pointer register (IP)
4. Address generation and bus control

2.3.2(A) The Instruction Queue or Implementation of Pipelining

- Pipelining is a special feature of 8086 that increases the speed of the processor. In earlier processors, the process of fetching the instructions and executing them was one at a time or not overlapping. This made the execution unit wait until the instruction is fetched and also made the fetching unit wait when the instruction is under execution.
- In 8086, pipelining refers to overlapping the process of fetching the instructions when the previous instruction is in execution. This is possible because of the prefetch queue
- 8086 has a 6 byte prefetch queue. The size of the queue is kept to be 6-byte so as to accommodate the largest instruction of 8086 which is of 6-byte.
- As the name says, 8086 (BIU) fetches instructions into this queue prior to execution.
- BIU fills in the queue, until the entire queue is full i.e. all 6 bytes are filled.
- BIU restarts filling in the queue when at least two locations of queue are vacant.
- In case, of a branching instruction like jump or call (we will study these instructions in later chapters), the sequence of execution of the program has to be changed. In such a condition, the instructions in the queue are of no use. These are to be removed from the queue and the process is called as flushing the queue. The instructions are then to be filled in the queue from the target location or branching location.

Advantage and disadvantage of pipelining

- As discussed earlier in this section, the main advantage of pipelining is that it increases the speed. This is because, the process of fetching instructions from the memory and the execution of the instructions is overlapping. The BIU fetches the instructions and puts them in the queue when the execution unit is executing previous instructions.
- The only disadvantage of pipelining is that there is requirement of extra hardware in the processor. There are some hardware units that are required for fetching the instructions as well as execution unit. These units are implemented doubly for both fetching unit as well as execution unit, thereby increasing slight cost of the processor.

2.3.2(B) Segmentation in 8086

University Questions

Q. What is segmented memory? State the advantages of segmentation with reference to the 8086 microprocessor. Also explain the default segment assignments.
MU - May 11, 10 Marks

Q. Explain the memory segmentation in Intel 8086 processor with it's advantages and disadvantages.
MU - Dec. 11, May 13, Dec. 18, 10 Marks

Q. What is segmented memory? State the advantages of it with respect to 8086 microprocessor.
MU - May 12, Dec. 12, May 14, 10 Marks

Q. What is segmentation? What are the merits and demerits of segmentation?
MU - Dec. 13, 10 Marks

Q. Write short note on : Advantages of memory segmentation in 8086. Memory can be thought of as a vast collection of bytes. These bytes need to be organized in some efficient manner in order to be of any use.
MU - Dec. 14, 5 Marks

Q. Write short note on: Advantages of memory segmentation in 8086.
MU - May 15, May 19, 5 Marks

Q. What is memory segmentation? State Advantages of memory segmentation.
MU - Dec. 15, May 17, 5 Marks

Q. Explain memory segmentation with pros and cons.
MU - May 16, 8 Marks

Q. Explain the memory segmentation of 8086.
MU - Dec. 17, 5 Marks

- 8086 has 20-bit address bus while the registers are of 16-bit. To access a memory location, we thus need to provide 20-bit address 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 and
  4. The Data Segment (DS) register

As discussed, 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 a the beginning of specific segments. As per the names 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

- Thus 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 we can say shifted left four times. Thus producing a 20-bit address.
- Now 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.
- In the later chapter we will see addressing modes, wherein we will go through the different methods of generating the 20-bit address. But for time being let us take a simple example.
- The CS as discussed 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. 2.3.8.

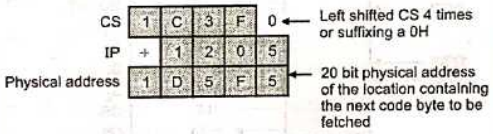


Fig. 2.3.8 : Computation

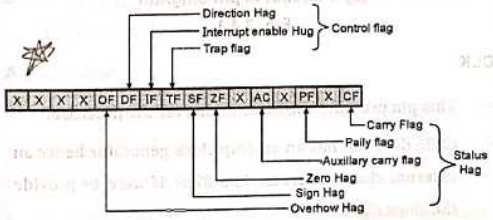


Fig. 2.3.9 : Example of Segments in 8086

Important points to be noted about segmentation in 8086

1. If non-overlapping, there can be 16 segments in all, each of 64KB (since, 1MB / 64KB = 16)
2. But, the segments can overlap each other. Hence there can be many segments.
3. A segment can begin at any location with the only condition that the starting address must be a multiple of 10H
4. At any given time a maximum of 4 segments (since there are 4 segment registers) and hence 256KB can be accessed (if all the 4 segments are non-overlapping).
5. The memory of 8086 is a wrap around memory. This means that once the address generated crosses the 1MB limit, it accesses the location at the top 00000H. For e.g. if CS = FFFF and IP = 0010. Physical address = CS*10H + IP = 100000H. The address lines in 8086 are only 20, so the MSB '1' is discarded and the location being accessed is 00000H.
6. In an instruction only the offset address in mentioned. The segment register is fixed for each of the pointer registers. Default segment register assignments are as follows :

Pointer Register	Default Segment Register
BX	DS
SI	DS
DI	DS (ES in case of string operations)
SP	SS
BP	SS
IP	CS

7. Using segment override prefix, one can change the above default segment register assignments.

Advantages of Segmentation

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. Finally, sharing of data or passing of data from one program to another is easily possible due to segmentation
5. Also the segmentation makes data relocatable as the program uses only offset register pointers while the segment points to the base of a segment.

Disadvantage of segmentation

It becomes complicated for the programmer as multiple registers (segment and pointer registers) to access a memory location.

2.4 Pin Definitions

Table 2.4.1 : Signal Description of 8086

Sr. No	Pins	Name of the pins
I.	Supply pins (3 pins)	V_{CC} GND, GND
II.	Clock related pins (3 pins)	CLK, RESET, READY
III.	Address and Data pins (21 pins)	$AD_0 - AD_{15}$, $A_{16}/S_3 - A_{19}/S_6$, $\overline{BHE}$ / S_7
IV.	Interrupt pins (2 pins)	NMI, INTR
V.	Other control (3 pins)	$\overline{TEST}$, MN / $\overline{MX}$, $\overline{RD}$
VI.	Mode multiplexed signals (8 pins) (MIN mode - MAX mode signals)	HOLD - $\overline{RQ_0}$ / $\overline{GT_0}$, HLDA - $\overline{RQ_1}$ / $\overline{GT_1}$, $\overline{WR}$ - $\overline{LOCK}$, $\overline{DEN}$ - $\overline{S_0}$, DT / $\overline{R}$ - $\overline{S_1}$, M / $\overline{IO}$ - $\overline{S_2}$, ALE - QS_0, $\overline{INTA}$ - QS_1

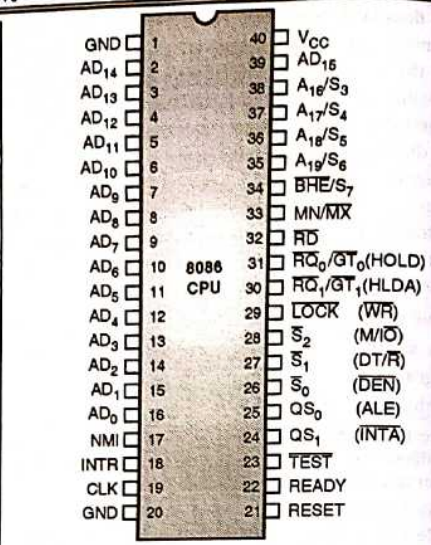
I. Supply pins (3 pins)

• V_{CC} • GND • GND

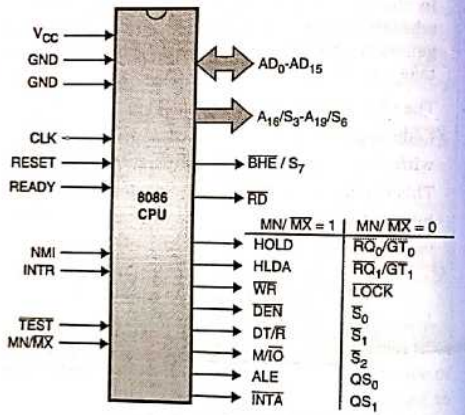
- Used for power supply i.e. +5V on V_{CC} w.r.t. GND.
- Two separate GND pins for two layers of 8086 chip, improves the noise rejection.

II. Clock related pins (3 pins)

• CLK • RESET • READY



(a) Pin configuration



(b) Functional pin diagram

Fig. 2.4.1

CLK

- This pin provides the basic timing for the processor.
- 8086 does not has an on-chip clock generator hence an external clock generator like 8284 is used to provide the clock signal.
- It is asymmetric with 33% duty cycle, TTL clock signal.

RESET

- It causes the processor to immediately terminate its present activity. The 8284 clock generator provides this signal.
- This signal must be active high for atleast 4 clock cycles.
- It clears all the flag register, the Instruction Queue, the DS, SS, ES and IP registers and sets the bits of CS register.
- Hence the reset vector address of 8086 is FFFF0H (as CS = FFFFH and IP = 0000H).

READY

- It is an acknowledgement from the addressed memory or I/O that it will complete the data transfer specially meant for slow devices.
- µP samples the READY input between T_2 and T_3 of a M/C cycle.
- If READY pin is LOW, µP inserts wait-states between T_3 and T_4, until READY becomes HIGH.

III Address and data pins (21 pins)

• $AD_{15} - AD_0$ • $A_{16}/S_3 \ldots A_{19}/S_6$ • $\overline{BHE}$ /S_7

$AD_{15} - AD_0$

- These are time multiplexed data address lines i.e. for some time they have address and for some time data.
- It gives the address $A_{15} - A_0$ during T1 of an Machine Cycle. (When ALE = 1)
- It gives the data $D_{15} - D_0$ after T1 of an M/C Cycle (Machine cycle).

$A_{16}/S_3 \ldots A_{19}/S_6$

- These lines work as Address bus ($A_{16} \ldots A_{19}$) during T1 of every M/C Cycle.
- T2 onwards these lines work as Status signals $S_3 \ldots S_6$.
- S_3 and S_4 gives the status of the memory segment currently accessed.
- S_5 gives the status of the Interrupt Enable Flag. S_6 goes low when 8086 controls the shared system bus.

S_4	S_3	Segment accessed
0	0	Extra Segment
0	1	Stack Segment
1	0	Code Segment or None
1	1	Data Segment

$\overline{BHE}$ /S_7

- This line carries the $\overline{BHE}$ signal during T1.
- $\overline{BHE}$ and A_0 are used together to access a word/byte from the memory as shown in the banking in section 2.5.
- Status line S_7 is reserved for "further development".

IV. Interrupt pins (2 pins)

• NMI • INTR

NMI

- This is a non-maskable, edge triggered that causes type 2 interrupt i.e. on receiving an interrupt on NMI line, the µP executes INT 2 and transfers the control to location 2 * 4 = 00008H in the Interrupt Vector Table (IVT). It reads 4 locations starting from this address to get values for IP and CS of the ISR address.
- It is not maskable internally by software.
- A transition from LOW to HIGH on this pin, causes the interrupt at the end of the current instruction.

INTR

- This is a non-vectored, maskable, level triggered interrupt sampled during last clock cycle of each instruction.
- To get the vector number for the interrupt the following procedure is followed.
- On receiving an interrupt on INTR line, the µP executes 2 $\overline{INTA}$ pulses.

- First $\overline{INTA}$ pulse → the interrupting device is indicated for its interrupt being accepted while the device calculates the vector number.
- Second $\overline{INTA}$ pulse → the interrupting device sends the vector number to the microprocessor on the data lines.
- Control shifts to location pointed by IP and CS which are loaded from IVT at Vector No * 4.

V. Other control (3 pins)

- $\overline{TEST}$ • MN/$\overline{MX}$ • $\overline{RD}$

$\overline{TEST}$

- It is an active low input line dedicated for 8087 Co-processor.
- In minimum mode it is connected to GND. In Maximum Mode whenever the Co-processor is busy it makes this pin HIGH.
- $\overline{TEST}$ input is examined by the WAIT instruction.
- If the TEST pin is high, the µPenters idle state; till TEST pin becomes low i.e. 8087 is free.

MN/$\overline{MX}$

- This is an input signal to 8086 that indicates the processor has to work in which mode.
- If this signal is HIGH, 8086 is in Minimum mode i.e. Single-processor system.
- If this signal is LOW, 8086 is in Maximum mode i.e. Multiprocessor system.

$\overline{RD}$

- It is an active low output signal. When it is low 8086 reads from memory or an I/O device.

VI. Mode multiplexed signals (8 pins)

(MIN Mode — Max Mode Signals)

- HOLD --- $\overline{RQ_0}$ / $\overline{GT_0}$ • HLDA --- $\overline{RQ_1}$ / $\overline{GT_1}$
- $\overline{WR}$ --- $\overline{LOCK}$ • $\overline{DEN}$ --- $\overline{S_0}$
- DT/$\overline{R}$ --- $\overline{S_1}$ • M/$\overline{IO}$ --- $\overline{S_2}$
- ALE --- QS_0 • $\overline{INTA}$ --- QS_1

HOLD — $\overline{RQ_0}$ / $\overline{GT_0}$

- In Minimum Mode this line carries the HOLD input signal from another master requesting a local bus.
- The DMA Controller issues the HOLD signal to request for the system bus.
- In response 8086 completes the current bus cycle and releases the system bus.
- In Maximum Mode it carries the bi-directional $\overline{RQ_0}$ /$\overline{GT_0}$ (Request/Grant) signal.
- The external bus master (8089 or 8087) sends an active low pulse to request for the control over the system bus.
- In response the 8086 completes the current bus cycle, releases the system bus and sends an active low Grant pulse on the same line to the external bus controller.
- 8086 gets back the system bus only after external bus master sends an active low release pulse on the same line.

HLDA — $\overline{RQ_1}$ / $\overline{GT_1}$

- In Minimum Mode, this line carries the HLDA signal.
- This signal is issued by 8086 after releasing the system bus.
- In Maximum Mode it functions as $\overline{RQ_1}$ /$\overline{GT_1}$ which is the same as $\overline{RQ_0}$ /$\overline{GT_0}$; but is of lower priority

$\overline{WR}$ --- $\overline{LOCK}$

- In Minimum Mode this line carries the $\overline{WR}$ signal indicates a write operation when this pin is Low.
- It is used with M/$\overline{IO}$ to write to Memory or IO Device.
- In Maximum mode it functions as the $\overline{LOCK}$ output line.

- When 8086 executes an instruction with the LOCK prefix this signal is active (i.e. low) remains active till next instruction, indicating the external bus master cannot take control of the system bus.

$\overline{DEN}$ --- $\overline{S_0}$

- In Minimum Mode it carries the $\overline{DEN}$ signal and is used to enable the Data transceivers (bidirectional buffer IC 8286).
- In Maximum Mode it carries the $\overline{S_0}$ signal. $\overline{S_0}$ is a status signal given to 8288.
- In Maximum Mode, Bus Controller (IC 8288) generates the DEN signal for 8286.

DT/$\overline{R}$ --- $\overline{S_1}$

- In Minimum Mode it carries the DT/$\overline{R}$ signal indicating Data Transmit or Receive.
- This signal goes low for a read operation and high for a write operation.
- In Maximum Mode it carries the $\overline{S_1}$ signal. $\overline{S_1}$ is a status signal given to 8288.
- In Maximum Mode, Bus controller issues the DT/$\overline{R}$ signal to 8286.

M/$\overline{IO}$ ---$\overline{S_2}$

- In Minimum Mode it carries the M/$\overline{IO}$ signal, to distinguish between Memory and IO access.
- In Maximum Mode it carries the $\overline{S_2}$ signal. $\overline{S_2}$ is a status signal given to 8288.
- In Maximum Mode $\overline{S_2}$, $\overline{S_1}$ and $\overline{S_0}$ are used to generate the appropriate control signal.

S_2	S_1	S_0	Machine cycle
0	0	0	Interrupt Acknowledge
0	0	1	Read I /O Port
0	1	0	Write I / O Port
0	1	1	Halt
1	0	0	Code Access
1	0	1	Read Memory
1	1	0	Write Memory
1	1	1	Passive

ALE — QS_0

- In Minimum Mode it carries the ALE signal, which is used to demultiplex the address data line by latching the address.
- In Maximum Mode it carries the QS_0 signal.
- It is used with QS_1 to indicate the Instruction Queue Status.
- In Maximum Mode, Bus Controller 8288 issues the ALE signal. (to Latches IC - 8282)

$\overline{INTA}$ — QS_1

- In Minimum Mode it carries the $\overline{INTA}$ signal.
- It is issued in response to an interrupt on the INTR line.
- It is used to read the vector number form the interrupting device.
- In Maximum Mode it carries the QS_1 signal.
- In Maximum Mode, Bus Controller issues the $\overline{INTA}$ signal. (to 8086)

QS_1	QS_0	Queue status
0	0	No Operation
0	1	First byte of Opcode from queue
1	0	Empty the Queue
1	1	Subsequent byte from queue

2.6 Memory Banking in 8086

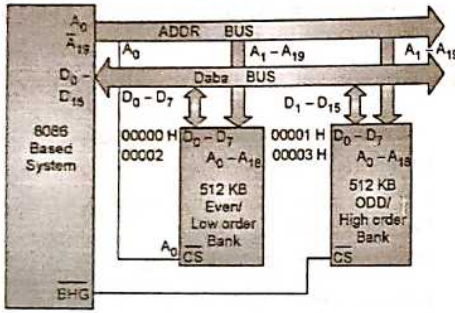
University Questions

Q. Write short note on : Memory banking in 8086.
MU - May 12, 7 Marks

Q. Explain memory banking in 8086.
MU - Dec. 19, 5 Marks

- 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.
- However, 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. 2.5.1 shows the different accesses possible for 8086. They are explained below.



(a)

Memory location	Address	Data Type	$\overline{BHE}$	A0	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. 2.5.1 : 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. 2.5.1(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. 2.5.1(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. 2.5.1(a).
  - (ii) One byte from 00001H i.e. first location of odd bank as shown in the Fig. 2.5.1(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. 2.5.1(a).
  - (ii) And another byte from 00002H i.e. second location of even bank as shown in Fig. 2.5.1(a).
- Since the locations in the two banks are not same, two accesses are required.

2.6 Exam Pack (Review and University Questions)

Q. 1 Explain architecture of 8086. **(Refer Section 2.3)** **(Dec. 13, May 14, 10 Marks)**

Q. 2 Write short note on : State use of control flags of 8086. **(Refer Section 2.3.1(B))** **(May 15, 5 Marks)**

Q. 3 State and explain General Purpose Registers. **(Refer Section 2.3.1(C))** **(5 Marks)**

Q. 4 What is segmented memory ? State the advantages of segmentation with reference to the 8086 microprocessor. Also explain the default segment assignments. **(Refer Section 2.3.2(B))** **(May 11, 10 Marks)**

Q. 5 Explain the memory segmentation in Intel 8086 processor with it's advantages and disadvantages. **(Refer Section 2.3.2(B))** **(Dec. 11, May 13, Dec. 18,10 Marks)**

Q. 6 What is segmented memory ? State the advantages of it with respect to 8086 microprocessor. **(Refer Section 2.3.2(B))** **(May 12, Dec. 12, Dec. 13, May 14, 10 Marks)**

Q. 7 What is segmentation ? What are the merits and demerits of segmentation. **(Refer Section 2.3.2(B))** **(Dec. 13, 10 Marks)**

Q. 8 Write short note on : Advantages of memory segmentation in 8086. Memory can be thought of as a vast collection of bytes. These bytes need to be organized in some efficient manner in order to be of any use. **(Refer Section 2.3.2(B))** **(Dec. 14, 5 Marks)**

Q. 9 Explain memory segmentation with pros and cons. **(Refer Section 2.3.2(B))** **(May 16, 8 Marks)**

Q. 10 Write short note on : Advantages of memory segmentation in 8086. **(Refer Section 2.3.2(B))** **(May 15, May 19,5 Marks)**

Q. 11 Explain the memory segmentation of 8086 **(Refer Section 2.3.2(B))** **(Dec.17, 5 Marks)**

Q. 12 What is memory segmentation ? State Advantages of memory segmentation. **(Refer Section 2.3.2(B))** **(Dec. 15, May 17 , 5 Marks)**

Q. 13 Write short note on : Memory banking in 8086 **(Refer Section 2.5)** **(May 12, 7 Marks)**

Q. 14 Explain memory banking in 8086. **(Refer Section 2.5)** **(Dec. 19, 5 Marks)**

□□□

Note

3

MODULE 1

Operating Modes : Minimum and Maximum Modes

3.1 8086 Configurations

8086 can work in either of the two modes :

1. **Minimum mode :** In this mode, 8086 is the only processor in the system. It is also called as **uniprocessor mode.**
2. **Maximum mode :** In this mode, multiple processors can be present in the system.

Some common components of 8086/8088 system are :

1. 8282 : Octal latch
2. 8284 : Clock generator and driver
3. 8286 : Octal bus transceiver
4. 8288 : Bus controller

Let us discuss these chips one by one.

3.2 8284 : Clock Generator and Driver

University Questions

Q. Write short note on : 8284 clock generator. **MU - May 11, May 12, Dec. 12, May 13, 7 Marks**

Q. Explain the hardware required to generate clock and reset signals. **MU - May 14, 10 Marks**

- Before discussing the clock generator let us understand the requirements of 8086/8088 as far as the clock input is concerned.

Clock

- Fig. 3.2.1 shows the clock signal required by 8086 with all the standard time durations and voltage levels.
- The clock signal required by 8086 should satisfy the following specifications.

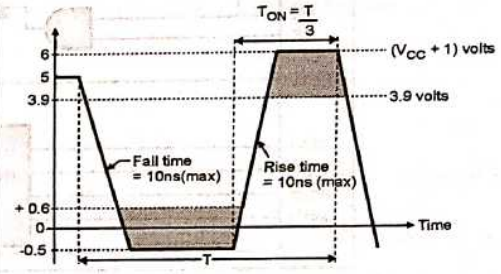


Fig. 3.2.1 : Clock signal required by 8086/8088

- 8086 doesn't have an on-chip clock generator and requires the following specifications from the externally generated clock pulse:
  1. The rise time and fall time must be very less, less than 10 nano seconds.
  2. The duty cycle i.e. the ration of ON time to the total time of the pulse must be 33%.
  3. The voltage ranges for logic '0' must be between -0.5 Volts to +0.6 Volts and that for logic '1' must be between 3.9 Volts to 6 Volts
  4. The ready signal and the reset signal must also be synchronized with the clock signal.
- Implementing the external clock circuit with all the above specifications is complicated and will require a lot of space on the motherboard.
- Hence Intel has developed 8284 clock generator that generates the clock pulses considering all the specifications.

3.2.1 8284 Block Diagram and Pin Configuration

- The internal block diagram and pin configuration of the clock generator 8284 are shown in Fig. 3.2.2(a) and 3.2.2(b) respectively.
- 8284 is a bipolar clock generator / driver which has been designed to provide the clock signals for 8086/8088 and peripherals.

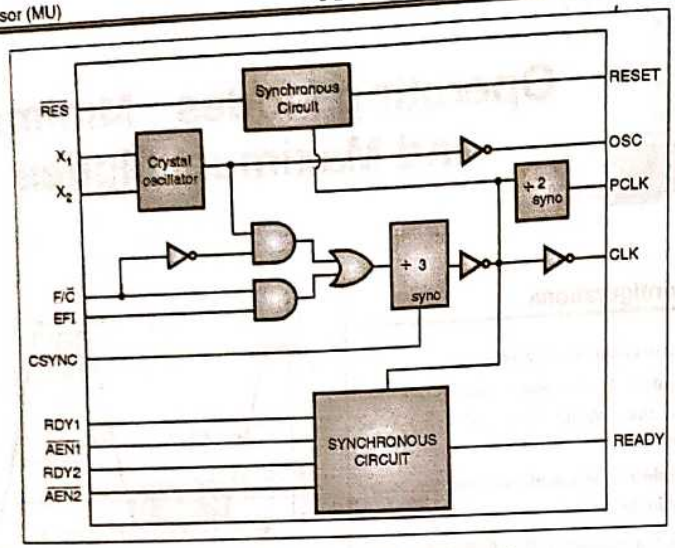


Fig. 3.2.2(a) : Internal block diagram of 8284

- This chip mainly produces the following three signals for microprocessor:
  1. Clock 2. Ready 3. Reset.
- It also provides the READY 1 and READY 2 signals along with their corresponding address enable signals $\overline{AEN1}$ and $\overline{AEN2}$ for the multibus system.

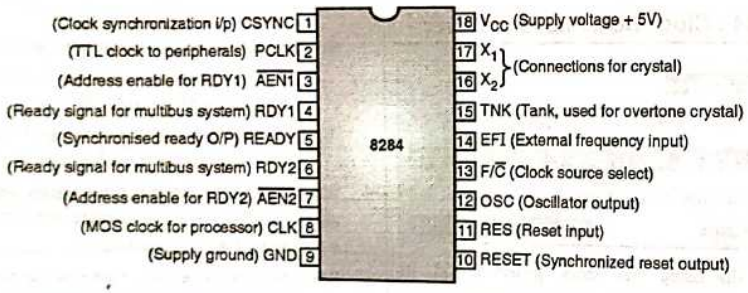


Fig. 3.2.2(b) : Pin configuration and explanation for 8284

3.2.2 Features of 8284

- Generates system clock signal for 8086, 8088 processors and the peripherals.
- Operates on a single + 5V supply.
- Available in the 18 pin package.
- Can use either a crystal or a TTL external signal as frequency source.
- Generates the system reset output.
- Provides the synchronization between local ready and microprocessor ready signals.
- Capable of clock synchronization with other 8284 chips.

3.2.3 Description of Operation of Various Pins

University Questions

Q. Draw the circuit diagram for generation of RESET signal and explain it's working. **MU - Dec. 11, 5 Marks**

Q. Write short note on : Generation of reset signals in 8086 based system. **MU - Dec. 15, 5 Marks**

Q. Explain power on reset circuit used in 8086 system. **MU - Dec. 17, 5 Marks**

1. X_1, X_2, TNK (Pin 17, 16, 15)

- These are the input lines of 8284. X1 and X2 are the inputs between which the crystal is actually connected.
- TNK stands for tank. These three inputs are interfaced to the internal oscillator of 8284.
- The internal oscillator circuit of 8284 has been designed to operate with an external crystal which operates in the series resonant fundamental mode.
- The external crystal acts as the source of basic operating frequency for the internal oscillator.
- The advantage of using the crystal is that the frequency remains dead constant.
- The crystal frequency should be equal to three times the required clock frequency. For example if f_{CLK} = 5 MHz then the crystal frequency should be 15 MHz.

2. EFI (External Frequency Input)

- Pin number 14 acts as the external frequency source (EFI) input.
- If the user has another high accuracy frequency source or an externally variable frequency source or a common source for driving multiple 8284 ICs, then he has to connect that source to EFI input.
- The external frequency source connected at EFI input should fulfil the following requirements.
  1. It must have 50% duty cycle.
  2. Its frequency should be equal to three times the desired clock frequency.
  3. It should be TTL compatible.

3. OSC (Pin No. 12) oscillator output

- This is an output pin for the 8284 chip.
- It is the TTL level output of the internal oscillator circuit.
- The frequency of the signal obtained at this pin is equal to the frequency of the crystal i.e. three times the desired clock frequency.

4. F/ $\overline{C}$ (Frequency / crystal select)

- F/ $\overline{C}$ is pin no. 13 of 8284 and it is the clock source select input pin.
- If F/ $\overline{C}$ = 0, then the processor clock signal is produced by using the crystal as clock source.
- Whereas if F/ $\overline{C}$ = 1, then the external source connected at EFI pin is used for generation of the processor clock.

$$\therefore F/\overline{C} = 0 \rightarrow \text{select crystal}$$

$$\text{and} \quad F/\overline{C} = 1 \rightarrow \text{select EFI}$$

5. CLK (Pin No. 8)

- This is the output pin and the processor clock signal is available on this pin.
- Therefore the CLK pin is actually connected to the clock input of the 8086 processor.
- It is also used by all those devices which are connected directly to the processors bus.
- The frequency of the rectangular signal obtained on the CLK output pin is equal to one third of the crystal frequency, or EFI input frequency.
- The duty cycle of the CLK signal is 33.33% that means the ON time is one third as compared to the total time.
- The output HIGH (1) logic level of 4.5 volts (with V_{CC} = 5V) is provided for driving the MOS devices.

6. PCLK (Peripheral clock)

- Pin no. 2 is PCLK i.e. peripheral clock output.
- On this pin, a TTL level clock signal which is required for the peripheral devices is produced.
- The PCLK signal frequency is half that of the CLK frequency and has a duty cycle of 50%.

7. CSYNC (Clock Synchronization)

- Pin number 1 of 8284 is the clock synchronization input. The signal applied to this pin synchronizes the CPU clock (CLK) to an external event.
- Such a synchronization is essential because the T-state of internal divide by three counter of 8284 is indeterminate at the time of power ON.
- When CSYNC = 1, the CLK and PCLK clock output signals are forced to become 1.

- When CSYNC = 0, then the next positive clock from the selected frequency source (crystal or EFI) will start the clock generation (CLK and PCLK signals).
- CSYNC should be active for at least two periods of the selected frequency source.
- If EFI is the selected frequency source, then the CSYNC needs to be externally synchronized with the EFI.
- If the crystal is the selected frequency source, then the CSYNC pin should be hardwired to GND.

8. $\overline{RES}$ (Reset In) Pin 11

- $\overline{RES}$ stands for reset input. This is an input pin.
- It is used for generating the "Reset" signal for the processor. It is an active low pin.
- An external RC integrator is connected to this pin in order to obtain the power ON reset signal of proper duration.

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 behaviour 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. 3.2.3 illustrates 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.
- The diode is connected across the resistor to discharge the capacitor rapidly when power supply is switched off.

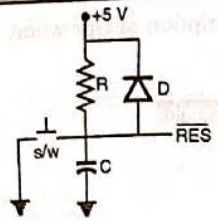


Fig. 3.2.3

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

9. RESET

- Pin no. 10 is the Reset output pin. It is an active high pin which is used to reset the 8086 family processors.
- The timing characteristics of the Reset signal on this pin are decided by the external circuitry connected to the $\overline{RES}$ input.
- The RESET pin is connected directly to the RESET input of 8086 (see interfacing of 8284 with 8086 in section 3.2.4).

10. READY (Pin 5)

- READY (pin 5) is an active high output signal generated by 8284. This pin is connected directly to the READY input of the 8086 processor as shown in Fig. 3.2.2(b).
- This output signal is synchronized with the two Ready input signals (RDY 1 and RDY 2) of 8284.

11. RDY 1 and RDY 2

- RDY 1 and RDY 2 are the input signals available from the external devices on pins 4 and 6 respectively.
- Whenever an external device wants to introduce a "wait" T-state for proper data transfer, it will activate (force to 1) the RDY 1 or RDY 2 input.
- A synchronized READY signal will be produced by 8284 and applied to 8086.

12. $\overline{AEN1}$ and $\overline{AEN2}$

- These are active low input signals. They are the enable inputs for RDY 1 and RDY 2 inputs respectively.
- A low $\overline{AEN1}$ signal validates RDY 1 and a low $\overline{AEN2}$ signal validates the RDY 2 input.

- $\overline{AEN1}$ and $\overline{AEN2}$ signals are also called as "Gated Signals" and they are generally used in the multiprocessor configuration.

3.2.4 Interfacing 8284 to 8086

The typical connections for 8284 and its interfacing with the 8086 processor is shown in Fig. 3.2.4.

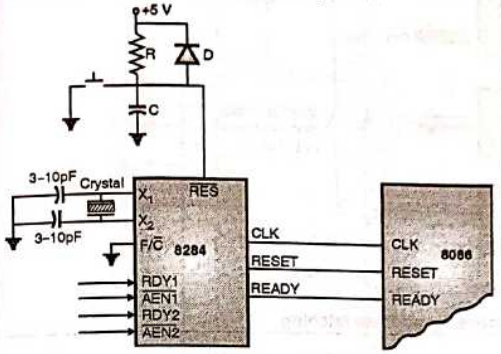


Fig. 3.2.4 : Interfacing 8284 to 8086

3.3 8282 /8283 : An Octal Latch

University Question

Q. Design interfacing of 8282 latches to 8086 system. MU- Dec. 15, 5 Marks

- "A latch is a register with a specific purpose. This group of D flip-flops is used to hold values to be output to other devices."
- For example, if you wanted to control a number of LEDs say for a level meter, a latch would hold the appropriate 1's and 0's to drive those LEDs while the processor went off to do other things.
- Before going into the details of 8282/8283 octal latch let us see its utility and why do we need to use the latch.

Address data bus of 8086

- The address data bus of 8086 is a **multiplexed** bus. That means the same pins (AD_0 to AD_{15}) are being used as address lines as well as data lines.
- The same lines are being used to carry the address information as well as data, on the time sharing basis, and hence are to be demultiplxed.
- The remaining four address lines A_{16} to A_{19} also are multiplexed with the S_3 to S_6 signals. (A_{16} / S_3 to A_{19} / S_6), and hence are to be demultiplexed.
- Also the $\overline{BHE}$ / S_7 signal is to be demultiplexed.
- The 8086 first outputs the 20 bit address on the multiplexed lines. The presence of address on the multiplexed bus is indicated by forcing the ALE (address latch enable) pin of 8086 to HIGH.
- This address should be latched into external latches. The ALE signal is used to enable the external latches, to store the address.
- The ALE signal then goes low (0), before the address disappears from the multiplexed bus as shown in Fig. 3.3.1. This will disable the external latches.
- The address which is stored by the external latches can now be used as address bus.
- The processor will output the data on the multiplexed bus, after ALE goes low.
- The stable address required for the system memory and peripherals for the duration of bus cycle is provided by the external latches at their outputs.

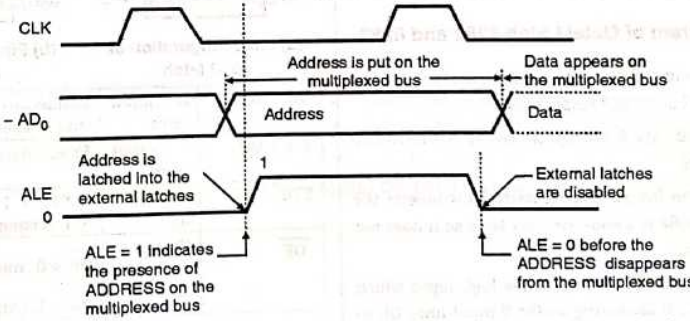


Fig. 3.3.1 : Timing diagram showing the role of ALE in the multiplexed address data bus

- The most popularly used latches are :
  1. 8282 : Non inverting octal latch.
  2. 8283 : Inverting octal latch.

3.3.1 Use of External Latches for Address Latching

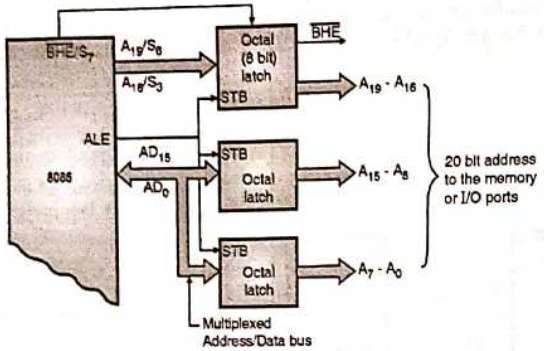


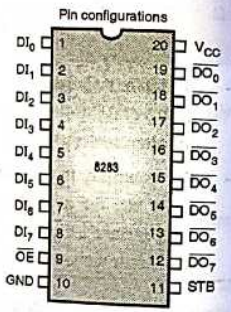
Fig. 3.3.2 : Use of external latches for address latching

- Fig. 3.3.2, shows diagrammatically the use of external latches to latch the address.
- The ALE signal is being used as the strobe (STB) input for the latches, which enables the latches and loads the address into the latches.
- Since each latch is an octal i.e. 8-bit latch, we have to use three such latches in order to latch a 20 bit address.
- On receiving the HIGH (1) ALE signal from the processor, the latches will be enabled and the A_0 to A_{19} address is latched.
- Before the address disappears from the multiplexed bus, the ALE reduces to 0 and the latches are disabled. So they do not respond to the data appearing on the multiplexed bus.

3.3.2 Pin Diagram of Octal Latch 8282 and 8283

- Fig. 3.3.3(a) and (b) show the pin diagram for the octal latch ICs 8282 and 8283 respectively.
- The 8282/8283 are 8 bit bipolar latches with tristate output buffers.
- The 8283 is an inverting octal latch. So it inverts the input while 8282 is a non-inverting latch so it does not invert the input.
- The strobe input (STB) is an active high input which latches the input appearing on the 8 input lines DI_0 to DI_7 into the latch.
- Whereas the output enable ($\overline{OE}$) is an active low input which enables the output buffers and connects the latched information to the output lines DO_0 to DO_7.
- If $\overline{OE}$ = 1 then the latch outputs are in the tri-state i.e. high impedance T-state.

(a) Pin configuration of 8282 latch



(b) Pin configuration of 8283 latch

$DI_0 - DI_7$	8 input lines	Connected to the multiplexed bus of 8086
$DO_0 - DO_7$	8 output lines	Produce the latched address
STB	Strobe input	If STB = 1, input is latched. STB is connected to ALE.
$\overline{OE}$	Output enable input	$\overline{OE}$ = 0, outputs are enabled $\overline{OE}$ = 1, outputs in tristate

(c) Pin definition of octal latch IC 8282 and 8283

Fig. 3.3.3

Fig. 3.3.4 shows the internal circuit diagram of 8282 and 8283.

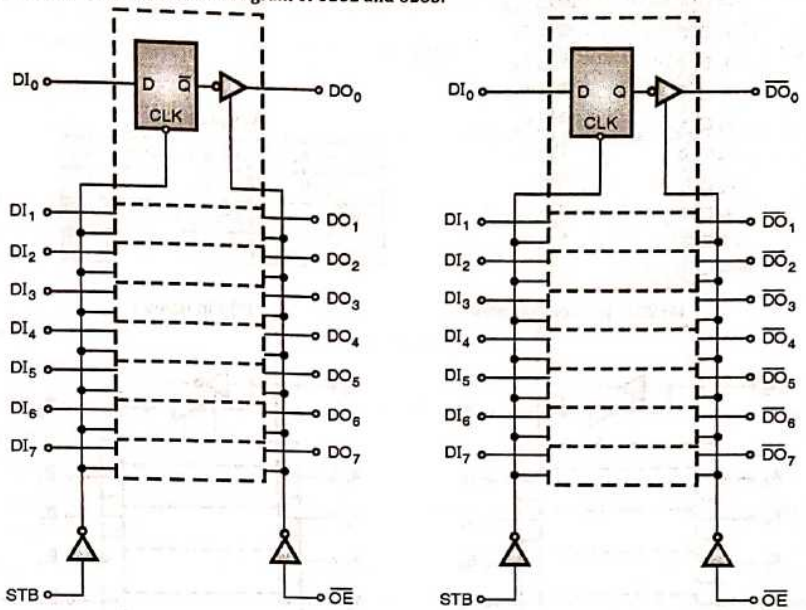


Fig. 3.3.4 : Internal circuit diagram of 8282 / 8283

3.3.3 Features of 8282 / 8283

- Available in 20 pin package.
- Operate on single polarity supply.
- Contain fully parallel 8 bit data registers and buffers.
- Tri-state (high impedance) outputs.
- High output drive capability.
- Transparent during active probe.
- Transparent during active probe.
- Support the 8085, 8086, 8088 microprocessor systems.

3.4 8286 : Octal Bus Transceiver

Q. Explain 8286 as octal bus receiver. (5 Marks)

- The 8286 is an octal bus transceiver which is used for obtaining the 16 bit data bus from the multiplexed address data bus of 8086 processor.
- The pin diagram of 8286 is shown in Fig. 3.4.1.

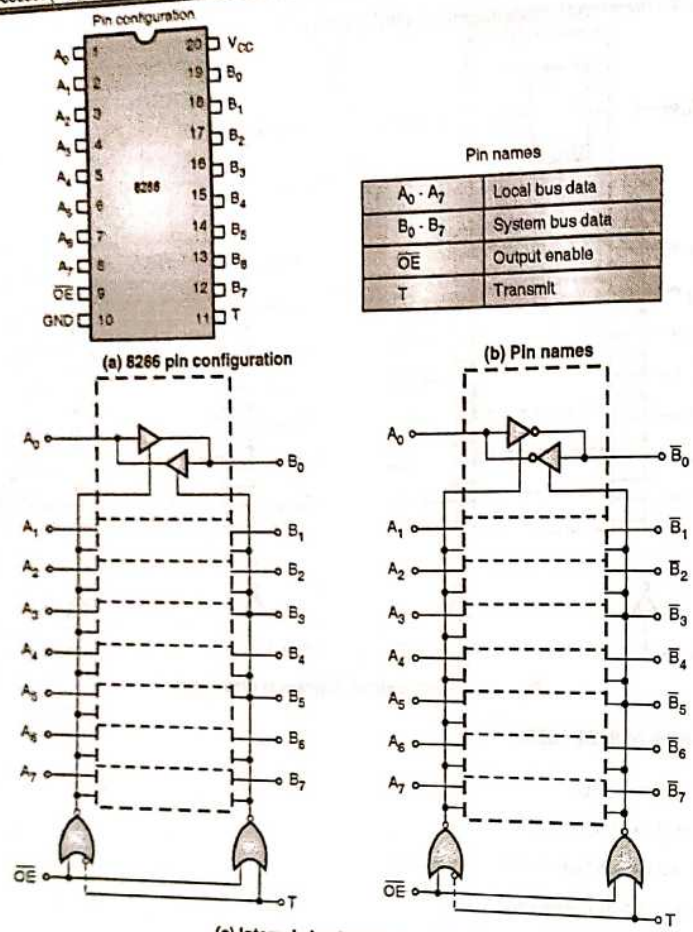


Fig. 3.4.1

Description

1. **$A_0 - A_7$**

- These 8-lines are called as the local bus data lines. They can act as input lines or as output lines depending on the status of the transmit (T) pin.
- If T = 1, then $A_0 - A_7$ act as inputs, whereas if T = 0, then $A_0 - A_7$ will act as output lines.

2. **$B_0 - B_7$**

- These 8-lines are called as system bus data lines. They too can act as input or output lines depending on the status of the transmit (T) pin.
- If T = 1, then $B_0 - B_7$ will act as output lines whereas with T = 0, they will act as the input lines.

3. **T (Transmit)**

- Pin 11 i.e. T or transmit pin is an input pin of 8286.

- It controls the direction of the transceivers, as follows :
  - If T = 1; $A_0 - A_7$ act as inputs and $B_0 - B_7$ act as outputs.
  - If T = 0; $A_0 - A_7$ act as outputs with $B_0 - B_7$ acting as inputs.

4. **Output Enable ($\overline{OE}$)**

- This is an active low input to 8286. It is used for enabling the appropriate driver to put his data onto the respective bus.
- If $\overline{OE}$ = 1 i.e. in active, then the outputs will be in the high impedance (tri-state) T-state.

3.4.1 Features of 8286

- Available in the 20 pin package.
- Tri state (high impedance) outputs.
- Contains fully parallel 8 bit transceivers (transmitters and receivers).
- Capable of driving system data bus.
- Used as data bus buffers for 8085, 8086 and 8088 based systems.

Interfacing with 8086

- The interfacing of the 8286 with 8086 is shown in Fig. 3.4.2.
- The multiplexed address data bus AD_0 to AD_{15} is applied at the A inputs (A_0 to A_7) of the two 8286 ICs ($AD_0 - AD_7$ to first and $AD_8 - AD_{15}$ to the second).
- The B outputs of the two 8286 chips act as the demultiplexed data bus.
- The 8286 transceivers allow bi-directional data transfer. The T (transmit) input of both 8286 chips is connected to the DT/$\overline{R}$ output of 8086.
- So when DT/$\overline{R}$ = 0, T = 0, so B lines act as input lines, a lines act as the output lines and the data will flow from memory or ports to 8086, that means the read operation will take place.
- On the other hand if DT / $\overline{R}$ = 1, then T = 1 and the data will get transferred from the processor to the memory or ports.
- The $\overline{OE}$ lines of both the 8286 chips are connected to the $\overline{DEN}$ output of 8086. This signal is used to enable or disable the transceivers. With $\overline{DEN}$ = 0, the transceivers are enabled and with $\overline{DEN}$ = 1 they are disabled and go into tri-state.

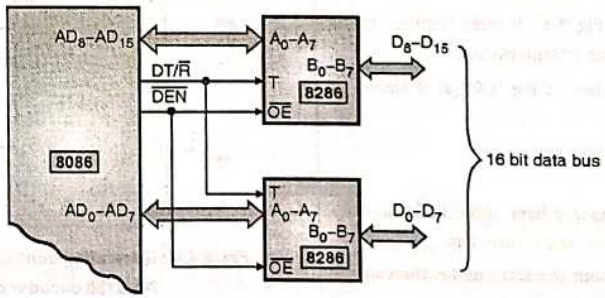
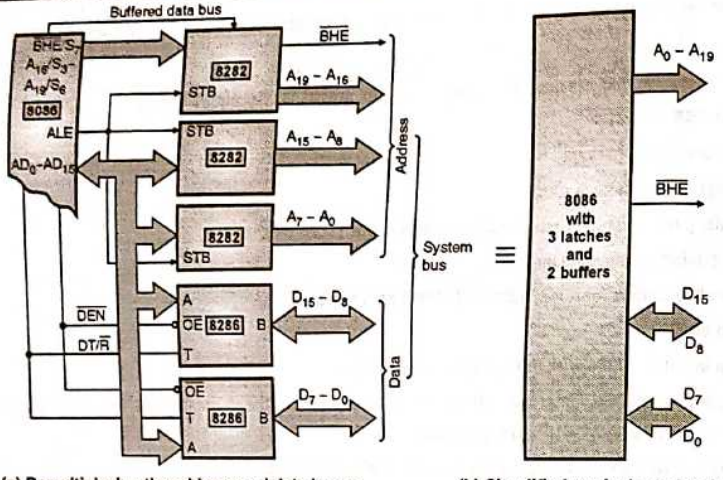


Fig. 3.4.2 : Interfacing of transceiver 8286 with 8086

3.4.2 Demultiplexing Address and Data Bus

Q. Explain demultiplexing address and data bus. (5 Marks)

Q. Draw and explain demultiplexing of address bus in 8086. MU - Dec. 19, 10 Marks



(a) Demultiplexing the address and data buses (b) Simplified equivalent circuit

Fig. 3.4.3

- We have discussed the use of 8282 latches in address latching and 8286 as data bus buffer.
- If we use them simultaneously then we can demultiplex the address data bus as shown in Fig. 3.4.3(a).
- Thus we require three octal latch chips (8282) and two octal transceiver chips (8286) in order to completely demultiplex the address and data bus.
- The basic diagram of Fig. 3.4.3 is often required in the memory interfacing and I/O interfacing.
- The simplified equivalent of Fig. 3.4.3(a) is shown in Fig. 3.4.3(b).

3.4.3 Control Bus

- The 8086/8088 processors have three buses namely the address bus, data bus and control bus.
- Let us now discuss about the last one i.e. the control bus.
- The 8086 provides three control lines namely $\overline{RD}$ (read), $\overline{WR}$ (write) and M/$\overline{IO}$ (Memory/input output port).
- We will use the decoder IC 74138 to convert these three lines into the following four signals.
  1. Memory Read $\overline{MEMR}$
  2. Memory Write $\overline{MEMW}$
  3. I/O Read $\overline{IOR}$
  4. I/O Write $\overline{IOW}$

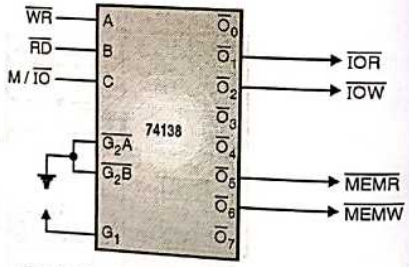


Fig. 3.4.4 : Generating control signals using 74LS138 decoder chip

Table 3.4.1 shows the generation of these signals in the tabulated form whereas Fig. 3.4.4 shows the connections to be made to 74138 in order to obtain the required signals.

Table 3.4.1 : Generating $\overline{MEMR}$, $\overline{MEMW}$, $\overline{IOR}$, $\overline{IOW}$

M/$\overline{IO}$	$\overline{RD}$	$\overline{WR}$	Operation
0	0	0	Invalid
0	0	1	I/O read ($\overline{IOR}$ to be generated)
0	1	0	I/O write ($\overline{IOW}$ to be generated)
0	1	1	No operation
1	0	0	Invalid
1	0	1	Memory read ($\overline{MEMR}$ to be generated)
1	1	0	Memory write ($\overline{MEMW}$ to be generated)
1	1	1	No operation

We will use decoder 74138 to generate control signals.

3.5 8288 Bus Controller

University Questions

Q. Write short note on : 8288 bus controller. MU - May 12, 7 Marks

Q. Explain the necessity of a bus controller in 8086 maximum mode operation. Also explain the 8288 bus controller in detail. MU - Dec. 12, 10 Marks

Q. Explain the 8288 bus controller block diagram. Explain it's use. MU - May 13, 10 Marks

Q. Write short note on 8288 bus controller. MU - Dec. 15, Dec. 16, 5 Marks

Use : In maximum mode 8086 doesn't provide various control signals (like ALE, DT/$\overline{R}$, $\overline{DEN}$ etc.) and command signals (like $\overline{WR}$, M/$\overline{IO}$ etc.). Hence 8288, bus controlled is used to provide all these signals.

- 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.

3.5.1 Internal Block Diagram and Pin Diagram

- Fig. 3.5.1(a) shows the simplified internal block diagram of 8288 bus controller and Fig. 3.5.1(b) shows the pin diagram of 8288.

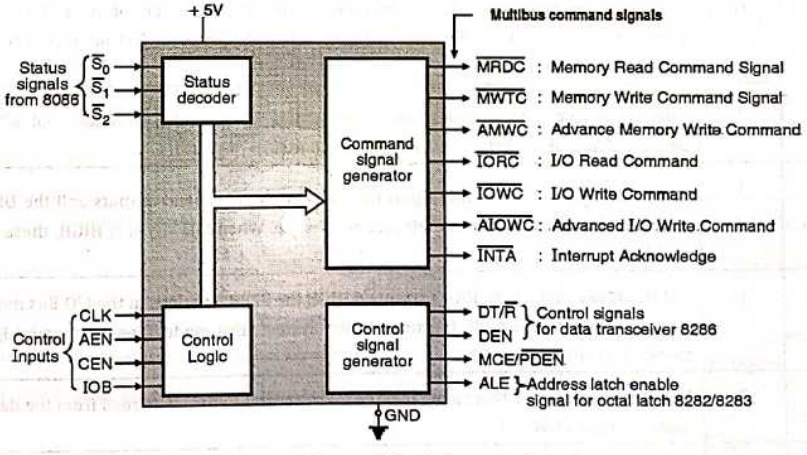


Fig. 3.5.1(a) : Simplified internal block diagram of 8288 bus controller

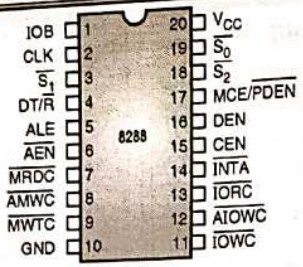


Fig. 3.5.1(b) : Pin diagram of 8288 bus controller

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).
$\overline{AEN}$	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. (See sections on I/O Bus and System Bus modes i.e. 3.5.3).
$\overline{IOWC}$	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.

Name	I/O	Function
$\overline{AIOWC}$	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.

3.5.2 Command and Control Logic

University Question

Q. Explain the status signals of Intel 8086 processor. Show all the possible combinations along with the processor state and 8288 command associated with each combination. **MU - Dec. 11, 5 Marks**

- It is possible to use the 8288 bus controller with the 8086, 8088 and 8089 processors.
- The CPU status lines $\overline{S_0}$, $\overline{S_1}$ and $\overline{S_2}$ are applied as the inputs to the 8288 bus controller.
- The status decoder and command logic inside the 8288 will generate a command depending on the status of the $\overline{S_0}$, $\overline{S_1}$ and $\overline{S_2}$ lines, as given in the Table 3.5.1.

Table 3.5.1 : Relation between the command words and status words

Status of CPU			Status word (8086)	8288 command
$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$		
0	0	0	Interrupt acknowledge	$\overline{INTA}$
0	0	1	Read I/O port	$\overline{IORC}$: I/O read command
0	1	0	Write I/O port	$\overline{IOWC}$, $\overline{AIOWC}$: I/O or advanced I/O write command
0	1	1	Halt	None
1	0	0	Code Access	$\overline{MRDC}$: Memory read command

Status of CPU			Status word (8086)	8288 command
$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$		
1	0	1	Read Memory	$\overline{MRDC}$: Memory read command
1	1	0	Write Memory	$\overline{MWTC}$, $\overline{AMWC}$: Memory or advanced memory write command
1	1	1	Passive	None

- The command generated by 8288 is issued in one of the two possible ways depending on the mode of 8288 bus controller.
- The modes of 8288 are :

1. I/O bus mode 2. System bus mode

3.5.3 I/O Bus Mode

- The 8288 operates in the I/O bus mode if the IOB pin (pin 1) is strapped HIGH (1).
- In the I/O bus mode, all the I/O command lines such as $\overline{IORC}$, $\overline{IOWC}$, $\overline{AIOWC}$, $\overline{INTA}$ are always enabled, independent of the $\overline{AEN}$ status.
- If an I/O command is initiated by the processor, then 8288 activates the command lines using the $\overline{PDEN}$ and $DT/\overline{R}$ signals to control the I/O bus transceivers.
- This mode allows the 8288 bus controller to handle two external buses and there is no waiting involved when the CPU wants to have access to the I/O bus.
- The I/O bus mode of 8288 should be preferred if the I/O or peripherals dedicated to single processor happen to exist in the multiprocessor system.

System bus mode

- The 8288 bus controller is in the system bus mode if the IOB input (pin 1) is strapped LOW (0).
- In this mode first the $\overline{AEN}$ line is activated (forced LOW), then there is a waiting period of 105 n sec during which no command is issued.
- After 105 n sec the required commands are issued.
- This mode assumes that the arbitration logic w
inform the bus controller on its $\overline{AEN}$ line, about wh
the bus is free for use.
- Both the I/O and memory commands will wait for t
$\overline{AEN}$ signal to get activated by the bus arbitration log
- The system bus mode is to be used when only one b
exists on which, both I/O and memory are shared
more than one processor.

Command outputs of 8288

- As shown in the internal block diagram of 8288
Fig. 3.5.1(a), the 8288 bus controller produces th
following seven command outputs.

1. $\overline{MRDC}$: Memory read command
2. $\overline{MWTC}$: Memory write command
3. $\overline{IORC}$: I/O read command
4. $\overline{IOWC}$: I/O write command
5. $\overline{AMWC}$: Advanced memory write command
6. $\overline{AIOWC}$: Advanced I/O write command
7. $\overline{INTA}$: Interrupt acknowledge.

- The first four command outputs correspond to the read and write operations or the memory and I/O ports. They are simple straight forward commands.
- The fifth and sixth command outputs correspond to the advanced memory or I/O writing. The advanced write commands are made available by 8288 so as to initiate the write procedures early in the machine cycles.
- The advanced write command signals are used to avoid the processor entering into unnecessary wait T-state.
- The last command output is $\overline{INTA}$ i.e. interrupt acknowledge. This output command actually acts as the I/O read, during the interrupt cycle.
- The $\overline{INTA}$ signal tells the interrupting device that its interrupt has been acknowledged by 8086 and now it should place the required information on the data bus.

3.5.4 Control Outputs

- As shown in the internal block diagram of 8288 in Fig. 3.5.1(a) the 8288 bus controller produces the following four control outputs.

1. DEN : Data enable
2. $DT/\overline{R}$: Data transmit / Receive
3. MCE / $\overline{PDEN}$: Master cascade enable / Peripheral data enable
4. ALE : Address latch enable

1. DEN (Data enable)

- The DEN control output is applied to the $\overline{OE}$ input of the 8286 transceivers.
- This output determines when the external data bus is to be enabled and connected onto the local bus.

2. $DT/\overline{R}$ (Data transmit/receive)

- The $DT/\overline{R}$ decides the direction of data transfer. This control output is connected to transmit (T) input of the 8286 transceivers.
- The MCE / $\overline{PDEN}$ control output changes its T-state depending on the operating mode of 8288 bus controller.
- If 8288 is in the I/O bus mode (IOB), then the $\overline{PDEN}$ signal works as a dedicated data enable signal for the I/O or peripheral system bus.
- MCE (Master control enable) is used during the interrupt acknowledge cycle if 8288 is in the system Bus Mode (IOB pin LOW).
- During every interrupt cycle there are two interrupt acknowledge cycles which occur back to back.
- During the first interrupt cycle no data or address transfers take place. Therefore logic '0' should be provided to mask the MCE signal during the first cycle.
- Just before the beginning of the second cycle, the MCE signal gates the cascade address of master priority interrupt controller (PIC) onto the processors local bus. The ALE then strobes it into the address latches.
- On the leading edge of the second interrupt cycle, the addressed slave PIC will gate an interrupt vector onto the system data bus where it is read by the processor.
- But if a system contains only one PIC, then the MCE signal is not used.
- Then the second interrupt acknowledge signal simply gates the interrupt vector onto the processor bus.

3. ALE and halt

- The ALE (address latch enable) control output is connected to the strobe (STB) input of the 8282 latches.
- The ALE signal is activated in each machine cycle in order to strobe the current address appearing on the address bus into the address latches (8282).
- There is one more application of the ALE signal. It strobes the 8086 status ($\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$) into a latch for the status decoding.

4. Command enable (CEN) control input

- This is one of the four control inputs to the 8288 bus controller. If CEN = 1 then 8288 functions normally.
- But if CEN = 0, then all the seven command outputs are forced into their inactive T-state (and not the tristate).

3.5.5 Interfacing of 8288 to 8086

- The interfacing of the bus controller 8288 with the 8086 processor is shown in Fig. 3.5.2.
- Note that the MN/$\overline{MX}$ pin of 8086 is connected to ground, which indicates that it is going to operate in the maximum mode.

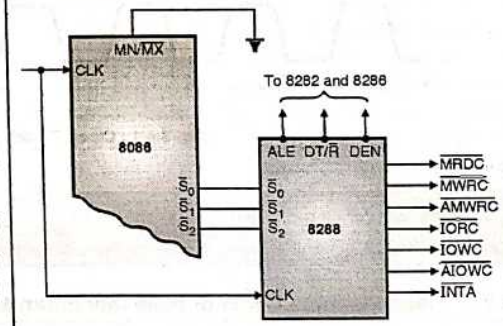


Fig. 3.5.2 : Interfacing of 8288 with 8086

3.5.6 Features of 8288 Bus Controller

- Available in 20 pin package.
- Operates on a single polarity supply.

- Generates the control and command signals useful for the operation in maximum mode.
- Bipolar drive capability.
- 3 T-state command output drivers.
- Configurable for use with the I/O bus.
- Facilitates the interface to one or two multimaster buses.
- Provides a wide flexibility in the system configurations.

3.6 8086 Timing Diagrams

Q. Explain 8086 timing diagrams. (5 Marks)

- All the operations in 8086 are carried out with a particular sequence in a synchronized manner.
- So to understand every operation it is necessary to understand the timing diagrams.
- Before going into the details of the timing diagrams for the read and write operations, we will first define some of the important terms.

3.6.1 T-State

- The smallest unit of time in the microprocessor timing is the T-state. It comprises of a falling edge, low level, rising edge and a high level. In figure 3.6.1, T_1 or T_2 are each a T-state.
- The time period for this T-state depends on the crystal frequency.

3.6.2 A Machine Cycle

- The time required to perform one access on the bus is called as a machine cycle. The access means read or write from memory or I/O device. Hence, for example, memory read is a machine cycle.
- The various operations to be performed in the execution of an instruction are such machine cycles.

3.6.3 Instruction Cycle

- Instruction cycle comprises of various machine cycles required to execute a given instruction.
- Hence it can be said that the instruction cycle time is the time required to execute an instruction for the processor.

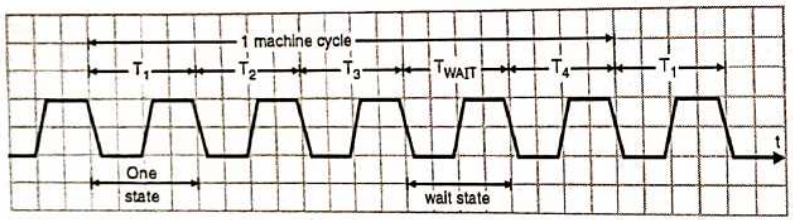


Fig. 3.6.1 : Clock, T-state, machine cycle etc.

Summary

1. A machine cycle is made up of some T- states.
2. An instruction cycle is made up of one or many machine cycles

3.7 Read Machine Cycle of 8086 (Minimum Mode)

University Questions

- Q. Draw the timing diagram of Memory read in minimum mode. MU - May 12, 5 Marks
- Q. Draw and explain timing diagram for read operation in minimum mode of 8086. MU - Dec. 14, Dec. 17, 5 Marks
- Q. Draw timing diagram of read operation on 8086 based system. MU - May 16, 10 Marks
- Q. Draw and explain memory read machine cycle timing diagram in minimum mode of 8086. MU - Dec. 18, 5 Marks

- Fig. 3.7.1(a) shows the timing diagram which shows the activities of various signals during the read operation.

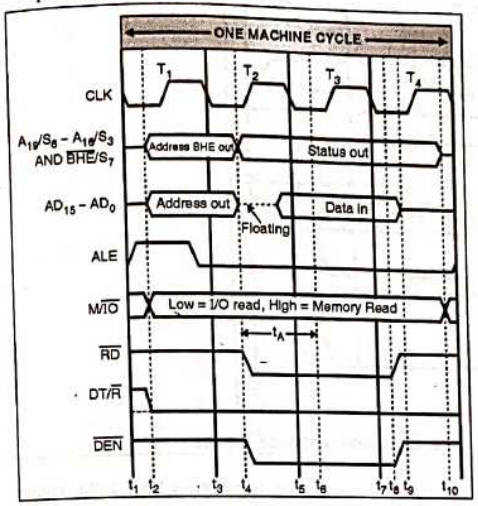


Fig. 3.7.1(a) : Timing diagrams for the read machine cycle of 8086

- 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.

- **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 3.7.1(b) shows wait states inserted in between of a machine cycle.

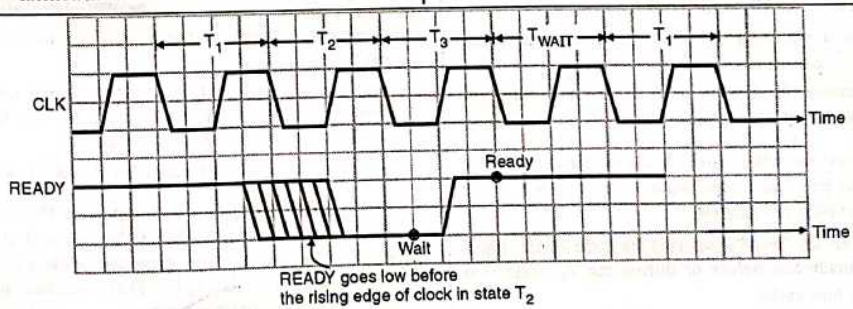


Fig. 3.7.1(b) : Concept of WAIT T-states

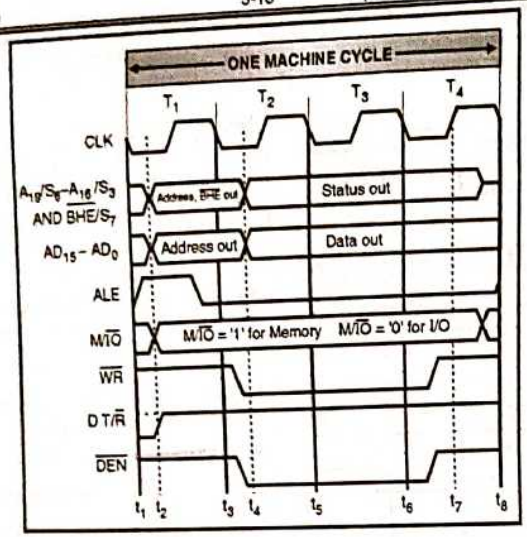


Fig. 3.7.2 : Write machine cycle in the 8086 minimum mode with wait T-state

3.8 8086 Write Machine Cycle in the Minimum Mode

University Question

Q. Draw and explain timing diagram for write operation in minimum mode of 8086. **MU - May 15, 5 Marks**

- Let us now see the activities taking place in the write machine cycle of 8086 in the minimum mode.

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.

3.9 Functioning of 8086 in Minimum Mode

University Question

Q. Explain minimum mode configuration of 8086 microprocessor. **MU - May 19, 5 Marks**

- The minimum mode system block diagram is as shown in Fig. 3.9.1.
- As shown in Fig. 3.9.1, the system in the minimum mode contains the support chips such as 8282, 8284 and if necessary, the 8286 data buffers.
- As shown in Fig. 3.9.1 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.

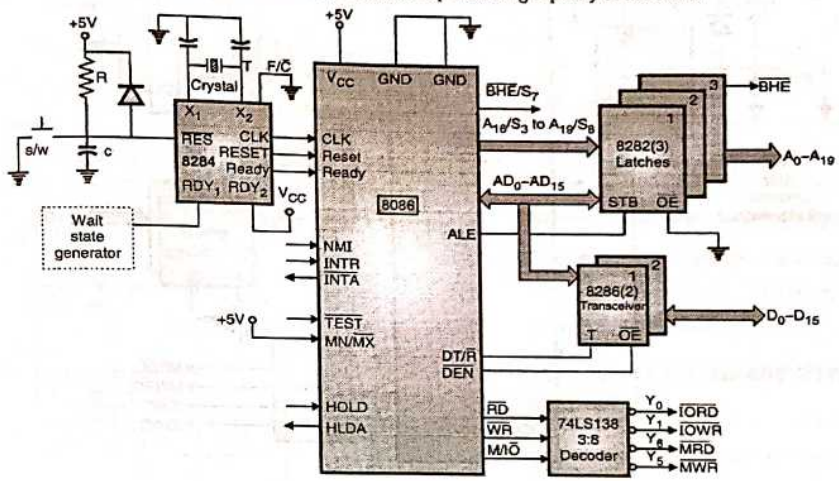


Fig. 3.9.1 : Block diagram of minimum system in minimum mode

3.10 Functioning of 8086 in Maximum Mode

University Questions

Q. What is maximum mode of 8086 ? **MU - Dec. 13, 10 Marks**

Q. Write short note on : Maximum mode of 8086. **MU - Dec. 14, 5 Marks**

Q. What is purpose of maximum mode of 8086 ? Give suitable example. **MU - Dec. 16, 5 Marks**

Q. Explain maximum mode of 8086 microprocessor. **MU - Dec. 17, Dec. 18, 10 Marks**

Q. Draw and explain maximum mode configuration of 8086 system. **MU - Dec. 19, 10 Marks**

- Fig. 3.10.1 shows the block diagram of the 8086 system in maximum mode.
- Note the use of the bus controller IC 8288 along with the other support chips.
- 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.

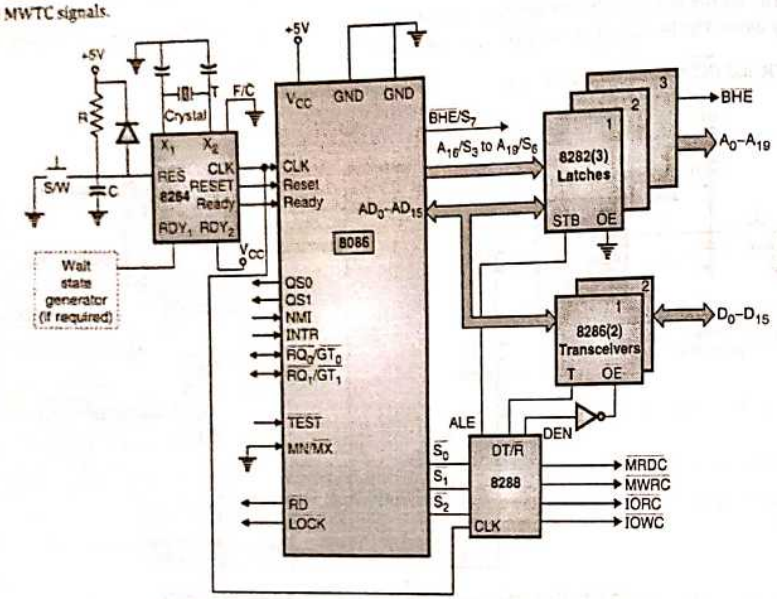


Fig. 3.10.1 : Block diagram of maximum mode minimum system

3.10.1 Comparison between Maximum Mode and Minimum Mode

University Questions

Q. How maximum mode differs from minimum mode ? **MU - Dec. 13, 10 Marks**

(i) The mode of operation i.e. maximum mode / minimum mode can be controlled by using pin 33 MN/$\overline{MX}$. The microprocessor system operates in the minimum mode by default on power on. If MN/$\overline{MX}$ pin is grounded, the system operates in maximum mode.

(ii) When only one CPU is to be used in a microcomputer system it is used in minimum mode operation. In the minimum mode CPU issues control signals that are required by the memory and I/O devices. For a multiprocessor system, the CPU operates in maximum mode. In maximum mode the control signals are issued by the Intel 8288 bus controller.

(iii) The pins 24 to 31 have alternate functions when the CPU operates in maximum and minimum mode. The pins used in minimum mode are M/$\overline{IO}$, $\overline{INTA}$, ALE, HOLD, HLDA, DT/$\overline{R}$, $\overline{DEN}$, $\overline{WR}$. The pins used in maximum mode are $\overline{S_2}$, $\overline{S_1}$, $\overline{S_0}$, $\overline{LOCK}$, QS_1, QS_0, $\overline{RQ_0}$ / $\overline{GT_0}$, $\overline{RQ_1}$ / $\overline{GT_1}$.

(iv) Minimum mode operation is less expensive because all the memory and I/O control signals are generated by the microprocessor itself. In maximum mode we require an external bus controller like 8288 that generates the control signals. Hence, maximum mode operation is expensive.

3.11 Maximum Mode Read/Write Cycle

University Questions

Q. Draw the timing diagram of memory write in maximum mode. **MU - May 12, 5 Marks**

Q. Explain with neat diagram memory read and write machine cycle in maximum mode. **MU - May 13, 10 Marks**

Q. Draw timing diagram of read operation on 8086 based system. **MU - May 16, 10 Marks**

Q. Draw and explain memory read and memory write machine cycle timing diagrams in maximum mode of 8086. **MU - May 19, 10 Marks**

READ CYCLE

Fig. 3.11.1 shows read cycle in a maximum mode of 8086. The cycle is identical to read cycle of 8086 in minimum mode of operation. The few differences we have those are as follows :

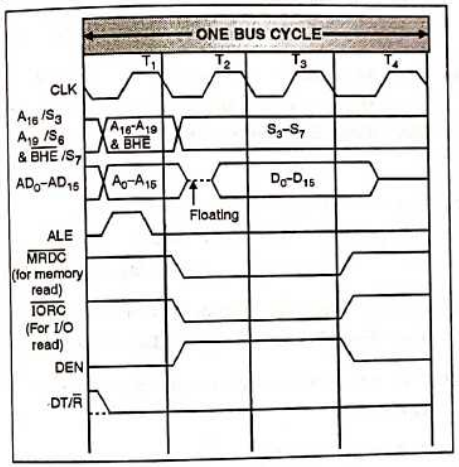


Fig. 3.11.1 : Maximum mode read bus cycle

(1) 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.

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

WRITE CYCLE

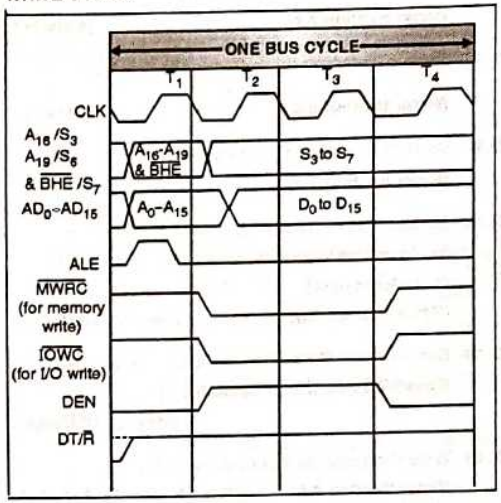


Fig. 3.11.2 : Maximum mode write cycle

Write cycle of 8086 in maximum mode is again similar to write cycle in minimum mode. Differences are already mentioned in points, in read cycle.

3.12 Exam Pack (Review and University Questions)

Q. 1 Write short note on : 8284 clock generator **(Refer Section 3.2)** **(May 11, May 12, Dec. 12, May 13, 7 Marks)**

Q. 2 Explain the hardware required to generate clock and reset signals. **(Refer Section 3.2)** **(May 14, 10 Marks)**

Q. 3 Draw the circuit diagram for generation of RESET signal and explain it's working. **(Refer Section 3.2.3)** **(Dec. 11, 5 Marks)**

Q. 4 Write short note on : Generation of reset signals in 8086 based system. **(Refer Section 3.2.3(8))** **(Dec. 15, 5 Marks)**

Q. 5 Explain power on reset circuit used in 8086 system. **(Refer Section 3.2.3)** **(Dec. 17, 5 Marks)**

Q. 6 Design interfacing of 8282 latches to 8086 system. **(Refer Section 3.3)** **(Dec. 15, 5 Marks)**

Q. 7 Explain 8286 as octal bus receiver. **(Refer Section 3.4)** **(5 Marks)**

Q. 8 Explain demultiplexing address and data bus. **(Refer Section 3.4.2)** **(5 Marks)**

Q. 9 Write short notes on : 8288 bus controller. **(Refer Section 3.5)** **(May 12, 7 Marks)**

Q. 10 Explain the necessity of a bus controller in 8086 maximum mode operation. Also explain the 8288 bus controller in detail. **(Refer Section 3.5)** **(Dec. 12, 10 Marks)**

Q. 11 Explain the 8288 bus controller block diagram. Explain it's use. **(Refer Section 3.5)** **(May 13, 10 Marks)**

Q. 12 Write short note on 8288 bus controller. **(Refer Section 3.5)** **(Dec. 15, Dec. 16, 5 Marks)**

Q. 13 Explain the status signals of Intel 8086 processor. Show all the possible combinations along with the processor state and 8288 command associated with each combination. **(Refer Section 3.5.2)** **(Dec. 11, 5 Marks)**

Q. 14 Explain 8086 timing diagrams. **(Refer Section 3.6)** **(5 Marks)**

Q. 15 Draw the timing diagram of Memory read in minimum mode. **(Refer Section 3.7)** **(May 12, 5 Marks)**

Q. 16 Draw and explain timing diagram for read operation in minimum mode of 8086. **(Refer Section 3.7)** **(Dec. 14, Dec. 17, 5 Marks)**

Q. 17 Draw timing diagram of read operation on 8086 based system. **(Refer Sections 3.7 and 3.11)** **(May 16, 10 Marks)**

Q. 18 Draw and explain timing diagram for write operation in minimum mode of 8086. **(Refer Section 3.8)** **(May 15, 5 Marks)**

Q. 19 What is maximum mode of 8086 ? How it differs from minimum mode ? **(Refer Sections 3.10 and 3.10.1)** **(Dec. 13, 10 Marks)**

Q. 20 Write short note on : Maximum mode of 8086. **(Refer Section 3.10)** **(Dec. 14, 5 Marks)**

Q. 21 What is purpose of maximum mode of 8086 ? Give suitable example. **(Refer Section 3.10)** **(Dec. 16, 5 Marks)**

Q. 22 Explain maximum mode of 8086 microprocessor. **(Refer Section 3.10)** **(Dec. 17, Dec. 18,10 Marks)**

Q. 23 Draw the timing diagram of memory write in maximum mode. **(Refer Section 3.11)** **(May 12, 5 Marks)**

Q. 24 Explain with neat diagram memory read and write machine cycle in maximum mode. **(Refer Section 3.11)** **(May 13, 10 Marks)**

Q. 25 Draw timing diagram of read operation on 8086 based system. **(Refer Sections 3.7 and 3.11)** **(May 16, 10 Marks)**

Q.26 Draw and explain memory read and memory write machine cycle timing diagrams in maximum mode of 8086. **(Refer Sections 3.7 and 3.11)** **(May 19, 10 Marks)**

Q.27 Explain minimum mode configuration of 8086 microprocessor. **(Refer Sections 3.9)** **(May 19, 10 Marks)**

Q. 28 Draw and explain demultiplexing of address bus in 8086. **(Refer Sections 3.4.2)** **(Dec. 19, 10 Marks)**

Q. 29 Draw and explain maximum mode configuration of 8086 system. **(Refer Sections 3.10)** **(Dec. 19, 10 Marks)**

□□□

MODULE 2

4 8086 Instruction Set

4.1 Programmer's Model of 8086

University Question

Q. Draw the programmers model of 8086 microprocessor and label it neatly. **(6 Marks)**

Q. Explain programming model of 8086. **MU - May 16, 5 Marks**

- 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)

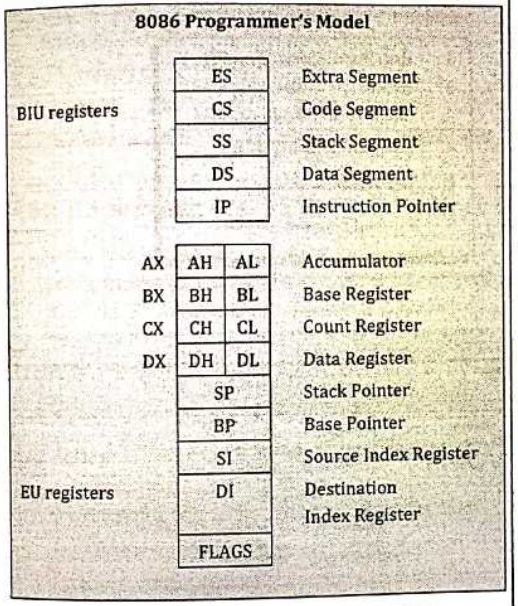


Fig. 4.1.1 : Programmer's model of 8086

- 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

Note : You need both a segment register and an offset to specify an address.

- 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).

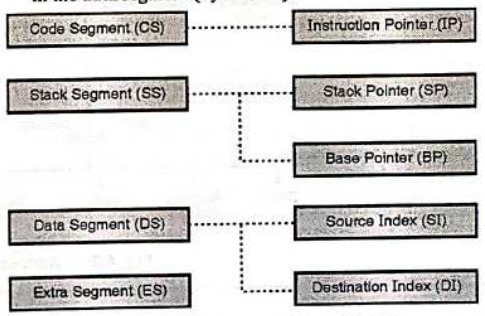


Fig. 4.1.2 : Address registers of 8086

Flag register

Fig. 4.1.3 : 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)

4.2 Addressing Modes

University Questions

Q. Explain the following addressing modes :
(a) Direct addressing
(b) Immediate addressing
(c) Register indirect addressing
(d) Based addressing
(e) Indexed addressing
Explain with suitable examples. **(5 Marks)**

Q. Write short note on : Addressing modes in the 8086. **MU - May 11 , Dec. 12 , Dec. 13 , 5 Marks**

Q. List different addressing modes of 8086. **MU - Dec. 16 , 5 Marks**

Q. Write addressing modes of following instructions of 8086.
(i) MOV al, [bx + si] (ii) AND cl, [2000]
(iii) ADD ax, [bx + si + 2000] (iv) IN al, dx
(v) POP BX **MU - Dec. 17 , 5 Marks**

Q. Explain different addressing modes of 8086 microprocessor. **MU - Dec. 18, 5 Marks**

Q. Write addressing modes of following instructions:
(i) MOV AX, [BX+SI]
(ii) AND CL,[2000]
(iii) IN AL,DX
(iv) JMP[BX+2]
(v) ADD AX,[BX+SI+5] **MU - Dec. 19, 5 Marks**

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

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

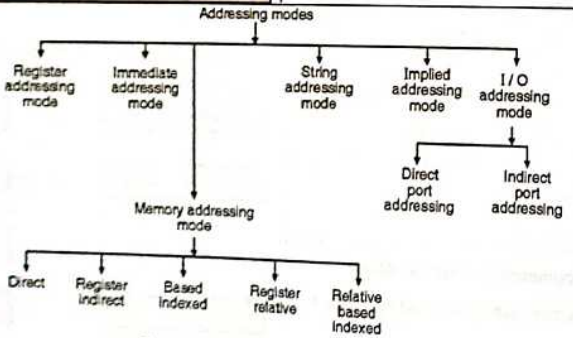


Fig. 4.2.1 : Addressing modes of 8086

4.2.1 Register Addressing Mode

In this mode of addressing, operand is in the register, and instruction specifies the particular register as shown in Fig. 4.2.2.

- The advantage of this addressing mode is that the access is faster.

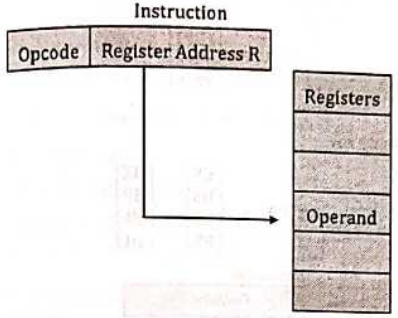


Fig. 4.2.2 : Register addressing

- 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.

4.2.2 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. 4.2.3 shows format of instruction encoded with immediate operand.

OPCODE	Immediate operand

Fig. 4.2.3 : Instruction encoded with an immediate operand

- The operand in this case could be either 8-bit or 16-bit.

Let's take one example :

e.g. MOV CL, 02 H

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

4.2.3 Memory Addressing Modes

University Question

Q. "Based indexed" addressing mode of 8086 is useful for an array element access explain. **(8 Marks)**

Q. For the following instruction compute the address of memory operand for 8086 :
(i) MOV AX, [BX]
(ii) MOV AL, [BP + SI]
Assume CS : 0100 H, DS : 0200 H, SS : 0400 H, ES : 0030 H, BP : 0010 H,
BX : 0020 H,
SI : 0030 H, SP = 0040 H. Clearly show computations. **(5 Marks)**

Q. Explain following addressing modes of Intel 8086. Write an instruction for each mode :
(a) Direct addressing mode
(b) Relative base indexed. **MU - Dec. 11, 5 Marks**

- For memory accesses, the processor needs to generate a 20-bit address. The registers in 8086 are of 16-bit. We have seen 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

The different memory addressing modes available are :

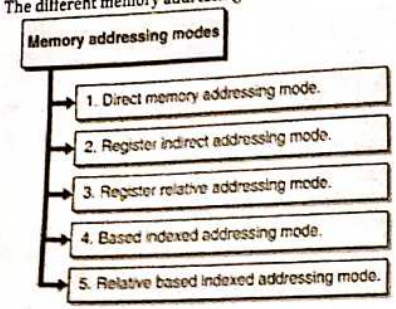


Fig. 4.2.4 : Memory Addressing Modes

1. 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. 4.2.5.

PA = segment : EA

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

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.

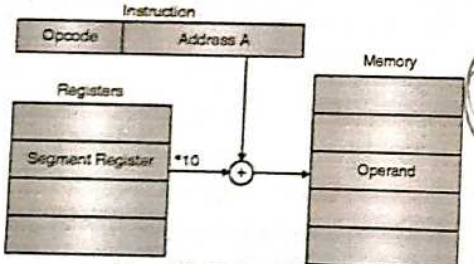


Fig. 4.2.5 : Direct addressing

Note : By default, it is DS segment register for direct addressing mode.

2. 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. 4.2.6.

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

Segment : EA

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

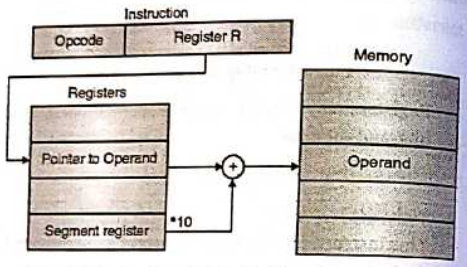


Fig. 4.2.6 : Register indirect addressing modes

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.

3. 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. 4.2.7.

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

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

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

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

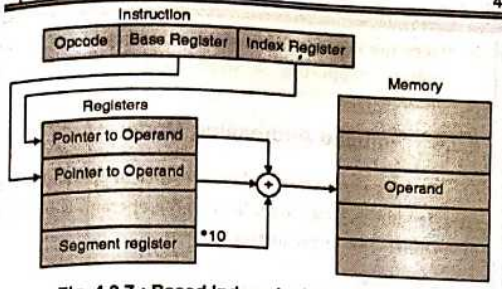


Fig. 4.2.7 : Based Indexed addressing mode

Let's see one example, to clear the concept.

MOV [BX + SI], AL

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

4. 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. 4.2.8.

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

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

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

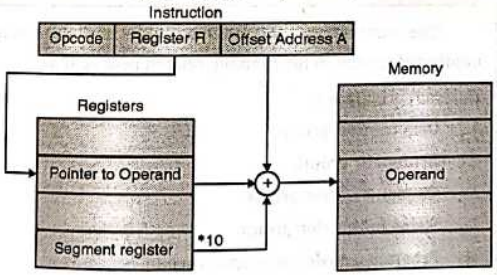


Fig. 4.2.8 : Register relative addressing mode

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.

5. Relative Based Indexed Addressing

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. 4.2.9.

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

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

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

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

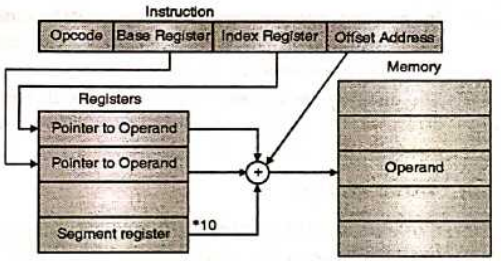


Fig. 4.2.9

Let's take one example :

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.

4.2.4 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.

4.2.5 I/O Addressing Mode in 8086

University Question

Q. Explain I/O related addressing mode of 8086. **MU - Dec. 14, 5 Marks**

This addressing mode is basically used for IOs. Under IO mapping we have :

1) Memory mapped I/O 2) I/O mapped I/O

1) Memory mapped I/O

- As the name says, 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 eof such I/O locations is that the access of I/O locations being normally slowers, it makes the processor to wait.

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.
- Direct Port addressing: As the name says, direct port addressing, in this case 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: In this case 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, we can have 16-bit address supporting a huge range of 65536 I/O locations

4.2.6 Implied Addressing Mode

In this case 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

4.3 Segment Override Prefix

- Normally for each offset, segment is fixed. But using ***segment override prefix*** one can change segment registers.
- For addressing modes DS may be overridden by CS, SS, or ES; and when BP is used, SS may be overridden by CS, DS, or ES. Specific cases that cannot involve overrides are as follows :
  1. The CS register is always used as the segment register when computing the address of the next instruction to be executed.
  2. For stack pointer SP, SS is THE SEGMENT REGISTER.
  3. For string operation ES is by default segment register for destination operand.

4.4 Instruction Set of 8086

Q. Explain instruction set of 8086. **(8 Marks)**

The instruction set of 8086/8088 is divided into number of groups, of functionally related instructions.

Different groups are :

1. Data transfer group.
2. Arithmetic group.
3. Bit manipulation group.
4. String instruction group.
5. Program transfer instruction group.
6. Process control instruction group.

Graphical presentation of different groups is as shown in Fig. 4.4.1.

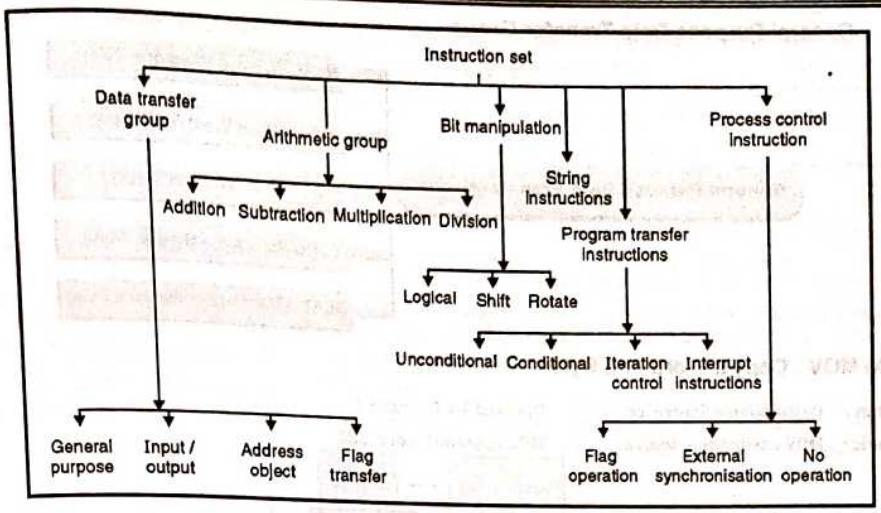


Fig. 4.4.1 : Instruction Set of 8086

4.5 Data Transfer Group

Q. Explain data transfer instructions. **(5 Marks)**

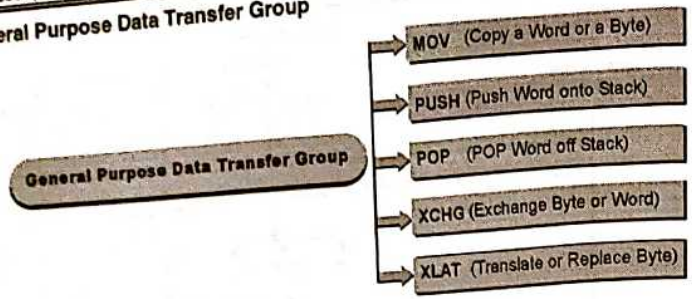
The 14 data transfer instructions are listed as follows :

Table 4.5.1 : Data transfer instructions

General Purpose		Address	
MOV	Move byte or word	LEA	Load effective address
PUSH	Push word onto stack		
POP	Pop word off stack	LDS	Load pointer using DS
XCHG	Exchange byte or word		
XLAT	Translate byte	LES	Load pointer using ES
Input/Output		**Flag Transfer**	
IN	Input byte or word	LAHF	Load AH register from flags
OUT	Output byte or word		
		SAHF	Store AH register in flags
		PUSHF	Push flags onto stack
		POPF	Pop flags off stack

These instructions transfer data from one location to another. Hence most of them do not require ALU and do not alter the contents of flag register, unless they transfer data into the flag register.

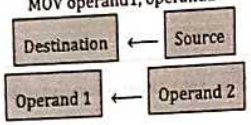
4.5.1 General Purpose Data Transfer Group



4.5.1(A) MOV : Copy a Word or a Byte

Algorithm : Destination = Source or Operand 1 = Operand 2

Mnemonic : MOV destination, source MOV operand1, operand2



Operation :

- The MOV instruction copies a word or a byte of data from a fixed/specified source to a fixed/specified destination.
- Following table contains valid source and destination operands.

Sr. No.	Destination	Source
1.	Memory	Accumulator
2.	Accumulator	Memory
3.	Register	Register
4.	Register	Memory
5.	Memory	Register.
6.	Register	Immediate
7.	Memory	Immediate
8.	Seg – Reg	Reg – 16
9.	Seg – Reg	Mem – 16
10.	Reg – 16	Seg – Reg
11.	Memory	Seg – Reg

Note: I have changed the tabular format of the first instruction. All the instructions are to be put in the same format

1. MOV memory, accumulator

Mnemonic	MOV accumulator, memory	**Flags**	No flags are affected.
Algorithm	memory, accumulator	**Add. Mode**	Indexed Addressing mode/Register Indirect addressing mode

Operation	Memory← accumulator	– This instruction copies the contents of accumulator to memory location specified in the instruction.
Example	MOV [SI], AL	– This instruction copies the contents of the AL register to memory location whose effective address is given by SI register.

2. MOV accumulator, memory

Mnemonic	MOV accumulator, memory	**Flags**	No flags are affected.
Algorithm	accumulator = memory	**Addr. Mode**	Register Direct Addressing Mode.

Operation	accumulator ↑ memory	– This instruction copies the contents of given memory locations to the accumulator.
Example	MOV AX, Temp_Result	– The contents of memory location Temp_result will be transferred/copied to the AL register. Then the IP will increment by 1 and contents of location after Temp_Result will be copied to the AH register. Refer Fig 4.5.1. – Let the contents of CS → 2105 H, IP → 187A H, DS → 2314 H and Temp_Result → FF H, AH → 00 H, AL → 0A H.

Generation of physical address (PA)

```
  2 3 1 4 0 H  →  Contents of DS
+     1 2 3 4 H  →  Offset of memory location
-----------------
  2 4 3 7 4 H  →  Physical address
```

The contents of Memory Location 24374 H will be transferred/copied to the AL Register & the contents of Memory Location 24375 H will be transferred/copied to the AH Register.

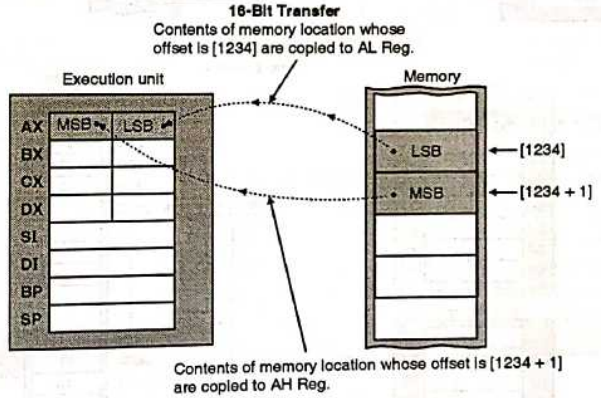


Fig. 4.5.1 : MOV AX, Temp_Result

Note : Temp_Result ← Memory location.

	Before execution	After execution	
Code Segment (CS)	2105 H	2105 H	
Instruction Pointer (IP)	**187AH**	**187 EH**	→ Note the change
Data Segment (DS)	2314 H	2314 H	
Register AL	**0A H**	**FF H**	→ Note the change
Register AH	**00 H**	**AA H**	→ Note the change
Memory location 24374H	FF H	FF H	Contents of memory
Memory location 24375H	AA H	AA H	location remain same
Figure	Refer Fig. 4.5.2	Refer Fig.4.5.3	

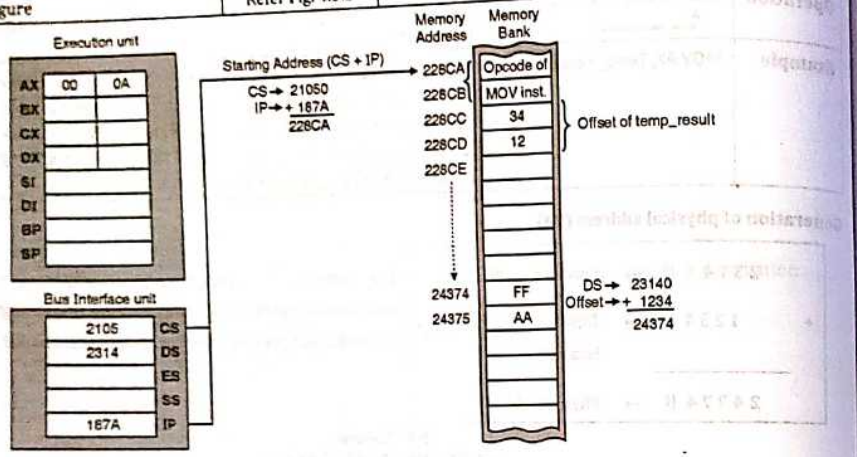


Fig. 4.5.2 : Before execution

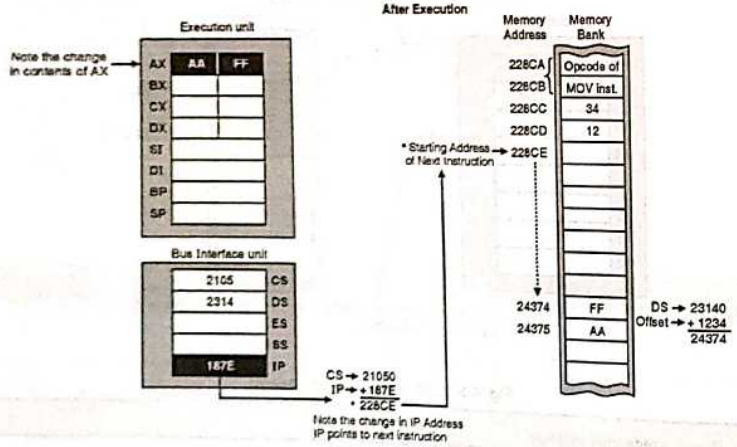


Fig. 4.5.3 : After execution

3. MOV destination register, source register

Mnemonic	MOV destination register, source register
Algorithm	destination register = source register.

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

Operation	dest reg source reg	This instruction will copy the contents of source register to the destination register. This instruction cannot be used to copy the data from one segment register to another segment register. The data from segment register needs to be copied to a general register first.
Example	MOV AX, BX	This instruction copies the contents of BX register to AX register. The LSB of BX i.e. BL is copied to AL and MSB of BX i.e. BH is copied to AH.

4. MOV register, memory

Mnemonic	MOV register, memory
Algorithm	register = memory

Flags	No flags are affected
Addr. Mode	Indexed / Register Relative addressing mode

Operation	reg memory	– This instruction will copy the contents of the memory location specified in the instruction to the destination register. – This instruction can be 8/16 bit.
Example	MOV CX, COUNT [DI]	– This instruction copies the contents of two memory locations to the CX register. Contents of first location are copied to CL and contents of the next location to CH register.

Note: Count is any variable which is defined while programming.

5. MOV memory, register

Mnemonic	MOV memory, register
Algorithm	memory = register

Flags	No flags are affected
Addr. Mode	Indexed / Register Relative addressing mode

Operation	memory, register	– This instruction will copy the contents of source register to the specified memory location in the address. This can be 8/16 bit transfer. – This instruction cannot be used to access the value of CS and IP registers.
Example	MOV COUNT [DI], CX	– This instruction copies the contents of CX to two memory locations. Contents of CL are copied to first location and CH to second location. – The effective address is represented by COUNT + contents of [DI].

Note : Count is any variable which is defined while programming.

6. MOV register, immediate

Mnemonic	MOV register, immediate
Algorithm	register = immediate data

Flags	No flags are affected.
Addr. Mode	Immediate addressing mode

Operation	register, immediate	– This instruction will copy the immediate data to the required destination register. – This instruction cannot be used to copy immediate data to a segment register.
Example	MOV CL, 02 H	– This instruction copies the immediate number 2H in the CL register.

7. MOV memory, immediate

Mnemonic	MOV memory, immediate
Algorithm	memory = immediate data

Flags	No flags are affected
Addr. Mode	Immediate addressing mode

Operation	memory immediate data	– This instruction will copy the immediate data to the memory location specified in the instruction.
Example	MOV COUNT [DI], 2DH i.e. MOV COUNT [DI], 2DH	– This instruction copies immediate number 2DH to the required memory location. – EA of the memory location is the sum of displacement COUNT and the contents of DI (EA = COUNT + DI)

Let the contents of CS → 2105 H, IP → 167A H, DI → 1100 H, COUNT → 1000 H, DS → 2314 H

Generation of EA			Generation of PA		
1000 H	→	COUNT	23140 H	→	[DS]
+ 1100 H	→	[DI]	+ 2100 H	→	EA
2100 H	→	EA	25240 H	→	PA

Immediate data 2D H will be loaded at location 25240 H.

8. MOV seg. reg., reg-16

Mnemonic	MOV seg. reg., reg-16
Algorithm	segment reg = 16 bit register

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

Operation	seg reg 16 bit reg	– This instruction will copy the 16 bit source register contents to the destination segment register. – It cannot be used to copy the contents to the CS register. – This instruction only allows 16 bit transfer, as segment register is 16 bit. – An 8 bit instruction like MOV DS, AL is illegal.
Example	MOV DS, AX i.e. i.e. MOV DS, AX	– This instruction copies a word from AX to data segment register.

9. MOV seg-reg, mem 16

Mnemonic	MOV seg-reg, mem 16
Algorithm	seg-reg = mem 16

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

Operation	seg-reg mem 16	– This instruction will copy the contents of specified memory locations to the desired segment register. – It cannot be used to alter the contents of the CS register. – This instruction allows only 16 bit transfer
Example	MOV DS, [SI] i.e. MOV DS , [SI]	– This instruction copies word from 2 memory locations to data segment register. – The memory location in DS is at displacement 1100. This offset 1100 is stored in the SI register.

Assume contents of DS : 2314 H. Let the contents of CS → 2105 H, IP → 187A H, DS → 2314 H.

Generation of physical address :

23140 H	→	Contents of DS
+ 1100 H	→	Memory location offset
24240 H	→	Physical address

The contents of memory location (24240) and location (24241) are copied to DS register.

10. MOV reg-16, seg reg.

Mnemonic	MOV reg-16, seg reg.
Algorithm	reg-16 = seg reg

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

Operation	reg 16, seg-reg.	– This instruction will copy the contents of a 16 bit segment register to a destination register specified in the instruction.
Example	MOV AX, DS. i.e. MOV AX, DS	– This instruction copies a word from the data segment register to AX register.

11. MOV memory, seg-reg

Mnemonic	MOV memory, seg-reg
Algorithm	memory = seg-reg

Flags	No flags are affected.
Addr. Mode	Register Direct Addressing

Operation	mem seg-reg	– This instruction will copy the contents of a 16 bit segment register to required destination memory locations.
Example	MOV count, DS i.e. MOV count, DS	– This instruction copies word from the data segment register to two memory locations, whose offset of displacement in DS represented by count.

Note : Count is a general variable.

4.5.1(B) PUSH : Push Word onto Stack

Mnemonic	PUSH Source
Algorithm	SP = SP – 2 SS : [SP] (Top of the Stack) = Operand

Flags	No flags are affected.
Addr. Mode	Register addressing mode

Operation	SP → SP - 2 SS → data from specified source	– This instruction decrements the stack pointer by 2 and copies a word from a specified source to the location in the stack segment where stack pointer points. – The source of operand (16 bit data to be stored on stack) can be a general purpose register, flag register, segment register or memory. – The stack segment register and stack pointer must be initialised before using this instruction. – PUSH can be used to save data on the stack so that it will not be destroyed by a execution of successive instructions.

Example : PUSH AX

Note : In this case physical address is the location of top of stack, where SP points before execution.

Note :
1. PUSH operates only on 16 bit data
2. PUSH CS is not allowed
3. PUSH cannot be used to push immediate data onto the stack.

4.5.1(C) POP : Pop Word off Stack

Mnemonic	POP Destination OR POP Operand	**Flags**	No flags are affected
Algorithm	Operand = SS : [SP] (top of stack) SP = SP + 2	**Addr. Mode**	Register Addressing mode

Operation	SS → Data copied to destination or operand SS → SS : [SP + 2]	– This instruction copies a word (two successive memory location contents) from stack segment in memory to a destination specified in the instruction. – The destination can be a general purpose register, flag register, a segment register or a memory location. – The data in the stack pointer is not changed. – After the word is copied to specified destination the stack pointer is automatically incremented by 2 to point to next word on stack.
Example	POP AX	– This instruction copies a word (two successive memory location contents) from stack segment in memory to the AX register.

Note: (1) POP CS is illegal. (2) POP operates only on 16 bit data

4.5.1(D) XCHG : Exchange byte or word

Mnemonic	XCHG destination, source.	**Flags**	No flags are affected
Algorithm	destination = source	**Addr. Mode**	Implied addressing mode

Operation	destination ↔ source	This instruction exchanges the contents of a register with contents of another register or the contents of a register with contents of memory location. The register can be 8/16 bit.

		XCHG cannot directly exchange the contents of two memory locations. e.g. XCHG [1234],[5068] ; This transfer is not possible The source and destination must both be words or they both must be bytes.
Example	XCHG AX, BX i.e. AX ↔ BX	This instruction will exchange the word in AX register with the word in BX register. Contents of AL ↔ Contents of BL, Contents of AH ↔ Contents of BH

Note :
1. The segment registers cannot be used as registers in this instruction. The memory can use DS, ES, SS as segment registers.
2. Since immediate operands cannot be changed neither operand in this instruction can be immediate.

4.5.1(E) XLAT : Translate or Replace Byte

University Question

Q. Explain the following instruction in 8086 : XLAT **MU - Dec. 19, 5 Marks**

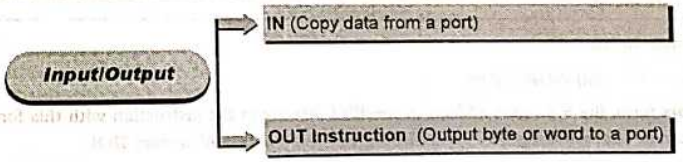
Mnemonic	XLAT/XLATB [B indicates byte operation] (meaning of XLAT and XLATB is same, either XLAT or XLATB can be written)	**Flags**	No flags are affected.
Algorithm	AL = DS : [BX + unsigned AL]	**Addr. Mode**	Implied Addressing mode

Operation	AL ← DS : [BX + AL]	– This instruction replaces a byte in AL register with a byte from a look up table in the memory. i.e. it copies the value of memory byte at location DS : [BX + unsigned AL] to the AL register. – Here contents of AL before execution acts as index to the desired location in lookup table. – This instruction is used to translate a byte from one code to another code. – Mostly this concept is used to convert BCD to seven segment code or ASCII to EDCBIC code conversion

4.5.2 Input / Output

University Question

Q. Write a short notes on : Fixed port and variable port addressing. **MU - Dec. 11 , May 13 , 5 Marks**



These instructions are basically related to communication with I/O devices, mapped in I/O map.

1. IN : Copy data from a port

Mnemonic	IN accumulator, port address.	**Flags**	No flags are affected
Algorithm	--	**Addr. Mode**	Direct port addressing mode

Operation	AX ← Contents of port or AL ← contents of port.	– This instruction will copy data from a port whose address is given in the instruction to AX register or AL register. – The data can be either 8 bit or 16 bit.
Example	IN AL, C8H.	– This instruction will copy the contents of port whose address is C8H to the AL register.

- The IN instruction has two possible formats

  (i) Fixed port (ii) variable port.

- For the **Fixed port**, the port address is specified directly in the instruction. The port numbers are from 00 to FF i.e. 8 bit address is directly specified. Thus addressing mode is Direct port addressing. The above example is an example of direct port addressing.
- For the **Variable port** IN instruction, the port address is loaded into the DX register before the IN instruction. Since DX is a 16 bit register, the port address can be any number between 0000H and FFFFH. Therefore it is possible to address up to 65, 536 ports in this mode.

  e.g. MOV DX, 0FF0H; Initialize DX to point to port.

  IN AL, DX: Input an 8 bit data from port 0FF0 to AL i.e. contents of port 0FF0 are copied to the AL register. The addressing mode is Indirect port addressing mode.

Note : This instruction is basically related to communication with I/O devices which are mapped in the I/O mapped I/O.

2. OUT Instruction - Output byte or word to a port

Mnemonic	OUT port address, Accumulator.
Algorithm	--

Flags	No flags are affected
Addr. Mode	Indirect port addressing mode

Operation	Contents of AX → Port address / Contents of AL → port address.	– This instruction copies a byte from AL or a word from AX to the specified port.
Example	MOV DX, FFF8H OUT DX, AL	– Load desired port address in DX. – Copies contents of AL to given port address in register DX

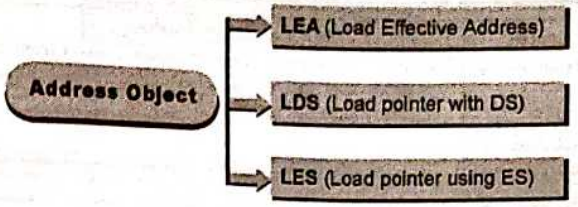
It has two possible forms.

(i) Fixed port (ii) Variable port

- In the **fixed port** form, the 8 bit port address is specified directly in the instruction with this form, any one of 256 possible ports can be addressed. e.g. OUT 3B H, AL ; copies the contents of AL to port 3B H.
- The addressing mode is Direct port addressing mode.
- In the **variable port** form, the contents of AL or AX will be copied to the port at an address contained in DX. The DX register should be loaded with the desired port address. The above example in table is an example of indirect port addressing.

4.5.3 Address Object

These instructions manipulate the addresses of the variables, rather than the contents or values of the variables. These are mainly used for list processing, based variables and string operation.



1. LEA : Load Effective Address

Mnemonic	LEA register, source.
Algorithm	REG = Address of memory (offset)

Flags	No flags are affected
Addr. Mode	Register Direct Addressing

Operation	REG ← source.	– This instruction determines the offset of variables or memory location named as source and puts the offset in the indicated 16 bit register. – Generally this instruction is replaced by MOV when assembling is possible. – Normally the offset is loaded into index register or base pointer registers such as SI, DI, BX, BP.
Example	LEA AX, COUNT	– This instruction loads AX with the offset of COUNT in DS. AX ← offset of COUNT. Say Offset of COUNT= 5051 in DS.

2. LDS : Load pointer with DS

(Load Register and DS with words from memory)

Mnemonic	LDS register, source.
Algorithm	REG = First word, DS = Second word

Flags	No flags are affected
Addr. Mode	Register Direct Addressing.

Operation	REG ← Source, DS ← (Source + 2)	– The source is always a memory location. DS is used as a segment register for memory. – This instruction is a 2 byte instruction. It copies a word from two memory locations into register specified in the instruction. It then copies a word from the next two memory locations into the DS register.
Example	LDS BX, Count	

Note : The source cannot be a register.

3. LES - Load pointer using ES

(Load register and ES with words from the memory)

Mnemonic	LES register, source.
Algorithm	REG = First word, ES = Second word

Flags	No flags are affected
Addr. Mode	Register Direct Addressing

Operation	REG ← Source, ES ← (source + 2)	The source is always a memory location. This is a 2 byte instruction. It copies a word from two memory locations into register specified in the instruction. It then copies a word from next two memory locations into the ES register.
Example	LES BX, Count	

Note: The source cannot be a register.

4.5.4 Flag Instructions (Flag Transfer)

University Question

Q. Explain the following instructions in 8086 : LAHF. **MU - Dec. 15, Dec. 19, 5 Marks**

These instructions are related to movement of flag register to/from a register and memory.

- **Flag Instructions (Flag Transfer)**
  - LAHF (Load AH register from flags)
  - SAHF (Store AH register in flags)
  - PUSHF (Push flags onto stack)
  - POPF (Pop flags off stack)

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

Note : U indicates undefined i.e. value of that bit (bits $D_1/D_3/D_5$) is in determined [i.e. it may be 0 or 1 anything]

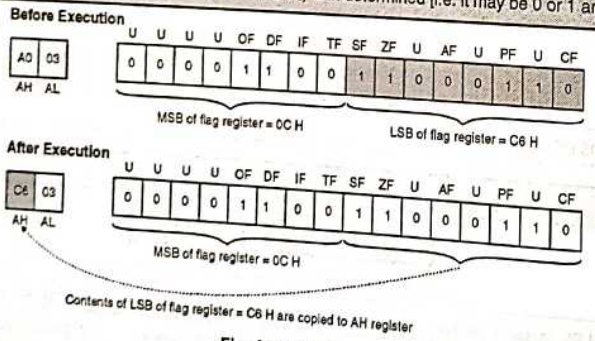


Fig. 4.5.4 : LAHF

2. SAHF – Store AH register in flags

(Copy contents of AH to lower byte of flag register)

Mnemonic	SAHF
Algorithm	AH = flag register

Flags	All the flags are changed.
Addr. Mode	Implied Addressing mode

Operation	AH → Lower byte of flag register	– This instruction copies the contents of AH register to the lower byte of flag register. – It is included for 8085 compatibility. – The OF, DF, IF and TF are not affected.

Let the contents of Flag register be 0CC6 H

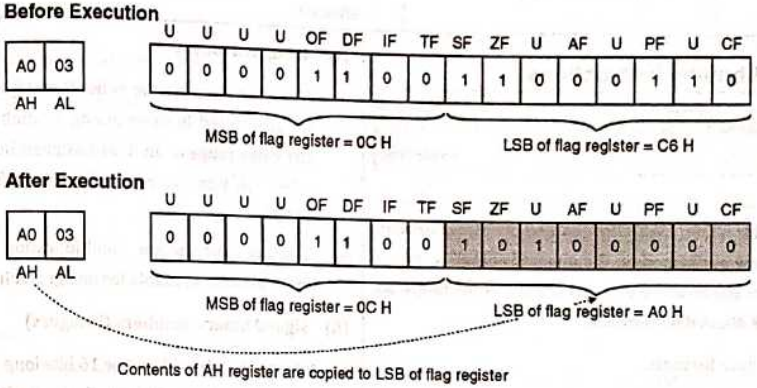


Fig. 4.5.5 : SAHF

3. PUSHF – Push flags onto stack

(PUSH flag register on the stack)

Mnemonic	PUSHF
Algorithm	SP = SP – 2 SS : [SP] (top of stack) = operand

Flags	No flags are changed.
Addr. Mode	Register Addressing mode

Operation	SP → SP – 2 SS → data from flag register.	– This instruction decrements the stack pointer by 2 and copies word in the flag register to the memory location pointed by stack pointer. – The stack segment register is not affected.

4. POPF – Pop flags off stack

Mnemonic	POPF
Algorithm	SS = data to flag register SP = SP + 2
Flags	All flags are affected.
Addr. Mode	Register Addressing mode
Operation	SS : [SP] → Copy data to flag register SP = SP + 2 SS : [SP] → SS : [SP + 2]

- This instruction copies a word from the two memory locations at the top of the stack to flag register and increments the stack pointer by 2.
- The stack segment register and word on the stack are not affected.

4.6 Arithmetic Instructions

Q. Explain arithmetic instructions. (5 Marks)

Under arithmetic operation, 8086/8088 provides an addition, subtraction, multiplication and division. These all operations are performed on the operand (data). As far as operands are concerned we have varieties of operands. So first let's talk about data formats.

Arithmetic data formats

8086/8088 arithmetic operations may be performed on four types of numbers :

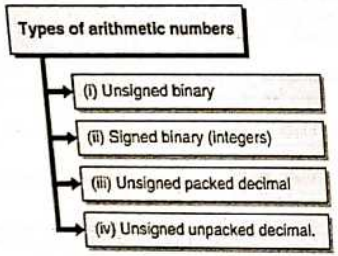


Fig. 4.6.1 : Types of arithmetic numbers

Here important point to be considered is ***the processor always assumes that the operands specified in arithmetic instructions contain data that represent valid numbers for the type of instruction being performed. Invalid data may produce unpredictable result.***

(i) Unsigned binary

- These numbers may be either 8 or 16 bits long; all bits are considered in determining a number's magnitude. The value range of an 8 bit unsigned binary number is 0-255; 16 bits can represent values from 0 through 65,535.
- Addition, subtraction, multiplication and division operations are available for unsigned binary numbers.

(ii) Signed binary numbers (Integers)

- Integers may be either 8 or 16 bits long. The high order (leftmost) bit is interpreted as number's sign. 0 → Positive, 1 → Negative.
- Negative numbers are represented in standard **two's complement notation**. The range of numbers for 8 bits are – 128 through + 127; 16 bit integers may range from – 32,768 through + 32.767. The value zero has positive sign. Multiplication and division operations are provided for signed binary numbers.
- Addition and subtraction are performed with the unsigned binary instructions. Instruction is also available to detect overflow into the sign bit. For signed operations the programmer has to interpret the result from the 'C', 'S' and 'O' flags.

(iii) Packed decimal numbers

- These numbers are stored as unsigned byte quantities.

- The byte is treated as having one decimal digit in each half byte (nibble); the digit in the high order half byte is the most significant. Hexadecimal values 0-9 are valid in each half byte, and the range of packed decimal number is 0-99.
- Addition and subtractions are performed in two steps :
  1. First an unsigned binary instruction is used to produce an intermediate result in register AL. (AF - Auxiliary carry flag is set reset).
  2. After that, an adjustment operation is performed which changes the intermediate value in AL to a final correct packed decimal result.

Multiplication and division adjustments are not available for packed decimal numbers.

(iv) Unpacked decimal numbers

- These numbers are stored as unsigned byte quantities. The magnitude of the number is determined from the low order half byte (nibble).
- Hexadecimal values 0-9 are valid data and are interpreted as decimal numbers. The high order half byte (nibble) must be zero for multiplication and division; it may contain any value for addition and subtraction.
- Arithmetic on unpacked decimal numbers is performed in two steps :
  1. The unsigned binary addition/subtraction and multiplication is used to produce an intermediate value (result) in register AL.
  2. After that an adjustment instruction then changes the value in AL to a final correct unpacked decimal number.
- Division is performed similarly, except that the adjustment is carried out on the numerator operand in register AL first, then a following unsigned binary division instruction produces a correct result.
- Unpacked decimal numbers are similar to ASCII character representation of the digits 0-9. Note, however, that the high-order half byte of an ASCII numerals is always 3H.
- Unpacked decimal arithmetic may be performed on ASCII numerals under following conditions :
  1) The high order nibble of an ASCII numeral must be set to 0H prior to multiplication or division.
  2. Unpacked decimal arithmetic leaves the high order half byte (nibble) set to 0H; it must be set to 3H to produce a valid ASCII numeral.

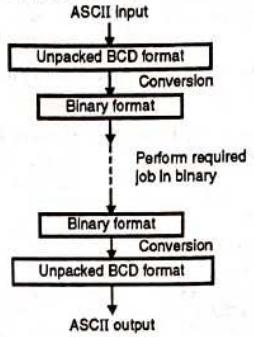


Fig. 4.6.2 : Conversion process from ASCII to binary and V.V.

At this stage it is needless to inform you that 8086/8088 is used in ***computer***. As far as computer is concerned, it follows ASCII input and output. We get from keyboard ASCII character and we have to send out ASCII character to monitor. But internally computer operates on binary only, therefore we have to convert ASCII to binary.

At this stage we have seen varieties of data formats. Let's say if a hex number is present in memory then how microprocessor looks at to it for different data formats. The same is shown in Table 4.6.1.

Table 4.6.1 : Arithmetic interpretation of 8-bit numbers

Hex	Bit Pattern	Unsigned Binary	Signed Binary	Unpacked Decimal	Packed Decimal
07	00000 111	7	+7	7	7
89	10001 001	137	– 119	invalid	89
C5	11000 101	197	– 59	invalid	invalid

- As seen 07H number fits within 0-9, therefore valid under all data format. 89H number is invalid under unpacked decimal (BCD), because in that upper nibble has to be zero. C5H is invalid for unpacked as well as packed BCD format because in BCD our number has to be within 0-9.
- As mentioned under arithmetic operations we have four basic subgroups i.e. addition, subtraction, multiplication and division. Table 4.6.2 lists out arithmetic instructions available in 8086 microprocessor, under these subgroups.

Table 4.6.2 : Arithmetic Instructions

Addition		Multiplication	
ADD	Add byte or word	MUL	Multiply byte or word unsigned
ADC	Add byte or word with carry	IMUL	Multiply byte or word signed
INC	Increment byte or word by 1	AAM	Integer multiply byte or word ASCII adjust for multiply
AAA	ASCII adjust for addition		
DAA	Decimal adjustment for addition		
Subtraction		**Division**	
SUB	Subtract byte or word	DIV	Divide byte or word unsigned
SBB	Subtract byte or word with borrow	IDIV	Integer divide byte or word
DEC	Decrement byte or word by 1	AAD	ASCII adjust for division
NEG	Negate byte or word (2's complement)	CBW	Convert byte to word
CMP	Compare byte or word	CWD	Convert word to doubleword
AAS	ASCII adjust for subtraction		
DAS	Decimal adjust for subtraction		

We will study the subgroups one by one.

4.6.1 Addition

University Question

Q. Explain the following Intel 8086 assembly language instructions giving example :
(i) DAA

MU - Dec. 11, Dec. 12, May 13, 2 Marks

- Addition
  - ADD (Add byte or word)
  - ADC (Add with carry)
  - INC (Increment byte or word by 1)
  - AAA (ASCII Adjust after addition)
  - DAA (Decimal adjustment for addition)

1. ADD – Add byte or word

Mnemonic: ADD destination, source

Algorithm: destination = destination + source

Operation: (Destination) ← (destination) + (source)

This instruction adds a number from source to number from destination and puts the result to specified destination.

Sr. No.	Destination	Source
1.	Register	Register
2.	Register	Memory
3.	Memory	Register
4.	Register	Immediate
5.	Memory	Immediate
6.	Accumulator	Immediate

Note : (i) Both the operands i.e. source and destination, cannot be memory locations.
e.g. ADD [1234], [5678] ⇒ Incorrect
(ii) Source and destination both have to be of same type i.e. byte or word.

1. ADD register, register

Mnemonic	ADD register, register.
Algorithm	register = register + register

Flags	All the flags are affected
Addr. Mode	Register addressing mode.

Operation	register ← register + register	– This instruction adds the data in registers. The result is stored in register. It can be 8 bit / 16 bit instruction. – This instruction can be 8/16 bit.
Example	ADD BL, CL	– This instruction adds the data in register CL and BL. The result is stored in BL register. It can be 8 bit/ 16 bit instruction BL ← BL + CL

Note: The register cannot be a segment register.

2. ADD register, memory

Mnemonic	ADD register, memory
Algorithm	register = register + memory

Flags	All the flags are affected.
Addr. Mode	Register addressing mode

Operation	register ← register + contents of memory location	– This instruction add a words from memory locations and required register. The result is stored in the register. – This instruction can be 8/16 bit.
Example	ADD AX, [2048] i.e. AX ← AX + contents of memory location	– This instruction adds the data at memory locations whose offset in DS are [2048] and [2049] with data in AX register. The result is stored in the AX register.

Note : 1. The register cannot be a segment register.
2. The addition of two memory locations is not allowed.
e.g. ADD [1024], [2048] is not allowed.
3. The memory uses DS as a segment register.

3. ADD memory, register.

Mnemonic	ADD memory, register.
Algorithm	memory = register + register

Flags	All the flags are affected.
Addr. Mode	Register Direct addressing mode

Operation	memory ← register + contents of memory location	– This instruction add a words from memory locations and required register. The result is stored at the memory location. – This instruction can be 8/16 bit.
Example	ADD [2048], AX	– This instruction adds the data at memory locations whose offset in DS are [2048] and [2049] with data in AL register and AH register. – The result is stored at memory locations whose offset in DS are [2048] and [2049] i.e. (25188 H and 25189 H).

Note : 1. The register cannot be a segment register.
2. The addition of two memory locations is not allowed.
e.g. ADD [1024], [2048] is not allowed .
3. The memory uses DS as a segment register.

4. ADD register, immediate / ADD accumulator, immediate.

Mnemonic	ADD register, immediate data.
Algorithm	register = register + immediate data

Flags	All flags are affected.
Addr. Mode	Immediate addressing mode.

Operation	register ← register + immediate data	– This instruction adds immediate number to contents of register. The result of addition is stored in the register. – This immediate data can be 8/16 bit .
Example	ADD AL, 74 H i.e. AL ← AL + 74	– This instruction adds the immediate number 74 H with the contents of AL register. The result is stored in AL.

Note : The segment registers cannot be used as registers.

5. ADD memory, immediate data.

Mnemonic	ADD memory, immediate data.
Algorithm	memory = memory + immediate data

Flags	All flags are affected.
Addr. Mode	Immediate addressing mode.

Operation	memory ← memory + immediate data	– This instruction will add immediate Data with contents of memory location whose offset is given in the instruction. The result of addition is stored at the memory location. – The immediate data can be 8 / 16 bit.
Example	ADD COST, 75 H. i.e. COST ← COST + 75 H [COST : is a memory location]	– This instruction will add immediate Data 75 H with contents of memory location COST whose offset is given in the instruction [i.e. memory location 28354 H] – Result of addition is stored at location COST.

2. ADC – Add with carry

Mnemonic	ADC destination, source.
Algorithm	destination = destination + source + CY

Flags	All flags are affected.
Addr. Mode	Register Immediate addressing mode.

Operation	destination ← destination + source + CY	– This instructions adds destination operand contents, source operand contents and carry flag and answer is stored back to destination operand. – The source and destination can be 8/16 bit register or memory location . The source and destination can also be a 8/16 bit register or memory location and immediate data. – The segment registers cannot be used. The memory uses DS as segment register.

Example	ADC AL, BL i.e. AL ← AL + BL + CY	– The addition of two memory locations along with carry is not possible . – It is easy to perform multiple- precision arithmetic by using ADC instruction. – This instruction adds the contents of AL register with BL register and contents of CY.

3. INC – Increment byte or word by 1

Mnemonic	INC Destination
Algorithm	destination =destination + 1

Flags	All the flags except carry flag are affected.
Addr. Mode	Implied addressing mode

Operation	destination ← destination + 1	– This instruction adds 1 to the destination operand. – The operand may be a byte or word and is treated as an unsigned binary number. – The destination operand may be a register or memory location.
Example	INC CX INC AL	– Add 1 to contents of CX – Add 1 to contents of AL register.

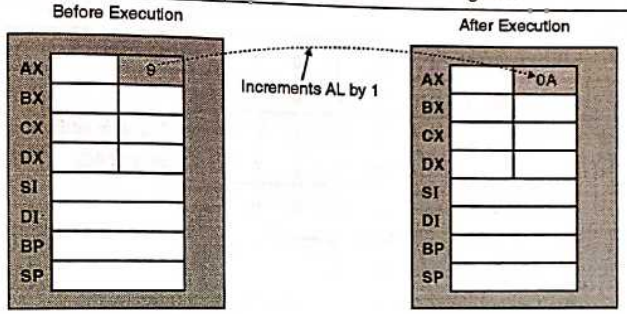


Fig. 4.6.3 : INC AL

Note : 1) Carry flag is not affected i.e. if 8 bit destination has FFH or 16 bit destination has FFFF, then incrementing the same will result to all 0's with No carry.
2) The segment registers cannot be incremented by this instruction.

4. AAA – ASCII Adjust after addition

Mnemonic	AAA
Algorithm	If lower nibble of AL > 9 or AF = 1 then : AL = AL + 6 AH = AH + 1 AF = 1

Flags	AF and CF flags are changed, while OF, PF, SF, ZF are left undersigned.
Addr. Mode	Implied addressing mode

	CF = 1 else : AF = 0 CF = 0 In both cases, clear the higher nibble of AL

Operation	– Numerical data coming into a computer from a terminal through keyboard is usually in ASCII code. The numbers 0 to 9 are represented by ASCII codes 30 H to 39 H. The 8086 allows to add the ASCII codes for two decimal digits without masking off "3" in the upper nibble of each. After addition, AAA instruction is used to make sure that the result is the correct unpacked BCD. – The AAA instruction works only on AL register. – If the lower nibble of AL register after addition is greater than 9, add 6 to it and increment AH by 1. The auxiliary carry flag and carry flag are set.
Example	Assume AL = 0 0 1 1 0 1 0 1 ASCII 5 BL = 0 0 1 1 1 0 0 1 ASCII 9 Add AL, BL

```
        0 0 1 1   0 1 0 1
+       0 0 1 1   1 0 0 1
  ------------------------
        0 1 1 0   1 1 1 0   →  Result of addition
+                 0 1 1 0      Invalid BCD
  ------------------------
AAA     0 0 0 0   0 1 0 0
Carry → [1]  Clear higher      ⇓
             nibble
```

(04 H) → Result in AL valid BCD.
(01 H) → Result in AH. The carry is stored in AH register

5. DAA – Decimal adjustment for addition

Mnemonic	DAA
Algorithm	If lower nibble of AL > 9 or AF = 1 then, AL = AL + 06 H AF = 1 If AL > 9 F H or CF = 1 then, AL = AL + 60 H CF = 1

Flags	It changes AF, CF, PF, ZF and SF.
Addr. Mode	Implied addressing mode

Operation	AL ← Sum in adjusted to packed BCD format

– This instruction is used to make sure that the result of adding two packed BCD numbers is adjusted to be a valid BCD number.
– It operates only on AL register.

– If number in the lower nibble of AL register after addition is greater than 9 or if the auxiliary carry flag is set add 6.
– If the higher nibble of AH is greater than 9 or if the carry flag is set then add 60 H.

Example
If AL = 59 H valid BCD, BL = 34 H valid BCD

```
ADD AL, BL
        0 1 0 1   1 0 0 1
+       0 0 1 1   0 1 0 0
  ------------------------
        1 0 0 0   1 1 0 1
          8           D
```

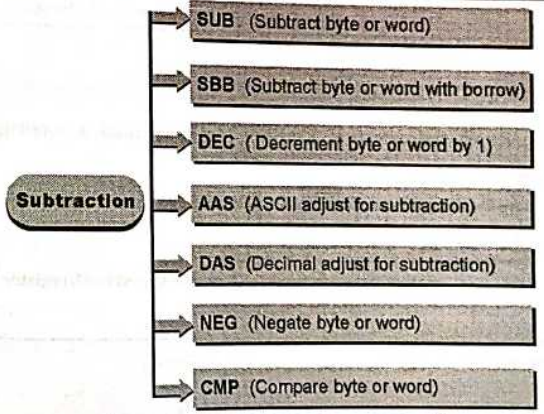
AL = 8 DH invalid BCD after addition of AL and BL

```
DAA →   1 0 0 0   1 1 0 1
+       0 0 0 0   0 1 1 0
  ------------------------
        1 0 0 1   0 0 1 1
          9           3
```

∴AL = 93 H BCD

4.6.2 Subtraction

University Question
Q. Explain following instructions with one example each (wrt 8086).
(i) CMP
MU - May 12, 2 Marks



1. SUB – Subtract byte or word

Mnemonic	SUB destination, source
Algorithm	destination = destination – source

Flags	The flags affected are AF, CF, OF, PF, SF and ZF.
Addr. Mode	–

Operation	destination ← destination – source	– This instruction subtracts a number from source with number from destination and puts result in destination location. – Both the operands i.e. source and destination, cannot be memory locations.

The type of both the operands should match i.e. byte or word.

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

1. SUB Register, Register

Mnemonic	SUB destination register, source register	**Flags**	All flags are affected
Algorithm	destination register = destination register – source register	**Addr. Mode**	Register addressing
Operation	destination register ← destination register – source register	This instruction subtracts a number from source register with number from destination register and puts result in destination register.	

Example : SUB BL, CL

BL ← BL – CL

This instruction subtracts the contents of CL register from BL and result is stored in BL.

Let CL → 04H , BL → 07 H

```
  07  →  BL
– 04  →  CL
  03  →  Result of subtraction which will be stored in BL register
```

Note : 1. The source and destination register cannot be a segment register.
2. The subtraction of two memory locations is not allowed .
e.g. SUB [1024] , [2048] is not allowed .

For signed numbers

Example 1 : SUB CH, BL :

Suppose CH → + 40 decimal = 00101000 binary

BL → + 70 decimal = 01000110 binary

Then after the subtraction, Contents of CH → – 30 decimal = 11100010 binary

CF = 1, OF = 0 magnitude fits within 7 bits, SF = 1, negative result

Example 2 : SUB CH, BL :

Suppose CH → 6AH = 0110 1010

BL = 7AH = 0111 1010 binary

Then after subtraction, contents of CH = (1111 000) binary.

Flags → ZF = AF = 0, PF = 1, SF = 1, 0F = 1, CF = 1

The overflow flag set, indicates that answer is not fitting within 7 bits.

2. SUB register, memory

Mnemonic	SUB register, memory	**Flags**	All the flags are affected.
Algorithm	register = register – memory	**Addr. Mode**	Direct addressing mode
Operation	register ← register – contents of memory location	– This instruction subtracts words from memory locations and required register. The result is stored in the register. – This instruction can be 8/16 bit.	
Example	SUB AX , [2048]	– AX ← AX - contents of memory location – This instruction subtracts the data at memory locations whose offset in DS are [2048] and [2049] with data in AX register. – The result is stored in the AX register.	

Note : 1. The register cannot be a segment register.
2. The subtraction of two memory locations is not allowed
e.g. SUB [1024] , [2048] is not allowed .

3. SUB memory, register

Mnemonic	SUB memory, register.	**Flags**	All the flags are affected.
Algorithm	memory = memory – register	**Addr. Mode**	Direct addressing mode
Operation	memory ← memory – register	– This instruction subtracts words from memory locations and required register. The result is stored in the memory locations. – This instruction can be 8/16 bit.	
Example	SUB [2048], AX	– [2048] ← [2048] – AL, [2049] ← [2049] - AH – This instruction subtracts the data at memory locations whose offset in DS are [2048] and [2049] with data in AL register and AH register. – The result is stored at memory locations whose offset in DS are [2048] and [2049] i.e. (25188 H and 25189 H).	

Note : 1. The register cannot be a segment register.
2. The subtraction of two memory locations is not allowed.
e.g. SUB [1024] , [2048] is not allowed.
3. The memory uses DS as a segment register.

4. SUB register, immediate / SUB accumulator, immediate.

Mnemonic	SUB register, immediate data
Algorithm	register = register – immediate data

Flags	All flags are affected.
Addr. Mode	Immediate addressing mode.

Operation	register ← register – immediate data	– This instruction subtracts the immediate number to contents of register. The result of subtraction is stored in the register. – This immediate data can be 8/16 bit.
Example	SUB AL, 24 H AL ← AL – 24	– This instruction subtracts the immediate number 24 H with the contents of AL register. The result is stored in AL.

Note : The segment registers cannot be used as registers.

5) SUB memory, immediate data

Mnemonic	SUB memory, immediate data.
Algorithm	memory = memory – immediate data

Flags	All flags are affected
Addr. Mode	Immediate addressing mode

Operation	memory ← memory – immediate data	– This instruction will subtract immediate Data with contents of memory location whose offset is given in the instruction. The result of addition is stored at the memory location. – The immediate data can be 8/16 bit
Example	SUB COST, 14 H. COST ← COST –14 H [Cost : is a memory location]	– This instruction will subtract the immediate Data 14 H with contents of memory location COST whose offset is given in the instruction [i.e. memory location 28354 H] – Result of subtraction is stored at COST.

Note : The segment register DS is used by the memory.

2. SBB – Subtract byte or word with borrow

Mnemonic	SBB destination, source
Algorithm	Destination = destination – source – CY

Flags	SF flag is not affected.
Addr. Mode	Register addressing mode.

Operation	Destination – source – CY	– This instruction subtracts source and carry flag (i.e. Borrow) from destination. – The source and destination can be 8/16 bit register or memory location. The source and destination can also be a 8/16 bit register or memory location and immediate data .

		– The segment registers cannot be used. The memory uses DS as segment register. – The result is stored in destination. – Both the source and destination may be words or bytes. – They can be signed or unsigned binary numbers. – It can be used to write routines that subtract the numbers longer than 16 bits i.e. double words ,quad words etc .
Example	SBB AL, BL AL ← AL – BL – CY	– This instruction subtracts contents of BL and CY from AL. – The result is stored in AL.

3. DEC – Decrement byte or word by 1

Mnemonic	DEC destination
Algorithm	destination = destination – 1

Flags	CF is not affected
Addr. Mode	Register Addressing mode

Operation	destination ← destination – 1	This instruction subtracts 1 from the destination word or byte. The destination can be a register or a memory location. This instruction cannot be used to decrement the contents of segment registers.
Example	DEC AL AL = AL – 1	This instruction subtracts 1 from the contents of CL and result is stored in CL.

Note: CF is not affected i.e. if an 8 bit destination containing 00H or 16 bit destination containing 0000H is decremented, the result will be FFH or FFFFH with no carry. (i.e. borrow)

4. AAS – ASCII adjust for subtraction

Mnemonic	AAS
Algorithm	If lower nibble of AL > 9 or AF = 1 then : AL = AL – 06 H AH = AH – 1 AF = 1 CF = 1 else AF = 0 CF = 0 In both cases clear the high nibble of AL

Flags	It updates AF, CF but OF, PF, SF and ZF are left undefined.
Addr. Mode	Implied addressing mode

Operation	The number 0 to 9 are represented as 30 – 39 in ASCII code The 8086 allows us to subtract the ASCII codes for two decimal digits without masking "3" in upper nibble of each. The AAS instruction is used to make sure that the result is in unpacked BCD form. It works only on the AL register.

5. DAS – Decimal adjust for subtraction

Mnemonic	DAS
Algorithm	If lower nibble of AL > 9 or AF = 1 then AL = AL – 06 H AF = 1 If AL > 9F H or CF = 1 then AL = AL – 60 H CF = 1

Flags	The OF is undefined.
Addr. Mode	Implied Addressing mode

Operation	AL ← difference in AL adjusted to packed BCD format	– This instruction is used after subtracting two packed BCD numbers to make sure that the result is correct packed BCD. The result of subtraction must be in AL for DAS to work correctly. – This instruction works only on AL register. – If the lower nibble in AL after a subtraction is greater than 9 or if the auxiliary carry flag is set then the DAS instruction will subtract 6 from the lower nibble of AL. – If the result in the upper nibble is greater than 9 or the carry flag is set the DAS instruction will subtract 60 H from the AL register.

6. NEG – Negate byte or word

Mnemonic	NEG destination
Algorithm	Invert all bits of operand and add 1 to inverted operand

Flags	All flags are affected.
Addr. Mode	Register Addressing mode

Operation
- This instruction replaces the number is a destination with 2's complement of that number
- The destination can be a register or memory location.
- This instruction forms the 2's complement by subtracting the original word or byte in the indicated destination from zero.
- It is useful for changing the sign of a signed word or byte.
- Attempt to negate a byte containing – 128 or a word containing – 32, 768 causes no change to the operand and sets Overflow flag.

Example : NEG AX

AX ← 2's complement of number in AX.

Suppose AX =00A3 H = 0000 0000 1010 0011

Then it's 2's complement will be,

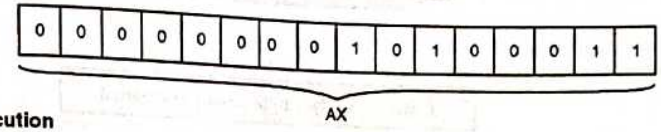
0000 0000 1010 0011 = 1111 1111 0101 1100 ← 1's complement of 00A3 H

\+ 1

1111 1111 0101 1101 ← 2's complement of number

= FF5D H ← contents of AX after execution

Before Execution



After Execution

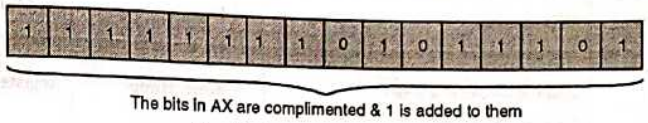


Fig. 4.6.4 : NEG AX

7. 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.

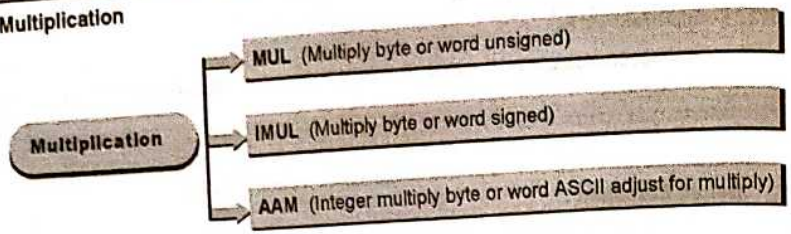
Compare	CF	ZF	SF	
Source > destination	1	0	1	Subtraction required borrow, so CF = 1
Source < destination	0	0	0	No borrow required CF = 0
Source = destination	0	1	0	Result of subtraction is zero

Example : CMP BH, CL

Compares a byte in CL with byte in BH.

Note : (i) The source and destination both cannot be memory locations.
(ii) The compare instructions are often used with conditional Jump instructions.
(iii) One of the operand must be in register.

4.6.3 Multiplication



1. MUL – Multiply byte or word unsigned

Mnemonic	MUL source
Algorithm	When operand is a byte : AX = AL * operand When operand is a word (DX : AX) = AX * operand

Flags	(i) AF, PF, SF, ZF are undefined. (ii) CF and OF will both be 0.
Addr. Mode	Register addressing mode

Operation	– This instruction multiplies an unsigned byte from source with an unsigned byte in the AL register or an unsigned word from source with an unsigned word in AX. – When a byte is multiplied by contents of AL, the result is stored in AX. The MSB of result is stored in AH register and the LSB of result is stored in the AL register. – When a word is multiplied by contents of AX, the product can be double word . The MSB of multiplication is stored in DX and LSB in AX register. – The source can be a register or memory location. – This instruction cannot be used to multiply immediate data. If we want to multiply immediate data. Then that immediate data has to be stored into some valid register and then multiplied with byte or word in AL or AX	
Example	MUL CX	DX:AX =CX * AX : Result of multiplication MSB will be stored in DX register and the LSB will be stored in AX register.

2. IMUL – Multiply byte or word signed

Mnemonic	IMUL source
Algorithm	When operand is a byte → AX = AL * operand When operand is a word → (DS : AX) = AX * operand

Flags	AF, PF, SF and ZF are undefined.
Addr. Mode	Register Addressing mode

Operation	– Byte operands → AX ← AL * source – Word operands → (DS : AX) ← AX * source – This instruction multiplies a signed byte from some source and a signed byte in AL, or a signed word from some source and a signed word in AX. – The source can be register or memory location. – When a signed byte is multiplied by AL a signed result will be put in AX. – When a signed word is multiplied by AX, the MSB 16-bits are put in DX and LSB 16-bits are put in AX. – If the magnitude of product does not require all bits of the destination, the unused bits are filled with copies of the sign bit. – To multiply a signed byte by a signed word it is necessary to move signed byte into lower byte of word and fill the upper byte of word with copies of sign bit. This can be done by CBW instruction. – If the upper byte of 16 bit result or upper word of 32 bit result contains only copies of sign bit (all 0's or all 1's) then the CF and OF will both be zeros. – If the upper byte of 16 bit result or upper word of 32 bit result contains part of the product then the CF and OF will both be zeros.

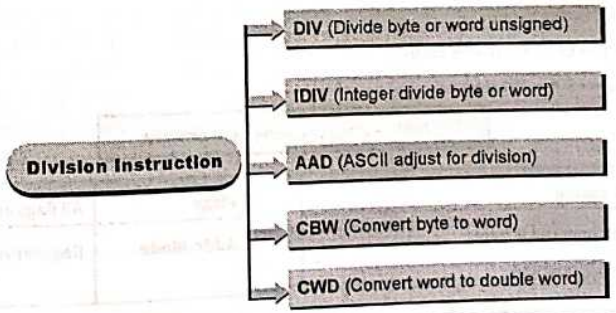
3. AAM – Integer multiply byte or word ASCII adjust for multiply

Mnemonic	AAM
Algorithm	AH = Quotient of AL/10 AL = remainder of AL/10

Flags	It updates PF, SF and ZF. The AF, CF and OF are left undefined.
Addr. Mode	Implied addressing mode

Operation	– Numerical data coming into a computer from a terminal through keyboard is usually in ASCII code. The numbers 0 to 9 are represented by ASCII codes 30 H to 39 H. – Before multiplying two ASCII digits, the upper nibble bits of each need to be masked. This leaves unpacked BCD in each byte . After the two unpacked BCD digits are multiplied, the AAM instruction is used to adjust the product of two unpacked BCD digits in AX. – It works only on register AL. – It is used after multiplying the two unpacked BCD numbers
Example	Let AL → 0000 0101 = unpacked BCD 5 BH → 0000 1001 = unpacked BCD 9 MUL BH : AL * BH → Result in AX register AX = 0000 0000 00101101 = 002D H AAM AX : 0000 01000 0000 0101 = 0405 H Which is unpacked BCD for 45.

4.6.4 Division Instruction



1. DIV – Divide byte or word unsigned

Mnemonic	DIV source
Algorithm	When operand is a byte : AL = AX/operand (Quotient) AH = remainder (Modulus) When operand is a word : AX = (DS : AX) / operand (Quotient) DX = remainder (modulus)

Flags	All flags are undefined . Let AX = 37D7 H = 14,295 decimal, BH = 97 H = 151 decimal
Addr. Mode	Register Addressing mode

Operation

This instruction basically performs two operations :

1. Divide an unsigned word by a byte.
2. Divide unsigned double word by a word.

For 1st operation, the word must be in AX register. After division,

- AL → 8 bit quotient, AH → 8 bit remainder
- If an attempt is made to divide by a or quotient is too large to fit in AL. 8086 will execute a type 0 interrupt.
- For 2nd operation, MSB of word of the double word must be in DX and LSB must be in AX.

  AX → 16 bit quotient, DX → 16 bit remainder

BYTE divisor

(AL) ← Quotient of (AX) / (source)

(AH) ← Remainder of (AX)/(source)

Word divisor

(AX) ← Quotient of (DX : AX)/ (source)

(DX) ← Remainder of (DX : AX)/(source)

Example : DIV BH

Divide word in AX by the byte in BH

AX / BH, AL → Quotient = 5 E H = 94 decimal.

AH → remainder = 65 H = 101 decimal.

2. IDIV – Integer divide byte or word

Mnemonic	IDIV source
Algorithm	When operand is byte : AL = AX/operand AH = remainder (modulus) When operand is a word : AX = (DX : AX) / operand DX = remainder (modulus)

Flags	All flags are undefined
Addr. Mode	Register Addressing mode

Operation

Byte division :

AL ← quotient of (DS : AX) / source

AH ← remainder of (DS : AX) / source

Word division :

AX ← quotient of (DS : AX) / source

DX ← remainder of (DS : AX) / source

This instruction performs two operation :

1. Divide a signed word by a signed byte
2. Divide a signed double word by a signed word.

Example

A signed word divided by a single byte

Let AX = 03 AB H BL = 00 D3 H

IDIV BL

Quotient in AL = EC H Remainder in AH = 27 H

3. AAD – ASCII adjust for division

Mnemonic	AAD
Algorithm	AL = (AH * 10) + AL AH = 0 AL ← 10* (AH) + AL AH ← 0

Flags	(i) The PF, SF and ZF are affected. (ii) AF, CF and OF are undefined after AAD.
Addr. Mode	Implied Addressing mode

Operation

– It converts two unpacked BCD digits in AH and AL to the equivalent binary number in AL. This adjustment must be must be made before dividing the two unpacked BCD digits in AX, by an unpacked BCD byte. After the division,

AL → unpacked BCD quotient

AH → unpacked BCD remainder

Example

AX = 0607H unpacked BCD for 67 decimal CL = 09 H, now adjust to binary using AAD. **AAD**	This instruction will perform following operation. $(AH)^* 10 = 06 * 10 = (60)_{decimal} = 3CH$ $(AH) * 10 + (AL) = (60)_{10} + (7)_{10} = 3\,C\,H + 7\,H = 43\,H$ AL = 43 H and AH = 00 H ∴ AX (New) = 0043 H
DIV CL	Divide AX by unpacked BCD in CL Quotient AL = 07 H unpacked BCD Remainder AH = 04 H unpacked BCD

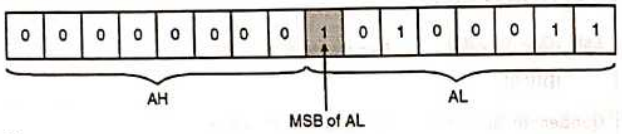
4. CBW – Convert byte to word

Mnemonic	CBW	Flags	No flags are affected.

Algorithm	If MSB bit of AL = 1 then
	AH = 255 (FF H)
	else
	AH = 0
		Addr. Mode	Implied Addressing mode

Operation	– This instruction copies the sign bit in AL to all the bits in AH. AH is then said to be a sign extension of AL. – This operation must be done before a signed byte in AL can be divided by another signed byte with IDIV instruction.		
Example	AX = 00AC H = – 163 decimal	CBW Convert signed byte in AL to signed word in AX. Result : 1111 1111 1010 0011 = – 163 decimal	

Before Execution



After Execution

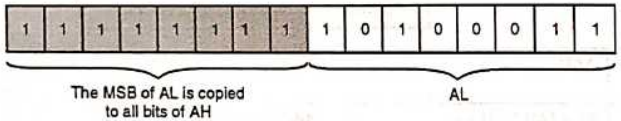


Fig. 4.6.5: CBW

5. CWD – Convert word to double word

Mnemonic	CWD	Flags	No flags are affected.
Algorithm	If MSB bit of AX = 1 then, DX = 655 35 (FFFF H) else DX = 0	Addr. Mode	Implied addressing mode
Operation	– This instruction copies the sign bit of word in AX to all the bits of DX register. DX is sign extension of AX then. – It must be done before signed word in AX can be divided by another signed word with IDIV instruction.		
Example	– If DX = 4AA3 H and AX = 00AC H		

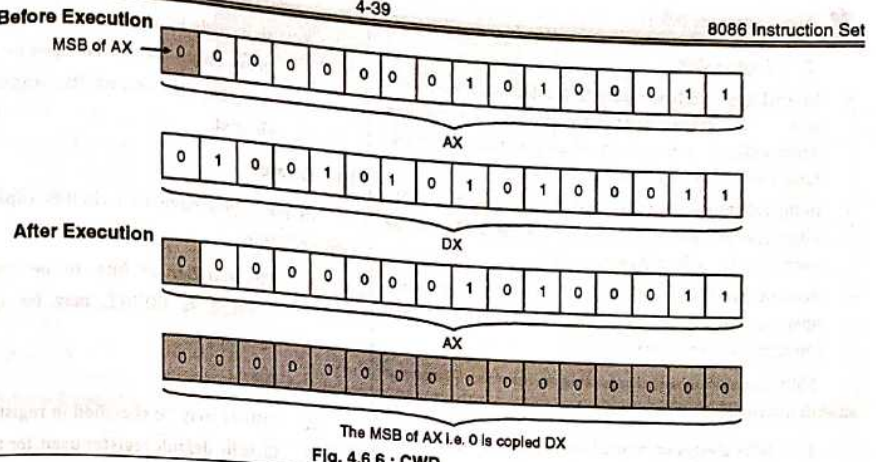


Fig. 4.6.6 : CWD

4.7 Bit Manipulation Instructions

The 8086/8088 provide three groups of instructions, for manipulating bits within both bytes and words. Three groups are listed in Table 4.7.1.

Table 4.7.1 : Bit manipulation instructions

LOGICALS	
NOT	Not byte or word
AND	And byte or word
OR	Inclusive or byte or word
XOR	Exclusive or byte or word
TEST	Test byte or word
SHIFTS	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
ROTATES	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word

I. Logical Instructions

- AND, OR, XOR and TEST affect the flags as follows :
- The overflow (OF) and carry (CF) flags are always cleared by logical instructions, and the content of the auxiliary carry (AF) flag is always undefined following execution of a logical instruction.
- The sign (SF), zero (ZF) and parity (PF) flags are always posted to reflect the result of the operation and can be tested by conditional jump instructions.
- The interpretation of these flags is the same as for arithmetic instructions.
- SF is set if the result is negative (MSB bit is 1), and is cleared if the result is positive (MSB bit is 0).
- ZF is set if the result is zero, cleared otherwise. PF is set if the result contains an even number of 1-bits (has even parity) and is cleared if the number of 1-bits is odd (the result has odd parity).

Note that NOT has no effect on the flags.

II. Shifts

In shift, we will shift the bits only either by a constant 1 or depending upon CL register (this register is also by default register for SHIFT operation). Upto 255 shift may be performed, according to the value of the count operand coded in an instruction. Shift operation can be subgrouped, as follows :

1. Arithmetic shift

2. Logical shift

- In arithmetic shift we have SAL (shift arithmetic left) and SAR (shift arithmetic right) instructions. Arithmetic shifts may be used to multiply and divide binary numbers by powers of two.
- In logical shift we have SHL (shift logical left) and SHR (shift logical right) instructions. Logical shifts can be used to isolate bits in bytes or words.
- Actually speaking SAL and SHL, both perform the same operation and one physically the same instruction. Therefore we will learn both at a same time.

Shift instruction affects the flag as follows : (Valid for all shift instructions) :

1. AF is always undefined following shift operation.
2. PF, SF and ZF are updated normally, as in the logical instructions. PF is set/cleared as per lower 8 bits of operand.
3. CF always contains, last bit shifted out of the destination operand.
4. The content of OF is always undefined following a multi-bit shift.
5. In a single bit shift, OF = 1, if the value of the high order (sign) bit was changed by the operation, if the sign bit retains its original value, OF is cleared.

III. Rotates

In 8086 you have two choices capable to perform single rotation :

1. The number of bits, to be rotated, normally referred as COUNT, may be specified as the constant 1.

   OR

2. COUNT may be specified in register CL. (Register CL is by default register used, for this operation).

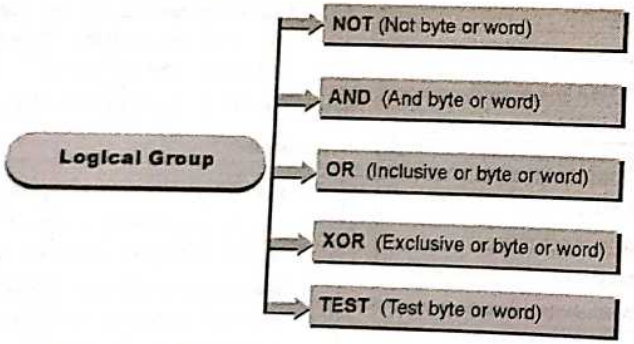
Common feature for all instructions are :

- Destination may be a word or byte. It may be a register or a memory location specified by any one of the 24 addressing modes.
- Number of bits to be shifted depends upon count in the instruction.
- Count may be a constant value 1 or CL register.
- Now Let us see one by one

4.7.1 Logical Group

Q. Explain the following Intel 8086 assembly language instruction giving example : TEST

MU - Dec. 11 , May 12 , Dec. 12 , May 13 , 2 Marks



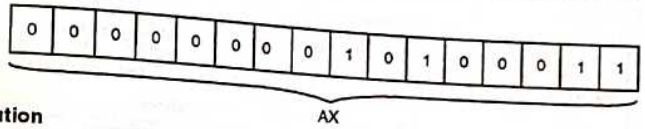
1. NOT – Not byte or word

Mnemonic	NOT destination.
Algorithm	If bit is 1 turn is to 0. If bit is 0 turn it to 1.

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

Operation	Destination	– This instruction inverts each bit of byte or word at the specified destination i.e. it finds 1's complement of the number. – The destination can be a register or a memory location. – The destination cannot be immediate data. – The destination cannot be a segment register.
Example	NOT AX $AX \leftarrow \overline{AX}$	– This instruction complements the contents of AX register.

Before Execution



After Execution

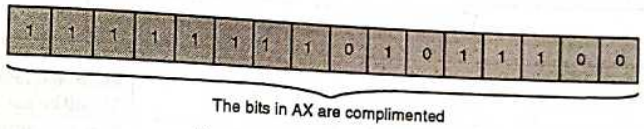


Fig. 4.7.1 : NOT AX

2. AND – And byte or word

Mnemonic	AND destination, source
Algorithm	Destination = destination ∧ source

Flags	CF and OF are both 0 after the execution of AND instruction. PF, SF and ZF are updated by AND instruction. AF is undefined.
Addr. Mode	Register addressing mode

Operation	1 AND 1 = 1 1 AND 0 = 0 0 AND 1 = 0 0 AND 0 = 0	– This instruction ANDs each bit in a source byte or word with the same number bit in a destination byte or word. The result is stored at specified location. – The contents of specified source do not change. – The source can be a register, memory location or an immediate number. – The destination can be register, or memory location. – The source and destination both cannot be memory locations . – The segment registers cannot be used as source or destination .
Example	AND AL, BL	– This instruction ANDs each bit in a AL register with the bits in BL register. – The result is stored in the AL register.

3. OR – Inclusive or byte or word

Mnemonic	OR destination, source.	**Flags**	CF = 0, OF = 0, after OR instruction PF, SF and ZF are affected. AF is undefined.
Algorithm	destination ← destination ∨ source.	**Addr. Mode**	Register addressing mode
Operation	1 OR 1 = 1 1 OR 0 = 1 0 OR 1 = 1 0 OR 0 = 0	– This instruction ORs each bit in a source with the corresponding bits in a destination. – The result is stored in the specified destination. – Contents of source will not change. – Source can be register, memory location or immediate number. – The source and destination both cannot be memory locations. – The segment registers cannot be used as source or destination . – Destination can be a register or Memory location.	
Example	OR AL, BL	– This instruction ORs each bit in a AL register with the bits in BL register. – The result is stored in the AL register.	

4. XOR – Exclusive or byte or word

Mnemonic	XOR destination, source.	**Flags**	CF = 0, OF = 0 after XOR instruction. PF, SF and ZF are affected AF will be undefined.
Algorithm	1 XOR 1 = 0 1 XOR 0 = 1 0 XOR 1 = 1 0 XOR 0 = 0	**Addr. Mode**	Register addressing mode
Operation	– This instruction logically XORs each bit of the source byte or word the corresponding bit in the destination and stores the result in the destination. – The source may be register, memory location or immediate number. – The destination may be register or memory location. – The source and destination both cannot be memory locations. – The segment registers cannot be used as source or destination.		
Example	XOR AL , BL	– This instruction XORs each bit in a AL register with the bits in BL register. – The result is stored in the AL register .	

5. TEST – Test byte or word

Mnemonic	TEST destination, source.	**Flags**	CF = 0, OF = 0 PF, SF, ZF will be affected to show result of ANDing.
Algorithm	destination ← destination ∧ source.	**Addr. Mode**	Immediate addressing mode
Operation	1 AND 1 = 1 1 AND 0 = 0 0 AND 1 = 0	– This instruction ANDs contents of a source byte or word with contents of specified destination word. – The source can be register, memory location, immediate data.	

	0 AND 0 = 0	– The source and destination both cannot be memory locations. – Segment registers are not allowed to be used as source or destination. – The destination can be a register or memory location. – Flags are affected, but neither operand is changed. – It is used to set flags before conditional JUMP.
Example	TEST AL, 75 H	Let AL = 0 1 1 1 0 1 0 1 AND Immediate number 75H with AL. PF = 0, SF = 0, ZF = 0, CF = 0, OF = 0

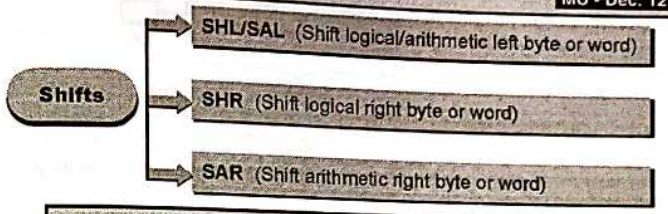
4.7.2 Shifts

University Questions

Q. Explain the following Intel 8086 assembly language instructions giving example :
(I) SAL

Q. Explain the following Intel 8086 assembly language instructions giving example.SAR. MU - Dec. 12, May 13, 2 Marks

MU - Dec. 11, May 12, 2 Marks



1. SHL/SAL – Shift logical/arithmetic left byte or word

Mnemonic	SAL/SHL destination, count.	**Flags**	All flags are affected.
Algorithm	Shift all bits left, the bit that goes off is set to CF. Zero bit is inserted in the right most position.	**Addr. Mode**	Immediate addressing mode
Operation	CF ← MSB ← LSB ← 0		

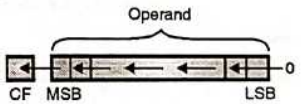


Fig. 4.7.2

– SAL/SHL are two mnemonics for the same instruction. This instruction shifts each bit of the specifies destination, some number of bit positions to the left. As left bit is shifted out of the LSB position, 0 is put in the LSB position. The MSB will be shifted into CF.
– The count can be any immediate number.
– In case of multiple bit shifts, CF will contain the bit most recently shifted in from the MSB.
– Bits shifted into CF previously will be lost.
– This instruction can be used to multiply an unsigned binary number by a power of 2.
– The destination can be a memory location or register.
– The count is specified in the CX register.
– Negative shifts are illegal.

Example : SHL AX , 01 ; This instruction shift AX by 1 bit to the left.

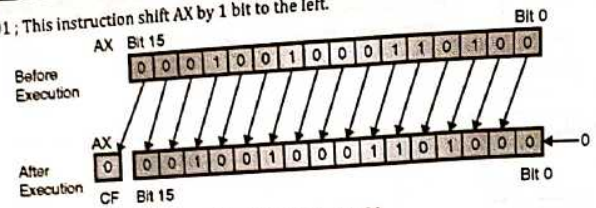


Fig. 4.7.3 : SHL AX, 01

2. SHR – Shift logical right byte or word

Mnemonic	SHR destination, count.
Algorithm	Shift all bits right, the bit that goes off is set to CF. Zero bit is increased to the right most position.

Flags	All the flags are affected.
Addr. Mode	Register addressing mode

Operation 0 → MSB → LSB → CF

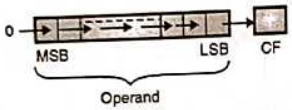


Fig. 4.7.4

- This instruction shifts each bit in specified destination some number of bit position to the right Bit shifted from LSB, goes to CF.
- For Multi bit shift CF will contain bit most recently shifted out from LSB position.
- Bits shifted into CF previously will be lost.
- This instruction is used to divide unsigned binary number by power of 2.
- Count is any immediate number.
- Negative shifts are illegal
- The count is specified in the CX register.

Example : SHR AX , CL ; Let AX = 1234 H and CL = 02 H

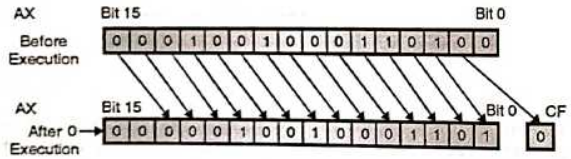


Fig. 4.7.5 : SHR AX, CL

3. 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

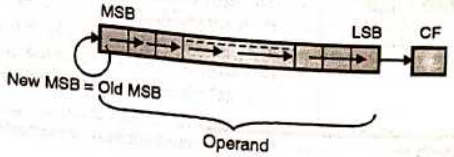


Fig. 4.7.6

- 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.

Example : SAR AX, 1

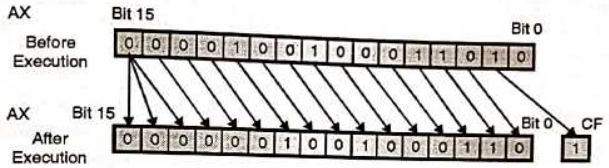
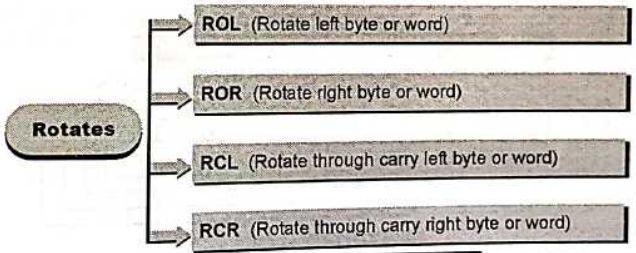


Fig. 4.7.7

4.7.3 Rotates



1. ROL – Rotate left byte or word

Mnemonic	ROL destination, source
Algorithm	Shift all bits left, the bit goes off is set to CF and the same bit is inserted to the right most position.

Flags	Only CF is affected.
Addr. Mode	Immediate addressing mode.

Operation	CF ← MSB ← LSB	– This instruction rotates all the bits in a specified word or byte to the left, by some bit positions. – The data bit rotated out of MSB is circled back into the LSB. The data bit rotated out of MSB is also copied to CF. – CL is default register used for rotate instruction when count is greater than 1.

Example : ROL AX, 01 H ; – This instruction rotates the contents of AX register by 1 bit to left.

Let AX = 0001 0010 0110 0100, CF = 0

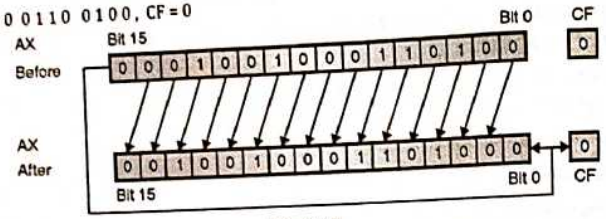


Fig. 4.7.8

2. ROR – Rotate right byte or word

Mnemonic	ROR destination, count.	Flags	Only CF
Algorithm	Shift all bits right, the bit that goes off is set to CF and the same bit is inserted into the left most position	Addr. Mode	Register addressing mode.

Operation	CF MSB → LSB	– This instruction rotates all the bits of specified destination operand to the right. The bit moved out of the LSB is rotated around into the MSB. – The data bit moved out of LSB is also copied into CF. – The destination may be memory location or register. – The count if greater than 1 should be specified in the CX register. – Negative shifts are illegal.

Example ROR AX, CL

Let AX = 0001 0010 0011 0100, CL = 04 H

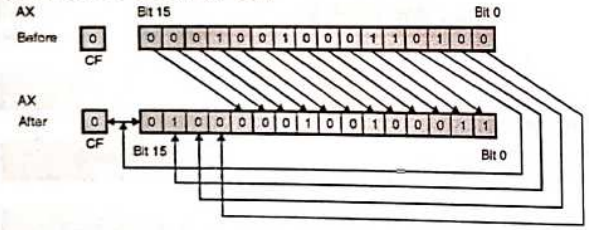


Fig. 4.7.9 : ROR AX, CL

3. RCL – Rotate through carry left byte or word

Mnemonic	RCL destination, count.	Flags	Only CF is affected.
Algorithm	Shift all bits left, the bit that goes is set to CF and previous value of CF is inserted to the rightmost position.	Addr. Mode	Immediate addressing mode

Operation	CF ← MSB ← LSB	This instruction rotates all the bits in specified destination by some number of bit positions to the left. The destination can be register or memory location. Negative shifts are illegal. CL is default register used for rotate instruction when count is greater than 1.

Example : RCL AX, 01 H

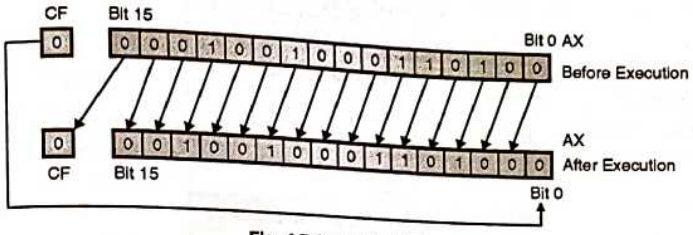


Fig. 4.7.10 : RCL AX, 01 H

4. RCR – Rotate through carry right byte or word

Mnemonic	RCR destination, count.	Flags	It affects only CF
Algorithm	Shift all bits right, the bit that goes off is set to CF and previous value of CF is inserted to the left most position	Addr. Mode	Immediate addressing mode

Operation	CF → MSB → LSB	– This instruction rotates all the bits is specified word or byte same number of bit positions to right. The destination can be register or memory location. – Negative shifts are illegal. – CL is default register used for rotate instruction when count is greater than 1.

Example : RCR AX, CL

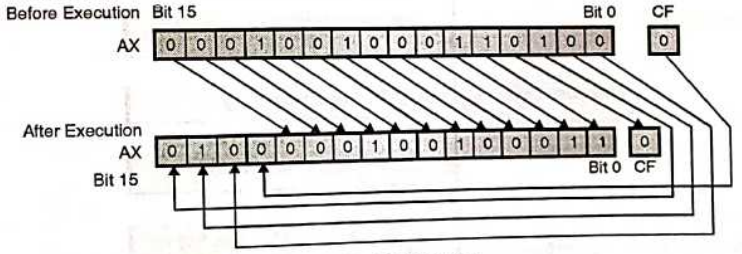


Fig. 4.7.11 : RCR AX, 04 H

4.8 Processor Control Instruction

Q. Explain processor control instructions. (5 Marks)

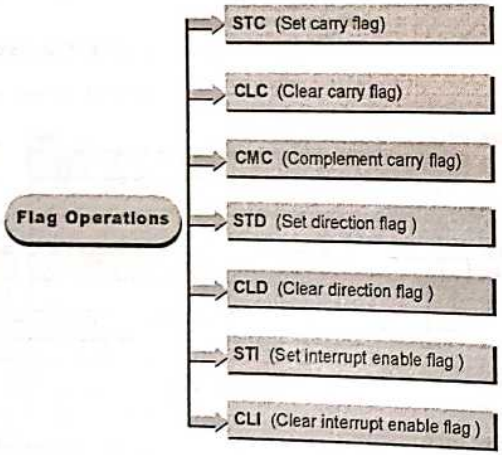
The instructions under this group are listed as follows :

Table 4.8.1 : Processor control instructions

FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
NO OPERATION	
NOP	No operation
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{TEST}$ pin active
ESC	Escape to external co-processor
LOCK	Lock bus during next instruction

These instructions allow programs to control various CPU functions. One group of instructions updates flags and another group is used primarily for synchronizing the 8086 or 8088 with external events. A final instruction causes the CPU to do nothing. Except for the flag operations, none of the processor control instructions affect the flags.

4.8.1 Flag Operations



1. CLC – Clear carry flag

Mnemonic	CLC	Flags	Except carry, no other flags are affected.
Algorithm	CF = 0	Addr. Mode	Implied addressing mode
Operation	CF ← 0	This instructions clears (resets) the carry flag to zero.	

2. STC – Set carry flag

Mnemonic	STC	Flags	Except carry no other flags are affected.
Algorithm	CF = 1	Addr. Mode	Implied addressing mode
Operation	This instruction sets the carry flag.		

3. CMC – Complement carry flag

Mnemonic	CMC	Flags	Except carry no other flags are affected.
Algorithm	if CF = 0 then CF = 1 if CF = 1 then CF = 0	Addr. Mode	Implied addressing mode
Operation	CF = $\overline{CF}$	This instruction inverts the value of carry flag	

4. CLD – Clear direction flag

Mnemonic	CLD	Flags	Except direction flag, no other flags are affected.
Algorithm	DF = 0	Addr. Mode	Implied addressing mode
Operation	– It resets (DF) direction flag to zero. If the direction flag is reset, SI and / or DI will be automatically incremented during string operations. – When one of the string instructions like MOVS, CMPS or SCAS executes it automatically increments the SI and / or DI registers.		

5. STD – Set direction flag

Mnemonic	STD	Flags	Except Direction no other flags are affected.
Algorithm	DF = 1	Addr. Mode	Implied addressing mode
Operation	It is used to set the direction flag, so that SI and / or DI will be decremented by string instructions. i.e. CMPSB, MOVSB, STOSB, STOSW etc.		

6. CLI – Clear interrupt enable flag

Mnemonic	CLI	Flags	Except interrupt flag, it does not affect any other flag.
Algorithm	IF = 0	Addr. Mode	Implied addressing mode

Operation	– This instruction resets the interrupt flag to zero. No other flags are affected. If the interrupt flag is reset, the 8086 will not respond to an interrupt signal on its INTR input. – This instruction has no effect on the NMI.

7. STI – Set interrupt enable flag

Mnemonic	STI	Flags	Except interrupt flag, it does not affect any other flag.
Algorithm	IF = 1	Addr. Mode	Implied addressing mode

Operation	– This interrupt sets the interrupt flag to 1. This enables INTR interrupt of the 8086.

4.8.2 NOP

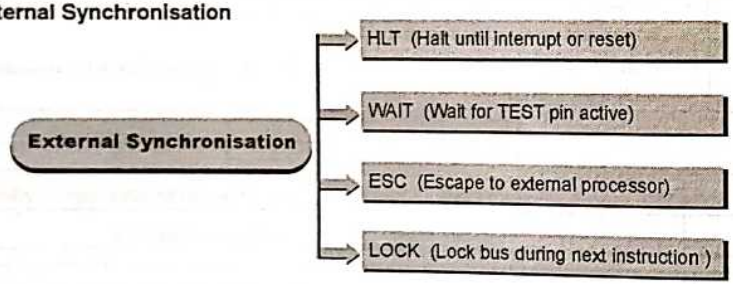
University Question

Q. Explain the following Intel 8086 assembly language instructions giving example :
NOP **MU - Dec. 11, May 13, 2 Marks**

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.

4.8.3 External Synchronisation



1. HLT – Halt until interrupt or reset

Mnemonic	Halt processing	Flags	No flags are affected.

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

2. WAIT – Wait for TEST pin active

Mnemonic	WAIT	Flags	No flags are affected.

Operation	– When this instruction executes, the 8086 enters on idle condition in which it is doing no processing. – The 8086 will stay in this idle state until 8086 $\overline{TEST}$ input pin is made low or on interrupt signal is received on the INTR or NMI interrupt pins. – If a valid interrupt occurs while the 8086 is in the idle state, the 8086 will return to idle state after the interrupt service procedure executes. – It is used to synchronize the 8086 with external hardware. Such as 8087 math processor.

3. ESC – Escape to external processor

Mnemonic	ESC external – opcode, source.

Operation	– This instruction is used to pass instruction to a coprocessor, such as 8087 math co-processor which shares the address and data bus with on 8086. – The instruction for the Co processor are represented by a 6 bit code embedded in the escape instruction. – When the 8086 fetches on ESC instruction, the coprocessor decodes the instruction and carries out the action specified by the 6 bit code specified in the instruction. – In most cases 8086 treats the ESC instruction as a NOP in some cases 8086 will access a data item in memory for co-processor.

4. LOCK – Lock bus during next instruction

Mnemonic	LOCK

Operation	– Many multiprocessor systems contain several microprocessors. Each microprocessor has its own local buses and memory. The individual microprocessors are connected together by a shared system bus so that each can access system resources such as disk drives or memory. – Each microprocessor takes control of the system bus. Only when it needs to access some resource.	– Lock prefix allows a microprocessor to make sure that another processor does not take control of the system bus. – While it is in the middle of a critical instruction which uses the system bus when an instruction with lock prefix executes the 8086 will assert its bus lock signal output. This signal is connected to an external bus controller, which then prevents any other processor from taking over the system bus.
		Example LOCK XCHG SEMAPHORE, AL : The XCHG instruction requires two bus accesses. The lock prefix prevents another processor from taking control of system bus between two accesses.

4.9 Program Transfer Group

University Questions

Q. Write a detailed note on : Branch instructions of the 8086. **MU - May 11, 10 Marks**

8086/8088 provides you :

1. Unconditional CALL
2. JMP — Conditional / Unconditional
3. INT (Software interrupt) and many more.

- These instructions are also referred to as Branch instructions.
- In 8086/8088, unconditional CALL and jump provides you, program transfer within the segment or outside the segment, anywhere in full memory space. The same is true for INT instruction. But additional jump instruction has very limited range in 8086/8088 i.e. + 127 bytes to – 128 bytes i.e. relative jump. Therefore care should be taken, so that you don't cross the boundary specified by microprocessor.
- Second important point is, these instructions ***operate*** on CS (Code segment) and IP (Instruction pointer), because THE SEQUENCE OF EXECUTION OF INSTRUCTIONS IN AN 8086/8088 PROGRAM IS DETERMINED BY THE CONTENT OF CS AND IP.
- The CS register contains the base address of the current code segment, the 64k portion of memory from which instructions are presently being fetched.
- The IP is used as an offset from the beginning of the code segment. The combination of CS and IP points to the memory location from which the next instructions is to be fetched.
- Here you can recall one point that, under most operating conditions, the next instruction to be ***executed*** has already been fetched from memory and is waiting in the CPU instruction queue.
- The program transfer instructions operate on the instruction pointer and on CS register.
- Changing contents of CS and IP, causes NORMAL SEQUENTIAL EXECUTION TO BE ALTERED.
- When a program transfer occurs, the queue no longer contains the correct instruction, and the BIU obtains the next instruction from memory using the new IP and CS values. It passes the instruction directly to EU, and then begins refilling the queue from the new location.
- Third point is, in 8086/8088 program transfer or branching can be within the segment or outside the segment. This is very important because, if you branch within the segment no need to change CS, change only IP, but if it is outside the segment then change both, CS as well as IP. This point in conjunction with, ***way of calculating branch address***, provides us four addressing modes for indicating branch addresses.

Those are listed as follows :

1)	Intrasegment Direct	Branching within the segment
2)	Intrasegment Indirect	
3)	Intersegment Direct	Branching outside the segment limit
4)	Intersegment Indirect	

1. Intrasegment Direct

- In this case the effective branch address is the sum of an 8 or 16 bit displacement and the current contents of IP. It means that branching is within the segment i.e. within 64k bytes, therefore no need to change CS. Fig. 4.9.1 shows graphical representation of the operation.

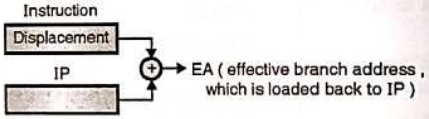


Fig. 4.9.1 : Intrasegment direct

- As seen, the displacement is compute ***relative*** to the IP, therefore this addressing mode is also sometime referred as ***Relative addressing***. When the displacement is 8 bits long, it is referred to as a ***short jump***. When displacement is 16 bit long, it is referred as ***NEAR Jump***.
- This type of addressing may be used by conditional or unconditional branching instructions, but for conditional branch instruction ONLY 8 bit DISPLACEMENT IS ALLOWED.

2. Intrasegment Indirect

- In this mode, the effective branch address is the contents of a register or memory location specified by one of the 24 addressing modes (EXCEPT Immediate mode).

- The contents of IP are replaced by effective branch address. Refer Fig. 4.9.2 for graphical representation of the mode.

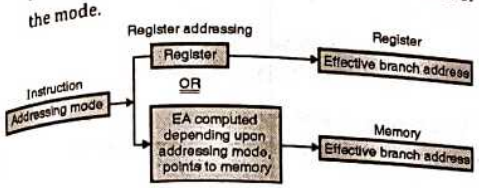


Fig. 4.9.2 : Intrasegment Indirect

Note : This addressing mode may be used only in unconditional branch instruction.

3. Intersegment Direct

- This mode deals with branching out of the present code segment. Therefore microprocessor requires information of new segment and new offset (i.e. new CS and new IP).
- In this mode in instruction itself new CS and new IP are mentioned. Therefore new CS and IP values ***replaces*** old CS and old IP. Fig. 4.9.3 shows graphical presentation.
- This addressing mode can also be used for unconditional branch instruction only.

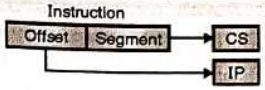


Fig. 4.9.3 : Intersegment direct

Note : As we change segment, this type of jump/CALL are referred as FAR jump/FAR CALL.

4. Intersegment Indirect

- This mode also deals with branching from one code segment to another. Therefore we require new CS and new IP. We get those from two consecutive words in memory that are referenced using any of the 24 addressing modes except immediate and register mode.
- Refer Fig. 4.9.4 for graphical presentation.
- This addressing mode can also be used for unconditional branch instruction only.

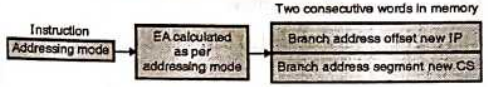


Fig. 4.9.4 : Intersegment Indirect

Now let's concentrate on program transfer group. We have four subgroups under this :

1. Unconditional transfers.	2. Conditional transfers.
3. Iteration control.	4. Interrupt relate instructions.

Instructions under these subgroups are listed as follows in Table 4.9.1.

Table 4.9.1 : Program transfer instructions

UNCONDITIONAL TRANSFERS		ITERATION CONTROLS	
CALL	Call procedure	LOOP	Loop
RET	Return from procedure	LOOPE / LOOPZ	Loop if equal/zero
JMP	Jump	LOOPNE/LOOPNZ	Loop if not equal/not zero
CONDITIONAL TRANSFERS		**INTERRUPTS**	
JA/JNBE	Jump if above / not below or equal	INT	Interrupt
JAE/JNB	Jump if above or equal / not below	INTO	Interrupt if overflow
JB/JNAE	Jump if below / not above nor equal	IRET	Interrupt return

JBE/JNA	Jump if below or equal / not above		
JC	Jump if carry		
JE/JZ	Jump if equal / zero		
JG/JNLE	Jump if greater / not less nor equal		
JGE/JNL	Jump if greater or equal / not less		
JL/JNGE	Jump if less / not greater nor equal		
JLE/JNG	Jump if less or equal / not greater		
JNC	Jump if not carry		
JNE/JNZ	Jump if not equal / not zero		
JNO	Jump if not overflow		
JNP/JPO	Jump if not parity / parity odd		
JNS	Jump if not sign		
JO	Jump if overflow		
JP/JPE	Jump if parity / parity even		
JS	Jump if sign		

4.9.1 Unconditional Transfers

The unconditional transfer instructions may transfer control to a target instruction within the current code segment (intrasegment transfer) or to a different code segment (intersegment transfer). The transfer is made unconditionally any time the instruction is executed.

1. CALL – Call a procedure

Mnemonic	CALL procedure
Operation	This instruction is used to transfer program control to a subroutine or a procedure. There are two basic types of CALLS : NEAR Call and FAR Call.

- **Near Call :** A near call is a call to a procedure which is in the same segment, which has the CALL instruction. It is also called as Intra segment call.

(i) If the call to the subroutine or procedure with a 16-bit signed displacement the 8086 will decrement the SP by 2 and push the IP contents onto the stack. Then adds the signed 16 bit value of DISP of IP. The contents of CS are unchanged. Such a call is an intra segment direct call.

e.g. DISP PROC NEAR :It indicates that DISP is the name of procedure which is in the same code segment. DISP is 16 bit signed displacement.

Addressing mode : Relative addressing mode

(ii) If the CALL is to a subroutine is in the same segment and is addressed by the contents of a 16-bit general register/memory. Then the 8086 decrements the SP by 2 and pushes the contents of IP onto the stack and then the contents of specified 16 bit register/memory location.

The registers can be BX, SI or DI to provide the new value of IP.

The memory location is addressed by contents of 16 bit register such as BX, SI, DI into IP.

The contents of CS remain unchanged. Such a call is called is an intra segment indirect call.

e.g. : CALL Reg 16.

- **Far Call :** A far is a call to a procedure which is in a different segment from that which contains the CALL instruction. Far calls are also called as intersegment calls.

(i) If the far call to a subroutine or procedure is with a signed displacement, the 8086 decrements SP by 2 and pushes the contents of CS onto the stack and moves the lower 16 bit value of the number like DISP in CALL DISP into CS.

The SP is again decremented by 2. The contents of IP are pushed onto the stack. The IP is then loaded with the higher 16 bits i.e. MSB value of DISP.

Thus, as this instruction CALLS a subroutine in another code segment it is an intersegment direct call.

(ii) If the CALL to a subroutine in another segment is addressed by the contents of a 16 bit register then the 8086 decrements the SP by 2 and pushes CS contents onto the stack.

The CS is then loaded with the contents of memory locations addressed by [reg 16 + 2] and [reg 16 + 3] in DS.

The SP is then again decremented by 2, IP is pushed onto the stack. The IP is then loaded with contents of memory locations addressed by [reg 16] and [reg 16 + 1] in DS.

The registers used as reg 16 are BX, SI, DI.

Such a CALL is called as intersegment indirect call.

e.g. : CALL DWORD, PTR [reg16]

2. RET – Return from Procedure

Mnemonic	RET optional – pop – value
Algorithm	– Return from near procedure POP from stack : IP If immediate operand is present : SP = SP + operand – Return from far procedure POP from stack IP CS If immediate operand is present SP = SP + operand
Operation	– The RET instruction will return execution from a procedure to the next instruction after CALL instruction. – If the procedure is a near procedure (i.e. in same code segment as CALL instruction), then the return will be done by replacing the IP with a word from the top of stack. This word from the top of stack is the offset of the next instruction after CALL. The SP will be incremented by 2. After return address is popped off the stack. – If the procedure is a far procedure (in a different code segment), the instruction pointer will be replaced by the word at the top of stack. The SP will be incremented by 2. The CS is then replaced with a word from new top of stack. After CS word is popped the SP will again incremented by two. – A RET instruction can be followed by a number.
Example	**RET 2.** – In this case the SP will be incremented by additional 2 addresses after the IP or IP and CS are popped off the stack. This form is used to increment the stack pointer UP over parameters passed to the procedure on the stack.

3. JMP - Jump

(Unconditional jump to specified destination)

Mnemonic	JMP	Flags	No flags are affected.
Algorithm	Always jump.		

Operation	– This instruction will cause the 8086 to fetch its next instruction from the location specified in the instruction rather than from next location after JMP instruction. – There are two basic types of JMPs, near and far. – Near JMP is a jump where destination location is in the same code segment only IP is changed to get destination location. It is known as intrasegment JMP. – If the destination is in a segment with a different name from the segment containing the JMP instruction, then both the IP and CS contents will be changed to get the destination location. Such a JMP is far JMP. A far JMP is an Inter segment JMP. – The near and far JMPs are further described as either direct or indirect. If the destination address is specified within instruction. It is a direct JMP, if the destination address is contained in register or memory location, the JMP is indirect, because 8086 has to access the specified register or memory to get the destination address.	
Example	JMP WORD PTR [BX]	– This instruction will replace IP with a word from memory location pointed by BX in DS. – This is an indirect near JMP.

4.9.2 Conditional Transfers

University Question

Q. Explain the conditional transfer instructions. **MU - Dec. 11, May 12 , Dec. 12, 5 Marks**

– Conditional transfer instructions are also referred as conditional jump instructions.

– There are total 18 instructions. Refer Table, each test a different combinations of flags, for a condition. If condition is true, then control is transferred to the target specified in the instruction.

– If the condition is ***false***, then control passes to the instruction that follows the conditional jump.

– Very important point regarding these instructions is, ALL CONDITIONAL JUMPS ARE ***SHORT***, that is, the target must be within the current code segment and within – 128 to + 127 bytes of the first byte of the next instruction.

– Since the jump is made by adding the relative displacement of the target to the instruction pointer, all conditional jumps are self relative and are appropriate for position independent routines. These instructions does not affect any flag.

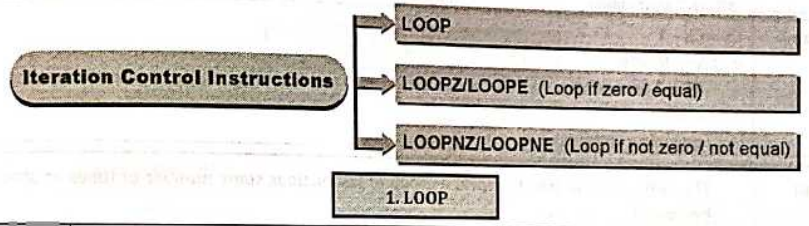
Common Operations		
JC	JUMP if carry	CF = 1
JNC	JUMP if not carry	CF = 0
JZ/JE	JUMP if zero (or equal)	ZF = 1
JNZ/JNE	JUMP if not zero (or not equal)	ZF = 1
JP/JPE	JUMP if parity (or even parity)	PF = 1
JNP/JPO	JUMP if not parity (or odd parity)	PF = 0
JCXZ	JUMP if CX is zero	CX = 0000H
Signed Operations		
JO	JUMP if overflow	0F = 1
JNO	JUMP if not overflow	0F = 0

Common Operations		
JS	JUMP if sign (– ve)	S = 1
JNS	JUMP if not sign (+ ve)	S = 0
JL/JNGE	JUMP if less (i.e. neither greater nor equal)	SF ⊕ 0F = 1
JNL/JGE	JUMP if not less (i.e. either greater or equal)	SF ⊕ 0F = 0
JLE/JNG	JUMP if less or equal (i.e. not greater)	(SF ⊕ 0F) + ZF = 1
JNLE/JG	JUMP if neither less nor equal (i.e. greater)	(SF ⊕ 0F) + ZF = 0

Common Operations		
Unsigned Operations		
JB/JNAE	JUMP if below (i.e. neither above nor equal)	CF = 1
JNB/JAE	JUMP if not below (i.e. either above or equal)	CF = 0
JBE/JNA	JUMP if below or equal (i.e. not above)	CF ⊕ ZF = 1
JNBE/JA	JUMP if neither below nor equal (i.e. above)	CF ⊕ ZF = 0

4.9.3 Iteration Control Instructions

The Instructions in this group are used to regulate the repetition of software loops. Like conditional transfers the iteration control instructions are self relative and may only transfer to targets that are within – 128 to + 127 bytes of themselves, i.e. they are short transfers.



1. LOOP

Mnemonic	LOOP short_label **LOOP** : Jump to specified lable if CX ≠ 0 after auto decrement	Flags	No flags are affected.
Algorithm	CX = CX – 1 If CX <> 0 then jump else no jump, continue		
Operation	This instruction is used to repeat a series of instructions some number of times. The number of times the instruction sequence is to be repeated is loaded into CX. Each time loop executes CX is decremented by 1. If CX ≠ 0 execution will jump to destination specified by label. If CX = 0 execution will go to the next instruction after loop.		

Example	MOV SI, offset ARRAY
	MOV AL, 00H
	MOV CX, 10H ; Counter loaded		

```
A1 : ADD AL, [SI]   ;  Add AL with contents of memory location
                    ;  pointed by SI.
     INC SI
     LOOP A1
```

2. LOOPZ/LOOPE – Loop if zero / equal

Mnemonic	Loop while CX ≠ 0 and ZF = 1 LOOPE short-label/LOOPZ short-label.	**Flags**	No flags are affected

Algorithm	CX = CX – 1
	If (CX <> 0) and
	ZF = 1 then jump
	else
	no jump, continue.		

Operation	This instruction is used to repeat a group of instructions some number of times or until zero flag becomes zero. Number of times of repetition is loaded in CX.		

3. LOOPNZ/LOOPNE – Loop if not zero / not equal

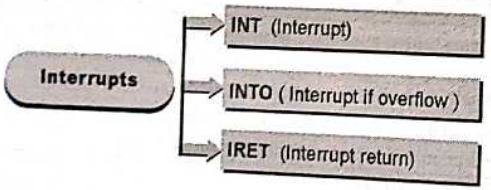
Mnemonic	Loop while CX ≠ 0 and ZF = 0 LOOPNZ short-label or LOOPNE short-label	**Flags**	No flags are affected.

Algorithm	CX = CX – 1
	if (CX <> 0) and
	ZF = 0 then jump
	else
	no jump, continue		

Operation	– This instruction is used to repeat a group of instructions some number of times or until zero flag becomes 1. – Number of times of repetition is loaded in CX.		

4.9.4 Interrupts

The interrupt instructions allow interrupt service routines to be activated by programs as well as by external hardware device. The effect of software interrupts is similar to hardware-initiated interrupts.



1. INT – Interrupt

Mnemonic	INT interrupt – type	**Flags**	IF = 0 and TF = 0 No other flags are affected
Algorithm	Push to stack flag register CS IP IF = 0, TF = 0 Transfer control to interrupt procedure.		
Operation	This instruction causes 8086 to call a far procedure. The term 'type' refers number between 0 to 255 which identifies the interrupt. When an 8086 executes an INT instruction, it will (i) Decrement SP by 2 and push flag register on stack. (ii) Decrement SP by 2 and push CS contents on stack. (iii) Decrement SP by 2 and push the IP after INT on stack. (iv) Get a new value for IP from a memory address of 4 times the type specified in instruction. e.g. For INT 8, the new IP will be read from 00020H. (v) Get new CS from memory address of 4 times the type specified in instruction plus 2. e.g. For INT 8, new value of CS will be read from 00022H. e.g. INT 35 : New IP from 008C H, New CS from : 008E H (vi) Reset IF and TF		

2. INTO – Interrupt if overflow

Mnemonic	INTO	**Flags**	IF = 0 and TF = 0 No other flag is affected.
Algorithm	If OF =1 then INT		
Operation	If the overflow flag is set, this instruction will cause the 8086 to do an indirect far call to a procedure you write to handle overflow condition. To do this call the 8086 will read a new value for IP from address 0010 H and CS from 0012 H.		

3. IRET – Interrupt return

Mnemonic	IRET	**Flags**	No flags are affected.
Algorithm	POP from stack IP CS Flag register		
Operation	The IRET instruction is used at the end of interrupt service routine to return execution to the interrupted program. The 8086 copies return address from stack into IP, and CS registers and stored value of flags back to flag register.		

Note : The RET instruction does not copy flags from the stack back to the flag register.

4.10 String Instructions Group

University Questions

Q. Explain following instructions with one example each (wrt 8086). STOS
MU - May 12, Dec. 12, May 13, 2 Marks

Q. Write a short notes on :String instructions
MU - Dec. 12, May 13, Dec. 13, 5 Marks

Q. Explain the following instructions in 8086 : LAHF and STOSB.
MU - Dec. 15, 5 Marks

Q. Briefly explain string instructions of 8086.
MU - May 17, 5 Marks

- As mentioned to you, 8086/8088 provides special instructions which performs some string related activities.
- We have 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. Let's refer Table 4.10.1, for varieties of string instructions.
- The five basic primitives (MOV, Compare, scan, load and store), may appear in one of the following form :

1. Operation B OR 2.Operation W.

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

Table 4.10.1 : String instructions

REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero
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 covered here :

1. A string instruction may 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. (One can use ***segment override prefix*** to change the segment forcefully).
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 auto-increment (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 4.10.2 summarises all 8 points.

Table 4.10.2 : 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 = auto increment SI, DI 1 = auto decrement SI, DI
ZF	Scan/compare terminator

Note : As mentioned, string instructions, operate on ONE element (byte/word) at a time. Therefore if you want to repeat the operation for N number of times, you can use prefix REP, REPZ, or REPNZ.

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.

To start with we will study string operations, which operate only ONCE and after that we will study REP prefix to repeat the operations.

1. MOVS/MOVSB/MOVSW : move string byte or string word

Mnemonic	MOVSB/MOVSW

Algorithm	**In case of byte :**
	ES : [DI] = DS : [SI]
	If DF = 0 then
	SI = SI + 1
	DI = DI + 1
	else
	SI = SI – 1
	DI = DI – 1
	In case of word :
	ES : [DI] = DS : [SI]
	If DF = 0 then,
	SI = SI + 2
	DI = DI + 2
	else
	SI = SI – 2
	DI = DI – 2

Flags	No flags are affected.
Addr. Mode	String addressing mode

Operation	– This instruction copies a byte or a word from a location in the data segment to a location in the extra segment. – The offset of the source byte/word in the DS must be SI register. – The offset of the destination in ES must be in DI register. – After byte or word is moved, SI and DI are automatically adjusted to point to the next source and next destination.
Example	LEA SI, SOURCE_STRING ; SI points to start of source string. LEA DI, DEST_STRING ; DI points to start of destination string.

	CLD	; Clear direction flag to use incrementing
	MOVSB	; MOV (DI) ← (SI), byte transfer

Note : 2 alternate mnemonics : MOVSB → move string bytes
MOVSW → move string words

2. CMPS : compare string byte or string words

Mnemonic	CMPSB/CMPSW Two alternate mnemonics for the same instruction CMPB, (compare string byte) CMPW, (compare string word)

Algorithm	For byte operation
	DS : [SI] – ES : [DI]
	Set flags according to result
	→ OF, SF, ZF, AF, PF, CF.
	If DF = 0 then
	SI = SI + 1
	DI = DI + 1
	else
	SI = SI – 1
	DI = DI – 1
	For word operation
	DS : [SI] – ES : [DI]
	Set flags, according to result
	→ OF, SF, ZF, AF, PF, CF.
	If DF = 0 then
	SI = SI + 2
	DI = DI + 2
	else
	SI = SI – 2
	DI = DI – 2

Flags	All flags are affected.
Addr. Mode	String addressing mode

Operation	– This instruction is used to compare a byte in one string with a byte in another string or to compare a word in one string with another string. – SI is used to hold the offset of a byte or word in the source string and DI is used to hold the offset of byte or word is another string. – Comparison is done by subtracting the byte or word pointed by DI from byte or word pointed by SI. – After comparison SI and DI will be automatically incremented or decremented according to direction flag to point to next element in the string.

Example	LEA SI, SRC	; SI points to start of source string labeled SRC.
	LEA DI, DEST	; DI points to start of destination string labeled DEST
	CLD	; Clear DF (incrementing is used).
	CMPS SRC, DEST	; Compare.

Note : Note that no operand is changed.

3. SCAS/SCASB/SCASW

(Scan a string Byte or Word)

Mnemonic	SCASB/SCASW SCASB (for byte operation) Or SCASW (for word operation)

Algorithm	**For SCASB**
	AL – ES : [DI]
	Set flags according to result
	If DF = 0 then DI = DI + 1
	For SCASW
	AX – ES : [DI]
	Set flags according to result
	If DF = 0 then
	DI = DI + 2
	Else
	DI = DI – 2

Flags	No flags are affected.
Addr. Mode	String addressing mode

Operation	This instruction compares a byte in AL or word in AX with a byte or word pointed to by DI in ES. Hence the string to be scanned must be in ES, and offset in DI. After comparison DI will be automatically incremented or decremented according to direction flag DF.
Example	Let BUFF ← 0EH LEA DI, BUFF ; Load offset Buffer in DI MOV AL, 0DH ; AL is loaded with the Byte to be compared. i.e. 0DH it is ; ASCII for carriage return it indicates line termination. CLD ; Clear direction flag DF to use incrementing SCASB ; Scan for byte in the string.

4. LODS

(Load string byte in AL or string word into AX.)

Mnemonic	LODSB/LODSW Or LODSB (For byte operation) Or LODSW (For word operation)

Flags	No flags are affected.

Algorithm	**LODSB** AL = DS : [SI] If DF = 0 then SI = SI + 1 else SI = SI – 1 **LODSW** AX = DS : [SI] If DF = 0 then SI = SI + 2 else SI = SI – 2
Addr. Mode	String addressing mode
Operation	This instruction copies a byte from a string location pointed to by SI to AX.
Example	**CLD** MOV SI, OFFSET A1 : SI points to start of A1. LODSB : Load byte in AL from the string

5. STOS/STOSB/STOSW

(Store byte or word in string)

Mnemonic	STOSB/STOSW STOSB (For byte operation) or STOSW (for word operation)
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
Flags	No flags are affected
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

6. REP/REPE/REPNE/REPNZ : (Prefix)

(Repeat string instruction until specified conditions exist).

Mnemonic	REP/REPE/REPNE/REPNZ
Flags	-
Algorithm	Check_CX For REP – CX = CX – 1 – Repeat the instruction to which it is a prefix – go back to check _ CX else – exit from REP cycle
Operation	– REP is a prefix written before one of the string instructions. These instructions repeat until specified condition exists.

Instruction code	Condition for exit
REP	CX = 0
REPE/REPZ	CX = 0 or ZF = 0
REPNE/REPNZ	CX = 0 or ZF = 1

Example

```
LEA SI, SOURCE_STRING  ;  SI having offset
LEA DI, DEST_STRING    ;  DI having offset of destination_string
MOV CX, 20             ;  Load counter
CLD                    ;  DF = 0
REP MOVSW              ;  Repeat, move string word operation for CX times.
```

If CMPS and SCAS operations are used, the two strings are compared and a byte in AL or word in AX will be compared with the string. In such cases, if we want to quit the loop or want to stop repeat instructions for the conditions mentioned as follows :

1. CX = 0, means string end is reached.
2. CX ≠ 0 (means string is not ended), but match found.
3. CX ≠ 0 (means string is not ended), but mismatch found.

Thus match (Z = 1) or mismatch (ZF = 0), these are the conditions to stop repeat operations.

1. REPE/REPZ (Repeat while equal)/Repeat while zero).
2. REPNE/REPNZ (Repeat while not equal/Repeat while not zero).

A. REPE/REPZ (Repeat while equal)/Repeat while zero).

Mnemonic	REPE/REPZ

Algorithm	Check CX
	If CX <> 0 then
	CX = CX - 1
	If ZF = 1 then
	repeat the instruction to which it is prefix
	go back to check CX
	else
	exit from REPE cycle
	else
	exit from REPE cycle.

Operation	– These are two mnemonics for the same instruction. – This prefix will cause the string instructions to be repeated as long as ZF = 1 and CX ≠ 0. – If ZF = 0 or CX = 0, string instructions will not be repeated.
Example	LEA DI, STR 1 ; DI = offset of STR 1 MOV AL, 05 H ; AL = 05 H CLD ; DF = 0 MOV CX, 35 H ; CX = 35 H i.e. counter = 35 H REPE SCASB ; Repeat, scan string byte operation till CX ≠ 0 ; and ZF = 1.

B. REPNE/REPNZ (Repeat while not equal/Repeat while not zero)

Mnemonic	REPNE/REPNZ

Algorithm	Check CX
	if CX <> 0 then
	CX = CX - 1
	If ZF = 0 then
	repeat the instruction to which it is prefix
	go back to check CX
	else
	exit from REPNE cycle
	else
	exit from REPNE cycle.

Operation	These are two mnemonics for the same instruction. This will cause string instructions to be repeated as long as ZF = 0 and CX ≠ 0. If ZF = 1 or CX = 0, string instructions will not be repeated.

Example	MOV DI, offset string 1	; DI points to string 1
	MOV AX, 24 H	; AX = 24 H
	MOV CX, 100	; Load count = 100
	CLD	; DF = 0
	REPNE SCASW	;Repeat scan word operation till ZF = 0 and CX ≠ 0.

4.11 Exam Pack (Review and University Questions)

Q. 1 Draw the programmers model of 8086 microprocessor and label it neatly. **(Refer Section 4.1)** **(6 Marks)**

Q. 2 Explain programming model of 8086. **(Refer Section 4.1)** **(May 16, 5 Marks)**

Q. 3 Explain the following addressing modes :

(a) Direct addressing

(b) Immediate addressing

(c) Register indirect addressing

(d) Based addressing

(e) Indexed addressing

(Refer Section 4.2) **(May 16, 5 Marks)**

Q. 4 Write short note on : Addressing modes in the 8086. **(Refer Section 4.2)** **(May 11, Dec. 12, Dec. 13, 5 Marks)**

Q. 5 List different addressing modes of 8086. **(Refer Section 4.2)** **(Dec.- 16, 5 Marks)**

Q. 6 Write addressing modes of following instructions of 8086.

(i) MOV al, [bx + si]

(ii) AND cl, [2000]

(iii) ADD ax, [bx + si + 2000]

(iv) IN al, dx

(v) POP BX **(Refer Section 4.2)** **(Dec. 17, 5 Marks)**

Q. 7 Explain following addressing modes of Intel 8086. Write an instruction for each mode :

(a) Direct addressing mode, **(Refer Section 4.2.3(1))**

(b) Relative base indexed. **(Refer Section 4.2.3(5))** **(Dec.- 11, 5 Marks)**

Q. 8 "Based indexed" addressing mode of 8086 is useful for an array element access explain. **(Refer Section 4.2.3)** **(8 Marks)**

Q. 9 For the following instruction compute the address of memory operand for 8086 :

(i) MOV AX, [BX]

(ii) MOV AL, [BP + SI]

Assume CS : 0100 H, DS : 0200 H, SS : 0400 H, ES : 0030 H, BP : 0010 H, BX : 0020 H,

SI : 0030 H, SP = 0040 H.

Clearly show computations. **(Refer Section 4.2.3)** **(5 Marks)**

Q. 10 Explain I/O related addressing mode of 8086. **(Refer Section 4.2.5)** **(Dec.- 14, 5 Marks)**

Q. 11 Explain instruction set of 8086. **(Refer Section 4.4)** **(8 Marks)**

Q. 12 Explain data transfer instructions. **(Refer Section 4.5)** **(5 Marks)**

Q.13 Write a short notes on : Fixed port and variable port addressing. **(Refer Section 4.5.2)** **(Dec. 11, May 13, 5 Marks)**

Q. 14 Explain the following instructions in 8086 : LAHF and STOSB.**(Refer Sections 4.5.4(1) and 4.10(5))** **(Dec. 15, Dec. 19, 5 Marks)**

Q.15 Explain arithmetic instructions. **(Refer Section 4.6)** **(5 Marks)**

Q.16 Explain the following Intel 8086 assembly language instruction giving example : DAA **(Refer Section 4.6.1(5))** **(Dec. 11, Dec. 12, May 13, 2 Marks)**

Q. 17 Explain following instructions with one example each (wrt 8086).

(iv) CMP **(Refer Section 4.6.2(7))**

(May 12, 2 Marks)

Q. 18 Explain the following Intel 8086 assembly language instructions giving example :

(i) TEST **(Refer Section 4.7.1(5))**

(Dec. 11, May 12, Dec. 12, May 13, 2 Marks)

Q. 19 Explain the following Intel 8086 assembly language instructions giving example :

(i) SAL **(Refer Section 4.7.2(1))**

(Dec. 11, May 12, 2 Marks)

Q. 20 Explain the following Intel 8086 assembly language instructions giving example.

SAR. **(Refer Section 4.7.2(3))**

(Dec. 12, May 13, 2 Marks)

Q. 21 Explain processor control instructions.

(Refer Section 4.8) **(5 Marks)**

Q. 22 Explain the following Intel 8086 assembly language instructions giving example :NOP

(Refer Section 4.8.2) **(Dec. 11, May 13, 2 Marks)**

Q. 23 Write a detailed note on : Branch instructions of the 8086. **(Refer Section 4.9)** **(May 11, 10 Marks)**

Q. 24 Explain the conditional transfer instructions.

(Refer Section 4.9.2) **(5 Marks)**

Q. 25 Explain the following Intel 8086 assembly language instructions giving example : JA

(Refer Section 4.9.2) **(Dec. 11, 2 Marks)**

Ans. :

JA Back : This instruction will branch to the lable 'Back' if CF⊕ ZF = 0

Q. 26 Explain following instructions with one example each (wrt 8086) : JC

(Refer Section 4.9.2) **(May 12, 2 Marks)**

Ans. :

JC Back : This instruction will branch to the lable 'Back' if CF = 1

Q. 27 Explain the following Intel 8086 assembly language instructions giving example : JC

(Refer Section 4.9.2) **(Dec. 12, 2 Marks)**

Ans. :

Example of JC : JC Next

Q. 28 Explain following instructions with one example each (wrt 8086) : STOS **(Refer Section 4.10(5))**

(May 12, Dec. 12, May 13, 2 Marks)

Q. 29 Write a short notes on : String Instructions **(Refer Section 4.10)**

(Dec. 12, May 13, Dec. 13, 5 Marks)

Q. 30 Explain the following instructions in 8086 : LAHF and STOSB.**(Refer Sections 4.5.4(1) and 4.10(5))**

(Dec. 15, 5 Marks)

Q. 31 Briefly explain string instructions of 8086.

(Refer Section 4.10) **(May 17, 5 Marks)**

Q. 32 Explain different addressing modes of 8086 microprocessor.

(Refer Section 4.2) **(Dec. 18, 5 Marks)**

Q. 33 Write addressing modes of following instructions:

(i) MOV AX, [BX+SI]

(ii) AND CL,[2000]

(iii) IN AL,DX

(iv) JMP[BX+2]

(v) ADD AX,[BX+SI+5]

(Refer Section 4.2) **(Dec. 19, 5 Marks)**

Q. Explain the following instruction in 8086 : XLAT **(Refer Section 4.5.1(E))** **(Dec. 19, 5 Marks)**

□□□

MODULE 2

5 Assembly Language Programming

5.1 Assembly Language Program Development Tools

Q. List and explain assembly language program development tools. **(10 Marks)**

In this section we will go into the details of some of the assembly language program development tools.

5.1.1 Editor

- The program is written on the computer using a software tool called as Editor.
- The editor is used to write and edit the programs.
- Once the program is written, it can be given to assembler to assemble the code

5.1.2 Assembler

- An assembler assembles i.e. converts the assembly language program to machine language
- The microprocessor as discussed in earlier chapters, can understand only binary or machine language code.
- The assembly instructions studied in previous chapter are converted to the programs
- The assembler accepts an assembly file or a file with extension of "asm" i.e. ".asm" file and produces an object file as output i.e. a file with extension ".obj"

5.1.3 Linker

- Linker links multiple object files and produces an executable file
- It can be said to join multiple object files.
- In case if a program is made with subroutines or functions in separate file, then the linker combines these files.

5.1.4 Locator

- The locator is a program which is used for assigning the specific address of the locations where the segments of the object code are to be loaded into the memory.
- A locator program called EXE2BIN comes with the IBM. PC disk operating system (DOS).
- This program converts an .EXE file into a .BIN file which has physical addresses.

5.1.5 Debugger

- As the name says, debugger is used to remove bugs from the program.
- A bug is an error. And debugging refers to removing the error.
- Using debugger, the programmer can execute the program step by step to locate the error. After every step, the contents of memory locations, registers and flag can be monitored, so as to understand where the program is not functioning properly.
- The debugger also provides break point besides the single step execution. In case of break point execution, the program can insert break points at certain places in the program. The debugger will execute the program until the break point and then user can check the contents of registers and memory locations. Once checked, the further part of the program can be executed.

5.1.6 Emulator

- Emulator is another tool used for debugging the program.
- It emulates or produces the same effect of execution of the program as the processor would do.
- In fact this software tool, needs a board which has the processor on it and the board is interfaced with the computer where the emulator resides.
- The emulator actually sends the instructions to the processor and shows the results.
- Special debugging like time requirement for the program execution can be checked using emulator.
- Emulators are normally used when the processor to be programmed are different than the one that resides in the computer.

5.2 Assembly Language Structure

- The general format of an assembler instruction is as follows :

Label	Mnemonic	Operand	Comment
(Optional) Used to identify the instruction. It is followed by colon (:)	Basically, this is the instruction. The mnemonic indicates the operation to be performed.	The data which is being operated upon. We have already seen in addressing modes, operand is not required in implied addressing mode	(Optional) These are followed by semicolon(;) and are useful for understanding the logic of Assembly Language program

eg: up: CMP AX, BX ; Compare contents of BX with AX

Label, Mnemonic, Source operand, Destination operand, Comment

5.3 Assembler Directives

University Question

Q. Explain assembler directives of 8086.

MU - May 12, Dec. 12, 10 Marks

Assembly language consists of two type of statements viz.

1. **Executable statements :** These are the statements to be executed by the processor. It consists of the entire instruction set of 8086 (as seen in the previous chapter.)

2. **Assembler directives :** These are the statements that direct the assembler to do something. As the name says, it direct the assembler to do a task.

We can divide the assembler directives into various categories as shown in Fig 5.3.1.

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

(a) Simplified segment directives
(b) Data allocation directives
(c) Segment directives
(d) Macros related directives
(e) Code label directives
(f) Scope directives
(g) Listing control directives
(h) Miscellaneous directive

1. CODE

- This indicates the assembler of the start of code segment.
- The CS register is initialized to such a value that it should point to this location in the beginning along with the Instruction pointer (IP).

2. DATA

- This indicates the assembler of the start of data segment.
- The DS register is initialized to such a value that it should point to this location in the beginning.

3. MODEL

- It is used to indicate the memory requirement of the program.

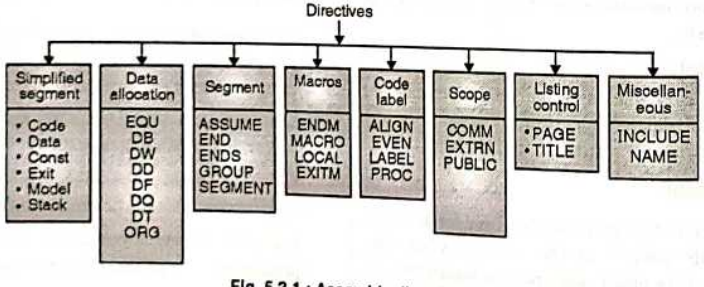


Fig. 5.3.1 : Assembly directives

- Along with the word "model", there is a word followed by it to indicate the size of memory block required by the program.
- It also indicates the number of segments that the program would require for each of the code and data.
- The various options for the size of memory block and the corresponding size of blocks allocated are listed in the Table 5.3.1.

Table 5.3.1

Model	Number of Code Segments	Number of Data Segments
Small	One code segment and of size < = 64 Kbytes	One of size < = 64 Kbytes
Medium	Code segment may be of any number and any size.	One of size < = 64 Kbytes
Compact	One of size < = 64 Kbytes	Data segment of any number and any size.
Large	Any number, any size	Any number, any size.
Huge	Any number, any size	Any number, any size.

4. STACK

- This indicates the assembler of the start of stack segment.
- The SS register is initialized to such a value that it should point to this location in the beginning. Its format is as follows :

STACK [size]

- The size of the stack is 1024 bytes by default but this size can be overridden.

5. EQU-Equate

- EQU stands for EQUAL or EQUATE.
- It is used to assign a value to a variable or constant.
- For e.g. X EQU 10.

6. Define Byte [DB]

- This directive is used to initialize a variable as a byte (8-bit) variable.
- Hence the memory space allocated to such a variable is one byte.
- The initial value to be assigned to the variable is also to be specified. If no value is to be assigned, then a question mark (?) is to be given in the place of value.

For e.g. X DB 5.

- This statement creates a byte variable named as X and the value initialised is 5.

7. Define Word or Word [DW]

- This directive is used to initialize a variable as a word (16-bit) variable.
- Hence the memory space allocated to such a variable is two bytes.
- The initial value to be assigned to the variable is also to be specified. If no value is to be assigned, then a question mark (?) is to be given in the place of value.

For e.g. X DW 5000H.

- This statement creates a word variable named as X and the value initialised is 5000H.

8. Define Double word or DWORD [DD]

- This directive is used to initialize a variable as a double word (32-bit) variable.
- Hence the memory space allocated to such a variable is four bytes.
- The initial value to be assigned to the variable is also to be specified. If no value is to be assigned, then a question mark (?) is to be given in the place of value.

For e.g. X DD ?

- This statement creates a word variable named as X and no value is initialised.

9. Define Quad word or QWORD [DQ]

- This directive is used to initialize a variable as a Quad word (64-bit) variable.
- Hence the memory space allocated to such a variable is eight bytes.
- The initial value to be assigned to the variable is also to be specified. If no value is to be assigned, then a question mark (?) is to be given in the place of value.

For e.g. X DQ 12345000H.

- This statement creates a quad word variable named as X and the value initialised is 12345000H.

10. Define Ten Bytes or TBYTE [DT]

- This directive is used to initialize a variable as a ten byte (80-bit) variable.
- Hence the memory space allocated to such a variable is ten bytes.
- The initial value to be assigned to the variable is also to be specified. If no value is to be assigned, then a question mark (?) is to be given in the place of value.

  For e.g. X DT 1234675000H.
- This statement creates a ten byte variable named as X and the value initialised is 1234675000H.

11. ORG-Originate

- This directive indicates the assembler, that the next data/code should origin from the specified location.

  e.g. ORG 1000 H ; The code/data following should be ; stored at 1000H.

12. ASSUME

- This directive is used to give another name for segment registers, so as to make it easy to give the name.
- The example of Assume directive is as follows,

  ASSUME CS : Code, DS : Data, SS : Stack
- In this example, DS : Data indicates the assembler to associate the name of data segment with DS register.
- Similarly CS : Code indicates the assembler to associate the name of code segment with CS register.

13. END

- This is placed at the end of a source and it acts as the last statement of a program.

14. SEGMENT and ENDS

- The SEGMENT directive is used to indicate the start of a logical statement.
- ENDS directive is used to indicate the end of segment.

  e.g. : DATA SEGMENT

  -

  -

  DATA ENDS.

15. MACRO AND ENDM

- The start of a macro is defined by the directive MACRO.
- The ENDM directive is used along with the Macro to indicate the end of macro.

e.g. : KBD MACRO;

Statements inside the Macro

ENDM

16. PROC-Procedure and ENDP-End Procedure

- This start of a procedure is defined by the directive PROC.
- The procedures are of two types
- Procedure in the same segment is called as NEAR procedure and
- Procedure in a different segment as the current code in execution is called as a FAR procedure.
- ENDP directive is used along with the name of the procedure to indicate the end of procedure
- ENDP defines the end of procedure.

e.g. : FACT PROC FAR ; Start procedure named FACT and

; indicates the assembler that the

; procedure is FAR.

-

-

FACT ENDP ; End of Procedure FACT.

17. PTR-Pointer

- It is used to assign a specific data size to a pointer.
- It is necessary to do this in any instruction where the size of operand (byte or word) is not clear.

e.g. INC BYTE PTR [BX] : It increments 8-bit data pointed to

; by [BX]

INC WORD PTR [BX] : It increments 16-bit data pointed to

; by [BX].

5.4 Structured Programming

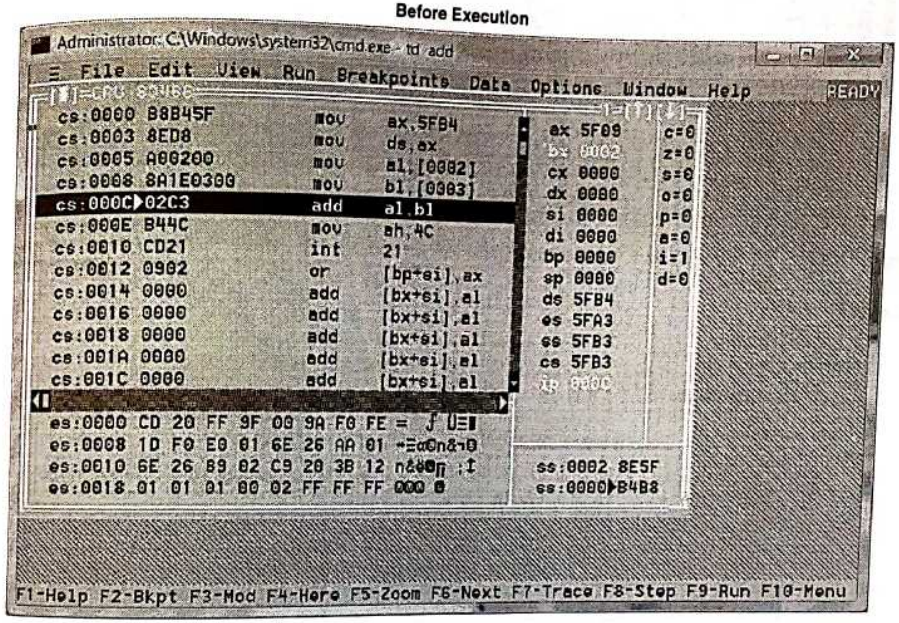
Program 5.4.1 : Write a program to add two 8 bit numbers stored in memory locations. The result must be stored in register AL **Lab Experiment**

Program

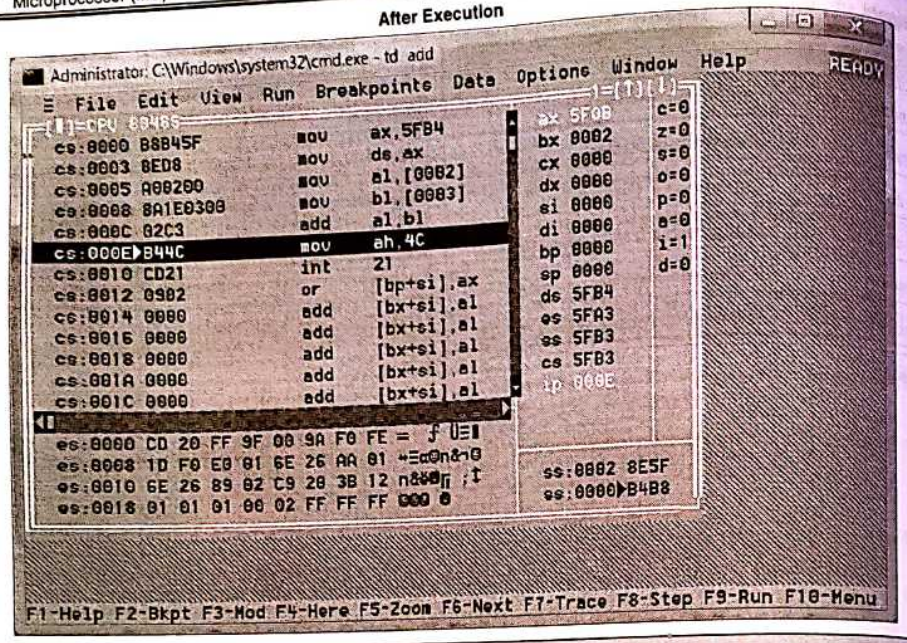
Instruction	Comment
.model small	
.data	Start the data segment
a db 09H	Load variable 'a' with data 09H
b db 02H	Load variable 'b' with data 02H
.code	Start of code segment
mov ax, @data	Initialize data segment (DS) register
mov ds, ax	to point to data segment
mov al, a	Load first operand in al
mov bl, b	Load second operand in bl
add al, bl	Add numbers and result in al
mov ah, 4CH	Terminate Program
int 21H	
end	End Program

Result :

Before Execution



After Execution



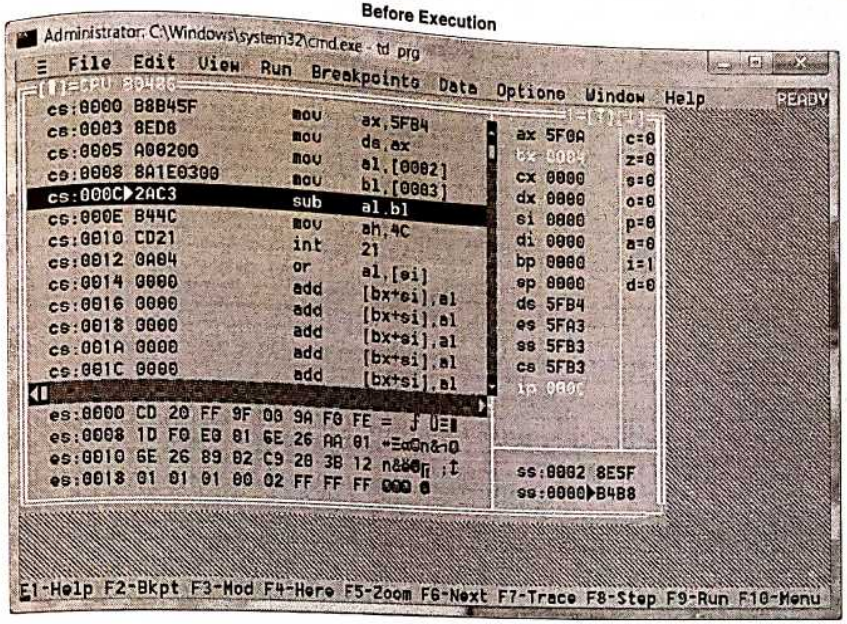
Program 5.4.2 : Write a program to subtract two 8 bit numbers stored in memory locations. The result must be stored in register AL. **Lab Experiment**

Program :

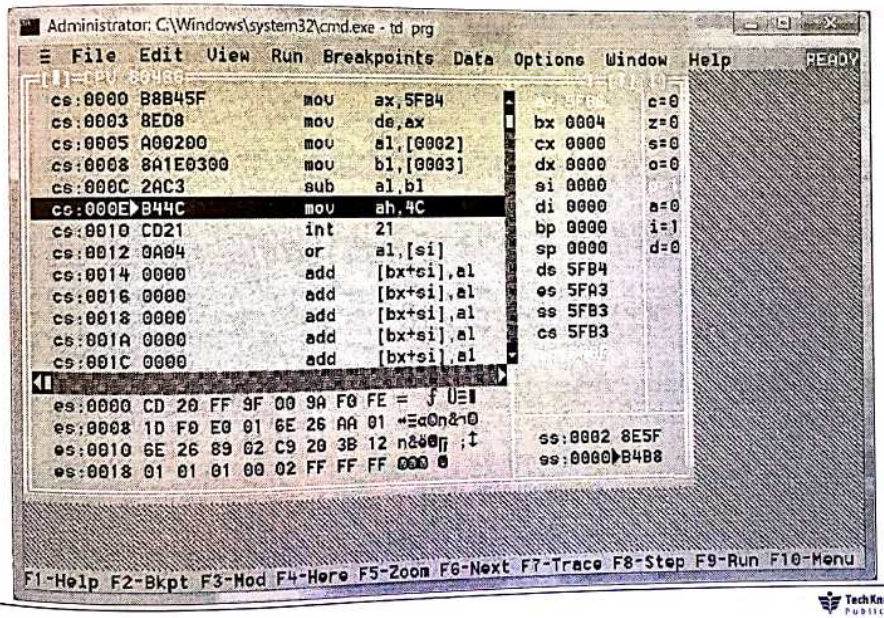
Instruction	Comment
.model small	
.data	Initialize data segment.
a db 0AH	Load variable 'a' with data 0AH
b db 04H	Load variable 'b' with data 04H
.code	Initialize code segment
mov ax, data	Initialize data segment (DS) register to point to data segment
mov ds, ax	
mov al, a	Load first operand in AL
mov bl, b	Load second operand in BL
sub al, bl	subtract numbers and store the result in AL
mov ah, 4ch	Terminate Program
int 21H	
end	End

Result

Before Execution



After Execution



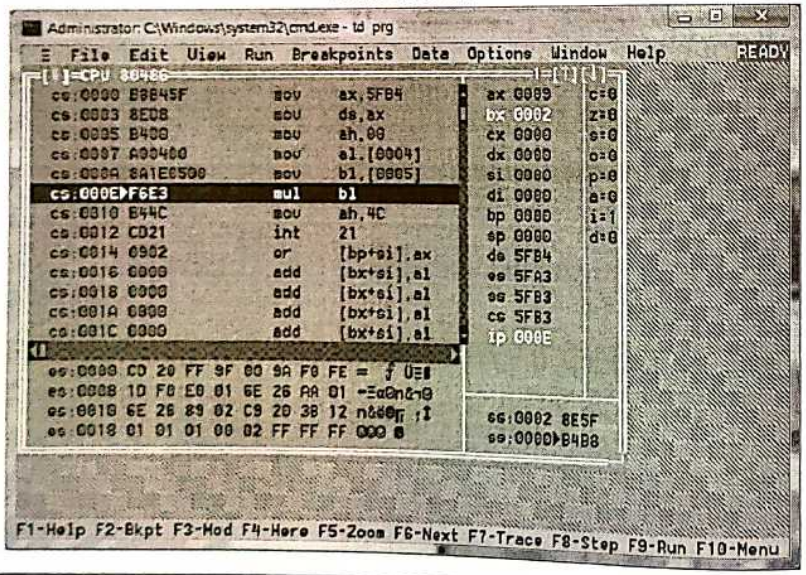
Program 5.4.3 : Write a program to multiply two 8 bit numbers stored in memory locations. The result must be stored in register AX. **Lab Experiment**

Program

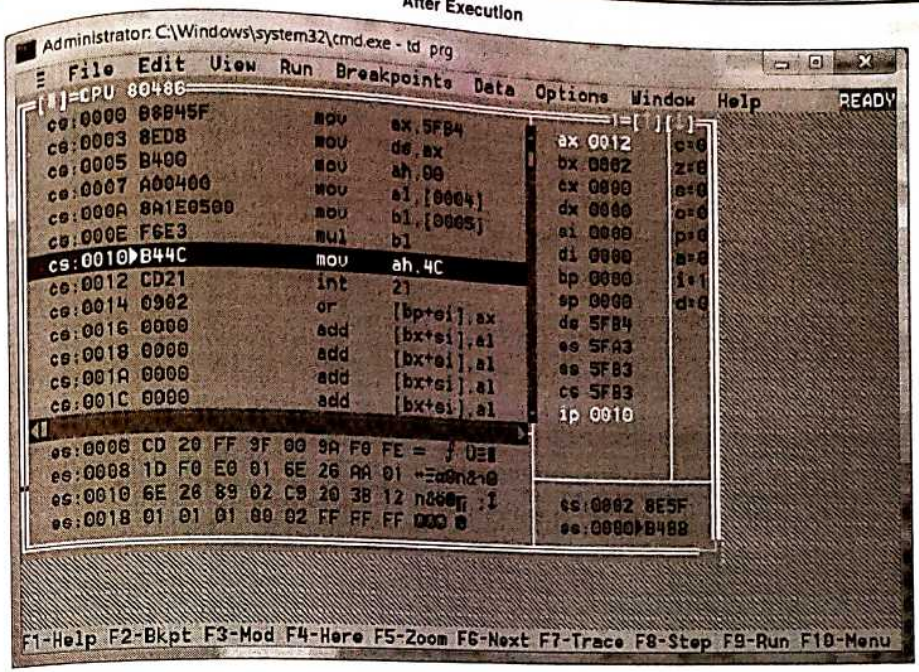
Instruction	Comment
.model small	
.data	Initialize data segment
a db 09H	Load variable a with data 09H
b db 02H	Load variable b with data 02H
.code	Initialize code segment
mov ax, @data	Initialize data segment (DS) register to point to data segment
mov ds, ax	
mov ah, 0	Clear register AH
mov al, a	Load first operand in al
mov bl, b	Load second operand in bl
mul bl	multiply numbers and store the result in ax
mov ah, 4cH	Terminate Program
int 21H	
end	End

Result :

Before Execution



After Execution



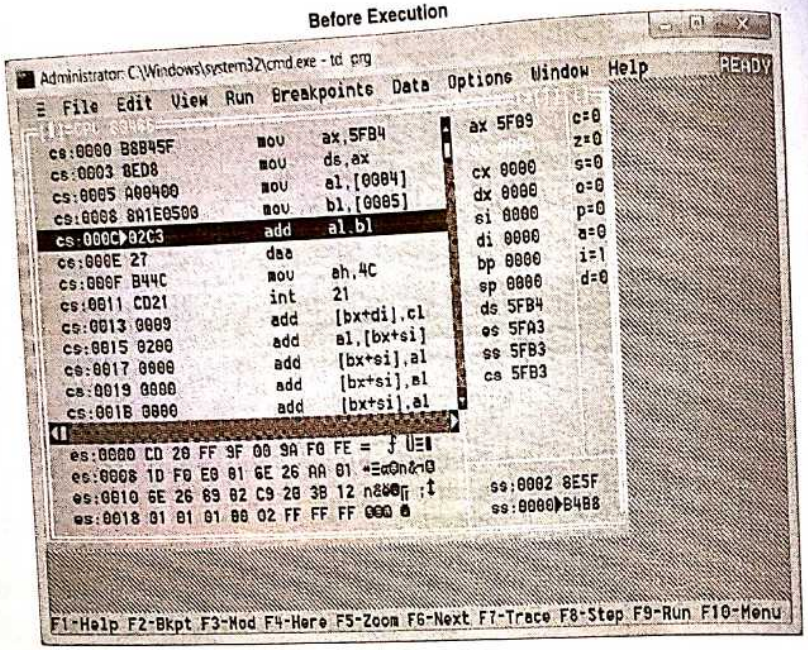
Program 5.4.4 : Write a program to add two 8 bit BCD numbers stored in memory locations. The result must be stored in register AL.

Program :

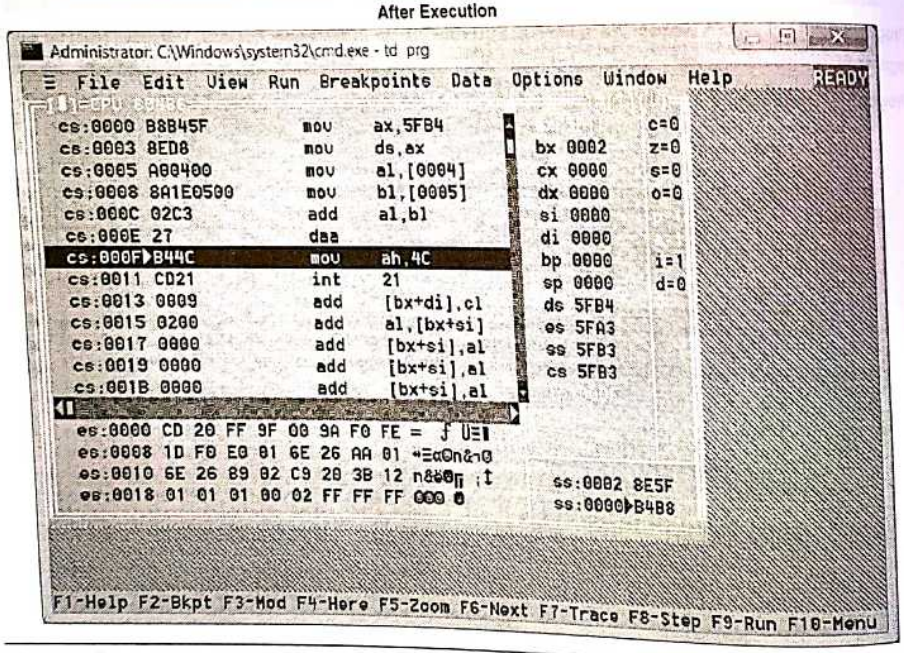
Instruction	Comment
.model small	
.data	Initialize data segment
a db 09H	Load variable a with data 09H
b db 02H	Load variable b with data 02H
.code	Initialize code segment
mov ax, @data	Initialize data segment (DS) register to point to data segment
mov ds, ax	
mov al, a	Load first operand in al
mov bl, b	Load second operand in bl
add al, bl	add numbers and result in al
daa	adjust result to valid BCD result
mov ah, 4cH	Terminate Program
int 21H	
end	End

Result

Before Execution



After Execution



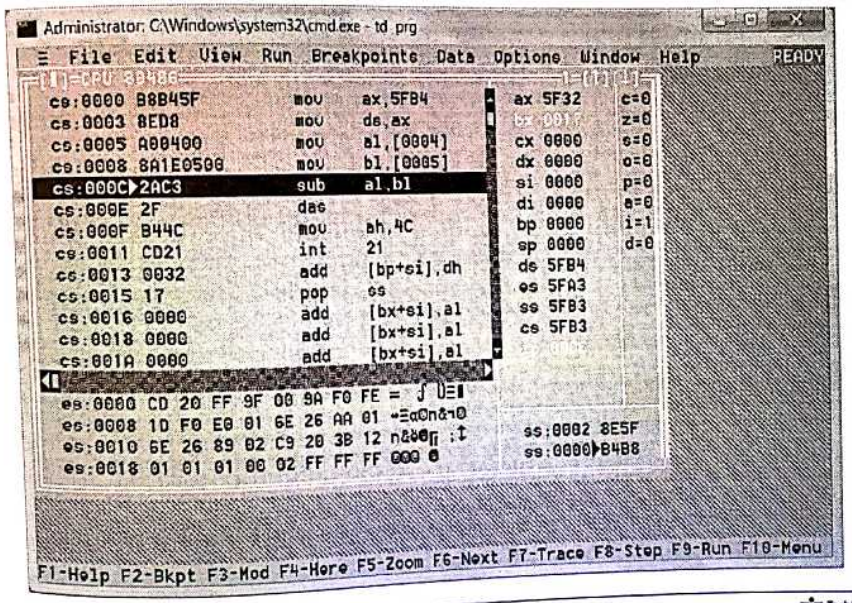
Program 5.4.5 : Write a program to subtract two 8 bit BCD numbers stored in memory locations. The result must be stored in register AL.

Program

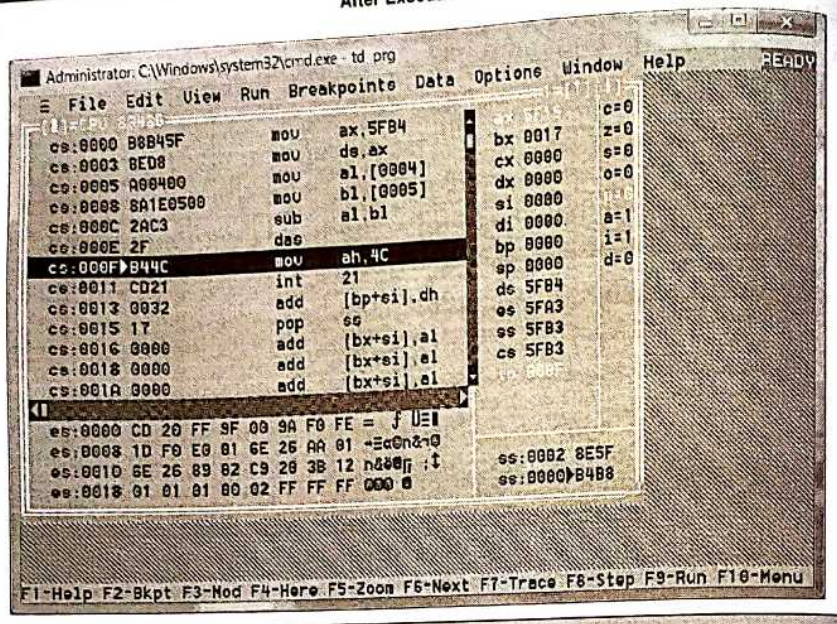
Instruction	Comment
.model small	
.data	Initialize the data segment
a db 32H	Load variable a with 32 H
b db 17H	Load variable b with 17 H
.code	Initialize the code segment
mov ax, @data	Initialize data segment (DS) register to point to data segment
mov ds, ax	
mov al, a	Load first operand in al
mov bl, b	Load second operand in bl
sub al, bl	subtract numbers and result in al
das	adjust result to valid BCD result
mov ah, 4cH	Terminate Program
int 21H	
end	

Result

Before Execution



After Execution



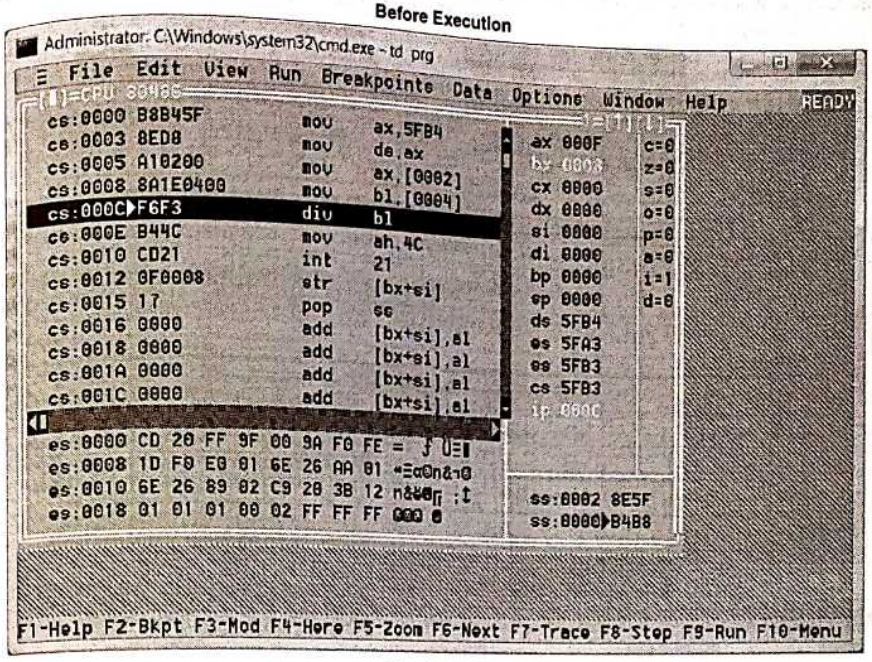
Program 5.4.6 : Write a program to divide a 16-bit number by an 8 bit numbers stored in memory locations. The result must be stored in register AX. **Lab Experiment**

Program :

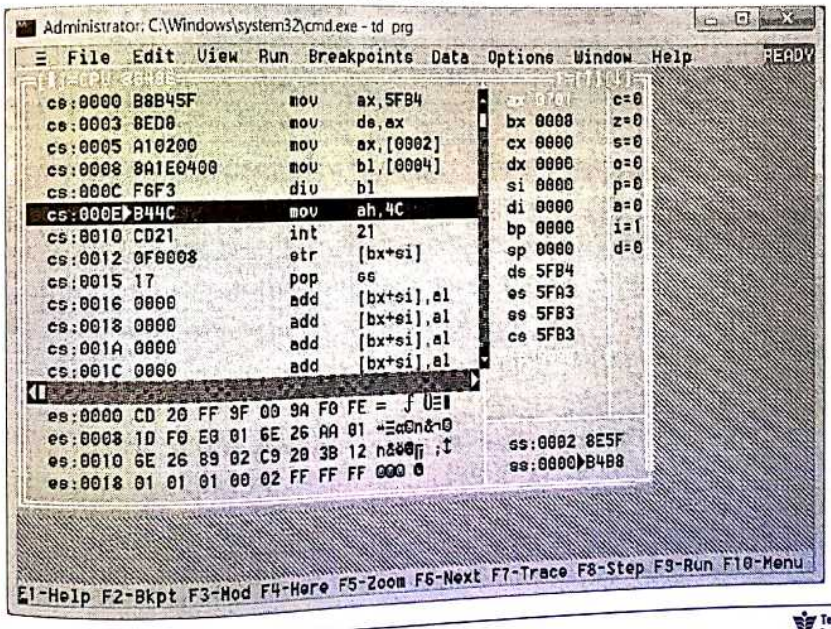
Instruction	Comment
.model small	
.data	Initialize the data segment
a dw 000FH	Load variable a with data 000FH
b db 08H	Load variable b with data 08H
.code	Initialize the code segment
mov ax, @data	Initialize data segment (DS) register to point to data segment
mov ds, ax	
mov ax, a	Load dividend in ax
mov bl, b	Load divisor in bl
div bl	divide numbers. Quotient will be in al and remainder in ah
mov ah, 4cH	Terminate Program
int 21H	
end	End

Result :

Before Execution



After Execution



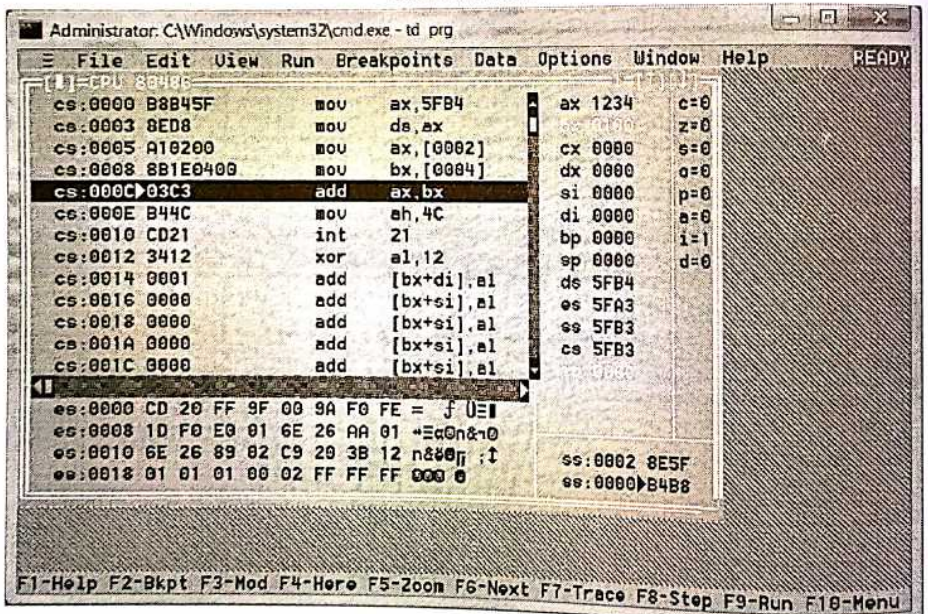
Program 5.4.7 : Write a program to add two 16-bit numbers stored in memory locations. The result must be stored in register AX. **Lab Experiment**

Program :

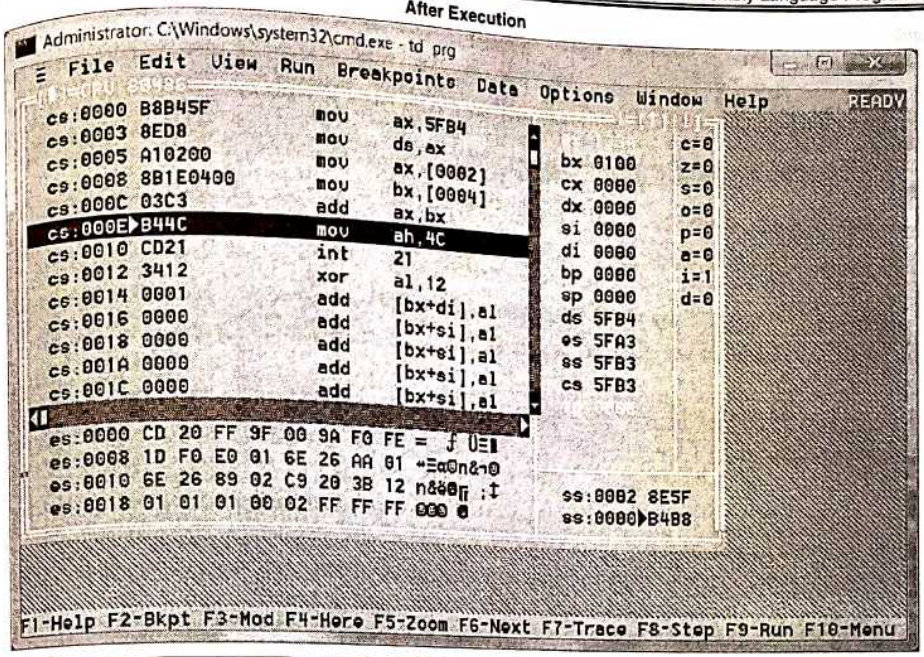
Instruction	Comment
.model small	
.data	Initialize data segment
a dw 1234H	Load variable a with data 1234H
b dw 0100H	Load variable b with data 0100H
.code	Initialize code segment
mov ax, @data	Initialize data segment (DS) register to point to data segment
mov ds, ax	
mov ax, a	Load first operand in ax
mov bx, b	Load numbsecond operander2 in bx
add ax, bx	add numbers. Result will be stored in ax
mov ah, 4cH	Terminate Program
int 21H	
end	End

Result :

Before Execution



After Execution



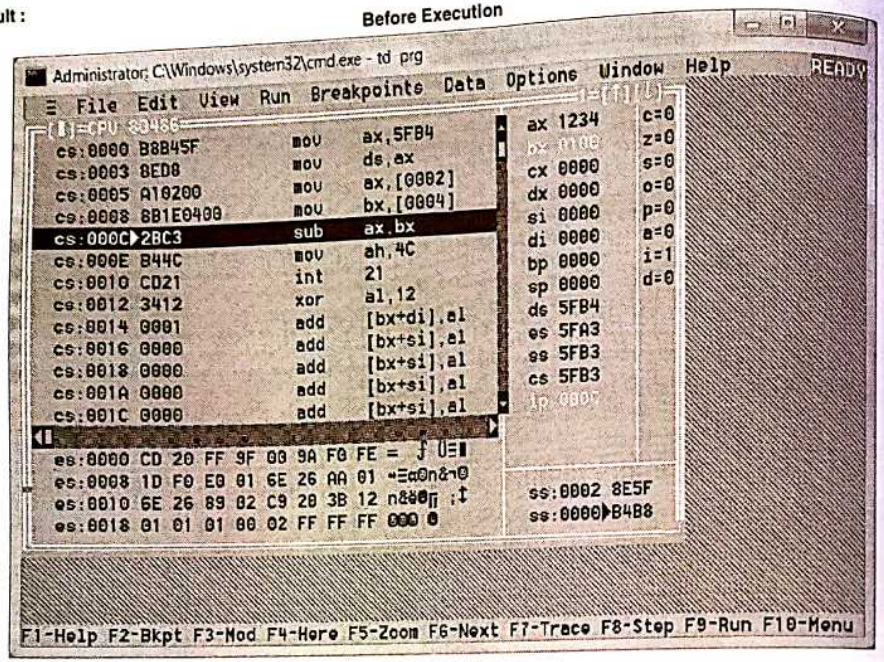
Program 5.4.8 : Write a program to subtract two 16-bit numbers stored in memory locations. The result must be stored in register AX. **Lab Experiment**

Program :

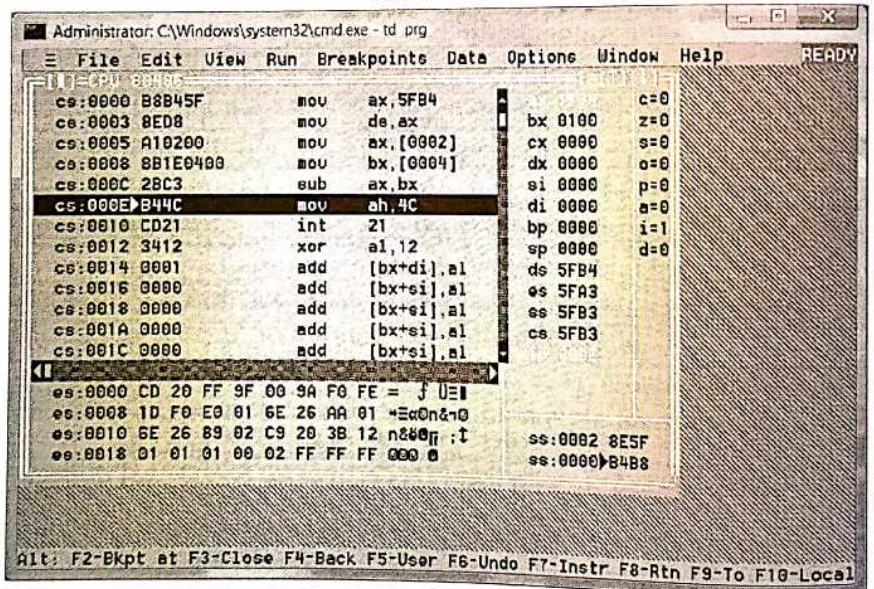
Instruction	Comment
.model small	
.data	Initialize data segment
a dw 1234H	Load variable a with 1234H
b dw 0100H	Load variable b with 0100H
.code	Initialize code segment
mov ax, @data	Initialize data segment (DS) register to point to data segment
mov ds, ax	
mov ax, a	Load first operand in ax
mov bx, b	Load second operand in bx
sub ax, bx	subtract numbers. Result will be stored in ax
mov ah, 4cH	Terminate Program
int 21H	
end	End

Result :

Before Execution



After Execution



Program 5.4.9 : Write and assembly program to divide 16 bit signed number by another 16 bit signed number.

Program

Instruction	Comment
.model small	
.data	Initialize data segment
a dw 8000H	Load variable a with 8000H
b dw 2000 H	Load variable b with 2000 H
.code	Initialize code segment
mov ax, @data	Initialize data segment (DS) register to point to data segment
mov ds, ax	
mov ax, a	Load dividend in AX
mov dx,0000h	Initialize upper word of dividend to 0000H
mov bx, b	Load second operand in bx
idiv bx	divide numbers. Quotient in AX and Remainder in DX
mov ah, 4ch	terminate program
int 21h	
end	End

Program 5.4.10 : Write an 8086 assembly language program for BCD to seven segment code conversion. Use XLAT instruction and common cathode display.

BCD Number	Seven segment conversion								Seven segment equivalent
	Dot	g	f	e	d	c	b	a	
0	0	0	1	1	1	1	1	1	3F
1	0	0	0	0	0	1	1	0	06
2	0	1	0	1	1	0	1	1	5B
3	0	1	0	0	1	1	1	1	4F
4	0	1	1	0	0	1	1	0	66
5	0	1	1	0	1	1	0	1	6D
6	0	1	1	1	1	1	0	1	7D
7	0	0	0	0	0	1	1	1	07
8	0	1	1	1	1	1	1	1	7F
9	0	1	1	0	0	1	1	1	67

Program

Instruction	Comment
.model small	
.data	Initialize data segment
lookup db 3Fh, 06h, 5Bh, 4Fh, 66H, 6Dh, 7Dh, 07h, 7Fh, 67h	Initialize look up table
key db 02h	Load the variable key
.code	Initialize code section
mov ax, @data	Initialize data segment (DS) register to point to data segment
mov ds, ax	
mov bx, offset lookup	Load offset of lookup in bx
mov al, key	Load key in al
xlat	translate byte in al
mov ah, 4cH	Terminate Program
int 21H	
end	End

Program 5.4.11 : Write a program in assembly language of 8086 to sort the given 10 numbers(8-bit) from a block in ascending order. **Lab Experiment**

Program :

Label	Instruction	Comment
	.MODEL SMALL	
	.DATA	Initialize the data segment
	NUM DB 01H, 02H, 12H, 54H, 00H, 70H, 15H, 45H, 11H, 25H	
	count db 0AH	
	.CODE	Initialize code segment
	mov ax, @data	Initialize data segment (DS) register to point to data segment
	mov ds, ax	
	mov cl,count	Initialize outer loop count
	mov ch,00h	
	dec cx	One less iteration than the number of elements
outer:	mov bx,cx	Initialize inner counter
	lea si,NUM	Load SI to point to the array
inner:	mov al,[si]	Load a data in accumulator
	inc si	Increment the pointer
	cmp al,[si]	Compare consecutive locations
	jc over	if the previous value is smaller, no swapping required and hence branch to label over
	mov dl,al	Swap the contents of two memory location if the previous is greater
	mov al,[si]	
	mov [si],al	
	mov [si-1],al	
over:	loop inner	Repeat for next consecutive memory locations
	dec bx	Decrement outer counter
	jnz outer	Repeat all comparisons if counter is not zero
	mov ah, 4ch	Terminate Program
	int 21h	
	end	End

Program 5.4.12 : Write a program in assembly language of 8086 to sort the given 10 numbers(8-bit) from a block in descending order.

MU - May 15, 10 Marks, Lab Experiment

Program :

Label	Instruction	Comment
	.MODEL SMALL	
	.DATA	Initialize the data segment
	NUM DB 01H, 02H, 12H, 54H, 00H, 70H, 15H, 45H, 11H, 25H	
	count db 0AH	
	.CODE	Initialize code segment
	mov ax, @data	Initialize data segment (DS) register to point to data segment
	mov ds, ax	
	mov cl,count	Initialize outer loop count
	mov ch,00h	
	dec cx	One less iteration than the number of elements
outer:	mov bx,cx	Initialize inner counter
	lea si,NUM	Load SI to point to the array
inner:	mov al,[si]	Load a data in accumulator
	inc si	Increment the pointer
	cmp al,[si]	Compare consecutive locations
	jnc over	if the previous value is greater, no swapping required and hence branch to label over
	mov dl,al	Swap the contents of two memory location if the previous is greater
	mov al,[si]	
	mov [si],al	
	mov [si-1],al	
over:	loop inner	Repeat for next consecutive memory locations
	dec bx	Decrement outer counter
	jnz outer	Repeat all comparisons if counter is not zero
	mov ah, 4ch	Terminate Program
	int 21h	
	end	End

Program 5.4.13 : Write an assembly language program to exchange the blocks of 1KB located at 01000H and 02000H using string instructions.

MU - May 12, Dec. 14, 10 Marks, Lab Experiment

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. We will 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.

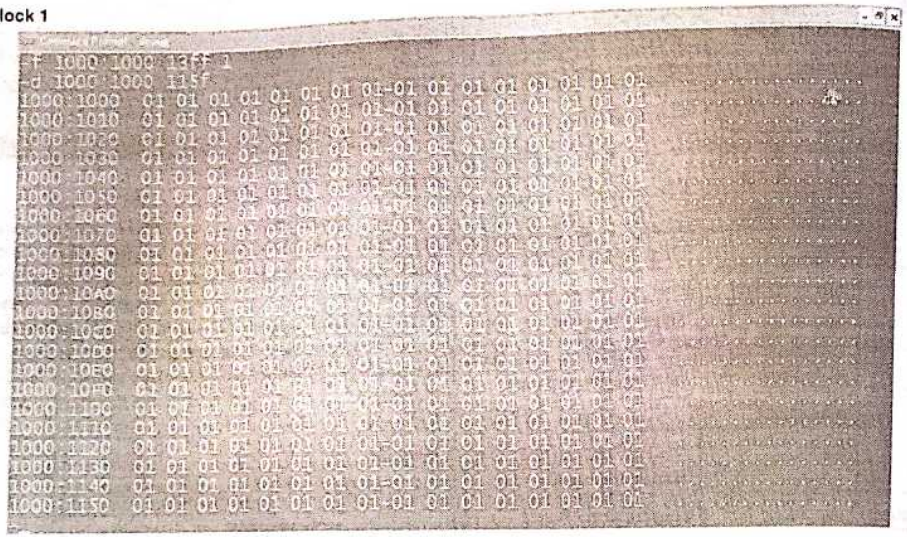
Program :

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

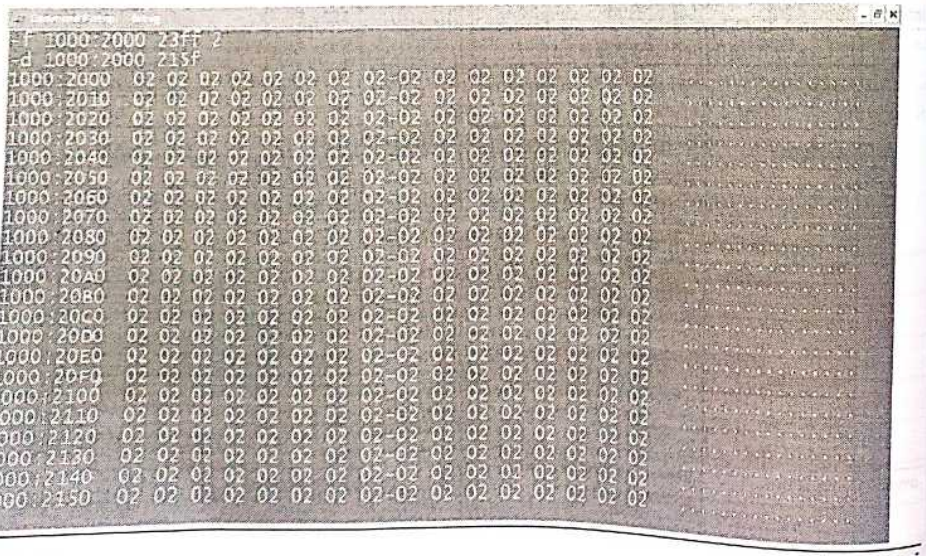
Note : For execution, we have considered different memory locations i.e. 11000H has the first block and 12000H has the second block. This is done as the first segment starting from 00000H has some important data of the computer and we cannot overwrite it. This program has been executed using the tool DEBUG of the DOS. Also since the block size is 1KB i.e. 1024 byte, the screenshots taken are shown for partial data only.

Before Execution

Block 1

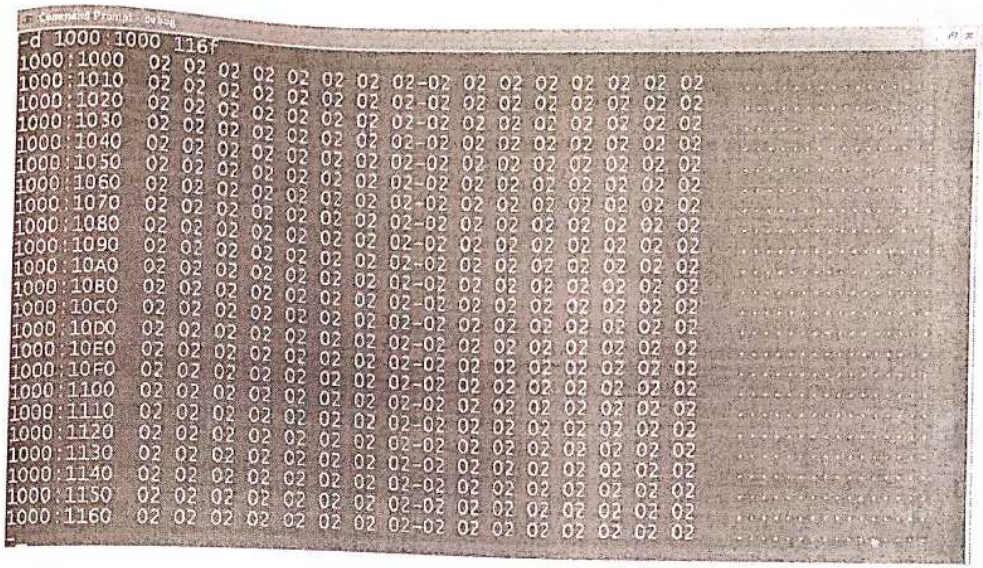


Block 2

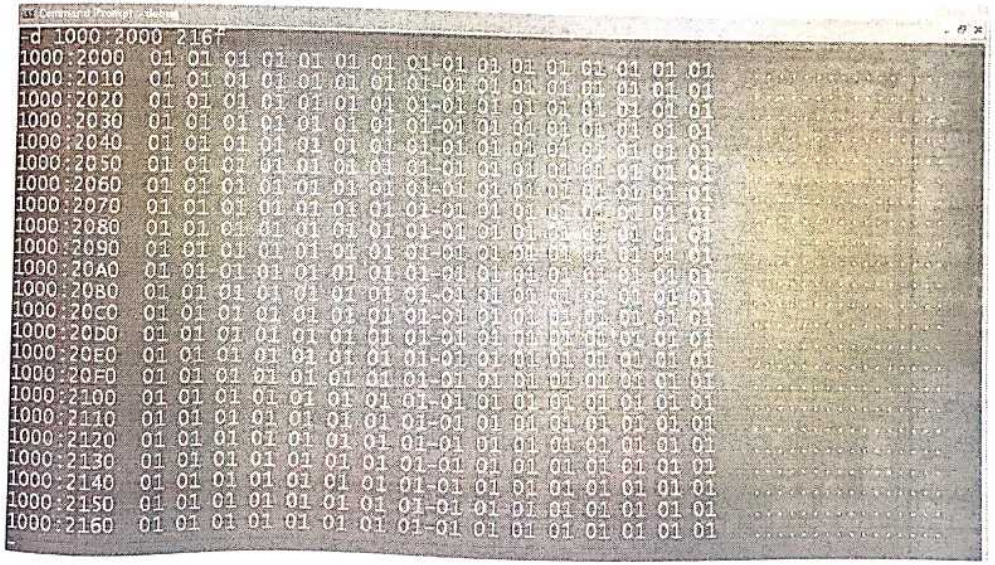


After Execution

Block 1



Block 2



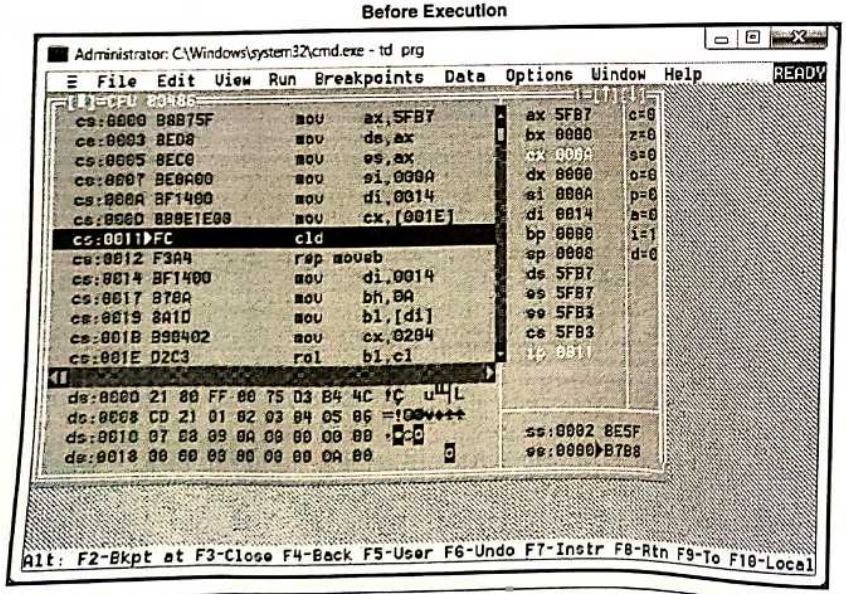
Program 5.4.14 : Write a program to transfer a block of N bytes from source to destination (Non overlapped block transfer).

Program

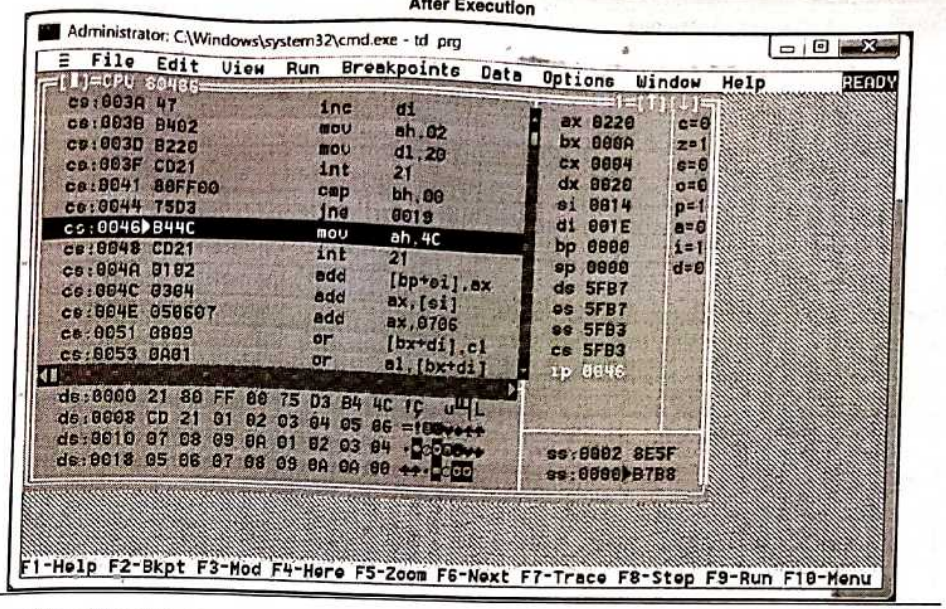
Label	Instruction	Comment
	.model small	
	.data	Initialize data segment
	src_blk db 01, 02, 03, 04, 05, 06, 07, 08, 09, 0AH	
	dest_blk db 10 dup(?)	
	count dw 0AH	
	.code	Initialize code segment
	mov ax, @data	Initialize data & extra segment registers
	mov ds, ax	
	mov es, ax	
	mov si, offset src_blk	Initialise SI to point to source block
	mov di, offset dest_blk	Initialise DI to point to destination block
	mov cx, count	initialize counter
	cld	df=0
again :	rep movsb	transfer contents
	mov ah, 4ch	normal termination to dos
	int 21h	
	end	

Result

Before Execution



After Execution



Program 5.4.15 : Write a program to check if the given string is palindrome.

Lab Assignment **MU - May 14, 10 Marks**

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

Program 5.4.16 : Write an assembly language program to subtract 10 digit decimal numbers which are stored at memory locations 10000H and 10005H respectively with least significant digit at first and most significant digit at last. Store the result at memory location 10010H on words with LSD at start and MSD at last.

Label	Instruction
	CLC
	MOV AX, 1000H
	MOV DS, AX
	MOV SI, 0000H
	MOV DI, 0005H

Label	Instruction
	MOV BX, 0010H
	MOV CX, 0005H
again :	MOV AL, [SI]
	SBB AL, [DI]
	MOV [BX], AL
	INC SI
	INC DI
	INC BX
	LOOP agn

Program 5.4.17 : Write an Assembly language program for Intel 8086 processor to perform the division of 2 digit BCD number which is in unpacked form. **MU - Dec. 11, 5 Marks**

Label	Instructions
	.model small
	.data
	d1 dw 0405H
	d2 db 09H
	Q db ?
	R db ?
	.code
START :	MOV AX, @data
	MOV DS, AX
	MOV AX, d1
	MOV BL, d2
	AAD
	DIV BL
	MOV Q, AL
	MOV R, AH
	MOV AH, 4CH
	INT 21H
	end start

Program 5.4.18 : Write an assembly language program for 8086 to transfer the block of 1 KB located at 0100H to 0200H using string instructions. **MU - May 12, Dec. 12, 10 Marks**

Program :

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

Program 5.4.19 : Write a 8086 program to add 4 digitBCD number in AX to the similar in BX maintaining valid BCD result.

Instruction	Comment
.model small	
.data	Initialize data segment
a dw 6789H	Load variable a with data 6789H
b dw 1232H	Load variable b with data 1232H
c dw ?	Variable c for result
car db ?	Variable car for carry
.code	Initialize code segment
mov ax, @data	Initialize data section
mov ds, ax	
mov ax, a	Load number1 in ax
mov bx, b	Load number2 in bx
add al, bl	add numbers and result in al
daa	adjust result to valid BCD number
mov cl,al	Store lower byte of result in cl
mov al,ah	Move one higher byte into al
adc al,bh	Add the higher bytes
daa	adjust result to valid BCD number
mov ch,al	Store higher byte of result in ch
mov c,cx	Store result in variable 'c'
mov al,00h	
adc al,al	Get carry into al
mov car,al	Store carry in the variable car
mov ah, 4cH	Terminate Program
int 21H	
End	End

Program 5.4.20 : Write a program in the assembly language of 8086 to compute the Fibonacci series of N terms.

Explanation :

The Fibonacci series is

0 1 1 2 3 5 8 13 21....

The first two terms are 0 1, the third term is computed as 0 + 1 = 1, fourth term 1 + 1 = 2, fifth term = 1 + 2 = 3. i.e. n^{th} term = $(n-2)^{th}$ term + $(n-1)^{th}$ term. We will find the series say for 10 terms i.e. N = 10. We will initialize count = 10. An array res is initialized, so that the computed terms will be stored in it. SI is initialized to start of res. The first term i.e. 0 is stored at start. SI is incremented and next term is stored i.e. 1.

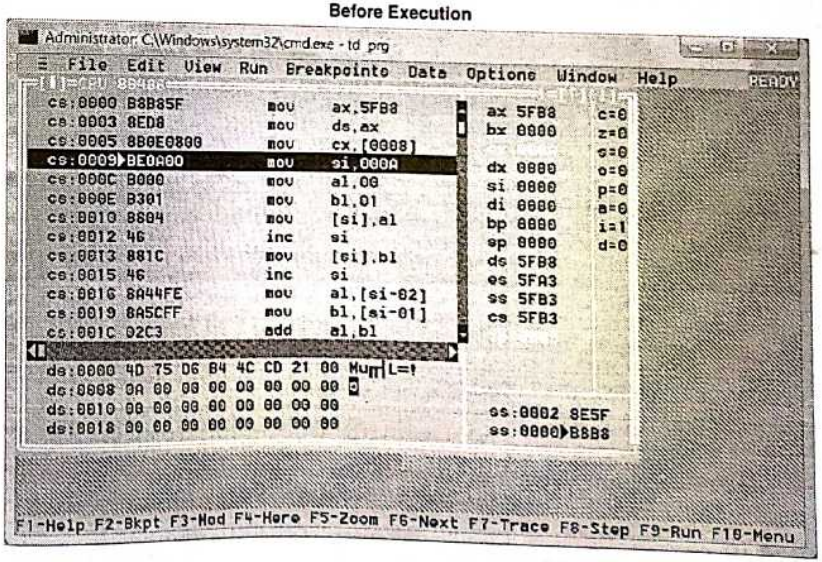
Then addition of contents of two locations i.e. 0 and 1 is done. The next term is computed and result is stored at third location. Process is repeated till all 10 terms are computed. Display the result. The result in Hex will be 0 1 1 2 3 5 8 0D 15 22.

Program

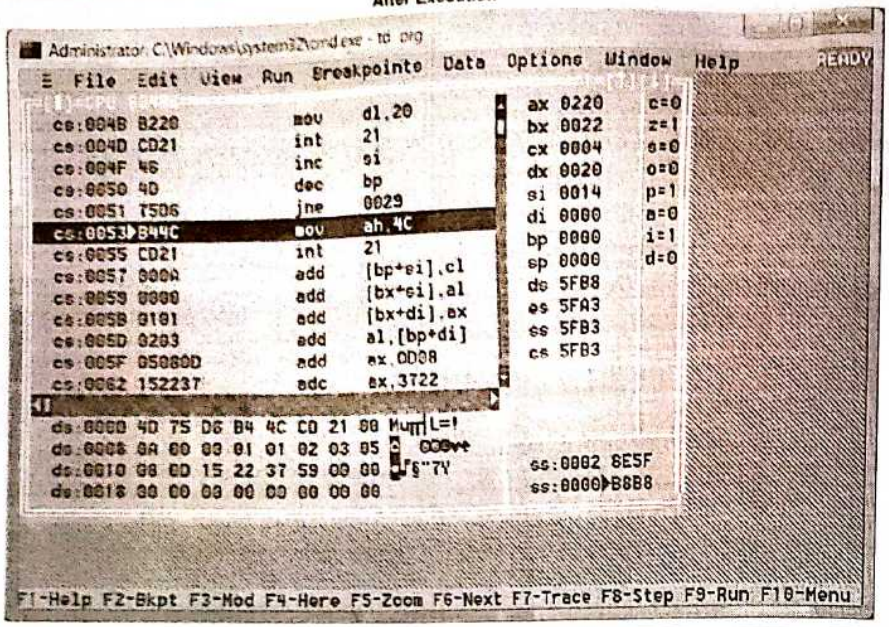
Label	Instruction	Comment
	.model small	
	.data	
	count dw 0AH	count of no. of terms let n=10
	res db 10 dup(?)	
	.code	
	mov ax, @data	initialize data segment
	mov ds, ax	
	mov cx, count	initialize counter
	mov si, offset res	initialize si
	mov al, 00h	initialize al=0 i.e.first term of series
	mov bl, 01h	initialize bl=1 i.e.next term of series
	mov [si], al	store the first term of series
	inc si	increment si to next location
	mov [si], bl	store next term in series
	inc si	increment si to next location
up :	mov al, [si-2]	
	mov bl, [si-1]	
	add al, bl	compute the next term in series
	mov [si], al	store the computed term
	inc si	increment si to next location
	loop up	repeat till all terms are calculated
	mov ah, 4ch	terminate program
	int 21h	
	End	

Result

Before Execution



After Execution



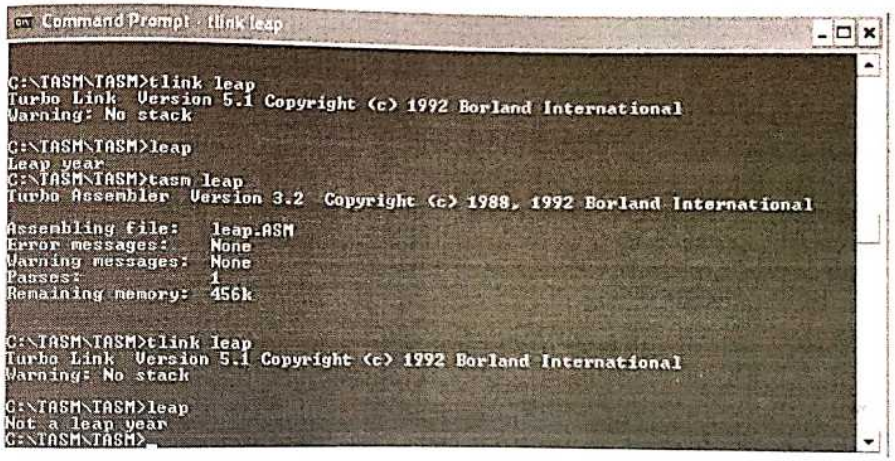
Program 5.4.21 : Write a 8086 assembly language program with appropriate comments, to find if the given year is a leap year or not. **MU - Dec. 19, 10 Marks**

Program

	Instruction	Comment
	.model small	
	.data	Initialize the data segment
	a dw 2021	Load variable 'a' with current year
	b dw 04H	Load variable b with data 08H
	msg1 db "Leap year$"	
	msg2 db "Not a leap year$"	
	.code	Initialize the code segment
Start:	mov ax, @data	Initialize data segment (DS) register to point to data segment
	mov ds, ax	
	mov ax, a	Load year number in ax
	mov dx,00h	Initialize upper word of dividend to zero
	mov bx, b	Load divisor in bl

	Instruction	Comment
	div bx	divide numbers. Quotient will be in al and remainder in ah
	add dx,0000H	Add remainder in AH register with 0
	lea dx,msg1	Initialize pointer DX to message 1
	jz over	If leap year then jump to label over
	lea dx,msg2	Else Initialize pointer to message 2
over:	mov ah,09h	Initialize ah for display routine of interrupt 21H
	int 21h	
	mov ah, 4cH	Terminate Program
	int 21H	
	end	End

Result



5.5 Exam Pack (Review and University Questions)

Q. 1 List and explain assembly language program development tools. **(Refer Section 5.1)** **(10 Marks)**

Q. 2 Explain assembler directives of 8086. **(Refer Section 5.3)** **(May 12, Dec. 12, 10 Marks)**

Q. 3 Write an Assembly language program for Intel 8086 processor to perform the division of 2 digit BCD number which is in unpacked form. **(Refer Program 5.4.17)** **(Dec. 11, 5 Marks)**

Q. 4 Write an assembly language program for 8086 to exchange the blocks of 1 kB located at 0100 H and 0200 H using string instructions. **(Refer Program 5.4.18)** **(May 12 , Dec.12, 10 Marks)**

Q. 5 Write an assembly language program to find whether given word is palindrome or not. **(Refer Program 5.4.15)** **(May 14, 10 Marks)**

Q. 6 Write assembly language program for 8086 to exchange contents of two memory blocks. **(Refer Program 5.4.13)** **(May 12, Dec. 14, 10 Marks)**

Q. 7 Write assembly language program for 8086 to reverse a string of 10 characters. **(Refer Program 5.4.12)** **(May 15, 10 Marks)**

Q. 8 Write a 8086 assembly language program with appropriate comments, to find if the given year is a leap year or not. **(Refer Program 5.4.21)** **(Dec. 19, 10 Marks)**

QQQ

MODULE 2

6 Stacks and Subroutines

6.1 Introduction to Stack

- The stack is a reserved area of the memory where temporary information may be stored. A n-bit stack pointer is used to hold the address of the most recent stack entry. This location which has the most recent entry is called as the **top of the stack.**
- When the information is written on the stack, the operation is called **PUSH.** When the information is read from the stack, the operation is called **POP.** The stack works on the principle of **Last in First Out or First in Last Out.**
- The microprocessor stores the information/data like stacking books. Fig. 6.1.1 shows stacked books. If we want to remove the first stack book, then we have to remove all the books above the first i.e. we have to remove the top book, second from top book, third from top book and so on, then finally the first book at the bottom can be removed. This indicates that the first book pushed onto the stack is the last one to be popped from the stack. This operation is called as **First In Last Out (FILO).**

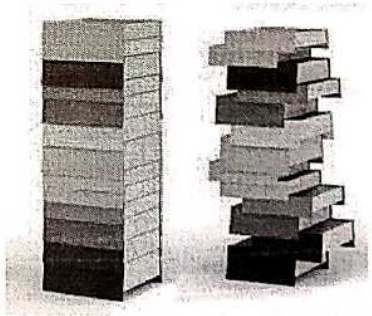


Fig. 6.1.1 : Stacked books

- The stack is implemented with the help of special memory pointer register called as the **stack pointer.** The stack pointer's contents are automatically adjusted to point to the stack.
- When data is to be stored or pushed on the stack, the SP increments before storing the data on stack and when the data is to be retrieved/popped from the stack the SP decrements to point next available byte of stored data.
- The stack array can reside anywhere in the on-chip memory.

Hence in short,
(a) Stack is used to store information (data) temporarily.
(b) Stack is a part of memory defined in program by THE USER.
(c) STACK, the name itself, informs about the way data is stored i.e. stack of data.

6.2 Procedures

Q. Explain procedures with example. **(10 Marks)**
Q. State basic requirements for calling a procedure with example. **(5 Marks)**

- Whenever we need to use a group of instructions several times throughout a program there are two ways we can avoid having to these instructions each time we want to use them by using procedure or macro.
- It is similar to functions or methods of high level language.
- A procedure or subroutine is invoked by the CALL instruction. Calls can be intrasegment (NEAR) and intersegment (FAR), and direct (address in instruction) or indirect (address in memory).
- The other instruction required with a procedure is the RET or return.
- Two assembler directives are used, the PROC and ENDP directives are used to begin and end the procedure. For example:

```
proc_1    proc  near
  ...
code for procedure
...
ret
proc_1    endp
```

- A near CALL pushes the IP onto the STACK before changing the IP to start executing the procedure. The RET restores the IP so the program can pick up from where it left off, i.e., the instruction right after the CALL.
- The far CALL pushes both CS and IP onto the STACK and the RET restores them to their values before the call (assuming the SP is pointing at the IP on the stack when the RET is executed).

6.3 Parameter Passing

There are several ways of passing information or parameters to a procedure.

1. Put the parameters in certain registers.
2. Put the parameters named memory locations. Uses memory locations.
3. Use the STACK. This also uses memory, but only temporarily.

- Since the 8086 has a limited set of registers, parameter passing on the stack is widely used, especially with compiled code.
- The primary instructions for dealing with the stack are the PUSH and POP instructions. These instructions only have word operands and manipulate the stack pointer.
- The PUSH instruction first decrements the SP by two before pushing the data, i.e., SP <-- SP - 2 and push word to SS:SP.
- The POP instruction first pops the word to the desired location and then increments SP by 2, i.e. push the word and then SP <--- SP + 2.

Program 6.3.1 : Addition of Two Arrays using Procedure.

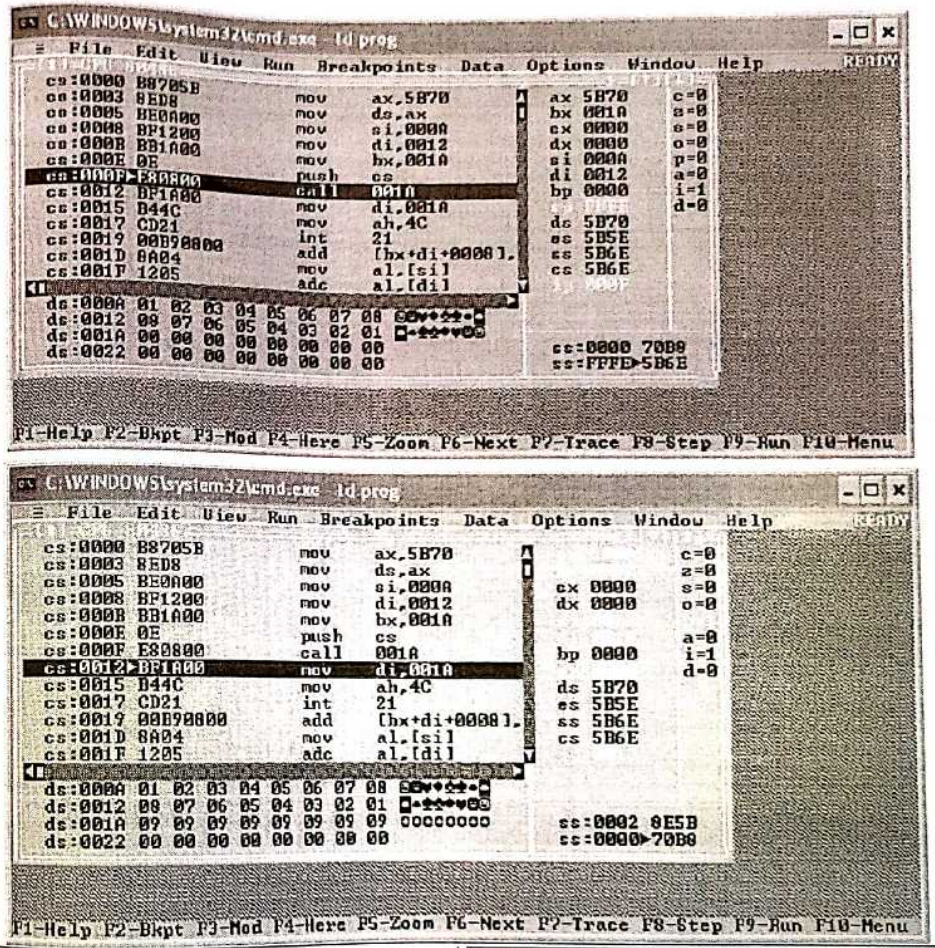
Program

Label	Instruction	Comment
	.model small	
	.data	Initialize data segment
	public arr1, arr2, arr3	declaration of data
	arr1 db 01, 02, 03, 04, 05, 06, 07, 08	
	arr2 db 08, 07, 06, 05, 04, 03, 02, 01	
	arr3 db 8 dup(?)	declared space for result
	.code	Initialize code segment
	mov ax, @data	initialize data segment register
	mov ds, ax	
	mov si, offset arr1	Initialize si to point to arr1
	mov di, offset arr2	Initialize di to point to arr2
	mov bx, offset arr3	bx to point to arr3
	extrn proadd:far	procedure in other file
	call proadd	call procedure
	mov ah, 4ch	Terminate program
	int 21h	
	end	End

Far procedure for addition of numbers in different arrays :

Label	Instruction	Comment
	.model small	
	.data	
	extrn arr1, arr2, arr3	declaring that variables are in other file
	.code	
	public proadd	making this procedure available to other file proadd proc
	mov cx, 08h	initialize counter=08
next:	mov al, [si]	get operand1 from arr1
	adc al, [di]	add=operand1+operand2+carry
	mov [bx], al	save result in location pointed by bx
	inc si	get next operand from arr1
	inc di	get next operand from arr2
	inc bx	point to next location from arr3
	loop next	repeat till all bytes are added
	Ret	
	proadd endp	end procedure
	end	end program

Result



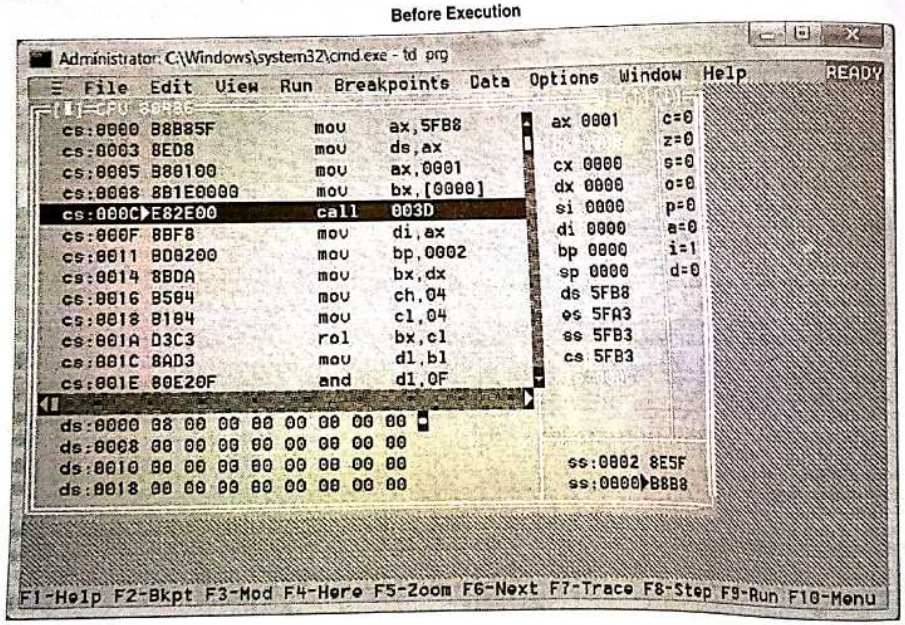
Program 6.3.2 : Write a program to find the factorial of a number using procedure. **MU - Dec. 18, 5 Marks**

Label	Instruction	Comment
	.model small	
	.data	
	num dw 08h	
	.code	
	mov ax, @data	initialize data segment
	mov ds, ax	
	mov ax, 01	initialise ax=1
	mov bx, num	load the number in ex
	call fact	call procedure
	mov di, ax	store the lsb of result in di

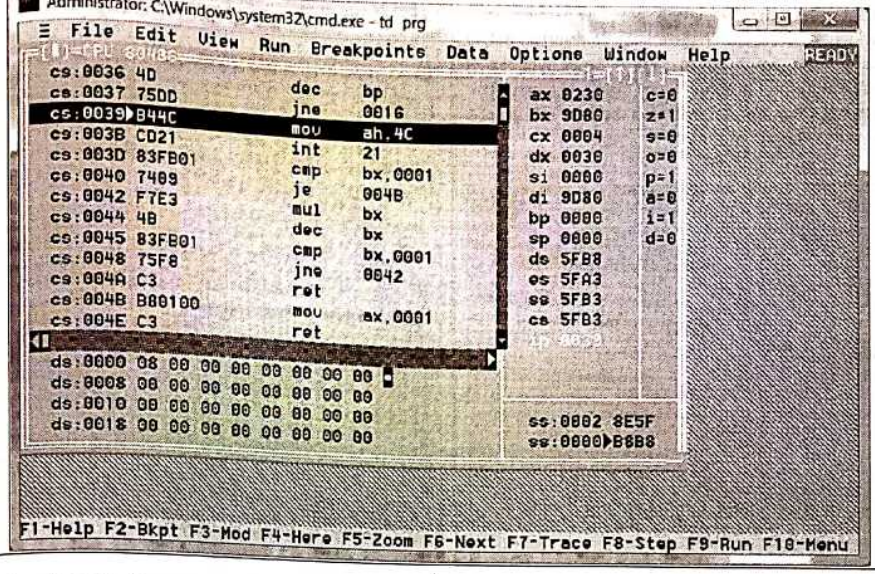
Label	Instruction	Comment
	mov bp, 2	initialise count for no of times display is called
	mov bx, dx	store msb of result in reg bx
	mov bx, di	store lsb of result in bx
	dec bp	decrement bp
	mov ah, 4ch	
	int 21h	
	factproc near	function for finding the factorial
	cmp bx, 01	is bx=1?
	jz l11	if yes, ax=1
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

Result

Before Execution



After Execution



6.3.1 Passing Parameters to and from Procedures

Procedures are written to process data or address variables from the main program. To achieve this, it is necessary to pass the information about address, variables or data. This technique is called as parameter passing. The four major ways of passing parameters to and from a procedure are :

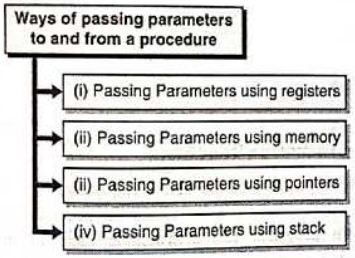


Fig. 6.3.1 : Ways at Passing Parameters to End From Procedure

(I) Passing Parameters using registers :

The data to be passed is stored in the registers and these registers are accessed in the procedure to process the data.

e.g. : .model small

```
.data.
MULTIPLICAND DW 1234 H
MULTIPLIER DW 4232 H
.code
MOV AX, MULTIPLICAND
MOV BX, MULTIPLIER
CALL MULTI
:
:
MULTI PROC NEAR
MUL BX ; Procedure to access data from BX register.
RET
MULTI ENDP
:
:
END
```

The disadvantage of using registers to pass parameters is that the number of registers limits the number of parameters you can pass.

e.g. : an array of 100 elements can't be passed to a procedure using registers.

(II) Passing parameters using memory

In the cases where we have to pass few parameters to and from a procedure registers are convenient. But, in cases when we need to pass a large number of parameters to procedure we use memory. This memory may be a dedicated section of general memory or a part of it.

e.g. : .model small

```
.data.
MULTIPLICAND DW 1234 H      ; Storage for
                              Multiplicand value
MULTIPLIER DW 4232 H        ; Storage for Multiplier
                              value.
MULTIPLICATION DW ?         ; Storage for
                              Multiplication
                            ; result.
.code
MOV AX, @data
MOV DS, AX
:
:
CALLMULTI
:
:
MULTI PROC NEAR.
MOV AX, MULTIPLICAND.
MOV BX, MULTIPLIER.
:
:
MOV MULTIPLICATION, AX  ;  Store the Multiplication
                        ;  value in named memory
                        ;  location.
RET
MULTI ENDP
END
```

(III) Passing Parameters using Pointers

A parameter passing method which overcomes the disadvantage of using data item names (i.e. variable names) directly in a procedure is to use registers to pass the procedure pointers to the desired data.

e.g.

```
.model small
.data
MULTIPLICAND DB 12 H ;  Storage  for  Multiplicand
value.
MULTIPLIER DB 42 H    ; Storage for Multiplier value.
MULTIPLICATION  DW ? ; Storage for Multiplication
                       result.
.code
MOV AX, @data
MOV DS, AX
MOV SI, OFFSET MULTIPLICAND
MOV DI, OFFSET MULTIPLIER
MOV BX, OFFSET MULTIPLICATON
CALLMULTI
:
:
MULTI PROCNEAR
:
:
MOV AL, [SI]      ; Get Multiplicand value pointed by
                    ; SI in accumulator.
MOV BL, [DI]      ; Get Multiplier value pointed by DI in
BL.
    :
    :
MOV [BX], AX      ; Store result in location pointed out by
BX.
RET
MULTI ENDP
END
```

(Iv) Passing Parameters Using Stack

In order to pass the parameters using stack we push them on the stack before the call for the procedure in the main program. The instructions used in the procedures read these parameters from the stack. Whenever stack is used to pass parameters it is important to keep a track of what is pushed on the stack and what is popped off the stack in the main program.

e.g. :

```
.model small
.data
MULTIPLICAND DW 1234H
MULTIPLIER DW 4232H.
.code
MOV AX, @data
MOV DS, AX.
:
:
PUSH MULTIPLICAND.
PUSH MULTIPLIER
CALL MULTI
:
:
MULTI PROCNEAR
PUSH BP                    ;
MOV BP, SP                 ; copies offset of SP into BP.
MOV AX, [BP + 6]           ; MULTIPLICAND value is
                           ; available at
                           ; [BP + 6] and is passed to AX.
MUL WORD PTR [BP + 4] ; MULTIPLIER value is passed.
POP BP
RET                        ; Increments SP by 4 to Return
                           ; Address
MULTI ENDP                 ; End procedure.
END
```

6.3.2 Re-entrant and Recursive Procedures

There are two kinds of procedures :

(1) Re-entrant procedure
(2) Recursive procedure.

(1) Re-entrant Procedure :

Fig. 6.3.2 shows execution flow for a reentrant procedure.

While executing a program it may happen that a procedure 1 is called from main program, procedure 2 is called from procedure 1 and procedure 1 is again called from procedure 2. The program reenters the procedure 1. This type of procedure is called as re-entrant procedure.

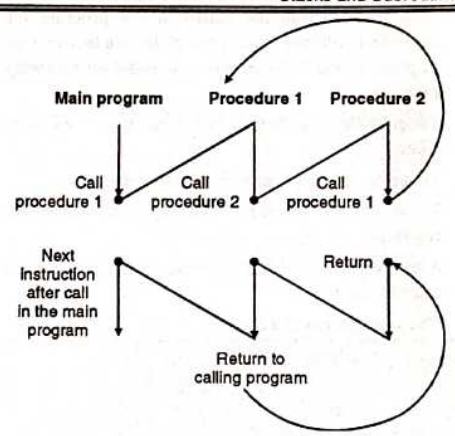


Fig. 6.3.2 : Program execution for re-entrant procedure

(2) Recursive Procedure :

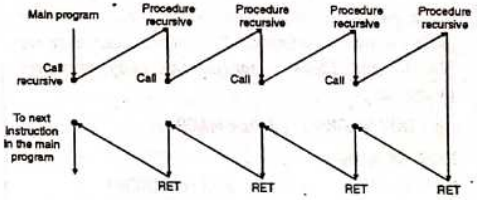


Fig. 6.3.3 : Flow diagram for recursive procedure with N = 4

- Fig. 6.3.3 shows flow diagram for N = 4.
- A procedure which calls itself is called a recursive procedure. They are used when we work with complex data structures called trees. The recursive loop takes a long time to execute and occupies huge stack space.
- For this type of procedures we normally define N i.e. the recursive depth. If N = 4, N is decremented by 1 after each procedure. Call the procedure until N = 0.

6.4 Macros

Q. Write short note on : Macros with example. (5 Marks)

- When the repeated group of instructions is too short not appropriate to be written as a procedure, we use a macro. A macro is a group of instructions that perform a task.]

- Each time we 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.
- To simplify and reduce the amount of repetitive coding.
- To reduce the errors caused by repetitive coding.
- To make the assembly language program more readable.
- 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.

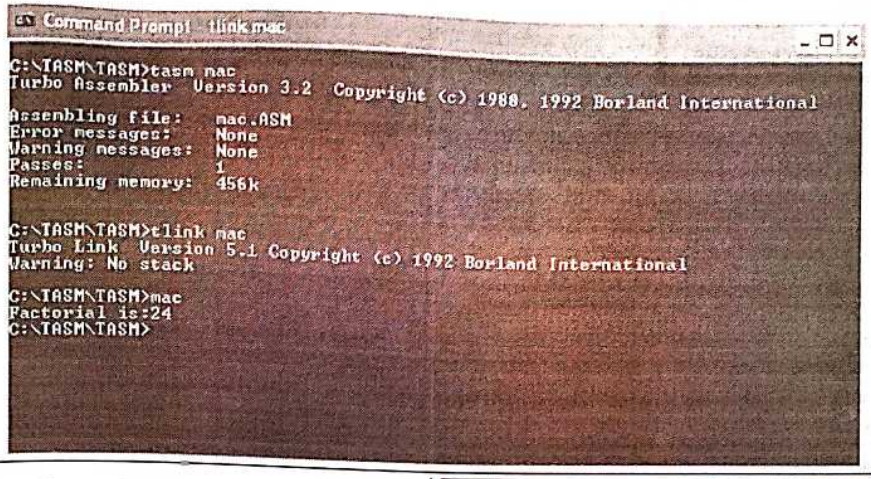
Program 6.4.1 : Assembly program using macro.

Soln. :

Program

Label	Instruction	Comments
	Write a program to find the factorial of a number using macro	
	.model small	
	.data	
	msg db "Factorial is:$"	
	num db 4	
	.code	
	fact macro	Macro for calculating factorial
	mov bl,num	Load the number in register bl
Start :	cmp bl,01h	Compare the number with 01
	jz over	If the number is 01H, then calculation done
	mul bl	If the number is not 01H, then multiply the number with the contents of register al and store the result in register ax (register al works as the variable fact)
	dec bl	Decrement the number
	jmp agn	Repeat the multiplication with the variable fact, until the number becomes 1
Over :	endm	End of macro
Start :	mov ax,@data	Initialize data segment
	mov ds,ax	
	mov ax,0001h	Initialize register ax=0001H as the fact variable
	fact	Call the macro
	mov bx,ax	Copy the factorial in register bx
	lea dx,msg	Display the message using INT21H type 09H
	mov ah,09h	
	int 21h	
	mov al,bl	Display the two digits of factorial by separating them. To separate, divide by the decimal number 10 or $(0A)_H$ and separate the quotient and remainder.
	mov ah,00h	
	mov bl,0ah	
	div bl	
	mov dl,al	Display the quotient and remainder after converting them to ASCII form. To convert in ASCII form, add with $(30)_H$.
	mov bl,ah	
	add dl,30h	
	MOV ah,02h	To display, use INT 21H and type 02H
	int 21h	
	mov al,bl	
	add al,30h	
	mov dl,al	
	MOV ah,02h	
	int 21h	
	mov ah, 4ch	
	int 21h	
	end start	

Output



6.4.1 Comparison of Procedure and Macro

University Question

Q. Differentiate procedure and macro.

MU - Dec. 18, May 19 ,5 Marks

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).

6.5 Timing and Delay Loops

Q. Explain software delay and hardware delay. **(5 Marks)**

- The speed at which the instructions in 8086 are executed is determined by a crystal-controlled clock with a frequency of few megahertz. Each instruction takes a fixed number of clock cycles to execute.

e.g. : DAA requires 4 clock cycles, JNZ requires 16 clock cycles.

- If an 8086 is running with a 5 MHz clock, then each clock takes $\left(\frac{1}{5\text{ MHz}}\right) = 0.2\ \mu s$. An instruction which takes 4 clock cycles, will take $4 \times 0.2 = 0.8\ \mu s$ to execute. A common programming problem is to introduce delay between executions of two instructions. E.g. traffic signal controller, real time clock, process control, serial communication.

- There are two types of delay
  (1) Software delay
  (2) Hardware delay

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.

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

Example 1 : 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

$= (\text{count} - 1) \times 16 + 4$

If count = 4

Timer delay = $4 + 4 \times 3 + 3 \times 16 + 4$

$= 4 + 12 + 48 + 4$

$= 68$ T states.

Operating frequency of 8086 = 5 MHz.

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

$\therefore$ Total Time Delay = 68×0.2

Total Time Delay = 13.6 µs.

Example 2 : Implement a timer delay using 16 bit counter.

```
. model small
. stack 100
. code                  T states required
    MOV AX, count       ; Load count          4
Loop 1 :  DEC AX        ; Decrement counter   2
          JNZ loop1     ; Repeat till count = 0  16/4
    end
```

$\therefore$ Time delay = $4 + \text{count} \times 2 + (\text{count} - 1) \times 16 + 4$

If count 3

Time delay = $4 + 3 \times 2 + 2 \times 16 + 4$

$= 4 + 6 + 32 + 4$

$= 46$ T states

8086 operates at 5 MHz.

$\therefore$ Total timer delay = 46×0.2

Timer delay = 9.2 µs.

6.5.1 Disadvantage of Software Delay

In software delay, microprocessor wastes its valuable time doing nothing. If microprocessor time is to be used at its best, hardware delay is preferred.

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.

Ex. 6.5.1 : Write an Assembly Language Program to generate a delay of 1 sec using a microprocessor running at 5 MHz. Also show the delay calculations.

Soln. :

Program :

```
Instruction   Clock cycles required
              MOV BX, Multiplier count
REPE :        MOV CX, Count          ; 4
              DEC CX                 ; 2
              JNZ BACK               ; 16/4
              DEC BX
              JNZ REPE
```


Step I : To calculate the delay generated by inner loop with maximum count (FFFF H).

Delay generated by = $[4 + (65535 - 1) \times (2 + 16) + (2 + 4)] \times 0.2 \ \mu s$

Inner loop for count = 235.9244 msec

FFFF H (i.e. 65535)

Step II : Calculate the multiplier count to get delay of 1 second

Multiplier count = $\frac{\text{Required delay}}{\text{delay provided by inner loop}}$

$= \frac{1 \times 1s}{235.9244 \text{ msec}}$

$= 4.239$

$= 04$ H

Ex. 6.5.2 : Write an 8086 ALP to generate a delay of 100 ms, if 8086 system frequency is 10 MHz.

Soln. :

Program :

```
        Instruction          Clock cycles required
        MOV CX, Count        ; 4
BACK :  DEC CX               ; 2
        JNZ BACK             ; 16/4
```

Step I : To calculate the number of required clock cycles.

Clock cycles required = $\frac{\text{Required delay time}}{\text{Time for 1 clock cycle}} = \frac{100 \text{ ms}}{0.1 \ \mu s}$

$= 1000000.$

Step II : To find the required count.

Count = $\frac{\text{Number of clock cycles required} - 4 - (2 + 4)}{\text{execution time for one loop}} + 1$

Count = $\frac{1000000 - 4 - 6}{(16 + 2)} + 1$

Count = $(55556)_{10}$

Count = D904 H

6.6 Mixed Language Programming using Assembly and C

University Questions

Q. Write short note on : mixed mode programming.
MU - May 14, Dec. 18, May 19, 5 Marks

Q. Write short note on : Mixed language programming.
MU - Dec. 15, Dec. 16, Dec. 19, 5 Marks

- '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 :
  - 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.

Combining C and assembly :

There are 2 ways of combining C and Assembly language.

Method 1 :

- Here 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.

Case 1 : You can prefix the keyword asm for and assembly instruction in a 'C or C++' program.

Lab Experiment

Example :

```
#include<iostream.h>
void main()
{
 int a, b, c;
 cout<<"Enter two numbers";
 cin>>a>>b;
 asm mov ax,a;
 asm mov bx,b;
 asm add ax,bx;
 asm mov c,ax;
 cout<<"The sum is "<<c;
}
```

Case 2 : You can prefix the keyword asm for a function and write assembly instruction in the curly braces in a 'C or C++' program.

Example :

```
#include<iostream.h>
void main()
{
 int a, b, c;
 cout<<"Enter two numbers";
 cin>>a>>b;
 asm{
 mov ax,a;
 mov bx,b;
 add ax,bx;
 mov c,ax;
 }
 cout<<"The sum is "<<c;
}
```

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.
- E.g.: 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, we 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. 6.6.1 below shows Compile, Assemble, and link processes using C compiler, MASM Assembler and TLINK.

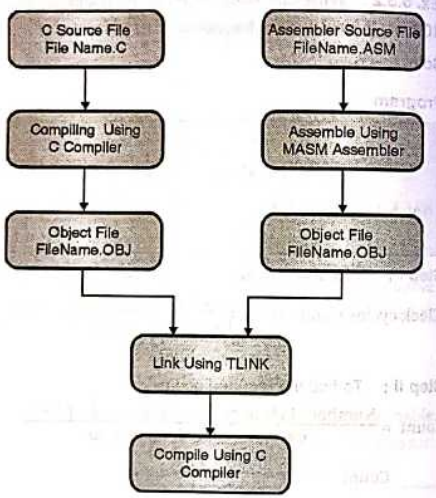


Fig. 6.6.1

6.7 Programming based on DOS and BIOS Interrputs (INT 21H,INT 10H)

- **INT 10h or INT 10H** is shorthand for BIOS interrupt call 10. The 17th interrupt vector in an x86-based computer system.
- The BIOS typically sets up a real mode interrupt handler at this vector that provides video services. Such services include setting the video mode, character and string output, and graphics primitives (reading and writing pixels in graphics mode)

- To use this call, load AH with the number of the desired sub function, load other required parameters in other registers, and make the call. INT 10h is fairly slow; so many programs bypass this BIOS routine and access the display hardware directly.
- Setting the video mode, which is done infrequently, can be accomplished by using the BIOS, while drawing graphics on the screen in a game needs to be done quickly, so direct access to video RAM is more appropriate than making a BIOS call for every pixel.

Function	Function code	Parameters	Return
Set video mode	AH=00h	AL = video mode	AL = video mode flag / CRT controller mode byte
Set text-mode cursor shape	AH=01h	CH = Scan Row Start, CL = Scan Row End Normally a character cell has 8 scan lines, 0-7. So, CX=0607h is a normal underline cursor, CX=0007h is a full-block cursor. If bit 5 of CH is set, that often means "Hide cursor". So CX=2607h is an invisible cursor. Some video cards have 16 scan lines, 00h-0Fh. Some video cards don't use bit 5 of CH. With these, make Start>End (e.g. CX=0706h)	
Set cursor position	AH=02h	BH = Page Number, DH = Row, DL = Column	
Get cursor position and size	AH=03h	BH = Page Number	AX = 0, CH = Start scan line, CL = End scan line, DH = Row, DL = Column
Read light pen position (Does not work on VGAsystems)	AH=04h		AH = Status (0=not triggered, 1=triggered), BX = Pixel X, CH = Pixel Y, CX = Pixel line number for modes 0Fh-10h, DH = Character Y, DL = Character X
Select active display page	AH=05h	AL = Page Number	
Function	Function code	Parameters	Return
Scroll up window	AH=06h	AL = lines to scroll (0 = clear, CH, CL, DH, DL are used), BH = Background Color and Foreground color. BH = 43h, means that background color is red and foreground color is cyan. Refer the BIOS color attributes CH = Upper row number, CL = Left column number, DH = Lower row number, DL = Right column number	

Function	Function code	Parameters	Return
Scroll down window	AH=07h	like above	
Read character and attribute at cursor position	AH=08h	BH = Page Number	AH = Color, AL = Character
Write character and attribute at cursor position	AH=09h	AL = Character, BH = Page Number, BL = Color, CX = Number of times to print character	
Write character only at cursor position	AH=0Ah	AL = Character, BH = Page Number, CX = Number of times to print character	
Set background/border color	AH=0Bh, BH = 00h	BL = Background/Border color (border only in text modes)	
Set palette	AH=0Bh, BH = 01h	BL = Palette ID (was only valid in CGA, but newer cards support it in many or all graphics modes)	
Write graphics pixel	AH=0Ch	AL = Color, BH = Page Number, CX = x, DX = y	
Read graphics pixel	AH=0Dh	BH = Page Number, CX = x, DX = y	AL = Color
Teletype output	AH=0Eh	AL = Character, BH = Page Number, BL = Color (only in graphic mode)	
Get current video mode	AH=0Fh		AL = Video Mode
Write string (EGA+, meaning PC ATminimum)	AH=13h	AL = Write mode, BH = Page Number, BL = Color, CX = String length, DH = Row, DL = Column, ES:BP = Offset of string	

- **INT 21H** Most of the general functions and services offered by DOS are implemented through this interrupt.The functions available are well standardised and should be common to all MSDOS, PCDOS and DOS Plus systems. Well behaved programs, therefore, should use these facilities in preference to any other methods available for the widest range of compatibility.
- INT 21h in the 512's implementation of DOS Plus 2.1 provides 77 official functions, two of which are non-functional and return with no action. Within this range some calls have sub functions which further extend the range of operations.
- In all calls, on entry AH defines the function. Other parameters may also be required in other registers. Where a memory block is used by the call this is specified in the normal segment:offset form. In all cases the general programming technique is to set AH to the function pointer, set up the required register contents (and the memory block if necessary) then to issue the call by the assembly code INT instruction.

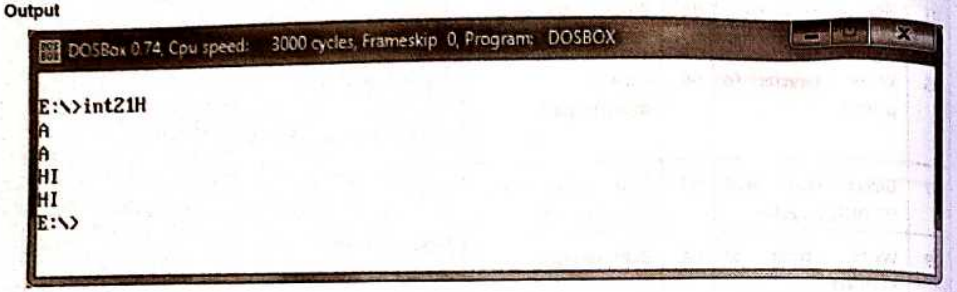
AH	Description	AH	Description
01	Read character from STDIN	02	Write character to STDOUT
05	Write character to printer	06	Console Input/Output
07	Direct char read (STDIN), no echo	08	Char read from STDIN, no echo
09	Write string to STDOUT	0A	Buffered input
0B	Get STDIN status	0C	Flush buffer for STDIN
0D	Disk reset	0E	Select default drive
19	Get current default drive	25	Set interrupt vector
2A	Get system date	2B	Set system date
2C	Get system time	2D	Set system time
2E	Set verify flag	30	Get DOS version
35	Get Interrupt vector		
36	Get free disk space	39	Create subdirectory
3A	Remove subdirectory	3B	Set working directory
3C	Create file	3D	Open file
3E	Close file	3F	Read file
40	Write file	41	Delete file
42	Seek file	43	Get/Set file attributes
47	Get current directory	4C	Exit program
4D	Get return code	54	Get verify flag
56	Rename file	57	Get/Set file date

Program

```
;int 21 type2 display char
;       type 1 read char
;       type 09  write string
;       type 0a read string
```

```
.model small
.data

ch1 db 0h
newl db 0ah,0dh,'$'

str1 db 20 dup(20,'$')
.code
start:

mov ax,@data
mov ds,ax

mov ah,01h
int 21h
mov ch1,al

lea dx,newl
mov ah,09h
int 21h
mov dl,ch1
mov ah,02h
int 21h

lea dx,newl
mov ah,09h
int 21h

lea dx,str1
mov ah,0ah
int 21h

lea dx,newl
mov ah,09h
int 21h

lea dx,str1+2   ; 0th posn stores the size of string declared
                ; and 1st posn stores the size of
                ; string inputted by user
mov ah,09h
int 21h

mov ah,4ch
int 21h
end start
```

Output



6.8 Exam Pack (Review and University Questions)

Q. 1	Explain procedures with example. **(Refer Section 6.2)**	**(10 Marks)**
Q. 2	State basic requirements for calling a procedure with example. **(Refer Section 6.2)**	**(5 Marks)**
Q. 3	Write short note on : Macros with example. **(Refer Section 6.4)**	**(5 Marks)**
Q. 4	Differentiate procedure and macro. **(Refer Section 6.4.1)**	**(Dec. 18, May 19, 5 Marks)**
Q. 5	Explain software delay and hardware delay. **(Refer Section 6.5)**	**(5 Marks)**
Q. 6	Write short note on : mixed mode programming. **(Refer Section 6.6)**	**(May 14, Dec. 18, 5 Marks)**
Q. 7	Write short note on : Mixed language programming. **(Refer Section 6.6)**	**(Dec. 15, Dec. 16, Dec. 19, 5 Marks)**

□□□

MODULE 1

7 8086 Interrupt Structure

7.1 8086 Interrupt Structure

University Questions

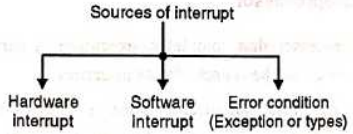
Q. Explain the interrupt structure of the 8086. **MU - May 11, May 17, Dec. 18, 10 Marks**

Q. Difference between Software and Hardware interrupts. **MU - Dec. 11, 5 Marks**

Q. Explain interrupts of 8086 in detail. **MU - May 12, May 13, Dec. 13, May 14, 10 Marks**

Q. Explain the following: Types of interrupts **MU - May 19, 5 Marks**

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)

- We know that 8086 supports division, multiplication, addition etc. Suppose by mistake 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"**. (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. 7.1.1 shows interrupt sources for 8086.
- As shown in Fig. 7.1.1 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.

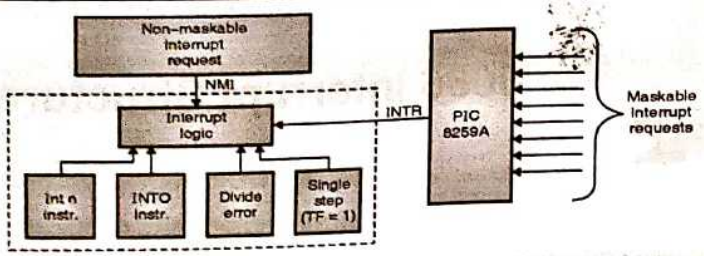


Fig. 7.1.1 : Interrupt sources (INTO-Interrupt on Overflow)

Sr. No.	Hardware Interrupts	Software Interrupts
1.	To implement a hardware interrupt a pin is given on the processor	To implement a software interrupt an instruction is given in the instruction set of the processor.
2.	Hardware interrupts are mostly maskable or non-maskable interrupts	Software interrupts are mostly non-maskable although may have lower priority.
3.	Hardware interrupts may be vectored or non-vectored	Software interrupts are mostly vectored.
4.	Hardware interrupts of 8086 are NMI and INTR	Software interrupts of 8086 are INT n (where n is any value from 00H to FFH)

7.2 Software Interrupts

- The INT instruction of the 8086 can be used to do one of the 256 interrupts (Type 0-255).
- The interrupt type is specified by the number as a part of the instruction. e.g. INT 0 instruction can be used to send execution to a divide by zero interrupt service routine.
- With the help of these software interrupts we can call the routines from different programs in the system e.g. BIOS. The BIOS routines are called with INT instructions.

7.3 Interrupt Service Routines

Q. What is an ISR ? How does 8086 acknowledge an interrupt ? Draw flowchart for interrupt processing sequence. (5 Marks)

- 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.

7.3.1 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. 7.3.1).

(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.

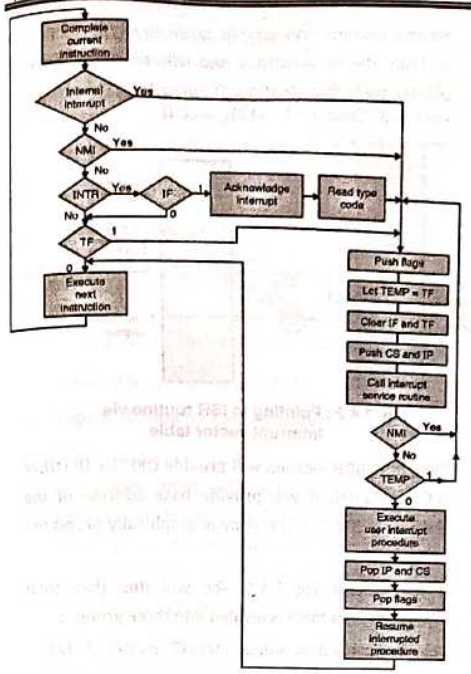


Fig. 7.3.1 : Interrupt processing sequence/ priority sequence

(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).

- 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. 7.3.2.

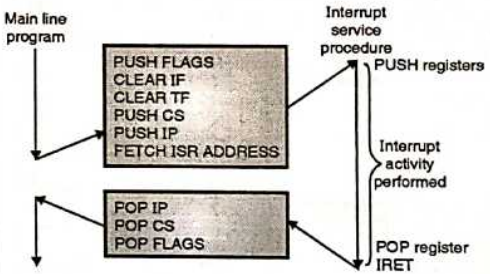


Fig. 7.3.2 : 8086 Interrupt response

- Going through the full discussion of interrupt response, you came across **"Vector table"** OR **"Interrupt vector table"**. So let's have a look at this table to see how it looks like.

7.4 Interrupt Vector Table (IVT)

Q. Explain Interrupt Vector Table (IVT) with diagram. (5 Marks)

- 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.
- In Section 7.3, point 10 (e), we specified that microprocessor receives NEW CS and NEW IP value from vector table, after calling internal service routine.

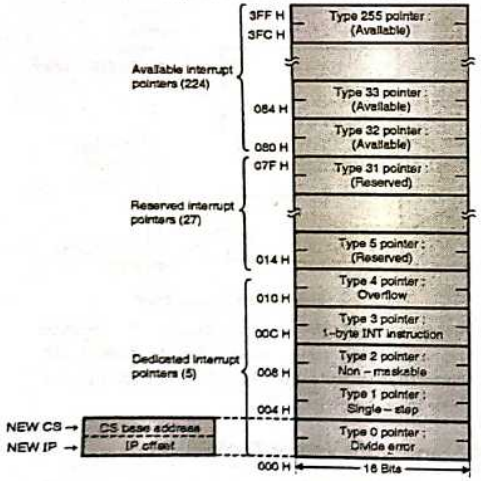


Fig. 7.4.1 : Interrupt vector table

- *This point is referred back here because, 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.

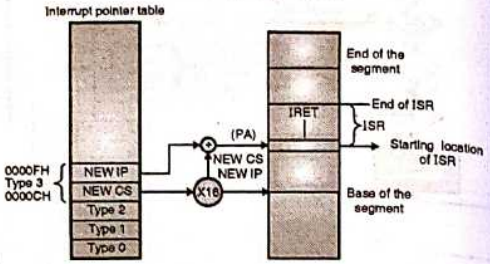


Fig. 7.4.2 : Pointing to ISR routine via interrupt vector table

- Then 0CH/0DH location will provide OFFSET IP (NEW IP) and 0EH/0FH will provide base address of the segment (NEW CS). The same is graphically presented in Fig. 7.4.2.
- If you observe Fig. 7.4.1, you will find that, total 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. Now let's start our study with dedicated interrupt pointers.

7.5 Dedicated Interrupts

University Questions

Q. Explain type 0, 1, 2 interrupts found in the interrupt vector table of 8086 microprocessor ? **(10 Marks)**

Q. Discuss the functions of the pre-defined/dedicated interrupts. **MU - May 11, 5 Marks**

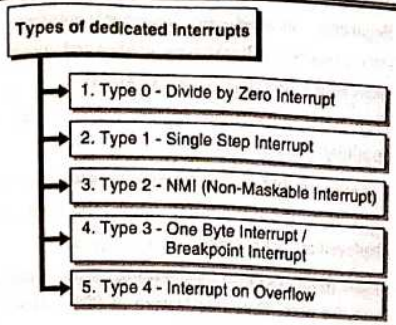


Fig. 7.5.1 : Types of dedicated interrupts

Under this, we have type 0 to 4.

1. Type 0 - Divide by Zero Interrupt

- 8086 supports division (unsigned / signed), instruction. 8086 will automatically do a type 0 interrupt if the result of DIV or IDIV operation is too large to fit in destination register. When type 0 interrupt is internally generated, microprocessor will :
  (a) Push flag register.
  (b) Reset TF and IF.
  (c) Push CS and IP (i.e. return address).
  (d) Get NEW CS and NEW IP. For this, microprocessor takes type no, i.e. '0', multiply by 4. Therefore we get 0 × 4 = 000 H. So microprocessor gets, NEW IP from 00000H / 00001H location and NEW CS from 00002H/00003H location.
  (e) NEW CS and NEW IP will be loaded into the CS and IP register. Thus we get branching to ISR routine.
  (f) After returning from ISR, microprocessor will pop CS and IP (OLD CS/OLD IP). Microprocessor will also pop flag register.
- Here important point regarding Type 0, is, it is automatic and cannot be disabled anyway i.e. non-maskable. User has to keep a account of it in the program where he/she uses DIV/IDIV instruction.
- Normally user will write an interrupt service procedure which takes desired action when an invalid division occurs.
- To avoid this interrupt, user can check, before division, that divisor is not zero.

2. Type 1 - Single Step Interrupt

- We use debug utility to debug the program. In that we can **"Single step"** the program. You must have found this utility very useful for the beginner.
- You must have observed that when you tell a system to single step, it will execute one instruction and stop.
- One can examine the contents of memory / register if required, else execute next instruction. The conclusion is, in this mode, a system will stop after it executes each instruction and wait for further direction from user.
- The 8086 trap flag and type 1 interrupt makes it easy to implement a single step feature.

How to set trap flag ?

8086 has no instructions to directly set/reset trap flag. Therefore to initiate single stepping follow next steps :

(1) Copy the flags onto the stack.
(2) Setting the TF bit on the stack.
(3) Popping the flag.

The same is achieved by following instructions :

```
    PUSHF          ; Push flag register
MOV   BP, SP       ; Use BP as a pointer as it is stack
                     segment
OR WORD PTR [BP + 0], 0100 H   ; Set TF bit
    POPF           ; Restore flag register
```

One more way we can have is,

```
    PUSHF                      ; Push flag register
    MOVSI, SP                  ; Read SI pointer
    OR  Word PTR [SI], 0100 H  ; Set TF = 1
    POPF                       ; Pop flag register
```

- After, TF = 1, microprocessor will automatically do a Type 1 interrupt after each instruction executes. The interrupt sequence saves the flags and program counter, then resets TF flag to allow the single step routine to execute normally.
- To return to the routine under test, an interrupt return restores the IP, CS and flags with TF set. This allows the execution of next instruction in the program under test before trapping back to the single step routine.
- Note that single step is not masked by the IF bit in the flag register.
- Locations used by type 1 interrupt are 0004H / 5H / 6H / 7H.

3. Type 2 - NMI (Non-Maskable Interrupt)

- This is the highest priority hardware interrupt and is non maskable. The input is edge triggered, but is synchronized with the CPU clock and must be active for two clock cycles to generate recognition.
- The interrupt signal may be removed prior to entry to the service routine. Since the input must make a LOW to HIGH transition to generate an interrupt, **"spurious"** transition on the input should be suppressed.
- If the input is normally HIGH, the NMI low time to guarantee triggering is two CPU clock times.
- When type 2 (NMI) is generated, in response to the same microprocessor executes steps which are same as that listed in section 7.3 point 10 (a) to (j).
- The only point to be added is, the location in interrupt pointer table.

  Type no. × 4 = Location in interrupt pointer table.

  2 × 4 = 00008 H
- Thus from 00008H/9H microprocessor gets NEW IP and from 0000AH/BH microprocessor gets NEW CS.
- Basically NMI interrupt input is used for catastrophic failures, for example power failure, time out of system watchdog timer.

4. Type 3 - One Byte Interrupt / Breakpoint Interrupt

- This type is invoked by a special form of the software interrupt instruction which requires a single byte of code space i.e. CCH (INT 3). This interrupt is primarily used as a breakpoint interrupt for software debug. When you insert a breakpoint in your program, the system executes instructions upto the breakpoint and then goes to the breakpoint procedure.
- When user informs debugger program to insert breakpoint at some point in your program, they actually do it by temporarily replacing the instruction byte at that address with CC H, i.e. code for INT 3 instruction.
- Thus this single byte instruction can be mapped into the smallest instruction for absolute resolution in setting breakpoints.
- The steps taken by microprocessor are same as that taken in section 7.3 point 10 (a) to (j).
- Regarding location in interrupt pointer table, microprocessor will take type number and multiply the same with 4, therefore 3 × 4 = 12 = 0C H.
- Then from 0000CH/DH we get NEW IP and from 0000EH/FH we get NEW CS.
- A breakpoint ISR routine usually saves all the register contents on the stack. If user wants, the same can be displayed on CRT screen.
- At this point system is now waiting for next command from the user. Thus this feature of 8086 allows fast debugging, i.e. wait or stop at a point suspicious by the programmer, else continue.

5. Type 4 - Interrupt on Overflow

- This interrupt occurs if the overflow flag (OF) is set in the flag register. The OF flag is set if the signed result of an arithmetic operation on two signed number is too large to be represented in destination register or memory location. Thus this interrupt is used to capture overflow errors.
- One more way of branching to this interrupt is INTO (Interrupt on Overflow). The action taken by microprocessor in response to type 4 and INTO is same, which is already given in section, 7.3, point 10 (a) to (j). For type 4/INTO, we get NEW IP from location 10H/11H and NEW CS from 12H/13H. This interrupt is non maskable.
- Interrupt types 0 and 2 can occur without specific action by the programmer (except for performing a divide for Type 0).
- While types 1, 3 and 4 require a conscious act by the programmer or generate these interrupt types. All, but type 2 are invoked through software activity and are directly associated with a specific instruction.

7.6 Hardware Interrupts

Q. How does 8086 respond to an INTR signal ? (5 Marks)

Q Write short note on : Hardware Interrupts. (5 Marks)

- 8086 provides this hardware pin so that some external signal can interrupt program sequence executed by 8086. Let's start INTR discussion in question - answer form.

Can we mask INTR ? If yes, how ?

INTR is maskable interrupt, so that INTR cannot cause an interrupt. Masking is achieved by forcing IF = 0 in flag register. IF can be cleared at anytime using CLI (Clear Interrupt Instruction).

What we have to do to unmask INTR ?

To unmask INTR, set IF = 1. The same is achieved by STI (Set Interrupt Instruction).

What will be the status of IF flag, when 8086 is resetted ? Why ?

When 8086 is resetted IF flag is automatically cleared. To allow 8086 to service INTR you have to use STI instruction. IF is cleared at power on reset or reset by user, just because, before starting actual operation 8086 has to perform following jobs i.e. :

(a) Initialize peripherals 8255, 8155, 8279 etc.
(b) Initialize pointers
(c) Initialize counters
(d) Initialize variables etc.

So it is expected that unless and until **"initialisation"** is over 8086 SHOULD NOT service INTR interrupt. In short, by not enabling INTR we are allowing 8086 to **"get ready"** to service INTR, 8086 will take some time to get ready therefore during that part, IF = 0.

How INTR signal is sensed by 8086 ? Is the instruction of 8086 affects the sensing of INTR signal ?

- During the last clock cycle of each instruction, the state of the INTR pin is sampled.
- The 8086 deviates from this rule when the instruction is MOV or POP to a segment register.
- For this case, the interrupts are not sampled until completion of the following instruction.
- This allows a 32 bit pointer to be loaded to the stack pointer registers SS and SP without the danger of an interrupt occurring between the two loads. Another exception is WAIT instruction which waits for a low active input on the $\overline{TEST}$ pin.
- This instruction also continuously samples the interrupt request during its execution and allows servicing interrupts during the wait.
- When an interrupt is detected, the WAIT instruction is again fetched prior to servicing the interrupt to guarantee the interrupt routine will return to the WAIT instruction.
- Also, since prefixes are considered part of the instruction they precede, the 8086 will not sample the interrupt line until completion of the instruction, the prefix (es) precede(s).
- An exception to this (other than HLT or WAIT) is the string primitives preceded by the Repeat (REP) prefix. The repeated string operations will sample the interrupt line at the completion of each repetition.
- This includes repeat string operations and also includes the lock prefix. If multiple prefix precedes a repeated string operation and the instruction is interrupted, only the prefix immediately preceding the string primitive is restored.

When 8086 responds to interrupt, it clears IF flag. Why ?

IF is cleared because of two reasons :

(a) It prevents a signal on the INTR input from interrupting a higher priority interrupt service procedure in progress.
(b) IF is made **'0'**, so that signal on INTR input does not cause 8086 to interrupt itself continuously INTR is active HIGH. So when INTR = 1, 8086 is interrupted. If INTR is not disabled during the first response, the 8086 would be continuously interrupted and would never get to the actual interrupt service routine.

Suppose user wants, another INTR input to be able to interrupt as interrupt procedure in progress, what should be done ?

Simple, use STI command to make IF = 1, to allow sensing of another interrupt on INTR line.

How 8086 responds, to INTR signal ? Do we need any extra hardware for the same ?

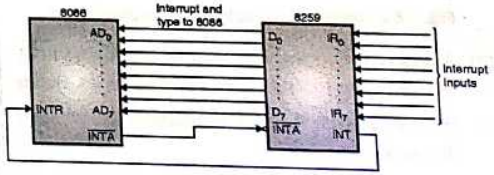


Fig. 7.6.1 : Interfacing 8259 to 8086

- 8086 response to INTR is somewhat different than the response to other interrupts. The main difference is, for INTR, interrupt type number is sent to 8086 from an external hardware.
- Thus we, do need external hardware. The most widely used device is 8259 (PIC). Refer Fig. 7.6.1 for interfacing.

Now we will study the interrupt acknowledge machine cycles.

(1) When 8259 receives an interrupt signal on one of its IR (Interrupt request) inputs, it sends an interrupt request signal to INTR input of 8086.

(2) If IF flag is set, then 8086 responds to 8259, which is diagrammatically presented in Fig. 7.6.2.

In response to INTR 8086 executes an interrupt acknowledge sequence. To guarantee the interrupt will be acknowledged, the INTR input must be held active until the interrupt acknowledge is issued by the CPU. If BIU is running a bus cycle when the interrupt condition is detected (as would occur if the BIU is fetching an instruction when the current instruction completes), the interrupt must be valid at 8086, 2 clock cycles prior to T_4 of the bus cycle if the next cycle is to be an interrupt acknowledge cycle. If the 2 clock setup is not satisfied another pending bus cycle will be executed before the interrupt acknowledge is issued. If a held request is also pending, the interrupt is serviced after the hold request is serviced.

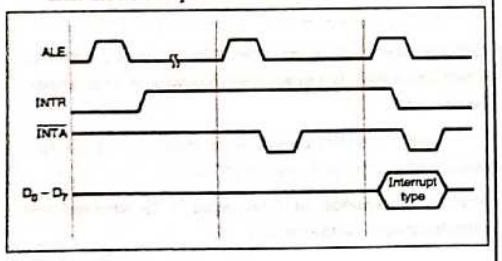


Fig. 7.6.2 : Interrupt acknowledge machine cycle

(3) The interrupt acknowledge cycle consists of two $\overline{INTA}$ bus cycles separated by two idle clock cycles. During the first bus cycle the $\overline{INTA}$ command is issued rather than read.

No address is provided by 8086 during either bus cycle ($\overline{BHE}$ and status are valid), however, ALE is still generated and will load the address latches with indeterminate information.

This condition requires that devices in the system do not drive their outputs without being qualified by the Read Command.

(4) During second $\overline{INTA}$ bus cycle DT/$\overline{R}$ and $\overline{DEN}$ are conditioned to allow the 8086 to receive a one byte interrupt type number from the interrupt system.

(5) The first $\overline{INTA}$ bus cycle signals an interrupt acknowledge cycle is in progress and allows the system to prepare to present the interrupt type number on the next $\overline{INTA}$ bus cycle. The CPU does not capture information on the bus during the first cycle, instead it will float its bus, AD_0 to AD_{15}.

(6) During the second interrupt acknowledge cycle the 8086 sends out another pulse on its $\overline{INTA}$ output pin.

In response to second pulse on, $\overline{INTA}$ the 8259 puts the interrupt type (number) on the lower eight lines of the data bus, where it is read by the 8086. This implies that devices which present interrupt type numbers to the 8086 must be located on the lower half of the 16 bit data bus.

(7) In the minimum mode system, the M/$\overline{IO}$ signal will be low indicating I/O during the $\overline{INTA}$ bus cycles. The 8086 internal $\overline{LOCK}$ signal will be active from T_2 of the first bus cycle until T_2 of the second to prevent the BIU from honouring a hold request between the two $\overline{INTA}$ cycles.

(8) Once the 8086 has the interrupt type number (from the bus for hardware interrupts, from the instruction stream for software interrupts or from the predefined condition), the type number is multiplied by four to form the displacement to the corresponding interrupt vector in the interrupt vector table. The four bytes of the interrupt vector are, for CS and IP register.

(9) During the transfer of control, the CPU pushes the flags, current code segment register and instruction pointer onto the stack. The new code segment and the instruction pointer values are loaded and the single step and interrupt flags are reset. Resetting the interrupt flag disables response to further hardware interrupts in the service routine unless the flags are specifically reenabled by the service routine. The CS and IP values are read from the interrupt vector table with data read cycles.

No segment registers are used when referencing the vector table during the interrupt context switch. The vector displacement is added to ZERO to form the 20 bit physical address. $S_4S_3 = (10)_2$ indicating code segment or none.

(10) The number of clock cycles from the end of the instruction during which the interrupt occurred to the start of interrupt routine execution is 61 clock cycles. For software generated interrupts, the sequence of the bus cycles is the same except no interrupt acknowledge bus cycles are executed. This reduces the delay to service routine execution to 51 clocks for "INT" instruction and single step, 52 clocks for INT3 and 53 clocks for INTO. If wait states are inserted by either the memories or the device supplying the interrupt type number, the given clock times will increase accordingly.

7.7 Interrupt Procedure

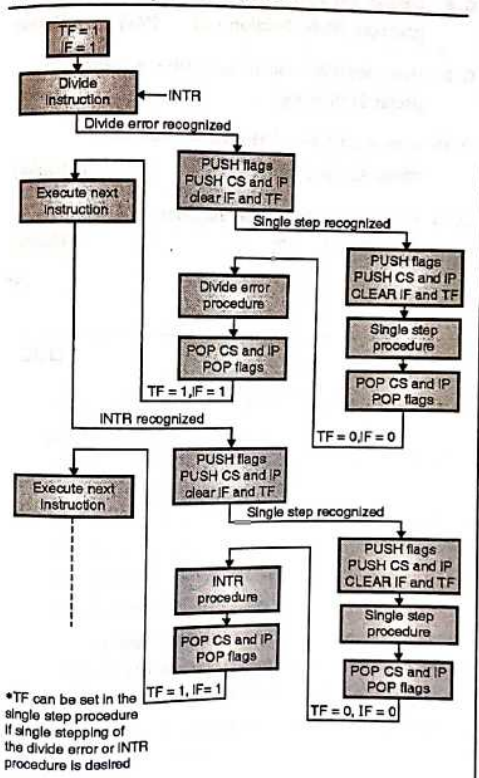


Fig. 7.7.1 : Processing simultaneous interrupts

- When an interrupt service procedure is entered, the flags, CS and IP are pushed onto the stack and TF and IF are cleared. The procedure may enable external interrupts with the STI (set interrupt-enable flag) instruction, thus allowing itself to be interrupted by a request on INTR. (Note, however, that interrupts are not actually enabled until the instruction *following* STI has executed).
- An interrupt procedure always may be interrupted by a request arriving on NMI. Software or processor-initiated interrupts occurring within the procedure also will interrupt the procedure. Care must be taken in interrupt procedures that the type of interrupt being serviced by the procedure does not itself inadvertently occur within the procedure.
- For example, an attempt to divide by 0 in the divide error (Type 0) interrupt procedure may result in the procedure being reentered endlessly. Enough stack space must be available to accommodate the maximum depth of interrupt nesting that can occur in the system.

7.8 Priority of 8086 Interrupts

Q. How does 8086 decide the priority of interrupts ? (5 Marks)

Here we have favorite question that, what will happen if two or more interrupt occur at the same time ? The answer is higher priority interrupt will be serviced and then next highest priority interrupt will be serviced. So let's know about priority of 8086 interrupt. Following Table 7.8.1 depicts the same.

Table 7.8.1

Interrupt	Priority
NMI	Highest
Divide Error, INT n, INTO	Higher
INTR	Lower
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 (will be serviced first) 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.

7.9 Exam Pack (Review and University Questions)

Q. 1 Explain the interrupt structure of the 8086.
(Refer Section 7.1)
(May 11, May 17, Dec.18, 10 Marks)

Q. 2 Difference between Software and Hardware interrupts. **(Refer Section 7.1)** **(Dec. 11, 5 Marks)**

Q. 3 Explain Interrupts of 8086 in detail.
(Refer Section 7.1)
(May 12, May 13, Dec. 13, May 14, 10 Marks)

Q. 4 Explain the following: Types of Interrupts.
(Refer Section 7.1) **(May 19 , 5 Marks)**

Q. 5 What is an ISR ? How does 8086 acknowledge an interrupt ? Draw flowchart for interrupt processing sequence. **(Refer Section 7.3)** **(5 Marks)**

Q. 6 Explain Interrupt Vector Table (IVT) with diagram.
(Refer Section 7.4) **(5 Marks)**

Q. 7 Explain type 0, 1, 2 interrupts found in the interrupt vector table of 8086 microprocessor ?
(Refer Section 7.5) **(10 Marks)**

Q. 8 Discuss the functions of the pre-defined/dedicated interrupts. **(Refer Section 7.5)** **(May 11, 5 Marks)**

Q. 9 How does 8086 respond to an INTR signal ?
(Refer Section 7.6) **(5 Marks)**

Q. 10 Write short note on : Hardware Interrupts.
(Refer Section 7.6) **(5 Marks)**

Q. 11 How does 8086 decide the priority of interrupts ?
(Refer Section 7.8) **(5 Marks)**

□□□

IC8259 Programmable Interrupt Controller

8.1 Polling and Interrupts

- Whenever more than one I/O devices are connected to a microprocessor based system, any one of the I/O devices may ask service at any time. There are two methods in which the microprocessor can service these I/O devices. One method is to use the **polling routine**, while the other method employs **interrupt**.
- In the polling routine the microprocessor checks whether any of the I/O devices is requesting for service.
- The polling routine is a simple program that keeps a check for the occurrences of interrupt. For e.g. : Let us assume that our polling routine is servicing I/O ports 1, 2, 3,.....8. The polling routine will check the status of the I/O ports in a proper sequence.
- The polling routine will first transfer the status of the I/O port 1 to the accumulator. It then checks the contents of accumulator to determine if the service request bit is set. If the bit is set then I/O port 1 service routine is called, otherwise the polling routine will move forward to check if port 2 is requesting service.
- On completion of the service to port 1, the polling routine will test port 2. The process is repeated till all the 8 ports are tested and all the I/O ports those are demanding service are processed. On completion of the polling routine, the microprocessor will resume with the execution of the program. Fig. 8.1.1 shows the sequence for polling routine.
- The polling routine has priorities assigned to the different I/O devices. Once the routine begins port 1 will always be checked first, then port 2 and so on.
- Another way that allows the microprocessor stop with the execution of the program and give service to the I/O devices is **interrupt**.
- It is an external asynchronous input that informs the microprocessor to complete the instruction that it is currently executing and fetch a new routine in order to offer service to the I/O device. Once the I/O device is serviced, the microprocessor will continue with the execution of its normal program.

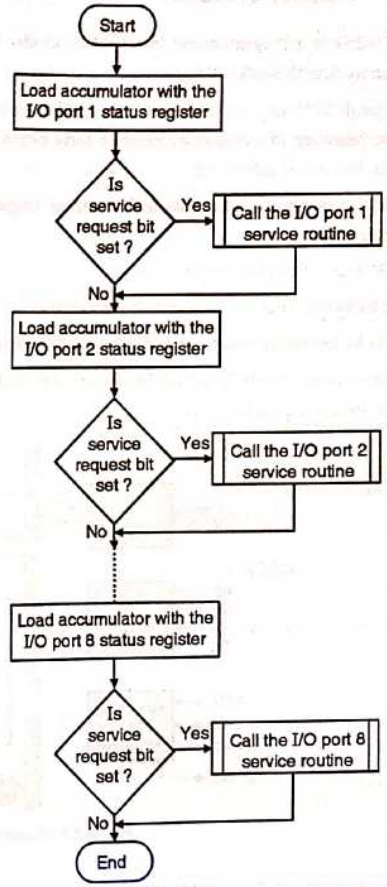


Fig. 8.1.1 : Polling sequence

8.2 8259A Programmable Interrupt Controller

- For applications where we require multiple interrupt sources, we need to use an external device called as a priority interrupt controller (PIC).
- By connecting a PIC to the microprocessor we can increase the interrupt handling capacity of the microprocessor.
- 8259A is the commonly used priority interrupt controller.
- It also helps giving interrupt type to the processor for non-vectored interrupt of 8086 i.e. INTR.

8.2.1 Features of 8259A

- IC 8259 is a Programmable Interrupt Controller that can work with 8085, 8086 etc.
- A single 8259 can handle 8 interrupts while a cascaded configuration of 1 master 8259 and 8 slave 8259s can handle upto 64 interrupts.
- 8259 can handle edge as well as level triggered interrupts.
- 8259 has a flexible priority structure.
- In 8259 interrupts can be masked individually.
- The Vector address of the interrupts is programmable.
- Status of interrupts (pending, in-service, and masked) can be easily read by the µP.

8.3 8259 Block Diagram

University Questions

Q. Explain the operation of IC 8259 with the block diagram. Explain all the signals in detail. **MU - May 11, Dec. 11, 10 Marks**

Q. Explain the operation of IC 8259 with block diagram. **MU - May 12, Dec. 12, Dec. 19, 10 Marks**

Q. Draw and explain block diagram of 8259 PIC. **MU - Dec. 14, 10 Marks**

Q. Write short notes on : 8259-PIC. **MU - May 16, 5 Marks**

The block diagram of 8259 is as shown in Fig. 8.3.1. It contains following blocks :

(1) Data bus buffer
(2) Read/write logic
(3) Cascade buffer and comparator
(4) Control logic
(5) IRR (Interrupt Request Register)
(6) InSR (In-Service Register)
(7) Priority resolver
(8) IMR (Interrupt Mask Register)

The architecture of 8259 can be divided into the following parts:

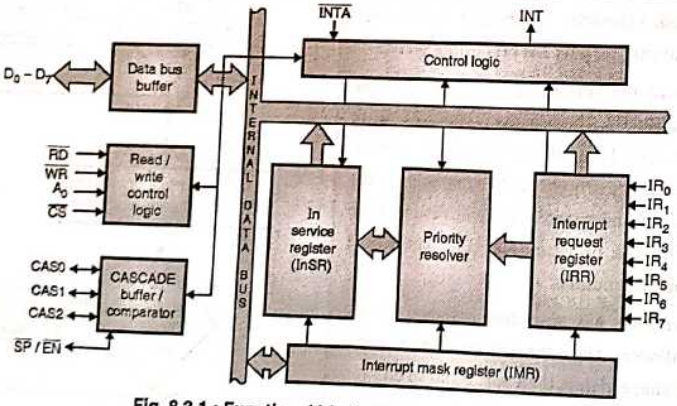


Fig. 8.3.1 : Functional block diagram of 8259

1. Interrupt Request Register (IRR)

- The IRR is an 8-bit register having one bit for each of the interrupt lines IR7 ... IR0.
- When an interrupt request occurs on any of these lines, the corresponding bit is set in the Interrupt Request Register (IRR).

2. In-Service Register (InSR)

It is an 8-bit register, which stores the information about the Interrupt Request, which is currently being serviced.

3. Interrupt Mask Register (IMR)

- It is an 8-bit register, which stores the masking information for each interrupt IR0 to IR7.
- This is written by the programmer.

4. Priority Resolver

It examines the IRR, InSR, and IMR and determines which interrupt is of highest priority and should be sent to the µP.

5. Control Logic

- It has INT output signal connected to the INTR of the µP, to send the Interrupt to the µP.
- It also has the INTA input connected to the INTA of the µP, to receive the interrupt acknowledge.
- It is also used to control the remaining blocks.

6. Data Bus Buffer

- It is a bi-directional buffer used to interface the internal data bus of 8259 with the external (system) data bus.
- It is used to send control and read status.
- It is used by the µP to read the interrupt type from the 8259.

7. Read/Write Logic

- It is used to accept the $\overline{RD}$, $\overline{WR}$, A_0 and $\overline{CS}$ signal.
- It is used to control data flow on the data bus.
- It also holds the Initialization Command Words (ICW's) and the Operational Command Words (OCW's).

8. Cascade Buffer / Comparator

- It is used in cascaded mode of operation.
- It has two components

(i) CAS2, CAS1, CAS0 lines

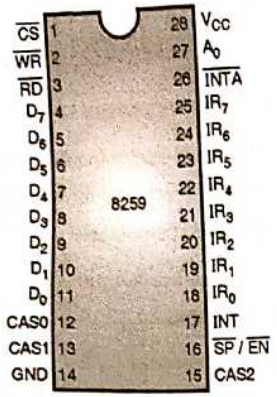
- These lines are output for the master, input for the slave.
- The master sends the address of the slave on these lines
- (hence output).
- The Slaves read the address on these lines (hence input).
- As there are 8 interrupt levels for the Master, there are 3 CAS lines ($2^3 = 8$).

(ii) $\overline{SP}$ / $\overline{EN}$ (Slave Program/Master Enable)

- In Buffered mode, it functions as the $\overline{EN}$ line and is used to enable the buffer.
- In non buffered mode, it functions as the $\overline{SP}$ output line.
- For Master 8259 $\overline{SP}$ should be high, and for the Slave SP should be low.

8.4 Pin Configuration of 8259

The pin configuration of 8259 programmable interrupt controller is as shown in Fig. 8.4.1



$D_0 - D_7$	I/O	Data bus
$\overline{RD}$	I	I/O read
$\overline{WR}$	I	I/O write
$\overline{CS}$	I	Chip select
INT	O	Interrupt request (To 8085)
$\overline{INTA}$	I	Interrupt acknowledge (From 8085)
A_0	I	Address line
$IR_0 - IR_7$	I	Interrupt request (To 8259)
$\overline{SP} / \overline{EN}$	I/O	Slave program / Enable buffer
CAS0 – CAS2	I/O	Cascade lines

Fig. 8.4.1 : Pin diagram of 8259

Table 8.4.1. Pin Description

Symbol	Pin No.	Type	Name and Function
Vcc	28	I	**SUPPLY :** + 5V Supply.
GND	14	I	**GROUND**
$\overline{CS}$	1	I	**CHIP SELECT :** A low on this pin enables $\overline{RD}$ and $\overline{WR}$ communication between the CPU and the 8259A. INTA function are independent of CS
$\overline{WR}$	2	I	**WRIYR :** A low on this pin when CS is low enables the 8259A to accept command words from the CPU
$\overline{RD}$	3	I	**READ :** A low on this when CS is low enables the 8259A to release status onto the data bus for the CPU.
$D_7 - D_0$	4-11	I/O	**BIDIRECTIONAL DATA BUS :** Control, status and interrupt-vector information is transferred via this bus.
$CAS_0 - CAS_2$	12,13,15	I/O	**CASCADE LINES :** The CAS lines form a private 8259A bus to control a multiple 8259A structure. These pins are outputs for a master 8259A and inputs for a slave 8259A.
$\overline{SP} / \overline{EN}$	16	I/O	**SLAVE PROGRAM/ENABLE BUFFER :** This is a dual function pin. When in the Buffered Mode it can be used as an output to control buffer transceivers (EN). When not in the buffered mode it is used as an input to designate a master (SP = 1) or slave (SP = 0)
INT	17	O	**INTERRUPT :** This pin goes high whenever a valid interrupt request is asserted. It is used to interrupt the CPU. Thus it is connected to the CPU's interrupt pin.
$IR_0 - IR_7$	18-25	I	**INTERRUPT REQUESTS :** Asynchronous inputs. An interrupt request is executed by raising an IR input (low to high), or just by a high level on an IR input (Level Triggered Mode).

Symbol	Pin No.	Type	Name and Function
$\overline{INTA}$	26	I	**INTERRUPT ACKHOWLEDGE :** This pin is used to enable 8259A interrupt-vector data onto the data bus by a sequence of interrupt acknowledge pulses issued by the CPU.
A_0	27	I	**AO ADDRESS LINE :** This pin acts in conjunction with the $\overline{CS}$, $\overline{WR}$ and $\overline{RD}$ pins. It is used by the 8259A to decipher various Command Words the CPU writes and status the CPU wishes to read. It is typically connected to the CPU A_0 address line (A_1 for 8086,8022)

8.5 Priority Modes

University Question

Q. Explain the fully nested mode of PIC 8259.

MU - Dec. 11, 5 Marks

The various priority modes of 8259A are :

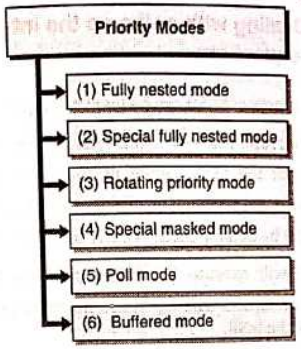


Fig. 8.5.1 : Priority modes

1. Fully Nested Mode (FNM)

- It is the default mode of 8259.
- It is a fixed priority mode.
- IR0 has the highest priority and IR7 has the lowest priority.

2. Special Fully Nested Mode

- This mode can be used for the Master 8259 in a cascaded configuration.
- Its priority structure is fixed and is the same as FNM (IR0 highest and IR7 lowest).
- Additionally, in FNM, the Master would recognize a higher priority interrupt form a slave whose another interrupt is currently being serviced. This is possible only in FSNM.
- In the fully nested mode, on the acknowledgement of an interrupt, the further interrupts of the same level are disabled.
- Consider a large system that uses cascaded 8259s and where the interrupt levels within each slave have to be considered. An interrupt request input to the slavecauses the slave to place an interrupt request to the master and on one of the master's inputs.
- Interrupts to the same slave will cause the slave to place the request to the master on same input to the master. However, these interrupts will not be recognized.
- This is because further interrupts to the same level are disabled by the master as its InSR bit is set.
- The special fully nested mode is used to prevent the problem.

Difference between special nested mode and fully nested mode

(i) Whenever an interrupt request form a slave is being serviced, the slave is allowed to place further requests and if these requests are of a higher priority than the request that is currently being serviced. These interrupts are recognized by the master. It initializes interrupt requests to the microprocessor unit.

(ii) Before termination form the ISR, a non-specific EOI must be sent to the slave. Its InSR must be read to determine if it was the only interrupt to the slave. If the InSR is empty, a non-specific EOI command can be sent to the master. If the InSR is not empty, then the same IR level input to the master can be reserviced. This is because there are multiple interrupts on the slave EOI must not be sent to the master.

3. Rotating Priority Modes

There are two rotating priority modes:

Automatic Rotation and Specific Rotation

(i) Automatic Rotation mode

- This is a rotating priority mode.
- It is preferred when several interrupt sources are of equal priority.
- In this mode, after a device receives service, it gets the lowest priority.
- All other priorities rotate subsequently.

(ii) Specific Rotation Mode

It is also a rotating priority mode, but here the user can fix all priorities.

4. Special Mask Mode

- Usually 8259 disables interrupt requests lower or equal to the interrupt, which is currently in service.
- In SMM 8259 permits interrupts of all levels (lower or higher) except the one currently in service.

5. Poll Mode

- Here the INT line of 8259 is disabled.
- The µP gives the Poll command to the 8259 using OCW3.
- In return, 8259 provides the Poll Word to the µP.
- Thereafter the µP services the interrupt.
- It is preferred when:
  - (i) Subroutine is common for several interrupt levels.
  - (ii) To expand number of interrupt levels more than 64.

6. Buffered Mode

- In this mode $\overline{SP/EN}$ becomes low during $\overline{INTA}$ cycle.
- This signal is used to enable the buffer.

8.5.1 End Of Interrupt (EOI)

- When the µP responds to an interrupt request by sending the first INTA signal, the 8259 sets the corresponding bit in the In Service Register (InSR). This begins the service of the interrupt.
- The µP has to service the interrupt and hence indicate the End of Interrupt(EOI) to 8259.
- There are three different ways/commands to send EOI.

(i) Automatic EOI (AEOI)

- In this mode, no command is necessary.
- During the third INTA cycle, the corresponding bit in the InSR is reset.

(ii) Non-Specific EOI

- This command is sent to 8259 at the end of a service routine.
- It would clear the bit of the currently serviced interrupt in the InSR.

(iii) Specific EOI

Here also a command is sent to 8259 at the end of a service routine, but this command specifies which InSR bit is to be reset.

8.5.2 Dealing with noise on the interrupt request pin

- The IR inputs of 8259 can be level or edge triggered.
- The interrupt request signal should be maintained high, until the falling edge of the first INTA pulse. Otherwise, it is treated as an invalid request (noise), and 8259 by default sends the IR7 level to the µP.
- The µP will execute the ISR corresponding to IR7 wherein it checks whether it is a noise by checking the D_7 bit of the InSR.
- If this bit is Zero, it will not execute the ISR.

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 slave.

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 $\overline{INTA}$ pulse. It can be used only for master and not for slave.

Ex. 8.5.1 : Draw the flow diagram for given following sequence of events, normal priority, and that EOI command must be output :

(i) Request on IR3 (ii) Request on IR2
(iii) Request on IR6 (iv) If reset to 1
(v) If reset to 1 (vi) EOI to clear ISR2
(vii) EOI to clear ISR3
(viii) If reset to 1
(ix) EOI to clear ISR6.

Soln. :

Each time when it branches to an ISR, the processor pushes flag register, clears IF and TF. Also it pushes CS and IP. On IRET instruction it pops CS, IP and flags from the stack.

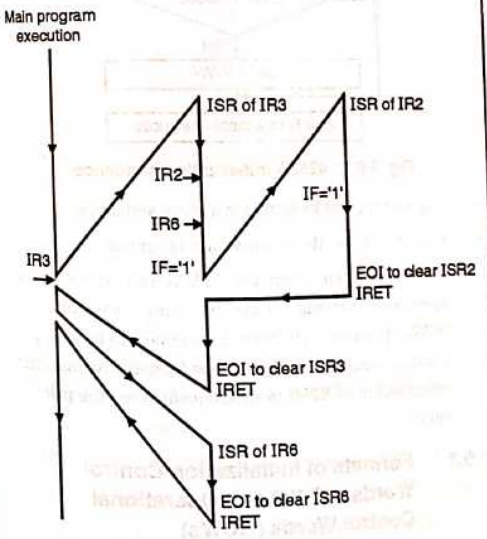


Fig. P. 8.5.1

3. Rotating priority mode

The rotating priority mode can be set as :

(i) Automatic rotation
(ii) Specific rotation

(i) Automatic rotation

- In this mode, a device after being serviced becomes the lowest priority, and consecutive next interrupt becomes highest priority.
- The device that has been just serviced will receive the seventh priority. Here IR_4 has just been serviced, hence it becomes lowest priority and IR5 becomes highest priority.

IR_0	IR_1	IR_2	IR_3	IR_4	IR_5	IR_6	IR_7
3	4	5	6	7	0	1	2

Lowest priority — Highest priority

(ii) Specific rotation

- In the automatic rotation mode, the interrupt request that is serviced last is assigned the lowest priority.
- In the specific rotation mode, lowest priority can be assigned to any interrupt input (IR_0 to IR_7) by a specific rotation command.
- e.g. if lowest priority is assigned to IR_3, all other interrupt priorities are shown.

IR_0	IR_1	IR_2	IR_3	IR_4	IR_5	IR_6	IR_7
4	5	6	7	0	1	2	3

4. Special masked mode

- If an interrupt is in service, then the corresponding bit in the InSR is set and the lower priority interrupts are inhibited.
- Sometimes it is desired for the interrupt service routine to dynamically after the systems interrupt priority structure. In such cases we have to used special masked mode.

- Special masked mode inhibits further interrupts at that level and enables interrupts form all other levels that are not masked. Thus, as interrupt can be enabled by loading the mask register.

8.5.3 Operating Modes

University Question

Q. Explain operating modes in 8259.

MU - Dec. 11, 5 Marks

8259 has two operating modes viz. interrupt driven and polling mode. In interrupt driven mode, 8259 interrupts the processor with the INT pin whenever it gets an interrupt.

Polled mode

- In this mode the INT output is not used. The microprocessor checks the status of the interrupt request by issuing poll command.
- The microprocessor reads contents of 8259 A after issuing the poll command.
- During the read operation the 8259A provides polled word and sets the InSR bit of highest active interrupt request in following format.

I	X	X	X	X	X	W_2	W_1	W_0

I = 1 — One or more interrupt, requests activated

Binary code of highest, priority active

I = 0 No interrupt request activated interrupt request

8.6 Programming the 8259A

University Questions

Q. Explain the initialization Command Words (ICWs) and Operational Command Words (OCWs) of the 8259 PIC ? **MU - May 11,10 Marks**

Q. Draw the flowchart for initialization sequence of PIC 8259. **MU - Dec. 11, 5 Marks**

Q. Explain ICWs of interrupt controller 8259. **MU - May 14,10 Marks**

Q. Explain the operational command words of PIC 8259 **MU - May 11,10 Marks**

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

(i)Initialization command words (ICWs)

(ii)Operation command words (OCWs)

- The 8259A can be initialised with four ICWs, the first two are compulsory and the other two are optional based on the modes being used.

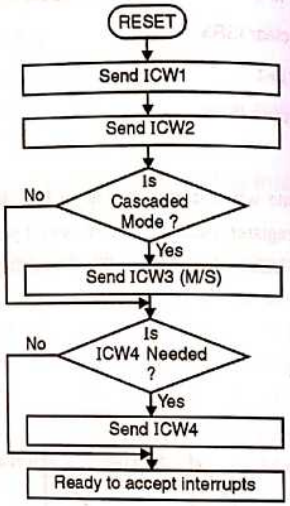


Fig. 8.6.1 : 8259A initialization sequence

- The words must be issued in a given sequence.
- Fig. 8.6.1 shows the initialisation sequence.
- After the initialisation the 8259A can be set up to operate in various modes by using three different OCWs. However, they are not necessary to be issued in a specific sequence. They may be loaded any time after initialisation of 8259 to dynamically alter the priority modes.

8.6.1 Formats of Initialization Control Words (ICWs) and Operational Control Words (OCWs)

The Fig. 8.6.2 below show the formats of ICWs and OCWs.

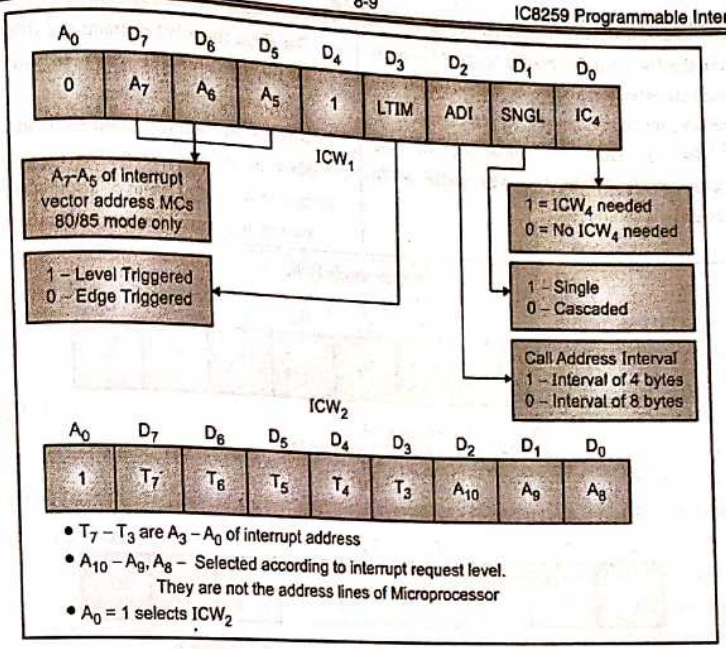


Fig. 8.6.2

8.6.1(A) ICW1

- ICW1 is compulsory as seen in Fig 8.6.2. 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.
- **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.

8.6.1(B) ICW2

- ICW2 is compulsory as seen in Fig 8.6.2. 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.

8.6.1(C) ICW3

- ICW3 is required on in cascaded mode as seen in Fig. 8.6.3. 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.

- As shown in the Fig. 8.6.3, 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. 8.6.3

8.6.1(D) ICW4

- ICW4 is required if IC4 bit of ICW1 was made '1' as seen in Fig 8.6.3. 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.

8.6.1(E) OCW1

The address bit, A_0 must be '1' while giving the operational control word 1 to the 8259 chip.

- M7 to M0 (D_7 to D_0) : These bits indicate masking or unmasking of a particular interrupt request IR0 to IR7 of 8259. If a bit is set to '1', the corresponding interrupt is disabled or masked, else its enabled or unmasked.

OCW$_1$

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	M_7	M_6	M_5	M_4	M_3	M_2	M_1	M_0

1– Mask Set
2– Mask Reset

Fig. 8.6.4 : OCW1 Format

8.6.1(F) OCW2

The address bit, A_0 must be '1' while giving the operational control word 2 to the 8259 chip.

- **R, SL and EOI (D_7 to D_0) :** These bits indicate different commands programming for 8259. The table for the different commands is shown in the Fig. 8.6.5.
- **L_2 to L_0 (D_2 to D_0) :** These bits are used to indicate the level or the interrupt request IR0 to IR7, when selecting a particular command using the bits D_7 to D_0. For example if the command is "specific EoI command", then L_2 to L_0 specify the interrupt level for which it is specific EoI. For certain commands like non-specific EoI, the bit D_2 to D_0 are not required.

OCW$_2$

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	R	SL	EOI	0	0	L_2	L_1	L_0

	R	SL	EOI			L_2	L_1	L_0	
End of interrupt	0	0	1						Non-Specification EOI command
	0	1	1						Specification EOI command
	1	0	1						Rotate On Non-Specific EOI mode (Set)
Automatic rotation	1	0	0						Rotate In Automatic EOI mode (Set)
	0	0	0						Rotate in Automatic EOI mode (Clear)
	1	1	1						Rotate on Specific EOI command •
Specific rotation	1	1	0						Set priority command
	0	1	0						No operation

	0	1	2	3	4	5	6	7
L_2	0	0	0	0	1	1	1	1
L_1	0	0	1	1	0	0	1	1
L_0	0	0	0	0	1	1	1	1

• - In this mode L_0 - L_2 are used

Fig. 8.6.5 : OCW2 Format

8.6.1(G) OCW3

The address bit, A_0 must be '0' while giving the operational control word 3 to the 8259 chip.

- **D_7 and D_4 :** These bits should always be 0.
- **ESMM and SMM (D_6 and D_5) :** These bits are used to enable and disable the Special mask mode. The table for the same is shown in the Fig. 8.6.6.
- **D_3 :** This bit should always be '1'
- **P (D_2) :** This bit is used to indicate giving of poll command from 8259. If the number of interrupt sources are too many and they cannot be interfaced using even cascaded mode, then the additional interrupt sources can be connected to8259s connected in poll mode. In this mode, the INTR pin of 8259 will not be connected to the INTR pin of 8086. This mode may also be required otherwise.
- **RR and RIS (D_1 and D_0) :** These bits are used to indicate 8259 that the processor wants to read the registers IRR (Interrupt request register) or ISR (In-service register).

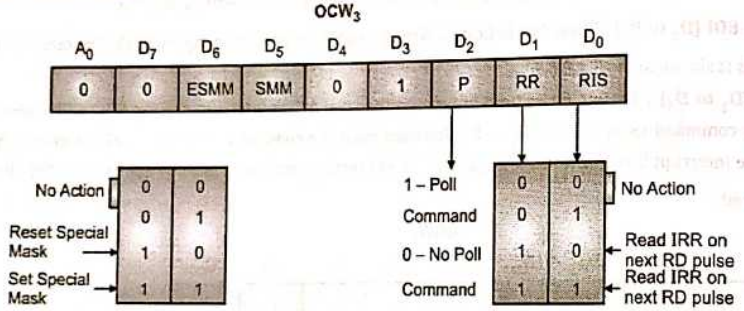


Fig. 8.6.6 : OCW3 Format

Ex. 8.6.1: Write the initialisation instructions for 8259A interrupt controller to meet the following specifications.

(i) Edge triggered, single.
(ii) Mask interrupts IR_1 and IR_3.
(iii) Interrupt vector type for IR_0 is 50 H.

Soln. : ICW 1

A_7	A_6	A_5	1	LTIM	ADI	SNGL	IC_4
0	0	0	1	0	0	1	1

= 13 H

ICW 2

It is type of Interrupt.

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
0	1	0	1	0	0	0	0

= 50 H

We do not require ICW3 as it is single 8259 connection.

ICW 4

D_7	D_6	D_5	SFNM	BUF	M / $\overline{S}$	AEIO	µPM
0	0	0	0	0	0	0	1

= 01 H

OCW 1

An OCW 1 must be sent to an 8259A to unmask any IR inputs. Here we want to mask IR_1 and IR_3, so we will put 1's in these two bits and 0's in the rest of bits.

M_7	M_6	M_5	M_4	M_3	M_2	M_1	M_0
0	0	0	0	1	0	1	0

= 0AH

Program

Instruction	Comments
MOV AL,13H	Edge triggered, single and ICW4 required
OUT 40H,AL	Send ICW1
MOV AL,50H	Interrupt type for IR0
OUT 41H,AL	Send ICW2
MOV AL,01H	Microprocessor 8086 selection
OUT 41H,AL	Send ICW4
MOV AL,0AH	Mask interrupts IR1 and IR3
OUT 41H,AL	Send OCW1

Ex. 8.6.2 : Write the initialisation instructions required for the master slave configuration. Assume the INTR of slave is routed through IR_2 of the master 8259 to the microprocessor. The addresses of the master and slave being D_0, D_1 H and F0 H and F1H. Write the initialisation for automatic rotation and auto end of interrupt. The master and slave interrupt vector type are 50 H and 60 H.

Soln. :

ICW 1 (Master)

A_7	A_6	A_5	1	LTIM	ADI	SNGL	IC_4
0	0	0	1	1	0	0	1

= 19H

ICW 2 (Master)

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
0	1	0	1	0	0	0	0

= 50 H

ICW 3 (Master)

S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0
0	0	0	0	0	1	0	0

= 04 H

ICW 4 (Master)

D_7	D_6	D_5	SFNM	BUF	M / $\overline{S}$	AEOI	µPM
0	0	0	0	0	0	1	1

= 03 H

Operation Command word 2 (OCW 2) (Master)

R	SL	EOI	D_4	D_3	L_2	L_1	L_0
1	0	0	0	0	0	0	0

= 80 H

ICW 1 (Slave)

A_7	A_6	A_5	1	LTIM	ADI	SNGL	IC_4
0	0	0	1	1	0	0	1

= 19H

ICW 2 (Slave)

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
0	1	1	0	0	0	0	0

= 60 H

ICW 3 (Slave)

–	–	–	–	–	ID_2	ID_1	ID_0
0	0	0	0	0	0	1	0

= 02 H

ICW 4 (Slave)

D_7	D_6	D_5	SFNM	BUF	M / $\overline{S}$	AEOI	µPM
0	0	0	0	0	0	1	1

= 03 H

Operation Command word 2 (OCW 2) (Slave)

R	SL	EOI	D_4	D_3	L_2	L_1	L_0
1	0	0	0	0	0	0	0

= 80 H

Program

Instruction	Comments
MOV AL,19H	Level triggered, cascaded and ICW4 required
OUT D0H,AL	Send ICW1 (Master)
MOV AL,50H	Interrupt type for IR0 (for Master)
OUT D1H,AL	Send ICW2 (for Master)
MOV AL,04H	Slave on IR2 of Master
OUT D1H,AL	Send ICW3 (for Master)
MOV AL,03H	Microprocessor 8086 selection and Automatic EOI
OUT D1H,AL	Send ICW4 (for Master)
MOV AL,80H	Enable Automatic rotation

Instruction	Comments
OUT D0H,AL	Send OCW2 (for Master)
MOV AL,19H	Level triggered, cascaded and ICW4 required
OUT F0H,AL	Send ICW1 (for Slave)
MOV AL,60H	Interrupt type for IR0
OUT F1H,AL	Send ICW2 (for Slave)
MOV AL,02H	Identity (ID) of the slave since connected on IR2
OUT F1H,AL	Send ICW3 (for Slave)
MOV AL,03H	Microprocessor 8086 selection and Automatic EOI
OUT F1H,AL	Send ICW4 (for Slave)
MOV AL,80H	Enable Automatic rotation
OUT F0H,AL	Send OCW2 (for Slave)

8.7 Interfacing 8259 with 8086

University Questions

Q. Explain interfacing of 8259 with 8086 in minimum mode. MU - May 15, 5 Marks

Q. Write short notes on : 8259-PIC. MU - May 16, 5 Marks

8259 can the interfaced with 8086 in following different possibilities.

1. 8259 in single mode and 8086 in minimum mode
2. 8259 in single mode and 8086 in maximum mode
3. 8259 in cascaded mode and 8086 in minimum mode
4. 8259 in cascaded mode and 8086 in maximum mode

The only difference in minimum and maximum mode connection is as seen in chapter 6. The control signals are provided by 8086 in case of minimum mode while in maximum mode, bus controller 8288 provides control signals. Fig. 8.7.1 shows 8259 in single mode and 8086 in maximum mode.

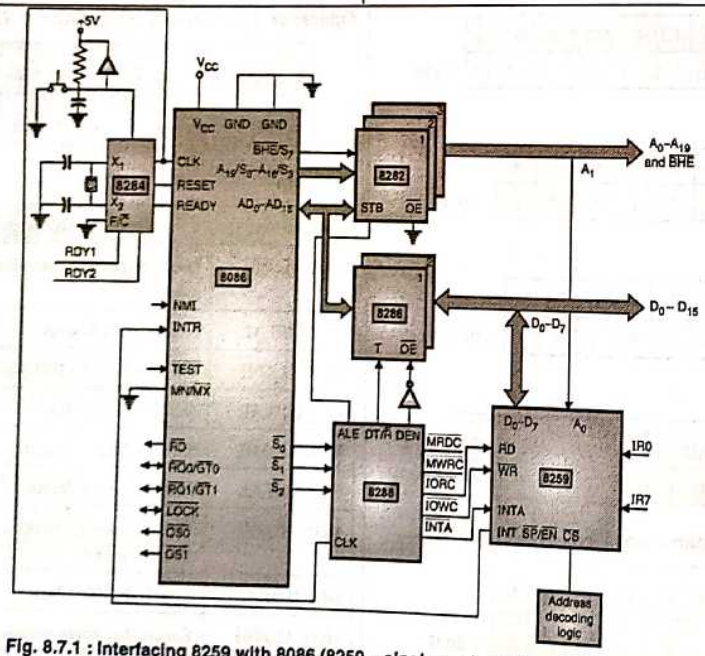


Fig. 8.7.1 : Interfacing 8259 with 8086 (8259 – single mode, 8086 – maximum mode)

8.7.1 Interfacing 8259 in Cascaded Mode

University Questions

Q. Explain the operation of three 8259 PIC in cascaded mode. MU - May 13, 10 Marks

Q. Interface three 8259s with 8086 in minimum mode and explain its functionality in fully nested mode. MU - Dec. 17, 10 Marks

Q. Explain the operation of three 8259 PIC in cascade mode. MU - May 19, 10 Marks

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. 8.7.2 shows cascaded 8259 interfaced with 8086 in maximum mode.

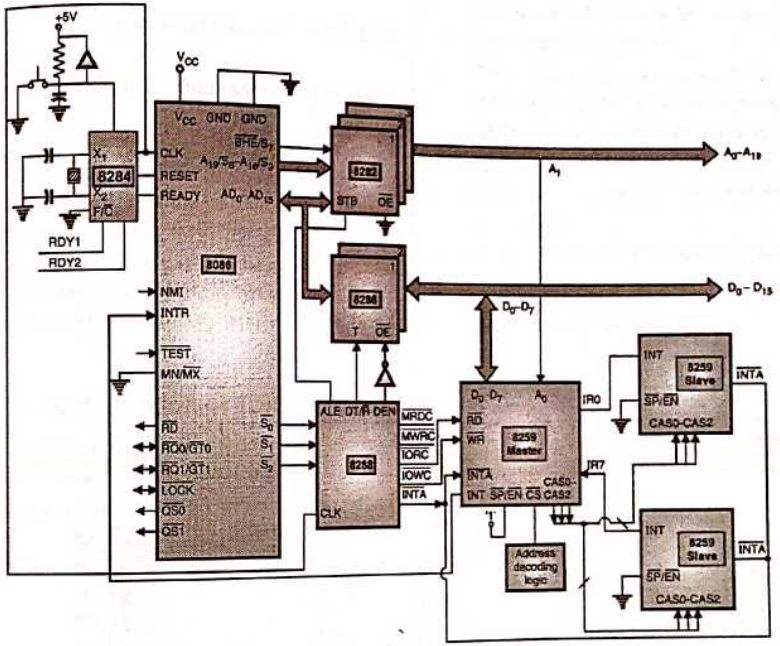


Fig. 8.7.2 : Interfacing 8259 with 8086 (8259 – cascaded, 8086 – maximum mode)

8.8 Exam Pack (Review and University Questions)

Q. 1 Explain the operation of IC 8259 with the block diagram. Explain all the signals in detail. (Refer Sections 8.3 and 8.4) (May 11, Dec. 11, 10 Marks)

Q. 2 Explain the operation of IC 8259 with block diagram. (Refer Section 8.3) (May 12, Dec. 12, Dec. 14, Dec.19, 10 Marks)

Q. 3 Draw and explain block diagram of 8259 PIC. (Refer Section 8.3) (Dec. 14, 10 Marks)

Q. 4 Explain the fully nested mode of PIC 8259. (Refer Section 8.5(1)) (Dec. 11, 5 Marks)

Q. 5 Explain operating modes in 8259. (Refer Section 8.5.3) (5 Marks)

Q. 6 Explain the initialization Command Words (ICWs) and Operational Command Words (OCWs) of the 8259 PIC ? (Refer Section 8.6) (May 11, 10 Marks)

Q. 7 Draw the flowchart for initialization sequence of PIC 8259. (Refer Section 8.6) (Dec. 11, 5 Marks)

Q. 8 Explain ICWs of interrupt controller 8259. **(Refer Section 8.6)** **(May 14, 10 Marks)**

Q. 9 Explain the operational command words of PIC 8259. **(Refer Section 8.6)** **(May 11, 10 Marks)**

Q. 10 Explain Interfacing of 8259 with 8086 in minimum mode. **(Refer Section 8.7)** **(May 15, 5 Marks)**

Q. 11 Explain the operation of three 8259 PIC in cascaded mode. **(Refer Section 8.7.1)** **(May 13, 10 Marks)**

Q. 12 Interface three 8259s with 8086 in minimum mode and explain its functionality in fully nested mode. **(Refer Section 8.7.1)** **(Dec. 17, 10 Marks)**

Q. 13 Explain the operation of three 8259 PIC in cascade mode. **(Refer Section 8.7.1)** **(May 19, 10 Aarks)**

□□□

IC8255 Programmable Peripheral Interface

9.1 8255

- The 8255 is a programmable peripheral interface i.e. PPI 8255.
- It is a general purpose programmable parallel I/O device.
- It contains 3 I/O ports which can be programmed in different modes.
- To program the function to all three I/O ports it contains a register called as control register. The control register defines the function of each I/O port and in which mode they should operate.
- 8255 is a general purpose in nature and provides many facilities for connecting different devices. So it is used frequently in different applications.

9.1.1 Features of 8255

1. 3, 8-bit IO ports PA, PB, PC
2. PA can be set for Modes 0, 1, 2. PB for 0,1 and PC for mode 0 and for BSR. Modes 1 and 2 are interrupt driven/ Handshaking mode.
3. PC has 2 4-bit parts: PC upper (PCU) and PC lower (PCL), each can be set independently for Input or Output. Each PC bit can be set/reset individually in BSR mode.
4. PA and PCU are Group A (GA) and PB and PCL are Group B (GB)
5. Address/data bus must be externally de-multiplexed.
6. TTL compatible.
7. Improved dc driving capability.

9.2 Pin Configuration of 8255

Q. Explain pin configuration of 8255 **(5 Marks)**

The pin configuration of 8255 programmable peripheral interface is as shown in Fig. 9.2.1.

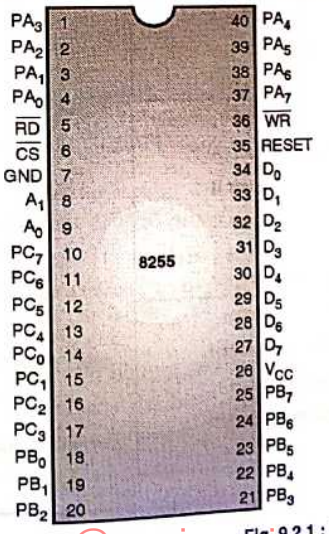


Fig. 9.2.1 : Pin configuration

PA_0-PA_7	I/O	Port A pins
PB_0-PB_7	I/O	Port B pins
PC_0-PC_7	I/O	Port C pins
D_0-D_7	I/O	Data pins
RESET	I	Reset input
$\overline{RD}$	I	Read input
$\overline{WR}$	I	Write input
A_0-A_1	I	Address pins
$\overline{CS}$	I	Chip select
V_{CC}-GND	I	+5 V supply ground

SYMBOL	TYPE	DESCRIPTION
V_{CC}		V_{CC} : The + 5V power supply pin. A 0.1 µF capacitor between V_{CC} and GND is recommended for decoupling.
GND		GROUND
D_0-D_7	I/O	**DATA BUS :** The Data Bus lines are bidirectional three state pins connected to the system data bus.
RESET	I	**RESET :** A high on this input clears the control register and all ports (A, B, C) are set to the input mode with the "Bus Hold" circuitry turned on.
$\overline{CS}$	I	**CHIP SELECT :** Chip Select is an active low input used to enable the 82C55A onto the Data Bus for CPU communication.
$\overline{RD}$	I	**READ :** Read is an active low input control signal used by the CPU to read status information or data via the data bus.
$\overline{WR}$	I	**WRITE :** Write is an active low input control signal used by the CPU to load control words and data into the 82C55A. .
A_0-A_1	I	**ADDRESS :** These input signals, in conjunction with $\overline{RD}$ and $\overline{WR}$ inputs, control the selection of one of the three ports of the control word register. A0 and A1 are normally connected to the least significant bits of the Address Bus A0, A1.
PA_0-PA_7	I/O	**PORT A :** 8-bit input and output port. Both bus hold high and bus hold low circuitry are present on this port.
PB_0-PB_7	I/O	**PORT B :** 8-bit input and output port. Bus hold high circuitry is present on this port.
PC_0-PC_7	I/O	**PORT C :** 8-bit input and output port. Bus hold circuitry is present on this port.

Table 9.2.1 : Port and register select signals summary

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	Operations
					Input (Read) Operation
0	0	0	1	0	Port A to data bus
0	1	0	1	0	Port B to data bus
1	0	0	1	0	Port C to data bus
					Output (Write) Operation
0	0	1	0	0	Data bus to port A
0	1	1	0	0	Data bus to port B
1	0	1	0	0	Data bus to port C
1	1	1	0	0	Data bus to control register
					Disable Function
×	×	×	×	1	Data bus tri-stated
1	1	0	1	0	Illegal condition
×	×	1	1	0	Data bus tri-stated

9.3 8255 Programmable Peripheral Interface(PPI) Functional Block Diagram

University Questions

Q. Draw and explain architecture of 8255. **MU - Dec. 13, Dec.15, May 16, 10 Marks**

Q. Draw and explain the block diagram of 8255 Programmable Peripheral Interface(PPI) with control word formats. **MU - May 19,10 Marks**

The block diagram of 8255 is as shown in Fig. 9.3.1.

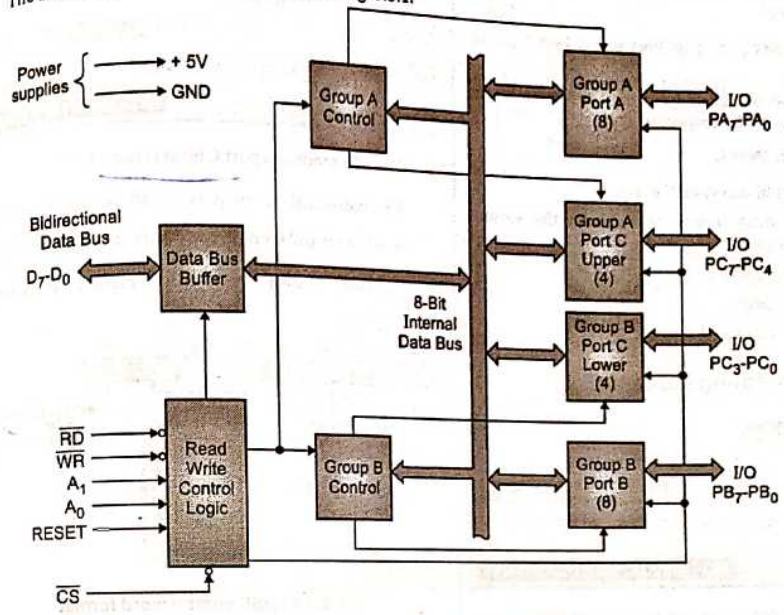


Fig. 9.3.1 : Block diagram of 8255

The architecture of 8255 can be divided into the following parts:

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.

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

9.4 8255 Operating Modes

University Questions

Q. Explain the operating modes of the 8255 PPI. **MU - May 11, 10 Marks**

Q. What are the various modes of operation of 8255 PPI ? **MU - May 12, May 16, 10 Marks**

- The 8255 IC provides one control word register.
- It is selected when $A_0 = 1$, $A_1 = 1$, $\overline{CS} = 0$ and $\overline{WR} = 0$.
- The read operation is not allowed for control register.
- The bit pattern loaded in control word register specify an I/O function for each port and the mode of operation in which the ports are to be used.
- There are 2 different control word formats which specify 2 basic modes : (1) BSR - Bit set reset mode (2) I/O mode.
- The two basic modes are selected by D_7 bit of control register. When $D_7 = 1$ it is a I/O mode and when $D_7 = 0$; it is a BSR mode.

9.4.1 BSR Mode

University Questions

Q. Write short note on : Control word register of 8255. **MU - Dec. 14, 5 Marks**

Q. Discuss control word format for Bit Set Reset (BSR) mode of 8255 PPI. **MU - Dec. 17, 5 Marks**

Q. Explain BSR mode of 8255 PPI. **MU - Dec. 19, 5 Marks**

- 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. 9.4.1.

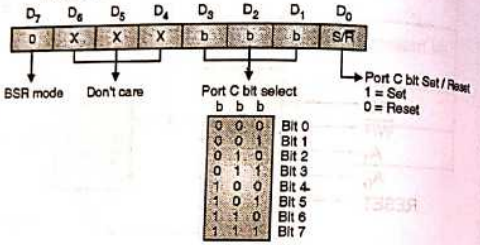


Fig. 9.4.1 : BSR control word format

- 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.

Ex. 9.4.1 : Write a set of instructions to perform the following :

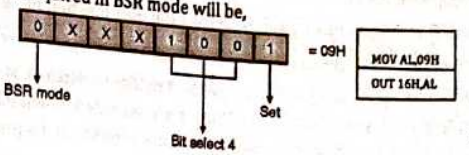
(1) Set bit 4 of port C.

(2) Reset bit 4 of port C.

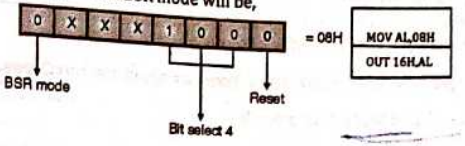
Assume the address of PA = 10 H, PB = 12 H, PC = 14 H and Control reg. = 16 H.

Soln. :

1. To set bit 4 of port C the bit pattern required in BSR mode will be,



2. To reset bit 4 of port C the bit pattern required in BSR mode will be,



9.4.2 I/O Modes

University Questions

Q. Write short note on : Control word register of 8255. **MU - Dec. 14, 5 Marks**

Q. Explain the I/O mode control word format of 8255 PPI. **MU - Dec. 18, May 19, 5 Marks**

Q. Explain Strobed Bi-directional I/O Mode 2 operation of 8255 PPI with control word and timing diagram. **MU - Dec. 19, 10 Marks**

There are three I/O modes of operation :

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

The I/O modes are programmed using control register. The control word format of I/O modes is as shown in Fig. 9.4.2.

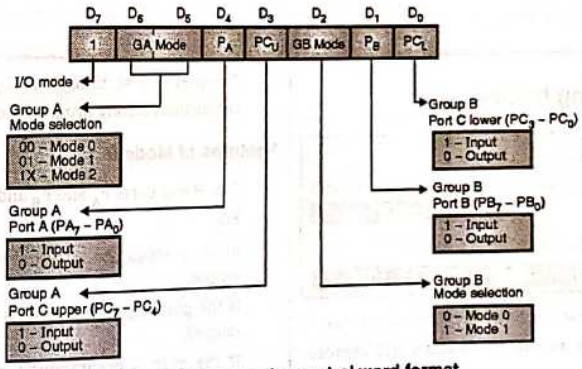


Fig. 9.4.2 : I/O modes control word format

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.

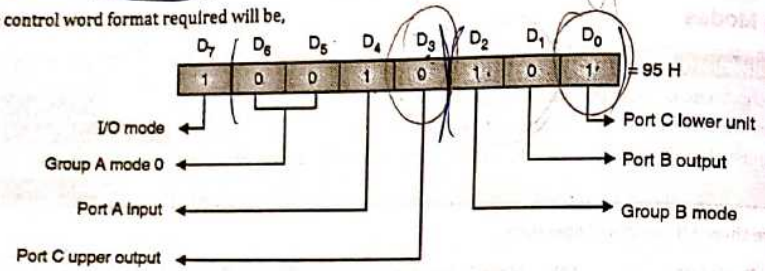
From the above explanation you can observe that all the 3 modes i.e. Mode 0, Mode 1 and Mode 2 are only for group A ports, but for group B only 2 modes i.e. Mode 0 and Mode 1 are provided. When 8255 is reset, it will clear control word register contents and all the ports are set to input mode. The ports of 8255 can be programmed for other modes by sending appropriate bit pattern to control register.

Ex. 9.4.2 : Write a set of instructions to perform the following :

(1) Initialise port A as input, port B as output, port C upper as output and port C lower as input.

(2) Use Mode 0 for group A and Mode 1 for group B.

Soln. : The control word format required will be,



The initialization instructions are :

```
MOV AL, 95H.
OUT CWR, AL.
```

9.5 I/O Operating Modes

University Questions

Q. What are the various modes of operation of 8255 PPI ? **MU - May 12, 10 Marks**

Q. Explain the operating modes of 8255 PPI. **MU - May 11, Dec. 12, 10 Marks**

The 8255 can operate in basic 3 I/O modes; mode 0, mode 1 and mode 2. Now we will see details of 8255 modes.

9.5.1 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.

Features of Mode 0

- Two 8 bits ports P_A and P_B and two 4 bit ports PC_L and PC_U.
- All the ports can be separately programmed as input or output.
- If the port is programmed as output, the outputs are latched.
- If the port is programmed as input, the inputs are buffered.
- As 4 ports are to be used, 16 different I/O configurations are possible.
- No facility for interrupt driven I/O.

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. 9.5.1.

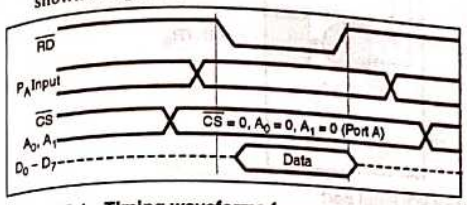


Fig. 9.5.1 : Timing waveforms for mode 0 input mode

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. 9.5.2.

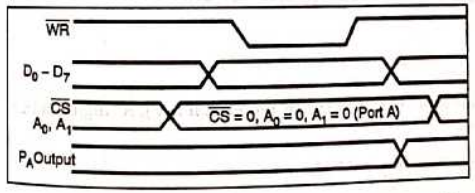


Fig. 9.5.2 : Timing waveforms for mode 0 output mode

9.5.2 Mode 1 (Strobed I/O)

University Questions

Q. Explain the handshaking operation for input and output in mode 1. **MU - May 11, Dec. 12, 10 Marks**

Q. Write short note on : Mode 1 of 8255 for input operation. **MU - May 15, 5 Marks**

- This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or "handshaking" signals. In mode 1, port A and Port B use the lines on port C to generate or accept these "handshaking" signals
- Two groups (Group A and Group B)
- Each group contains one 8-bit data port and one 4-bit control/data port
- The 8-bit data port can be either Inputs or output both Inputs and outputs are latched.
- The 4-bit port is used for control and status of the 8-bit data port.
- INTR (Interrupt Request). A high on this output can be used to interrupt the CPU for both input or output operations.
- Output Operations
- OBF (Output Buffer Full). The OBF output will go "low" to indicate that the CPU has written data out to port A.
- ACK (Acknowledge). A "low" on this input enables the tri-state output buffer of port A to send out the data. Otherwise, the output buffer will be in the high impedance state.
- INTE (The INTE Flip-Flop Associated with OBF). Controlled by bit set/reset of PC6

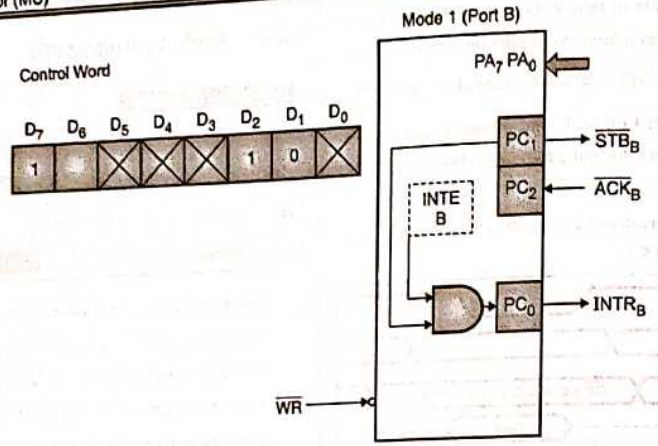


Fig. 9.5.3 : Mode 1 (Strobed I/O) Input port

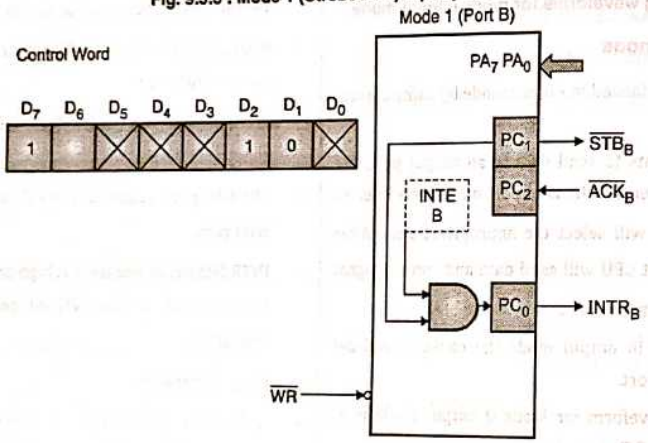


Fig. 9.5.4

Input Operations

- STB (Strobe Interrupt)
- STB (Strobed Input). A "low" on this input loads data into the input latch. IBF (Input Buffer Full F/F). A "high" on this output indicates that data has been loaded into the input latch.
- INTE (The INTE Flip-Flop Associated with IBF). Controlled by bit set/reset of PC4.

Input mode

- D4, D2: Set/Reset INTE using BSR. STB input is connected to external peripheral's strobe output (i.e. PC2, PC4 pin to external strobe).
- INTE is internal connection. STB is external connection.

Output mode

- D6, D2: Set/Reset INTE using BSR. ACK input is connected to external peripheral's acknowledge output (i.e. PC2, PC6 pin to external ack).

- INTE is internal connection. ACK is external connection.

D7	D6	D5	D4	D3	D2	D1	D0
PC7	PC6	IBF-A	INTE-A / STB-A-bar	INTR-A	INTE-B / STB-B-bar	IBF-B	INTR-B

Fig. 9.5.5 : PC bits in Input Mode

D7	D6	D5	D4	D3	D2	D1	D0
OBF-A-bar	INTE-A / ACK-A-bar	PC5	PC4	INTR-A	INTE-B / ACK-B-bar	OBF-B-bar	INTR-B

Fig. 9.5.6 : PC bits in Output Mode

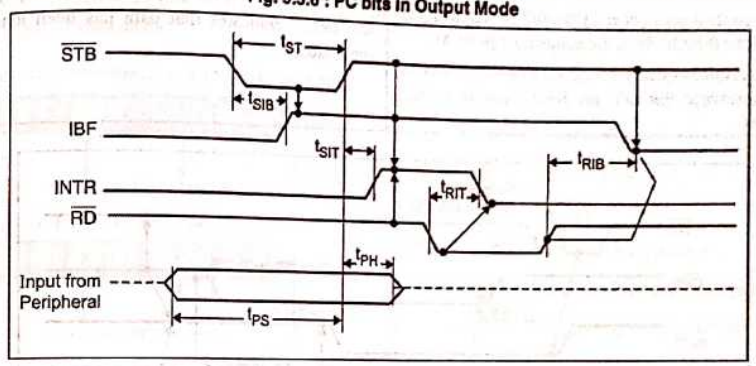


Fig. 9.5.7 : Mode 1 (Strobed I/O) Input port Timing diagram

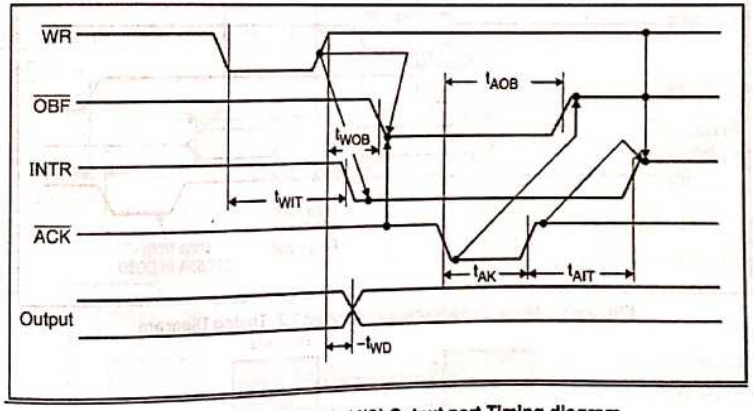


Fig. 9.5.8 : Mode 1 (Strobed I/O) Output port Timing diagram

9.5.3 Mode 2 - Strobed Bi-directional I/O

University Question

Q. Explain the operating modes of 8255 PPI including the handshaking operation. **MU - May 13, 10 Marks**

- In this mode, Port A is used as an 8-bit bi-directional Handshake I/O Port.
- This functional configuration provides a means for communicating with a peripheral device or structure on a single 8-bit bus for both transmitting and receiving data (bi-directional bus I/O). "Handshaking" signals are provided to maintain proper bus flow discipline in a similar manner to MODE.
- One 8-bit, bi-directional bus Port (Port A) and a 5-bit control Port (Port C).
- Both Inputs and Outputs are latched.
- The 5-bit control port (Port C) is used for control and status for the 8-bit,bi-directional bus port (Port A).
- INTR (Interrupt Request). A high on this output can be used to interrupt the CPU for both input or output operations.

Output Operations

OBF (Output Buffer Full). The OBF output will go "low" to indicate that the CPU has written data out to port A. ACK (Acknowledge). A "low" on this input enables the tri-state output buffer of port A to send out the data. Otherwise, the output buffer will be in the high impedance state. INTE 1 (The INTE Flip-Flop Associated with OBF). Controlled by bit set/reset of PC6

Input Operations

STB (Strobed Input). A "low" on this input loads data into the input latch. IBF (Input Buffer Full F/F). A "high" on this output indicates that data has been loaded into the input latch.

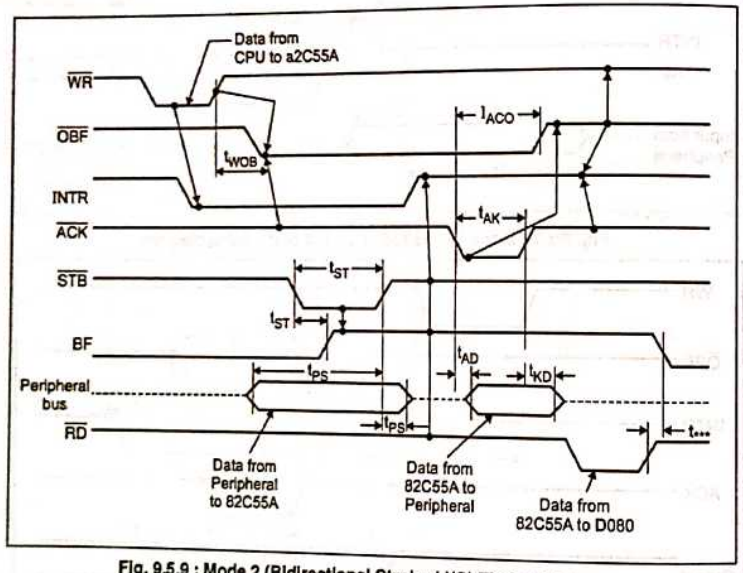


Fig. 9.5.9 : Mode 2 (Bidirectional Strobed I/O) Timing Diagram

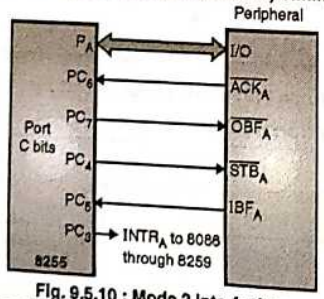


Fig. 9.5.10 : Mode 2 Interfacing

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

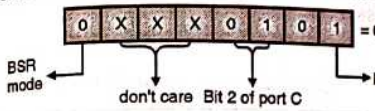
$\overline{OBF_A}$	$INTE_1$	IBF_A	$INTE_2$	$INTR_A$	X	X	X

Ex. 9.5.1

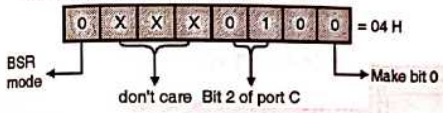
Blink port C bit 2 of 8255.

Program statement : Write a program to blink port C bit 2 of 8255. Assume address of control word register of 8255 as 83 H. Use Bit Set / Reset mode.

Soln. : Let us write the control word to make bit 2 high.



Control word to make bit 2 low



Program

Label	Instruction	Comments
L1 :	MOV AL, 05H	Load control word to make PC_2 high
	OUT 83H,AL	
	CALL DELAY	
	MOV AL, 04H	Load control word to make PC_2 low
	OUT 83H,AL	
	CALL DELAY	
	JMP L1	

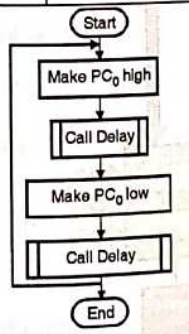


Fig. P. 9.5.1

9.6 Applications using 8255 Chip

University Question

Q. Draw and explain the 4 × 4 keyboard interface using 8255. **MU - May 14, 5 Marks**

8255 is most widely used chip for many applications. For example,

1. LED / Relay Interface
2. Key Board Interface
3. Display Interface
4. ADC / DAC Interface
5. Stepper motor Interface
6. Traffic Signal Controller
7. Lift Controller etc.

9.6.1 Matrix Type Keyboard

- In this keyboard, the keys are arranged in matrix form.
- It requires very few input/output lines.
- The number of lines required to interface n × m matrix keyboard is n + m and the number of keys is equal to n × m, where n is the number of columns and m is the number of rows.
- Each key is identified by a row and column number.
- Software and hardware techniques are adopted to identify the key closure and encode the pressed key.
- These techniques are also called scanning techniques.

9.6.2 Software Techniques

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. 9.6.1 shows a connection of hexadecimal key board with 8255.

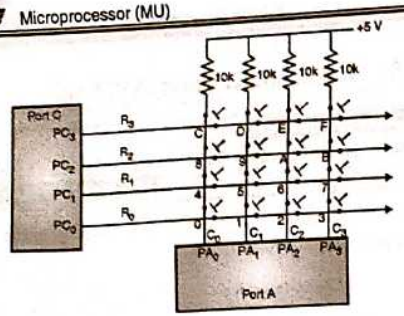


Fig. 9.6.1: Keyboard matrix

1. 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.

9.7 Interfacing 8255 with 8086

Q. Explain with neat diagram interfacing 8255 with 8086 **(6 Marks)**

Fig. 9.7.1 shows interfacing of 8255 with 8086 in I/O mapped I/O technique. Only lower data bus $D_0 - D_7$ is used as 8255 is an 8 bit device. The Reset out signal is connected to the Reset signal of 8255. In case of interrupt driven I/O INTR signal (PC_3 or PC_0) from 8255 is connected to INTR input of 8086.

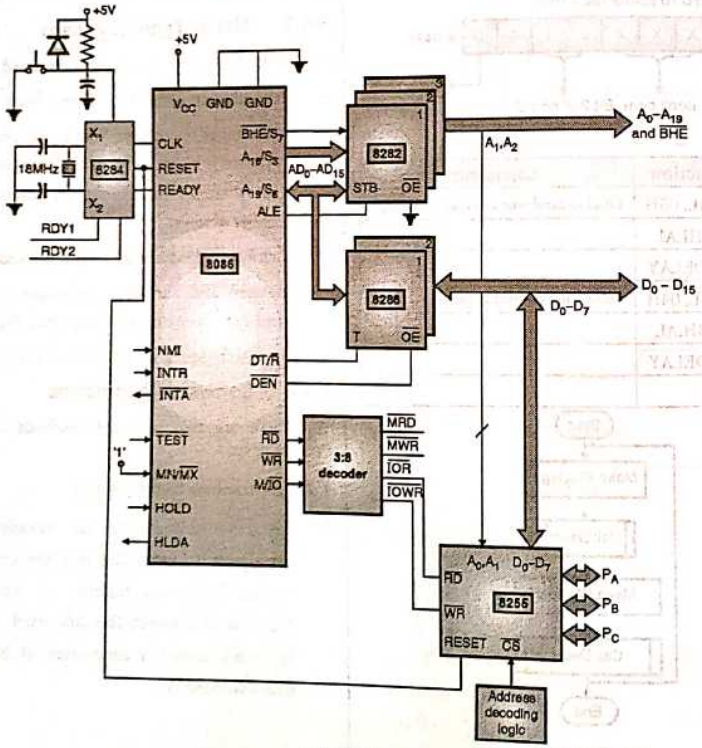


Fig. 9.7.1

9.8 Design Problems

Ex. 9.8.1 : Design a system using 8255 to generate a square wave of 1KHz. Write corresponding program.

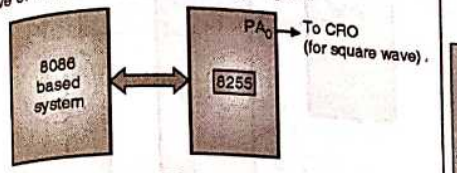


Fig. P. 9.8.1

Soln. :

Program

```
        MOV AL, 00H
again : OUT 80H, AL
CALL delay _500 µsec
NOT AL
JMP again
```

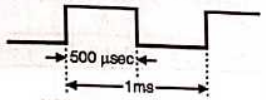


Fig. P. 9.8.1(a)

(Assume PA address = 80H)

delay 500 µsec subroutine assumed :

Generation of delay of 500 µsec :

Crystal frequency : 15 MHz

Operating frequency : 5 MHz

1 machine cycle = 4 * T-state

$= 4 * \frac{1}{5}$ µsec = 0.8 µsec

Delay required = 500 µsec

Machine required = $\frac{500}{0.8} = (625)_{10} \Rightarrow (0271)_{16}$

Delay_500µsec

```
        MOV CX, 026F H      3B    2m/c
next :  Loop next 2B        1m/c
        RET
```

∴ Total machine required = 2 + 1 * count

625 = 2 + 1 * count

∴ count = $(623)_{10} = (26F)_{16}$

∴ Program assume PA address = 80H

CW address = 86H

```
        MOV AL, 80H
        OUT 86 H, AL
        MOV AL, 00H
again : OUT 80H, AL
        CALL delay_500 µs
        NOT AL
        JMP again
```

Ex. 9.8.2

Interfacing key board and display to 8255. Interface a key to 8255 and indicate its position using a LED i.e. :

If switch is closed – LED should be ON

If switch is open – LED should be OFF.

Soln. :

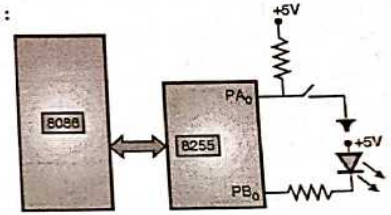


Fig. P. 9.8.2

Control Word

I/O	GA mode		PA	PCU	GB mode	PB	PCL
1	0	0	1	0	0	0	0

9 0

Assume address

PA = 80H PB = 82H

PC = 84H CW = 86H

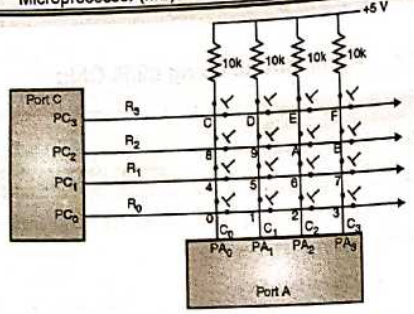


Fig. 9.6.1: Keyboard matrix

1. 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.

9.7 Interfacing 8255 with 8086

Q. Explain with neat diagram interfacing 8255 with 8086 **(6 Marks)**

Fig. 9.7.1 shows interfacing of 8255 with 8086 in I/O mapped I/O technique. Only lower data bus $D_0 - D_7$ is used as 8255 is an 8 bit device. The Reset out signal is connected to the Reset signal of 8255. In case of interrupt driven I/O INTR signal (PC_3 or PC_0) from 8255 is connected to INTR input of 8086.

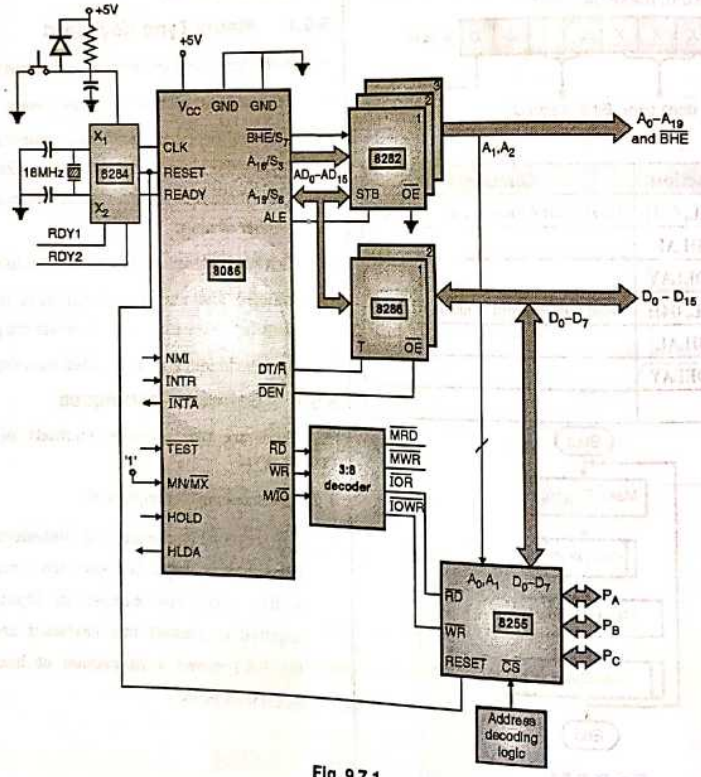


Fig. 9.7.1

9.8 Design Problems

Ex. 9.8.1 : Design a system using 8255 to generate a square wave of 1KHz. Write corresponding program.

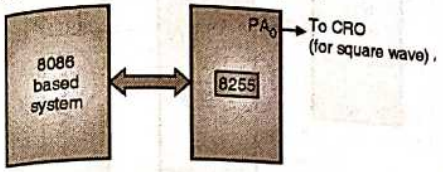


Fig. P. 9.8.1

Soln. :

Program

```
        MOV AL, 00H
again : OUT 80H, AL
CALL delay _500 µsec
NOT AL
JMP again
```

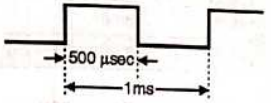


Fig. P. 9.8.1(a)

(Assume PA address = 80H)

delay 500 µsec subroutine assumed :

Generation of delay of 500 µsec :

Crystal frequency : 15 MHz

Operating frequency : 5 MHz

1 machine cycle = 4 * T-state

$= 4 * \frac{1}{5}$ µsec = 0.8 µsec

Delay required = 500 µsec

Machine required = $\frac{500}{0.8} = (625)_{10} \Rightarrow (0271)_{16}$

```
Delay_500µsec
    MOV CX, 026F H      3B    2m/c
next :Loop next 2B      1m/c
    RET
```

∴ Total machine required = 2 + 1 * count

625 = 2 + 1 * count

∴ count = $(623)_{10} = (26F)_{16}$

∴ Program assume PA address = 80H

CW address = 86H

```
        MOV AL, 80H
        OUT 86 H, AL
        MOV AL, 00H
again : OUT 80H, AL
        CALL delay_500 µs
        NOT AL
        JMP again
```

Ex. 9.8.2

Interfacing key board and display to 8255. Interface a key to 8255 and indicate its position using a LED i.e. :

If switch is closed – LED should be ON

If switch is open – LED should be OFF.

Soln. :

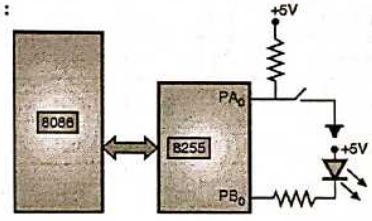


Fig. P. 9.8.2

Control Word

I/O	GA mode		PA	PCU	GB mode		PB	PCL
1	0	0	1	0	0		0	0
9				0				

Assume address

PA = 80H PB = 82H

PC = 84H CW = 86H

Program

```
        MOV AL, 90H  }
        OUT 86H, AL  }  initialize CW
again : IN AL, 80H
        OUT 82, AL   }  input to PA = output of PB
        JMP again    }
```

[Logic : if switch is closed – Logic 0 If switch is open – Logic 1]

Ex. 9.8.3 : Write a program to interface 8 keys using 8255 and display the key pressed on 7 segment display.

No	a	b	C	d	e	f	g	dp	Common anode Hex no.	Hex
0	1	1	1	1	1	1	0	0	FCH	03H
1	0	1	1	0	0	0	0	0	60H	9FH
2	1	1	0	1	1	0	1	0	DAH	25H
3	1	1	1	1	0	0	1	0	F2H	0DH
4	0	1	1	0	0	1	1	0	66H	99H
5	1	0	1	1	0	1	1	0	B6H	49H
6	1	0	1	1	1	1	1	0	BEH	41H
7	1	1	1	0	0	0	0	0	F0H	1FH
8	1	1	1	1	1	1	1	0	FEH	01H
9	1	1	1	1	0	1	1	0	F6H	09H
A	1	1	1	0	1	1	1	0	EEH	11H
B	0	0	1	1	1	1	1	0	3EH	C1H
C	1	0	0	1	1	1	0	0	9CH	63H
D	0	1	1	1	1	0	1	0	7AH	85H
E	1	0	0	1	1	1	1	0	9EH	61H
F	1	0	0	0	1	1	1	0	8EH	71H

Assume this look up table for common anode is stored on memory location from 3200 H on word.

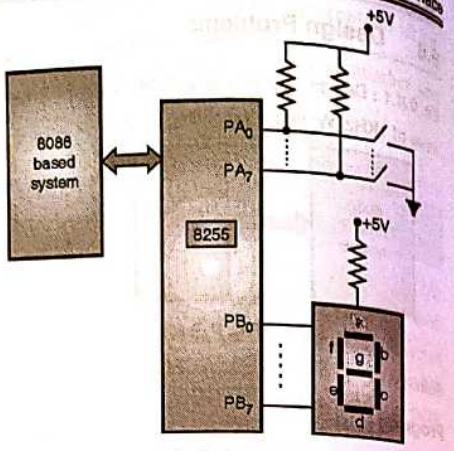


Fig. P. 9.8.3

Control word :	I/O	GA mode		PA	PCU	GB mode	PB	PCL
(90 H)	1	0	0	1	0	0	0	0

Assume : PA = 80H PB = 82H

PC = 84H CW = 86H

Program

Label	Instruction	Comments
	MOV AX, 3000H	Initialize Data Segment (DS) Register
	MOV DS, AX	
	MOV BX, 2000H	Initialize pointer to the Look-up table
	MOV AL, 90H	Initialize Control Word
	OUT 86H, AL	
	MOV CH, 00H	
	MOV AL, FFH	Initially assume that no switch is pressed, hence all SSD switched off.
	OUT 82H, AL	
again :	IN AL, 80H	Input from the port used for connecting switches
	MOV CL, 08H	Initialize counter to check 8 bits (switches)
next:	SHL AL, 1	Shift the data received from switches so as to get the MSB in carry flag and hence check if the switch is pressed or not

Label	Instruction	Comments
	JC over	If the switch is not pressed goto check the next switch by decrementing the counter
	MOV AL, CL	If the switch is recognized move the count of switch number in register AL
	DEC AL	Decrement the value of AL as the switch numbers are from 0 to 7, while count is from 1 to 8
	XLAT	Get the equivalent SSD code from the look-up table
	OUT 82H, AL	Give this code to the SSD
	JMP again	Repeat the checking of switch pressed
over :	LOOP next	Decrement the counter and check for next switch
	MOV AH, 4CH	Terminate program
	INT 21H	

Interface Hex keypad with 7 segment display using 8255 to display switch press.

Ex. 9.8.4 : Switch debouncing.

When human being presses a switch there are some bounces as shown in Fig. P. 9.8.4(a).

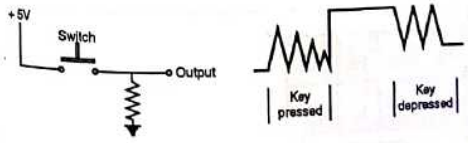


Fig. P. 9.8.4(a)

A bounce laps for not more than 20 msec.

Hence switch debouncing implements checking of the key pressed 2 times at the interval of 20 msec to confirm a valid switch pressed.

Control word

I/O	GA mode		PA	PCU	GB mode	PB	PCL
1	0	0	0	0	0	0	0

8 0

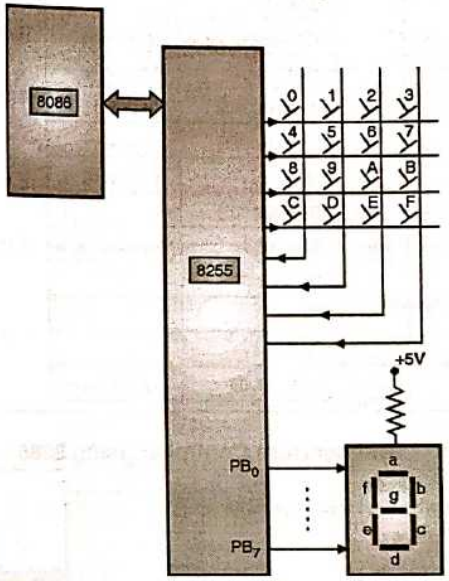


Fig. P. 9.8.4(b)

Switch pressed = row number * 4 + column number

Assume PA ⇒ 80H (address)

PB ⇒ 82H (address)

PC ⇒ 84H (address)

CW ⇒ 86H (address)

Program

Label	Instruction	Comments
	MOV AX, 3000H	
	MOV DS, AX	
	MOV BX, 2000H	
	MOV AL, 81H	

Label	Instruction	Comments
	OUT 86H, AL	
next :	MOV AL, 00H	
	OUT 80H, AL	
again :	IN AL, 84H	
	AND AL, 0FH	
	CMP AL, 0FH	
	JZ again	
	CALL delay_20 ms	debouncing
	IN AL, 84H	debouncing
	AND AL, 0FH	debouncing
	CMP AL, 0FH	debouncing
	JZ again	debouncing
	MOV CL, 00H	
rep1 :	ROR AL, 1	to get column no
	JNC over	
	INC CL	
	CMP rep1	
over :	MOV CH, 00	to get column no

Label	Instruction	Comments
	MOV AL, 0E H	
rep2 :	MOV DL, AL	
	OUT 80H, AL	
	IN AL, 0FH	
	CMP AL, 0FH	
	JNZ over 1	
	INC CH	
	MOV AL, DL	
	SHL AL, 01	
	JMP rep2	
over1:	MOV AL, 40H	
	MOV CH	
	ADD AL, CL	
	XLAT	
	OUT 82H, AL	
	JMP next	

9.9 Temperature Controller using 8086

Let addresses of 8255 ports be as follows :

PA – 80 H

PB – 82 H

PC – 84 H

CW – 86 H

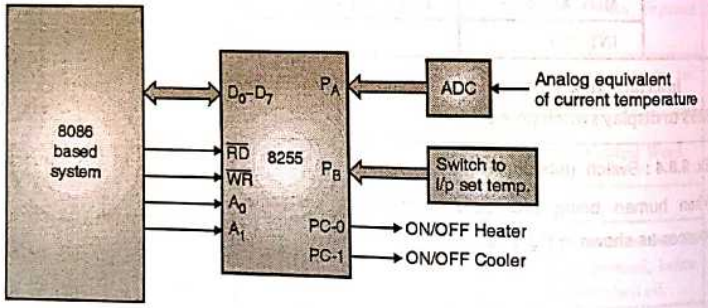


Fig. 9.9.1

I/O	GA mode		PA	PCU	GB mode	PB	PCL
1	0	0	1	0	0	1	0
	9				2		

Program for temperature controller

Label	Instruction	Comments
	MOV AL, 92H	
	OUT 86H, AL	
back :	IN AL, 80H	
	MOV BL, AL	
	IN AL, 82 H	
	CMP AL, BL	
	JC over	
	MOV AL, 02 H	
	OUT 84H, AL	
	JMP back	
over :	MOV AL, 01H	
	OUT 84H, AL	
	JMP back	
	HLT	

9.10 Exam Pack (Review and University Questions)

Q. 1 Explain pin configuration of 8255
(Refer Section 9.2) **(5 Marks)**

Q. 2 Draw and explain architecture of 8255.
(Refer Section 9.3)
(Dec.13, Dec.15, May 16, 10 Marks)

Q. 3 Explain the operating modes of the 8255 PPI. Also, explain the handshaking operation for input and output in mode 1.
(Refer Section 9.4 and 9.5.2)
(May 11, Dec. 12, 10 Marks)

Q. 4 What are the various modes of operation of 8255 PPI ? **(Refer Section 9.4 and 9.5)**
(May 12, May 16 10 Marks)

Q. 5 Write short note on : Control word register of 8255.
(Refer Sections 9.4.1 and 9.4.2) **(Dec.14, 5 Marks)**

Q. 6 Discuss control word format for Bit Set Reset (BSR) mode of 8255 PPI.
(Refer Section 9.4.1) **(Dec.17, 5 Marks)**

Q. 7 Write short note on : Mode 1 of 8255 for input operation. **(Refer Section 9.5.2)** **(May 15, 5 Marks)**

Q. 8 Explain the operating modes of 8255 PPI including the handshaking operation.
(Refer Section 9.5.3) **(May 13, 10 Marks)**

Q. 9 Draw and explain the 4 × 4 keyboard interface using 8255. **(Refer Section 9.6)** **(May 14, 5 Marks)**

Q. 10 Explain with neat diagram interfacing 8255 with 8086
(Refer Section 9.7) **(6 Marks)**

Q. 11 Explain the I/O mode control word format of 8255 PPI.
(Refer Section 9.4.2) **(Dec. 18, May 19, 5 Marks)**

Q. 12 Draw and explain the block diagram of 8255 Programmable Peripheral Interface(PPI) with control word formats.
(Refer Section 9.3) **(May 19,10 Marks)**

Q. 13 Explain BSR mode of 8255 PPI.
(Refer Section 9.4.1) **(Dec. 19, 5 Mark)**

Q. 14 Explain Strobed Bi-directional I/O Mode 2 operation of 8255 PPI with control word and timing diagram.
(Refer Section 9.4.2) **(Dec. 19, 10 Marks)**

□□□

Note

10 MODULE 3

IC8257 Direct Memory Access Controller (DMAC)

10.1 Introduction

University Question

Q. Draw and explain the block diagram of 8257 DMA controller. MU - Dec. 18, 10 Marks

- 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.

Let's understand the concept clearly.

1. Initially, switches S_1, S_2 and S_3 are at position A.
2. No direct access to memory; by I/O.
3. Microprocessor is MASTER of all three buses; address, data and control.
4. Microprocessor treats **DMA controller,** as I/O device ONLY.
5. Using IN/OUT instruction, you can program **DMA controller** chip, for various modes.
6. Whenever, peripheral device is ready to transfer data, DIRECTLY to memory, it will generate REQUEST (DRQ → DMA REQUEST), to DMA controller, asking for direct access.
7. In response to DRQ, DMA controller will activate, HRQ (HOLD request); connected to HOLD pin of microprocessor. By activating HOLD line, DMA controller request microprocessor, to HOLD for 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 (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.
11. 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.**

This is about basics of DMA. Now let's study the different methods of transferring data.

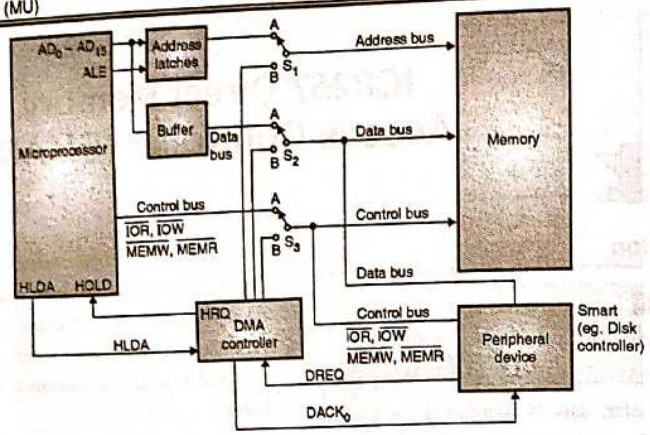


Fig. 10.1.1 : DMA controller interconnection scheme

10.1.1 DMA Controller: Data Transfer Modes

University Question

Q. Write short note on : Modes of DMA transfers. **(5 Marks)**

Q. Discuss different data transfer modes of DMA controller 8257. **(5 Marks)**

Q. Explain data transfer modes of DMAC 8257 **MU - Dec. 19, 5 Marks**

The DMA controller functions as a bus master and bus slave. It performs data transfer operations. DMA controlled input/output is further divided into the following categories :

(1) Burst or Block transfer DMA.

(2) Cycle steal or Single byte transfer DMA.

(3) Transparent or Hidden DMA.

10.1.1(A) 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.

10.1.1(B) 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.

10.1.1(C) 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.

10.2 Programmable DMA Controller 8257

Q. Write short note on DMA controller. **(10 Marks)**

General features :

- MCS-85® Compatible 8257-5
- 4-channel DMA Controller
- Priority DMA Request Logic
- Channel Inhibit Logic
- Terminal Count and Modulo 128 Outputs
- Single TTL Clock
- Single + 5 V Supply
- Auto Load Mode
- Available in EXPRESS
- Standard Temperature Range

10.2.1 Comparison between Slave and Master Mode

Sr. No.	Slave mode (Normal mode of the system)	Master mode (DMA mode of the system)
(1)	In this mode, microprocessor functions as a bus master and DMA controller functions as a bus slave.	In this mode, the DMA controller functions as a bus master and the microprocessor is disconnected from the system bus.
(2)	The data is transferred through microprocessor, hence two operations are required to transfer one byte between I/O device and a memory. As a result transfer rate is reduced.	The data is not transferred through microprocessor as well as DMA controller. The data is transferred between I/O device and memory directly. Hence only one operation is required to transfer one byte between I/O device and a memory. As a result the transfer rate is increased.
(3)	The microprocessor performs arithmetic, logical data transfer and decision making operations, hence data can be processed.	The DMA controller performs only data transfer operation, hence data cannot be processed.
(4)	All address and control signals are generated by microprocessor.	All address and control signals are generated by DMA controller.
(5)	The initialization of DMA controller, DMA and non DMA I/O devices is done by microprocessor.	The DMA controller cannot initialize devices.

10.3 Pin Configuration of 8257

Q. Write short note on Pin Configuration of 8257. **(10 Marks)**

The pin diagram of 8257 is as shown in Fig. 10.3.1.

$D_7 - D_0$	I/O	Data bus
$\overline{IOR}$	I/O	I/O read bi-directional
$\overline{IOW}$	I/O	I/O write bi-directional
$\overline{MEMR}$	O	Memory read
$\overline{MEMW}$	O	Memory write

$A_0 - A_3$	I/O	Address bus bi-directional
$A_4 - A_7$	O	Address bus
CLK	I	Clock input
RESET	I	Reset input
READY	I	Ready input
HRQ	O	HOLD request
HLDA	I	Hold acknowledge
AEN	O	Address enable
ADSTB	O	Address strobe
TC	O	Terminal count
MARK	O	Modulo 128 mark
$DRQ_0 - DRQ_3$	I	DMA request input
$\overline{DACK\ 0 - DACK\ 3}$	O	DMA acknowledge output
$\overline{CS}$	I	Chip select
V_{CC} / V_{SS}	I	+ 5 volts / Ground

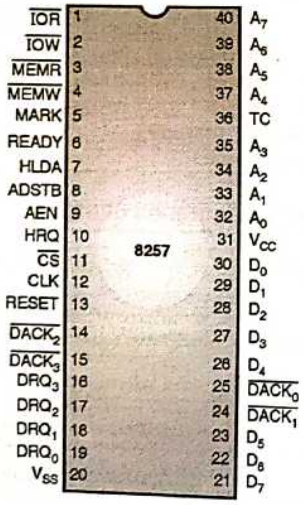


Fig. 10.3.1 : Pin diagram of 8257

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.
$\overline{IOW}$	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.

Symbol	Description
$A_0 - A_3$	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.
$\overline{MEMR}$	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.

Symbol	Description
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 system lines. 2. Disconnect data lines of 8257 from the system data bus and connect these lines to the high order system address lines ($A_8 - A_{15}$). 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.
$\overline{DACK\ 0 - DACK\ 3}$	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.

10.4 8257 Block Diagram

University Question

Q. Draw and explain the block diagram of 8257 DMA controller. **MU - Dec. 18, 10 Marks**

The block diagram of 8257 is as shown in Fig. 10.4.1. It contains following blocks :

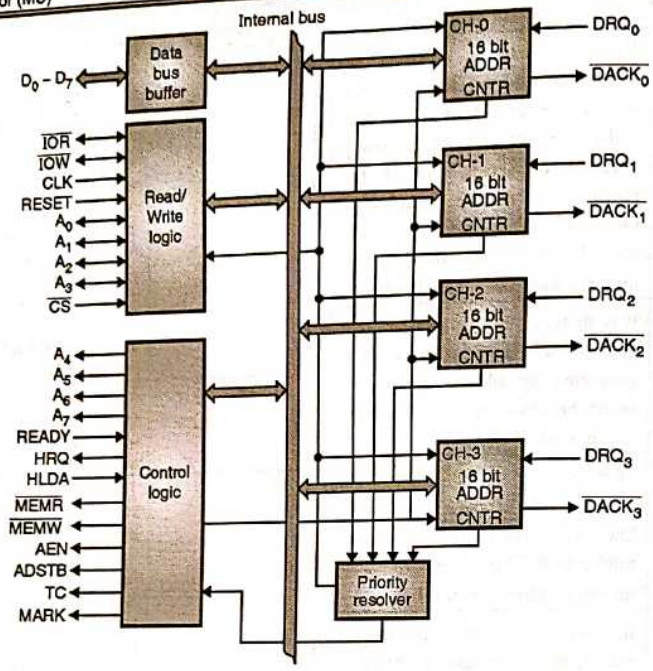


Fig. 10.4.1 : 8257 Functional block diagram

1. DMA Channels

- The 8257 provides four separate DMA channels (labeled CH-0 to CH-3).
- Each channel includes two sixteen-bit registers.

  (1) a DMA address registers and

  (2) a terminal count register Both registers must be initialized before a channel is enabled.
- The DMA address register is loaded with the address of the first memory location to be accessed.
- 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.
- In general, 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.
- These two bits are not modified during a DMA cycle, but can be changed between DMA blocks.
- Each channel accepts a DMA Request (DRQn) input and provides a DMA Acknowledge (DACKn) output.

(DRQ 0 – DRQ 3)

DMA request

- 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.
- A request can be generated by raising the request line and holding it high until DMA acknowledge.
- For multiple DMA cycles (Burst mode) the request line is held high until the DMA acknowledge of the last cycle arrives.

(DACK 0 – DACK 3)

DMA acknowledge

- 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.

($D_0 - D_7$)

Data Bus Lines

- 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.
- Note that 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.

($\overline{I/OR}$)

I / O Read

- 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.

($\overline{I/OW}$)

I/O write

- 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.

(CLK)

Clock input : Generally from an Intel® 8224 clock generator device. (ϕ2 TTL) or Intel® 8085A CLK output.

(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.

($A_0 - A_3$)

Address lines : 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.

($\overline{CS}$)

Chip select : 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.

($A_4 - A_7$)

Address lines : 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.

(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.

(HRQ)

Hold acknowledge : This input from the CPU indicates that the 8257 has acquired control of the system bus.

($\overline{MEMR}$)

Memory read : This active-low three-state output is used to read data from the addressed memory location during DMA read cycles.

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

(ADSTB)

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

(AEN)

Address enable

- This output is used to disable (float) the system data bus and the system control bus.
- It may 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 may be further 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.

(TC)

Terminal count

- 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.

(MARK)

Modulo 128 mark

- This output notifies the selected peripheral that the current DMA cycle is the 128th cycle since the previous MARK output.

- MARK always occurs at 128 (and all multiples of 128) cycles from the end of the data block. Only if the total number of DMA cycles (n) is evenly divisible by 128 (and the terminal count register was loaded with n-1), will MARK occur at 128 (and each succeeding multiple of 128) cycles from the beginning of the data block.

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 :

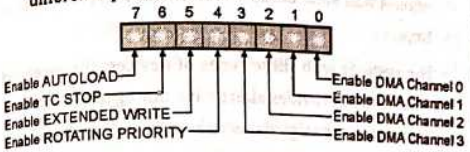


Fig. 10.4.2

- 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.

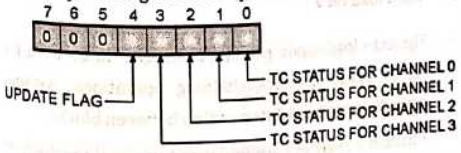


Fig. 10.4.3

- 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, however, 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.

10.5 Operating Modes of 8257

Q. Explain various priority modes and transfer modes of DMA controller. **(8 Marks)**

Q. Explain different modes of operation of DMA Controller. **(10 Marks)**

Q. Write short note on : DMA controller modes. **(5 Marks)**

The 8257 operates in the following modes :

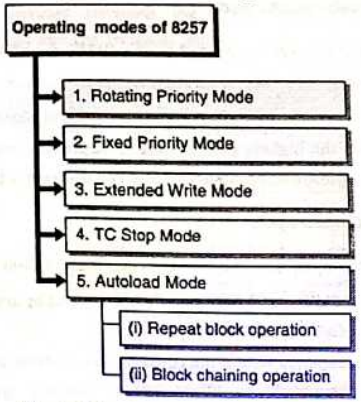


Fig. 10.5.1 : Operating modes of 8257

- Note that rotating priority will prevent any one channel from monopolizing the DMA mode; consecutive DMA cycles will service different channels if more than one channel is enabled and requesting service.
- There is no overhead penalty associated with this mode of operation.
- All DMA operations begin with channel 0 initially assigned to the highest priority for the first DMA cycle.

1. Rotating Priority Mode

- In this case, the DMA channel which is just serviced becomes the lowest priority and the next one gets the highest priority.
- This allows all the channels to get a chance to be serviced.
- This kind of priority must be used if all the devices are of same priority and no device is more important or less important

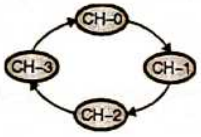


Fig. 10.5.2

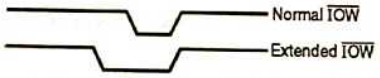


Fig. 10.5.3

2. Fixed Priority Mode

- If the devices connected to the DMAC are of different priority then this mode must be used.
- In this case the device connected on the channel 0 is given the highest priority and in decreasing order the one connected on channel 3 is given the least priority

3. Extended write

- If the EXTENDED WRITE bit is set, the duration of both the $\overline{MEMW}$ and $\overline{I/OW}$ signals is extended by activating them earlier in the DMA cycle.
- Data transfers within micro-computer systems proceed asynchronously to allow use of various types of memory and I/O devices with access times.
- If a device cannot be accessed specific amount of time it returns a 'not ready' to the 8257 that causes the 8257 to insert one or more wait states in its internet sequencing.
- Some devices are fast enough to be accessed without the use of wait states, but if they generate their READY response with the leading edge of the $\overline{I/OW}$ or $\overline{MEMW}$ signal (which generally occurs late in the transfer sequence), they would normally cause the 8257 to enter a wait state because it does not receive READY in time.
- For systems with these types of devices, the extended write option provides alternative timing for the I/O and memory write signals which allows the devices to return an early READY and prevents the unnecessary occurrence of wait states in the 8257, thus increasing system throughput.

4. TC stop

- In the TC STOP bit is set, a channel is disabled (i.e. its enable bit is reset) after the terminal count (TC) output goes true, thus automatically preventing further DMA operation on that channel.
- The enable bit for that channel must be re-programmed to continue or begin another DMA operation.
- If the TC STOP bit is not set, the occurrence of the TC output has no effect on the channel enable bits.
- In this case, it is generally the responsibility of the peripheral to cease DMA requests in order to terminate a DMA operation.

5. Auto load bit 7

- The auto load mode permits channel 2 to be used for repeat block or block chaining operations, without immediate software intervention between blocks.
- Channel 2 registers are initialized as usual for the first data block.
- Channel 3 registers, however are used to store the block re-initialization parameters (DMA starting address, terminal count and DMA transfer mode).

- After the first block of DMA cycles is executed by channel 2 (i.e. after the TC output goes true), the parameters stored in the channel 3 registers are transferred to channel 2 during an 'update' cycle. Note that the TC STOP feature, described above, has no effect on channel 2 when the auto load bit is set.
- If the auto load bit is set, the initial parameters for channel 2 are automatically duplicated in the channel 3 registers when channel 2 is programmed.
- This permits repeat block operations to be set up with the programming of a single channel. Repeat block operations can be used in applications such as CRT refreshing.
- Channels 2 and 3 can still be loaded with separate values if channel is loaded before loading channel 3.
- Note that in the auto load mode, channel 3 is still available to the user if the channel 3 enable bit is set, but use of this channel will change the values to be auto loaded into channel 2 at update time.
- All that is necessary to use the auto load feature for chaining operations is to reload channel 3 registers at the conclusion of each update cycle with the new parameters for the next data block transfer.
- Each time that the 8257 enters and update cycle, the update flag in the status register is set and parameters in channel 3 are transferred to channel 2, non-destructively for channel 3.
- The actual re-initialization of channel 2 occurs at the beginning of the next channel 2 DMA cycle after the TC cycle.
- This will be the first DMA cycle of the new data block for channel 2.
- The update flag is cleared at the conclusion of this DMA cycle. For chaining operations, the update flag in the status register can be monitored by the CPU to determine when the re-initialization process has been completed so that the next block parameters can be safely loaded into channel 3.

10.6 8257 DMA Operation State Diagram

Programming and reading the 8257 registers

- There are four pairs of "channel registers" each pair consisting of a 16-bit DMA address register and a 16-bit terminal count register (one pair for each channel).
- The 8257 also includes two "general registers" one 8-bit mode set register and one 8-bit status register.
- The registers are loaded or read when the CPU executes a write or read instruction that addresses the 8257 device generates the appropriate read or write control signal (generally I/OR or I/OW while the CPU places a 16-bit address on the system address bus and either outputs the data to be written onto the system data bus or accepts the data being read from the data bus.
- All or some of the most significant 12 address bits A_4 – A_{15} (depending on the systems memory, I/O configuration) are usually decoded to produce the chip select ($\overline{CS}$) input to the 8257.
- An I/O write input (or memory write in memory mapped I/O configurations, described below) specifies that the addressed register is to be programmed, while an I/O read input (or memory read) specifies that the addressed register is to be read address bit 3 specifies whether a "channel register" ($A_1 = 0$) or the mode set (program only)/Status (read only) register ($A_3 = 1$) is to be accessed. The least significant three address bits, A_0 – A_2 indicate the specific register to be accessed.
- When accessing the mode set or status register, A0 – A2 are all zero. When accessing a channel register bit A_0 differentiates between the DMA address register ($A_0 = 0$) and the terminal count register ($A_0 = 1$) four channels.

Control input	$\overline{CS}$	$\overline{I/OW}$	$\overline{I/OR}$	A_3
Program half of a channel register	0	0	1	0
Read half of a channel register	0	1	0	0
Program mode set register	0	0	1	1
Read status register	0	1	0	1

- Because the "channel registers" are 16-bits, two program instruction cycles are required to load or read an entire register.
- The 8257 contains a first/last (F/L) flip flop determines whether the upper or lower byte by the RESET input and whenever the Mode Set register is loaded.
- To maintain proper synchronization when accessing the "channel registers" all channel command instruction operations should occur in pairs, with the lower byte of a register always being accessed first.
- Do not allow $\overline{CS}$ to clock while either $\overline{I/OR}$ or $\overline{I/OW}$ is active, as this will cause an erroneous F/L flip flop state. In systems utilizing an interrupt structure, interrupts should be disabled prior to any paired programming operations to prevent an interrupt from splitting them.
- The result of such a split would leave the F/L F/F in the wrong state. This problem is particularly obvious when other DMA channels are programmed by an interrupt structure.

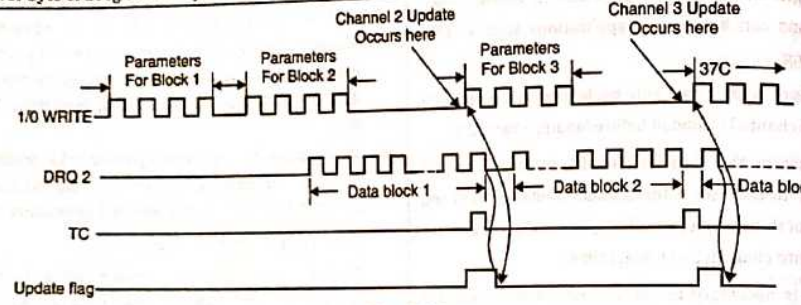


Fig. 10.6.1

8257 Register Selection :

Table 10.6.1

Register	BYTE	ADDRESS INPUTS A_3	A_2	A_1	A_0	F/L	DI-DIRECTIONAL DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CH-0 DMA Address	LSB	0	0	0	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	MSB	0	0	0	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-0 Terminal count	LSB	0	0	0	0	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	MSB	0	0	0	0	-1	Rd	Wr	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-1 DMA Address	LSB	0	0	0	0	0	Same as channel 0							
	MSB	0	0	0	0	1								
CH-2 Terminal count	LSB	0	0	0	0	0								
	MSB	0	0	0	0	1								
CH-2 DMA Address	LSB	0	0	0	0	0	Same as channel 0							
	MSB	0	0	0	0	1								
CH-1 Terminal count	LSB	0	0	0	0	0								
	MSB	0	0	0	0	1								
CH-3 DMA Address	LSB	0	0	0	0	0	Same as channel 0							
	MSB	0	0	0	0	1								

Register	BYTE	ADDRESS INPUTS A_3	A_2	A_1	A_0	F/L	DI-DIRECTIONAL DATA BUS D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CH-3 Terminal count	LSB	0	0	0	0	0								
	MSB	0	0	0	0	1								
MODE SET (Program only)	-	1	0	0	0	0	AL	TCS	EW	RP	En3	EN2	EN1	EN0
STAUTS (Read only)	-	1	0	0	0	1	0	0	0	UP	TC3	TC2	TC1	TC0

A_0-A_{15} : DMA Starting Address, C_0 - C_1 : Termal count value (N-1), **Rd and Wr** : DMA Verif (00), Write (01) or Read (10) cycle selection, **AL** : Auto Load, TCS, TC STOP, **EW** : Extended Write, **RP** : Rotating Prioroty, **EN3-EN0** : Channel Enable Mask Up Datae Flag, **TC3-TC0** : Terminal Count Status Bits

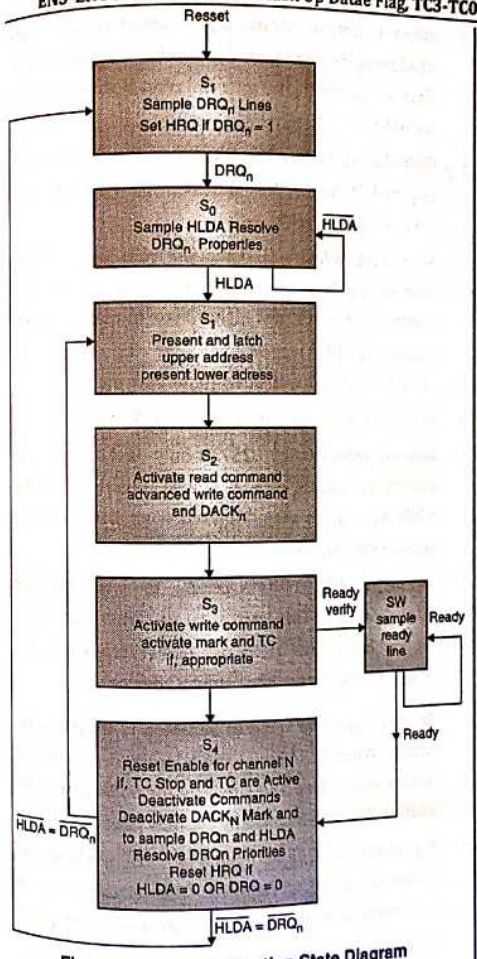


Fig. 10.6.2 : DMA Operation State Diagram

- Read and write commands and byte transfer occurs between the selected I/O device and memory.
- After the transfer is complete, the line is set HIGH and the HRQ line is set LOW to indicate to the CPU that the bus is now free for use.
- DRQ must remain HIGH until is issued to be recognized and must go LOW before S4 of the transfer sequence to prevent another transfer from occurring. (See timing diagram).

Consecutive transfers

- If more than one channel requests service simultaneously, the transfer will occur in the same way a burst does.
- No overhead is incurred by switching from one channel to another.
- In each S4 the DRQ lines are samples and the highest priority request is recognized during the next transfer. A burst mode transfer in a lower priority channel will be overridden by a higher priority request. Once the high priority transfer has completed control will return to the lower priority channel if its DRQ is still active. No extra cycles are needed to execute this sequence and the HRQ line remains active until all DRQ lines go LOW.

Control Override

- The continuous DMA transfer mode described above can be interrupted by an external device by lowering the HLDA line.
- After each DMA transfer the 8257 samples the HLDA line to insure that it is still active.

- If it is not active, the 8257 completes the current transfer, releases the HRQ line (LOW) and returns to the idle state.
- If DRQ lines are still active the 8257 will raise the HRQ line in the third cycle and proceed normally. (See timing diagram).

Not Ready

- The 8357 has a Ready input similar to the 8080A and the 8085A.
- The Ready line is sampled is State 3. If Ready is LOW the 8257 enters a wait state.
- Ready is sampled during every wait state. When ready returns HIGH the 8257 proceeds to State 4 to complete the transfer.
- Ready is used to interface memory or I/O devices that cannot meet the bus set up times required by the 8257.

Speed

- The 8257 uses four clock cycles to transfer a byte of data.
- No cycles are lost in the master to master transfer maximizing bus efficiency.
- A 2 MHz clock input will allow the 8257 to transfer at a rate of 500 K bytes/seconds.

Memory Mapped I/O Configurations

- The 8257 can be connected to the system bus as a memory device instead of as an I/O device for memory mapped I/O configurations by connecting the system memory control lines to the 8257's I/O control lines and the system. I/O control lines to the 8257's memory control lines.
- This configuration permits use of the 8080's considerably larger repertoire of memory instructions when reading or loading the 8257's registers. Note that with this connection, the programming of the Read (bit 15) and Write (bit 14) bits in the terminal count register will have a different meaning.

10.7 8257 Interfacing

Q. Interface DMA controller 8257 with 8086 microprocessor. (5 Marks)

Address generation

- The 8257 places lower byte of memory address on $A_0 - A_7$ lines and higher byte on $D_0 - D_7$ lines. It places higher address byte during S_1 of a DMA cycle.
- In block and demand transfer mode, the address generated will be sequential.
- When the borrow or carry is generated from lower byte of address, the 8257 places new value of higher byte on data bus. i.e. the higher byte will remain same for many transfers.
- Once the higher byte is latched then the state S_1 is not required. When the higher byte is constant the 8257 executes only S_2 S_3 and S_4 states.
- When borrow/carry is generated from lower address byte during decrement/increment address, the 8257 executes S_1 to latch new higher byte. In most cases, the length of DMA cycle will be 3S. Fig. 10.7.1 Interfacing 8257 with 8086 (minimum mode)
- Fig. 10.7.1 shows the interfacing of 8257 with 8086.
- Address generated by 8257 is only 16-bit; $A_0 - A_3$ is directly connected, $A_4 - A_7$ is also directly connected while $A_8 - A_{15}$ is demultiplexed from DB_0 to DB_7 as shown in the Fig. 10.7.1.
- The upper 4-bit address i.e. A_{16} to A_{19} is provided through a latch. This latch is to be written on by the programmers. This latch can be written with 8086 program, treating the latch as an IO device.
- Another important thing to be noted is $\overline{OE}$ pin of the latches. When 8257 issues the address, corresponding latches are enabled and similarly for 8086 issuing the address. This is controlled by the AEN pin of 8257.
- For interfacing 8257 with 8086 in maximum mode, the following circuit is required. This circuit is to be implemented as there is no HOLD and HLDA pin in maximum mode. Instead $\overline{RQ}/\overline{GT}$ is to be used.

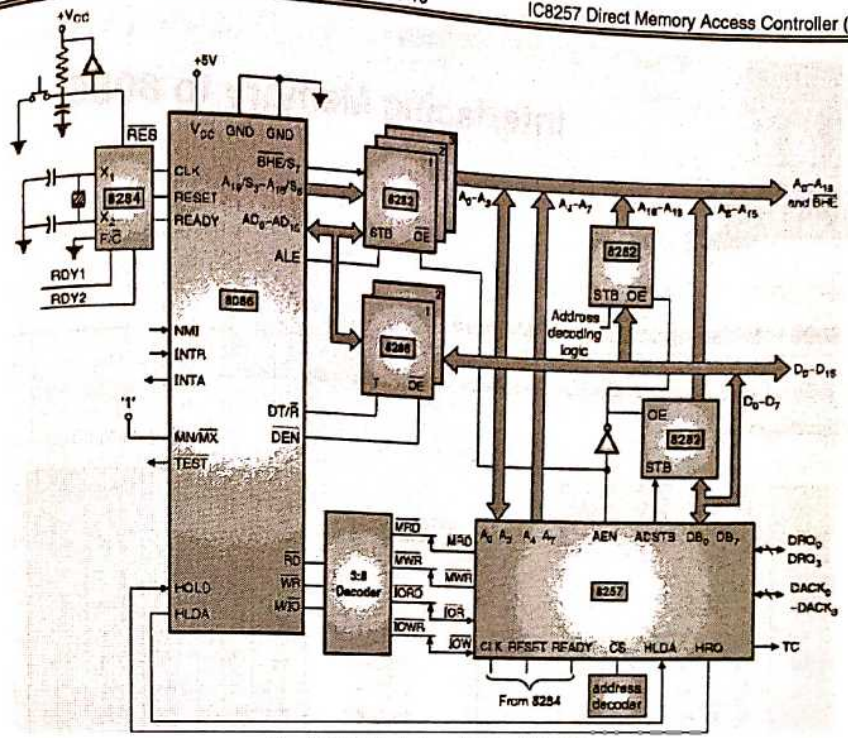


Fig. 10.7.1 : Interfacing of 8086 with 8257

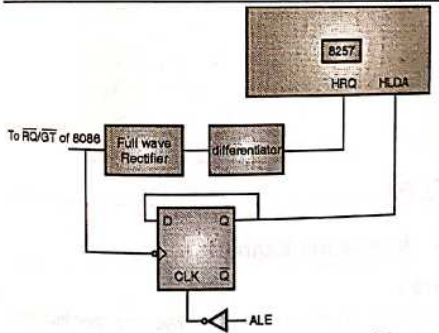


Fig. 10.7.2 : Interfacing 8257 with 8086 in maximum mode

10.8 Exam Pack (Review and University Questions)

Q. 1 Draw and explain the block diagram of 8257 DMA controller. **(Refer Section 10.1)** **(Dec. 18, 10 Marks)**

Q. 2 Write short note on DMA controller. **(Refer Section 10.1)** **(10 Marks)**

Q. 3 Draw and explain the block diagram of 8257 DMA controller. **(Refer Section 10.1)** **(5 Marks)**

Q. 4 Write short note on : Modes of DMA transfers. **(Refer Section 10.1.1)** **(5 Marks)**

Q. 5 Discuss different data transfer modes of DMA controller 8257. **(Refer Section 10.1.1)** **(5 Marks)**

Q. 6 Write short note on DMA controller. **(Refer Section 10.2)** **(10 Marks)**

Q. 7 Write short note on Pin Configuration of 8257. **(Refer Section 10.3)** **(10 Marks)**

Q. 8 Draw and explain the block diagram of 8257 DMA controller. **(Refer Section 10.4)** **(Dec. 18, 10 Marks)**

Q. 9 Explain various priority modes and transfer modes of DMA controller. **(Refer Section 10.5)** **(8 Marks)**

Q. 10 Interface DMA controller 8257 with 8086 microprocessor. **(Refer Section 10.7)** **(5 Marks)**

Q. 11 Explain data transfer modes of DMAC 8257 **(Refer Section 10.1.1)** **(Dec. 19, 5 Marks)**

□□□

11 Interfacing Memory to 8086

MODULE 3

11.1 8086 Interfacing to External Memory

The different memory chips available are listed below with their pin configuration.

Pin configuration

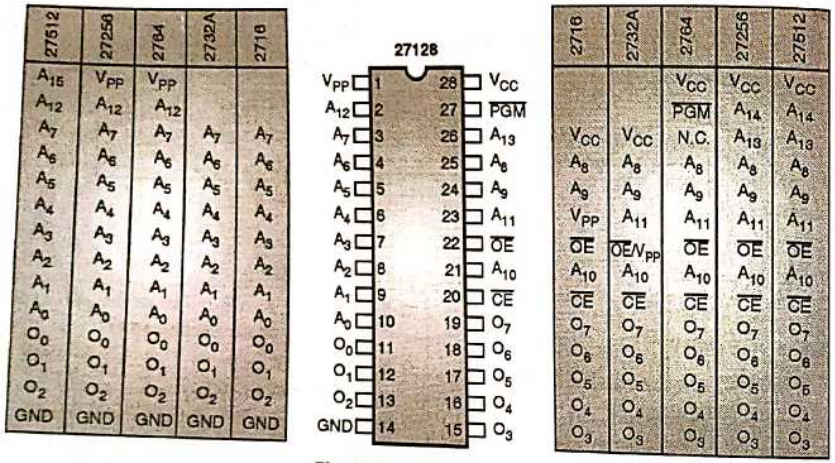
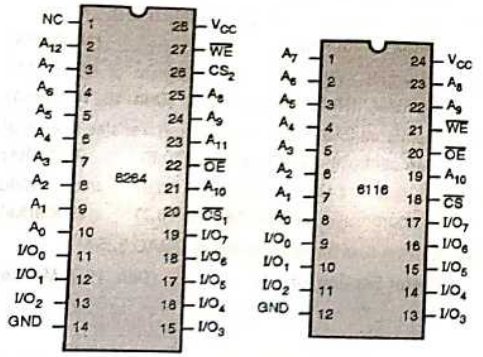


Fig. 11.1.1 : EPROM series

Pin configuration



(a) 6264 pin configuration (b) 6116 pin configuration

Fig. 11.1.2

11.2 Interfacing Examples

Ex. 11.2.1

Design a 8086 based system with the following specifications

(1) 8086 working at 10 MHz at minimum mode

(2) 32 KB EPROM using 16 KB devices

(3) 256 KB RAM using IC 62512

Soln. : Step 1 : Total EPROM required is equal to 32 KB

Chip size available = 16 KB

$\therefore$ Number of chips required $= \frac{32\text{ k}}{16\text{ k}} = 2$

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

SET 1 : Ending address = FFFFFH

Set size = chip size × 2

= 16 KB × 2 = 32 KB

32 KB = 2^{15} = 0000 0111, 1111, 1111, 1111

= 07FFF

Starting address = Ending address – SET size.

= FFFFFH – 07FFFH = F8000H

		Even Bank	Odd Bank
ROM	Starting Address	F8000H	F8001H
SET 1	Ending Address	FFFFEH	FFFFFH

Step 2 : Total RAM required is 256 KB

Chip size available = 64 KB (IC 62512 i.e. 512/8 = 64 KB)

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

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

SET 1 : Starting address = 00000 H

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

2^{17} = 128 KB

= 0001 1111 1111 1111 1111

1 F F F 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

Table P. 11.2.1

			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 $\bar{y}_0, \bar{y}_1, \bar{y}_2, \bar{y}_3$	EB	SA = 00000H	0 0 0 0 0	0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
		EA = 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 $\bar{y}_4, \bar{y}_5, \bar{y}_6, \bar{y}_7$	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 $\bar{y}_{31}$	EB	SA = F8000H	1 1 1 1 1	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 = F8001H	1 1 1 1 1	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

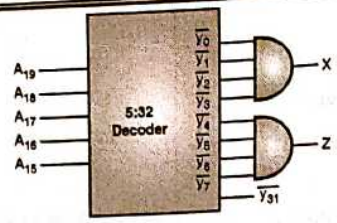


Fig. P. 11.2.1

Step 4 : Implementation

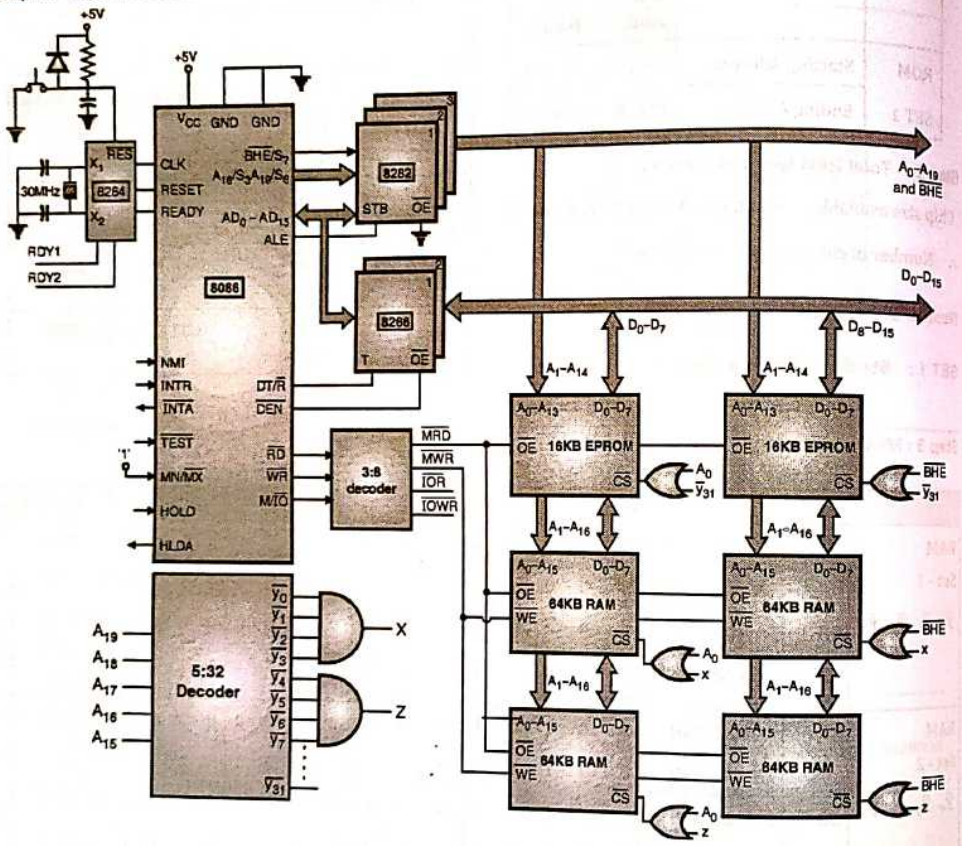


Fig. P. 11.2.1(a)

Ex. 11.2.2 : 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.

MU - Dec. 13, Dec. 16, 20 Marks

Soln. :

Step 1 : Total ROM required = 256 KB

Chip size available = 32 KB

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

$\therefore$ 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

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

$\therefore$ 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)

$\therefore$ 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)

$\therefore$ Ending Address = 60000H + 1FFFFH = 7FFFFH.

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

Step 3 : Memory Map

Table P. 11.2.2

			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 $\bar{y}_0 \cdot \bar{y}_1$	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	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 $\bar{y}_2 \cdot \bar{y}_3$	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	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
RAM Set 3 $\bar{y}_4 \cdot \bar{y}_5$	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	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	OB	SA = 40001H	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		EA = 5FFFFH	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM Set 4 $\bar{y}_6 \cdot \bar{y}_7$	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	0	1	1	1	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	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EPROM Set 4 $\bar{y}_{12}$	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	0	0	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	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EPROM Set 3 $\bar{y}_{13}$	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	0	1	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	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EPROM Set 2 $\bar{y}_{14}$	EB	SA = E0000H	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	OB	EA = EFFFEH	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
		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	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EPROM Set 1 $\bar{y}_{15}$	EB	SA = F0000H	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	OB	EA = FFFFEH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
		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	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 Even bank A_7	PA = 00H	0 0 0 0 0 0 0 0
	PB = 02H	0 0 0 0 0 0 1 0
	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 $\overline{A_7}$	Addr 1 = 81H	1 0 0 0 0 0 0 1
	Addr 2 = 83 H	1 0 0 0 0 0 1 1

Step 5 : Final Implementation

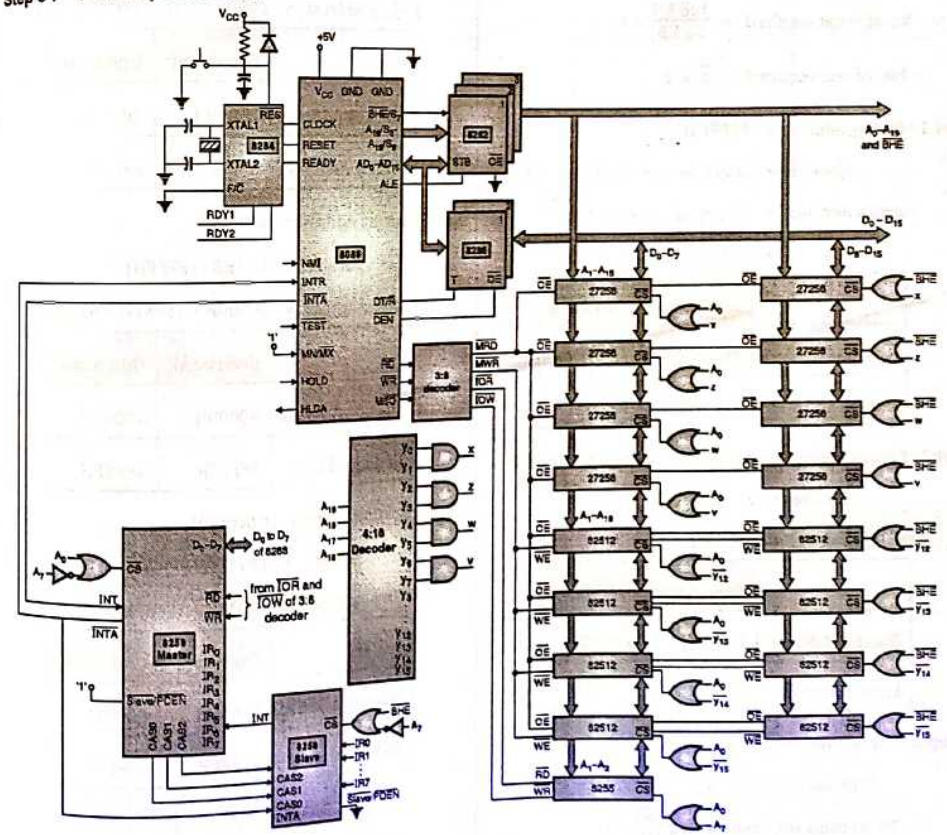


Fig. P. 11.2.2

Ex. 11.2.3

(a) Design 8086 based system in minimum mode system for following requirements:
- (i) 128 KB ROM using 32 KB × 8 memory device
- (ii) 512 KB RAM using 64 KB × 8 memory device

(b) Interface following I/O devices to system designed in (a)
- (i) Three 16 bit ports using 8255
- (ii) 15 interrupt support using 8259

MU - May 14, 20 Marks

Soln. :

Step 1 : Total ROM required = 128 KB

Chip size available = 32 KB

$\therefore$ No. of chips required = $\frac{128\ KB}{32\ KB} = 4$

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

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

Step 2 : Total RAM required = 512 KB

Chip size available = 64 KB

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

$\therefore$ 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)

$\therefore$ 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)

$\therefore$ Ending Address = 60000H + 1FFFFH = 7FFFFH.

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

Step 3 : Memory Map

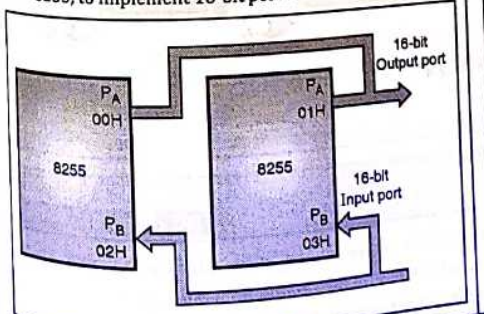
Here we use, partial decoding :

Table P. 11.2.3

			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	00000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		1FFFEH	0	0	0	1	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
		1FFFFH	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM Set 2 $\overline{y_2}\cdot\overline{y_3}$	EB	20000H	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		3FFFEH	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	OB	20001H	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		3FFFFH	0	0	1	1	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	40000H	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		5FFFEH	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	OB	40001H	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		5FFFFH	0	1	0	1	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	60000H	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		7FFFEH	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	OB	60001H	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		7FFFFH	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EPROM Set - 2 $\overline{y_{14}}$	EB	E0000H	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		EFFFEH	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	OB	E0001H	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		EFFFFH	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EPROM Set - 1 $\overline{y_{15}}$	EB	F0000H	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		FFFFEH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	OB	F0001H	1	1	1	1	0	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

Step 4 : I/O Map

- For 3 nos., 16-bit parallel port we need four chips of 8255, to implement 16-bit ports as shown below :



- 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 Even bank A_7	PA = 00H	0	0	0	0	0	0	0	0
	PB = 02H	0	0	0	0	0	0	1	0
	PC = 04H	0	0	0	0	0	1	0	0
	CW = 06H	0	0	0	0	0	1	1	0
8255 Odd bank A_7	PA = 01H	0	0	0	0	0	0	0	1
	PB = 03H	0	0	0	0	0	0	1	1
	PC = 05H	0	0	0	0	0	1	0	1
	CW = 07H	0	0	0	0	0	1	1	1

		A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
8255 Even bank	PA = 80H	1	0	0	0	0	0	0	0
	PB = 82H	1	0	0	0	0	0	1	0
	PC = 84H	1	0	0	0	0	1	0	0
$\overline{A_7}$	CW = 86H	1	0	0	0	0	1	1	0
8255 Odd bank	PA = 81H	1	0	0	0	0	0	0	1
	PB = 83H	1	0	0	0	0	0	1	1
	PC = 85H	1	0	0	0	0	1	0	1

		A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
$\overline{A_7}$	CW = 87H	1	0	0	0	0	1	1	1
8259 Even bank	Addr 1 = C0 H	1	1	0	0	0	0	0	0
	Addr 2= C2H	1	1	0	0	0	0	1	0
Odd bank	Addr 1 = C1H	1	1	0	0	0	0	0	1
$\overline{A_6}$	Addr 2 = C3 H	1	1	0	0	0	0	1	1

Step 5 : Final Implementation

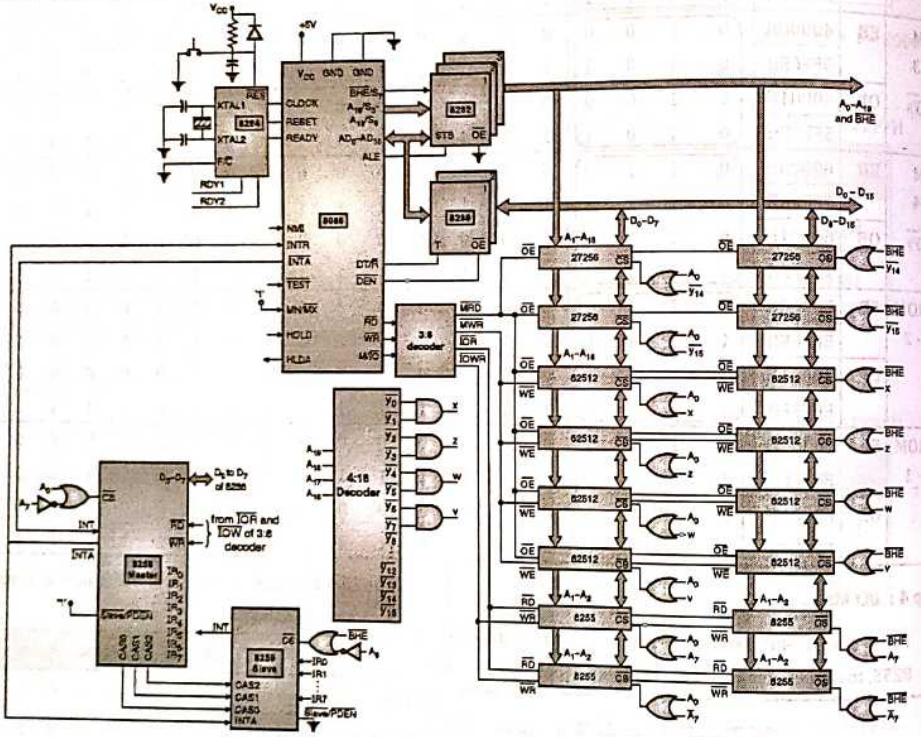


Fig. P. 11.2.3

11.3 Decoding Techniques

University Question

Q. Explain in brief address decoding techniques. **MU - Dec. 11, 5 Marks**

- After deciding the arrangement of memory modules to produce the desired memory storage, the user has to design an **address decoding circuit** for it. The address decoding circuit will receive the address from microprocessors address bus and convert it to chip select signals required for selecting each memory module or I /O device.

- A number of lines are connected to the address decoder as inputs that are coming from the microprocessor. Generally, these lines are the higher order address lines.
- The number of lines required for address decoding depends on the number of memory modules.

11.3.1 Address Decoding Techniques

The various address decoding techniques are :

(I) Absolute Decoding

- Fig. 11.3.1 shows the memory interface with absolute decoding technique.

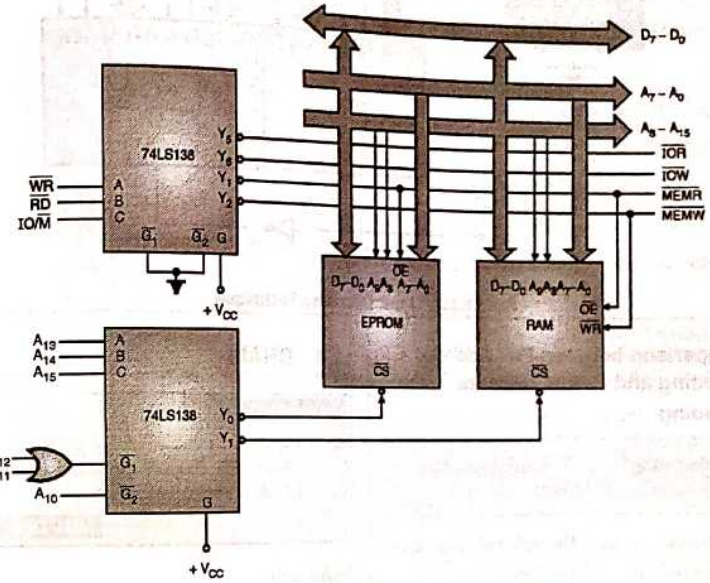


Fig. 11.3.1 : Absolute Decoding Technique / Full Decoding

- In absolute decoding technique, the high order address lines are decoded to select the memory chip. The memory chip is selected for specified logic levels on the high order address lines. Other logic levels cannot select the chip
- It is also called Full Decoding Technique.

  Such an addressing scheme is used for large memory systems.

(II) Linear Decoding

- In case of small systems, the hardware for the decoding logic can be eliminated by using individual higher order address lines to select memory chips. This is technique is also called **partial decoding.**
- This reduces the cost of decoding circuit, but it has a drawback of multiple addresses (shadow addresses).
- Fig. 11.3.2 shows the addressing of RAM and EPROM with linear decoding technique.
- A_{13} address line, is directly connected to chip select signal of the RAM. So when the status of the A_{13} line is zero, the EPROM gets selected and when the status of A_{13} line is one, the RAM gets selected.

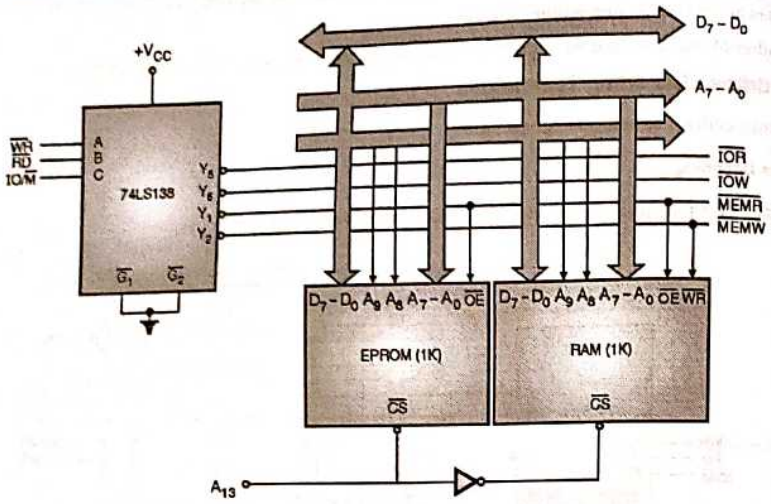


Fig. 11.3.2 : Linear Decoding Technique

11.3.2 Comparison between Full Address Decoding and Partial Address Decoding

Sr. No.	Full decoding	Partial decoding
1.	Also referred to as absolute decoding	Also referred to as linear decoding
2.	All address lines are considered	Few address line(s) are ignored.
3.	No shadowing or wrap around effect	Shadow or wrap around, of interfaced device will be present
4.	More hardware required for decoding the number of bits.	Decoder hardware is simple
5.	Flexible size(s) of the memory(s) may be interfaced	Flexible size(s) of the memory(s) may be interfaced

11.4 DRAM

University Question

Q. Explain the need of DRAM controller for interfacing DRAM with 8086. Draw and explain interfacing of DRAM controller with 8086.

MU - Dec. 16, 10 Marks

DRAM cell

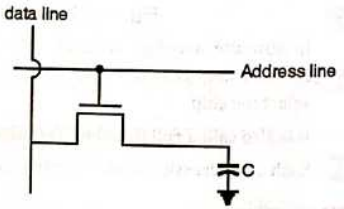


Fig. 11.4.1

Operation

When a particular address is issued, the corresponding address line is enabled switching on the MOSFET. This connects the capacitor to data line 8.

∴ The data can be read from or written to the capacitor.

Advantages of DRAM

- Smaller size / more dense on the chip
- Cheaper (cost is less)

Disadvantage

It requires refreshing as capacitor looses the charge frequently.

IC 8203 (DRAM Controller)

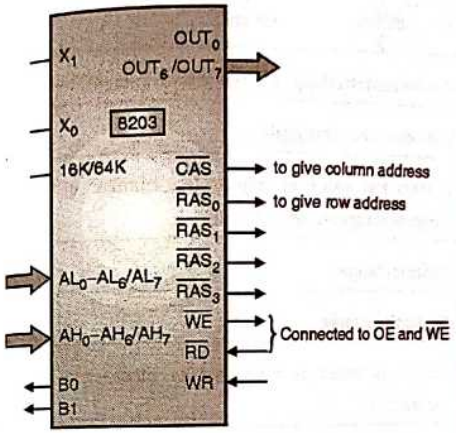


Fig. 11.4.2

Functions

(1) Power supply pins
- VCC
- GND

(2) Clock pins
- X_1
- X_0

Either a crystal can be connected across this pin or clock can be directly given to X_1 pin.

(3) B_0 B_1 (Bank select pins) : These 2 pins are used to select the bank of chips connected on I/O-address bus. For 64 K mode only B_0 is available.

∴ 2 banks in 64 K mode while 4 banks in 16 K mode.

(4) Mode select pin

16 K / $\overline{64}$ K

The pin is used to select mode of operation i.e. 16 KB mode or 64 KB mode.

(5) Input address pins :

AL_0-AL_6 / AL_7

These are lower address lines AL_0-AL_6 are available in 16 K mode while AL_0-AL_7 in 64 KB mode.

AH_0-AH_6 AH_7

These are higher order address lines. AH_0-AH_6 are available in 16 K mode while AH_0-AH_7 in 64 K mode

(6) Output address pins :

OUT_0 - OUT_6 / OUT_7

These address lines carry the row address or column address

OUT_0 - OUT_6 are available in 16 K mode

OUT_0 - OUT_7 are available in 64 K mode.

$\overline{CAS}$ (Column address select)

This is used to select a particular column indicating the address given on OUT_0 – OUT_6 / OUT_7 is a column address.

$\overline{RAS}$ (Row address select) : $\overline{RAS_0}$ – $\overline{RAS_3}$

These pins are used to indicate that the address given OUT_0 – OUT_6 / OUT_7 is a row address and also it indicates for which row.

(In $\overline{64}$ K mode only $\overline{RAS_0}$ and $\overline{RAS_1}$ are available) there are 4 rows possible ∴ $\overline{RAS_0}$ to $\overline{RAS_3}$

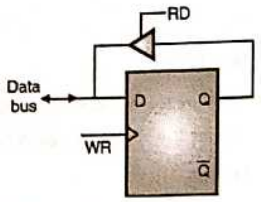
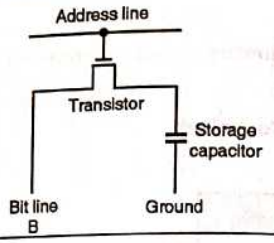
(7) Control signal :

$\overline{RD}$: This signal is given from 8086 to indicate the processor wants to read.

$\overline{WR}$: This signal is also given by 8086 to indicate it wants to write.

$\overline{WE}$: This signal is given from 8203 to indicate the processor wants to write to DRAM.

11.4.1 Difference between SRAM and DRAM

Sr. No.	SRAM	DRAM
1.		
2.	No refreshing required.	Continuous refreshing required (disadvantage).
3.	It is faster for accessing data.	It is slower in accessing data.
4.	It takes more space on chip as more number of components are required per bit.	It takes less space on ship as less number of components are required per bit.
5.	Hence is also costly.	Hence is cheaper.
6.	Bit density is lesser.	Bit density is more.
7.	The bit is stored in a flip-flop	The bit is stored as a charged or discharged capacitor.
8.	Mainly used in Cache memory.	Mainly used for semiconductor main memory.

11.5 Memory Mapped I/O and I/O Mapped I/O

University Question

Q. Differentiate between the Memory mapped I/O and I/O mapped I/O. (Dec. 11, May 12, Dec. 12, 5 Marks)

Sr. No.	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.

Ex. 11.5.1 : 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 **MU - May 16, 12 Marks**

Soln. :

Step 1 : Total EPROM required is equal to 128 KB

Chip size available = 64 KB

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

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

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 = $\frac{\text{no. of chips}}{\text{no. of banks}} = \frac{4}{2} = 2$

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 $\bar{y}_0$	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	1
	OB	SA = 00001H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		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 $\bar{y}_1$	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	1
	OB	SA = 20001H	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		EA = 3FFFFH	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	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
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	1
	OB	SA = E0001H	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		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.

			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

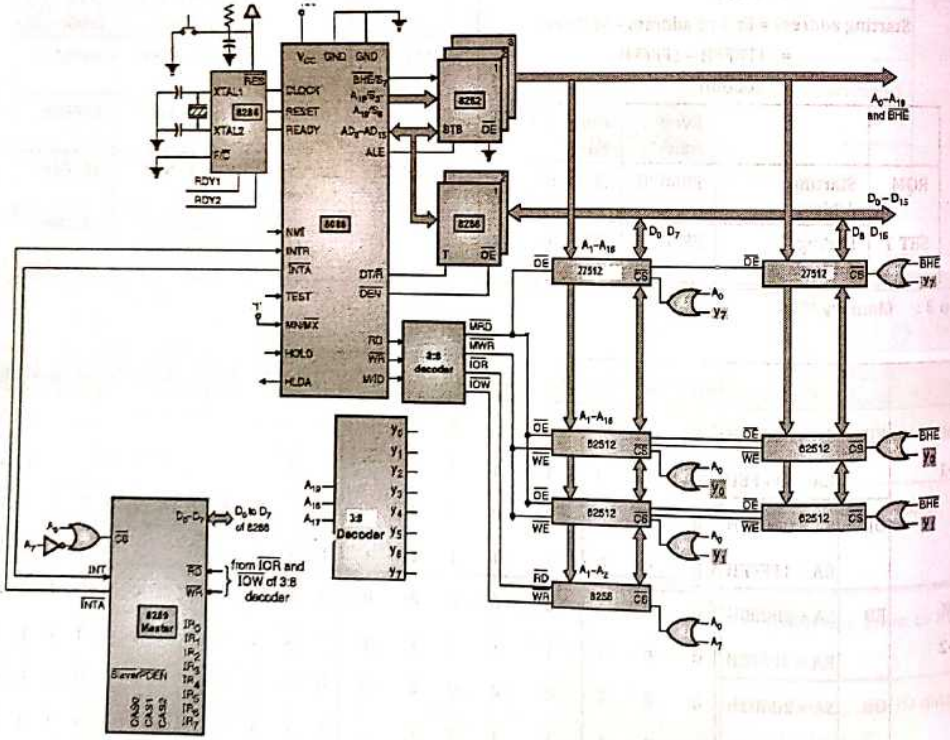


Fig. P. 11.5.1

Ex. 11.5.2 : Interface 2 sets of 32 KB DRAM with 8203

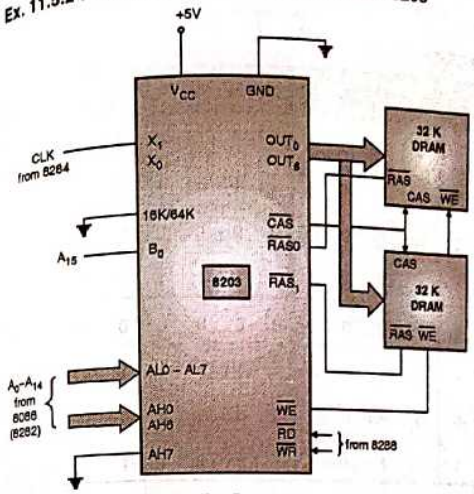


Fig. P. 11.5.2

Ex. 11.5.3 : Interface 4 sets of 8 KB DRAM

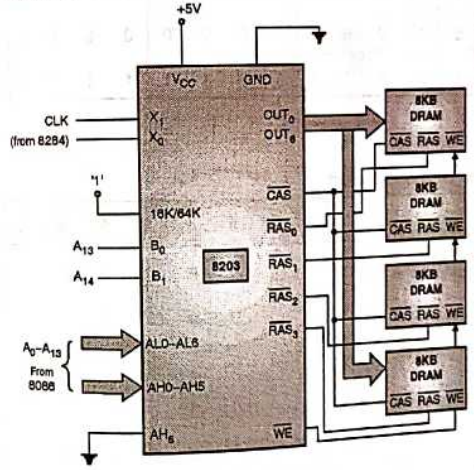


Fig. P. 11.5.3

Ex. 11.5.4 : Design a 8086 based system consisting of the following :

(a) 8086 microprocessor working at 8 MHz.

(b) EPROM of 64 KB using 32 KB devices.

(c) SRAM of 64 KB using 16 KB devices.

(d) 1 input, 1 output port (both 16-bit) interrupt driven.

Explain the design. Also show the memory and I/O map.

MU - May 11, 20 Marks

Soln. :

Step 1 : Total EPROM required = 64 KB

Chip size available = 32 KB

∴ No. of chips required = 2

∴ No. of sets required = $\frac{2}{2} = 1$

Set 1 : Ending address = FFFFFH

Set size = 32 KB × 2 = 64 KB ⇒ 0FFFFH

∴ Starting address = FFFFF – 0FFFF = F0000H

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

Step 2 : Total SRAM required = 64 KB

Chip size available = 16 KB

∴ No. of chips required = 4

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

Set 1 : Starting Address = 00000H

Set size = 16 KB * 2 = 32 KB ⇒ 07FFFH

∴ Ending Address = 00000H + 07FFFH = 07FFFH

	Even Bank	Odd Bank
Starting Address	00000H	00001H
Ending Address	07FFEH	07FFFH

Set 2 : Starting Address =08000H

Set size = 16 KB * 2 = 32 KB

⇒ 07FFFH

∴ Ending Address = 08000H + 07FFH

= 0FFFFH

	Even Bank	Odd Bank
Starting Address	08000H	08001H
Ending Address	0FFFEH	0FFFFH

Step 3 : Memory Map

Table P. 11.5.4

			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 = 07FFEH	0	0	0	0	0	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 = 07FFFH	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM Set - 2 $\overline{y}_1$	EB	SA = 08000H	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		EA = 0FFFEH	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	OB	SA= 080001H	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		EA = 0FFFFH	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EPROM Set - 1 $\overline{y}_{30}$ $\overline{y}_{31}$	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	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	1	1	1	1

Step 4 : I/O Map

Fig. P. 11.5.4 shows the implementation of a 16-bit input and 16-bit output port.

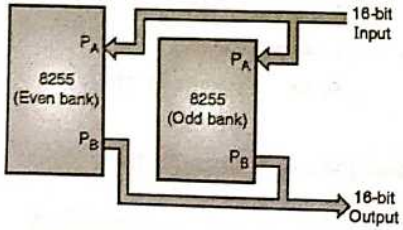


Fig. P. 11.5.4

			A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
8255 Set 1	EB	PA = 00H	0	0	0	0	0	0	0	0
		PB = 02H	0	0	0	0	0	0	1	0
		PC = 04H	0	0	0	0	0	1	0	0
		CW = 06H	0	0	0	0	0	1	1	0
	OB	PA = 01H	0	0	0	0	0	0	0	1
		PB = 03H	0	0	0	0	0	0	1	1
		PC = 05H	0	0	0	0	0	1	0	1
		CW = 07H	0	0	0	0	0	1	1	1

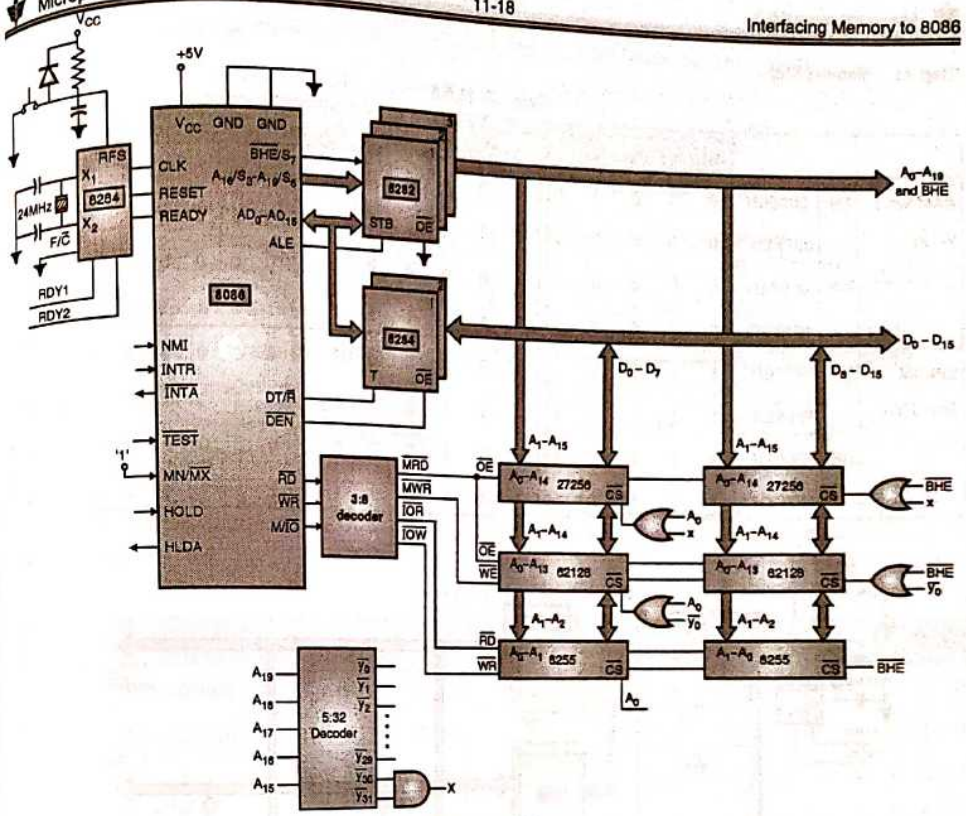


Fig. P. 11.5.4(a)

Ex. 11.5.5 : 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.

MU - May 13, Dec. 15, 10 Marks

Soln. : 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.

∴ number of chips = 2.

∴ 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. P. 11.5.5

			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 $\bar{y}_0 \cdot \bar{y}_1 \cdot$	EB	00000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		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 Set - 1 $\bar{y}_{31}$	EB	F8000H	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		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

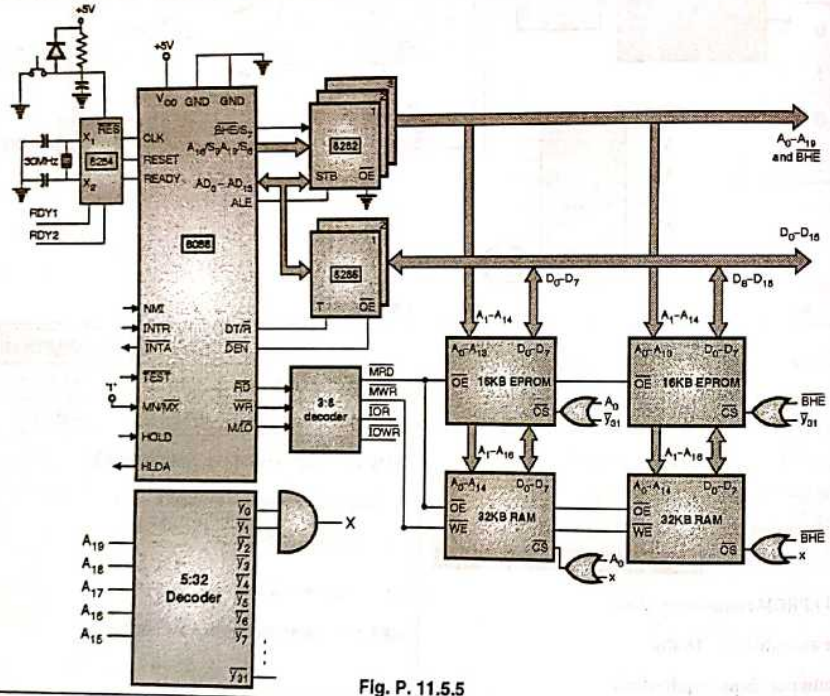


Fig. P. 11.5.5

Ex. 11.5.6 : Design 8086 based system with following specifications :

(a) Interface 32 KB SRAM. Use IC 6264.

(b) Interface 16 KB DRAM. Use 8 KB chips.

MU - Dec. 11, 10 Marks

Soln. : Step 1 : Total SRAM required = 32 KB

Chip size available = 8 KB (IC 6264)

∴ No of chips required = 4

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

Set 1 : Starting address = 00000 H

Set size = 8 KB × 2 = 16 KB = 03FFFH

∴ Ending address = 03FFFH

	Even bank	Odd bank
Starting address	00000H	00001H
Ending address	03FFEH	03FFFH

Set 2 : Starting address = 04000H

Set size = 16 KB = 03FFFH

Ending address = 07FFFH

	Even bank	Odd bank
Starting address	04000H	04001H
Ending address	07FFEH	07FFFH

Step 2 : Total DRAM required = 16 KB

Chip size available = 8 KB

∴ No of chips required = $\frac{16\text{ KB}}{8\text{ KB}}$ = 2

No. of sets = $\frac{2}{2}$ = 1

Set 1 : Starting address = 08000H

Set size = 8 KB × 2 = 16 KB = 03FFFH

∴ Ending address = 0BFFFH

	Even bank	Odd bank
Starting address	08000H	08001H
Ending address	0BFFEH	0BFFFH

Step 3 : Memory map

Table P. 11.5.6

			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
SRAM Set-1 $\bar{y}_0$	EB	SA = 00000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		EA = 03FFEH	0	0	0	0	0	0	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	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
SRAM Set - 2 $\bar{y}_1$	EB	SA = 04000H	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		EA = 07FFEH	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	OB	SA = 04001H	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		EA = 07FFFH	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
DRAM Set - 1 $\bar{y}_2$	EB	SA = 08000H	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		EA = 0BFFEH	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0
	OB	SA = 08001H	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
		EA = 0BFFFH	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Ex. 11.5.7 : Design an 8086 based system with the following specifications.

(i) 8086 is in minimum mode

(ii) 64 Kbyte EPROM using 52 KB devices

(iii) 64 Kbyte RAM using 32KB devices.

Draw the complete schematic of the design indicating address map.

MU - Dec. 12, 10 Marks

Soln. :

Note : Since there is no chip of 52 KB, assumption is made that the chip is of 32 KB

Step 1 : Total EPROM required is equal to 64 KB

Chip size available = 32 KB

$\therefore$ Number of chips required $= \frac{64\ KB}{32\ KB} = 2$

$\therefore$ Number of sets required $= \frac{\text{Number of Chips}}{\text{Number of Banks}} = \frac{2}{2} = 1$

SET 1 : Ending address = FFFFFH

Set size = chip size × 2 = 32 KB × 2 = 64 KB

64 KB = 2^{16} = 0000 1111 1111 1111 1111

= 0FFFF

Starting address = Ending address – SET size.

= FFFFFH – 0FFFFH = F0000H

		Even Bank	Odd Bank
ROM SET 1	Starting Address	F0000H	F0001H
	Ending Address	FFFFEH	FFFFFH

Step 2 : Total RAM required is 64 KB

Chip size available = 32 KB

$\therefore$ Number of chips required $= \frac{64\ KB}{32\ KB} = 2$

Number of sets required $= \frac{\text{no. of chips}}{\text{no. of banks}} = \frac{2}{2} = 1$

SET 1 : Starting address = 00000 H

Set size = chip size × 2 = 32 K × 2 = 64 KB

2^{16} = 64 KB

$= \frac{0000}{0}\ \frac{1111}{F}\ \frac{1111}{F}\ \frac{1111}{F}\ \frac{1111}{F}$

Ending address = starting address + SET size.

= 00000H + 0FFFFH

= 0FFFFH = 3FFFFH

		Even bank	Odd bank
RAM SET 1	Starting Address	00000H	00001H
	Ending Address	0FFFEH	0FFFFH

Step 3 : Memory Map

Table P. 11.5.7

			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 = 0FFFEH	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 = 0FFFFH	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ROM 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	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	1	1	1	1

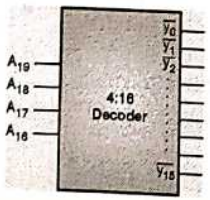


Fig. P. 11.5.7

Step 4 : Implementation

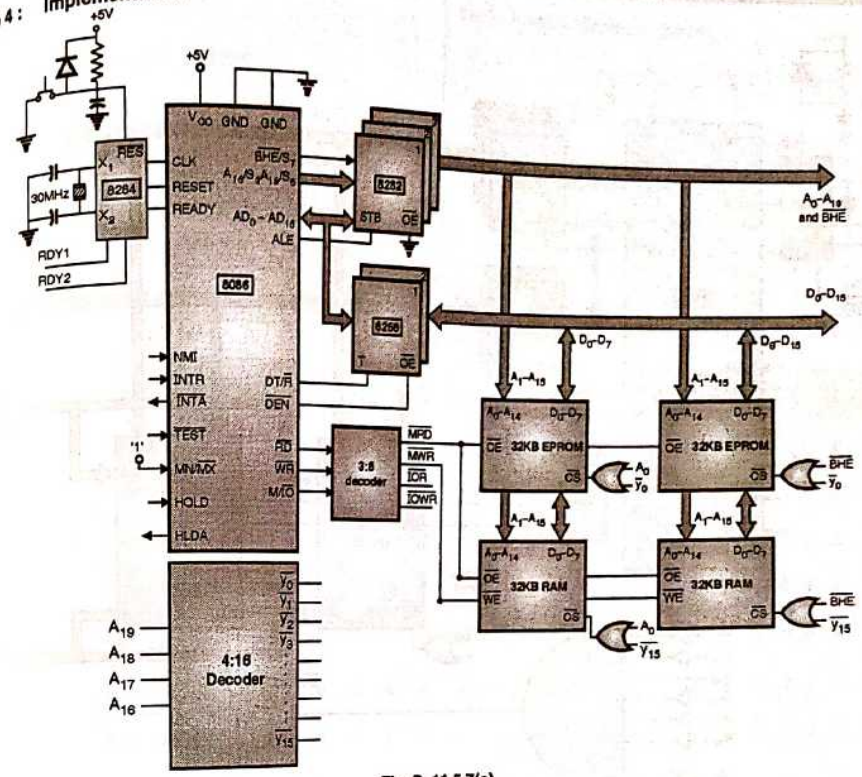


Fig. P. 11.5.7(a)

Ex. 11.5.8 : Design an interface for 8255 with the following requirements : 1, 16-bit input and 1, 16-bit output port. Starting address is 2000 H. **MU - Dec. 11, 5 Marks**

Soln. :

Table P. 11.5.8

			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
8255 Set - 1	Even bank	PA = 2000 H	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
		PB = 2002 H	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0
		PC = 2004 H	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0
		CW = 2006 H	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0
	Odd bank	PA = 2001 H	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
		PB = 2003 H	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1
		PC = 2005 H	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1
		CW = 2007 H	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1

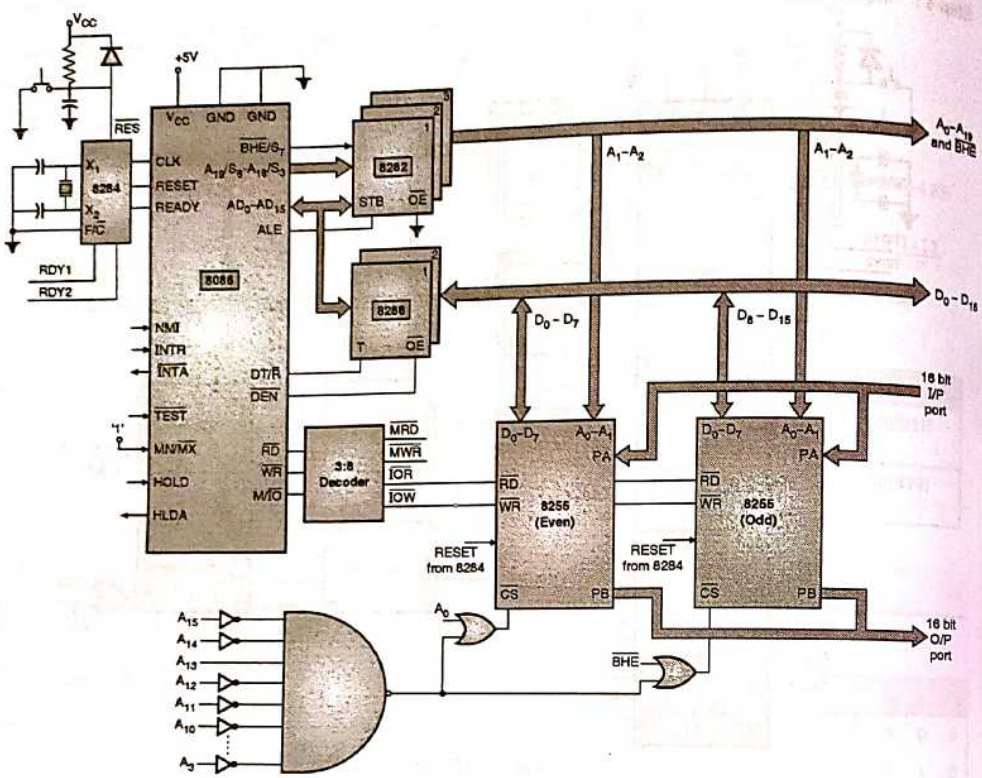


Fig. P. 11.5.8

Ex. 11.5.9 : Design 8086 microprocessor based system with following specifications

(a) Microprocessor 8086 working at 10 MHz in minimum mode

(b) 32 KB EPROM using 8 KB chips

(c) 16 KB SRAM using 4 KB chips

Explain the design along with memory address map.

MU - Dec. 14, 10 Marks

Soln. :

Step 1 : Total EPROM required = 32 KB

Chip size available = 8 KB (IC 2764)

∴ No of chips required = 4

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

Set 1 : Ending address = FFFFF H

Set size = 8 KB × 2 = 16 KB = 03FFFH

∴ Starting address = FC000H

	Even bank	Odd bank
Starting address	FC000H	FC001H
Ending address	FFFFEH	FFFFFH

Set 2 : Ending address = FBFFFH

Set size = 16 KB = 03FFFH

Starting address = F8000H

	Even bank	Odd bank
Starting address	F8000H	F8001H
Ending address	FBFFEH	FBFFFH

Step 2 : Total SRAM required = 16 KB

Chip size available = 4 KB

∴ No of chips required = $\frac{16}{4}$ = 4

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

Set 1 : Starting address = 00000H

Set size = 4 KB × 2 = 8 KB = 01FFFH

∴ Ending address = 01FFFH

	Even bank	Odd bank
Starting address	00000H	00001H
Ending address	01FFEH	01FFFH

Set 2 : Starting address = 02000H

Set size = 4 KB × 2 = 8 KB = 01FFFH

∴ Ending address = 03FFFH

	Even bank	Odd bank
Starting address	02000H	02001H
Ending address	03FFEH	03FFFH

Step 3 : Memory map

Table P. 11.5.9

			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
SRAM Set-1 $\bar{y}_0$	EB	SA = 00000H	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0
		EA = 01FFEH	0 0 0 0 0 0 0 0	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 = 01FFFH	0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1 1 1
SRAM Set-2 $\bar{y}_1$	EB	SA = 02000H	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0
		EA = 03FFEH	0 0 0 0 0 0 0 1	1 1 1 1 1 1 1 1 1 1 1 0
	OB	SA = 02001H	0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 1
		EA = 03FFFH	0 0 0 0 0 0 0 1	1 1 1 1 1 1 1 1 1 1 1 1
DRAM Set-1 $\overline{y_{124} \cdot y_{125}}$	EB	SA = F8000H	1 1 1 1 1 0 0	0 0 0 0 0 0 0 0 0 0 0 0
		EA = FBFFEH	1 1 1 1 1 0 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 0 1
		EA = FBFFFH	1 1 1 1 1 0 1	1 1 1 1 1 1 1 1 1 1 1 1
DRAM Set-1 $\overline{y_{126} \cdot y_{127}}$	EB	SA = FC000H	1 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 1 0
	OB	SA = FC001H	1 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 1

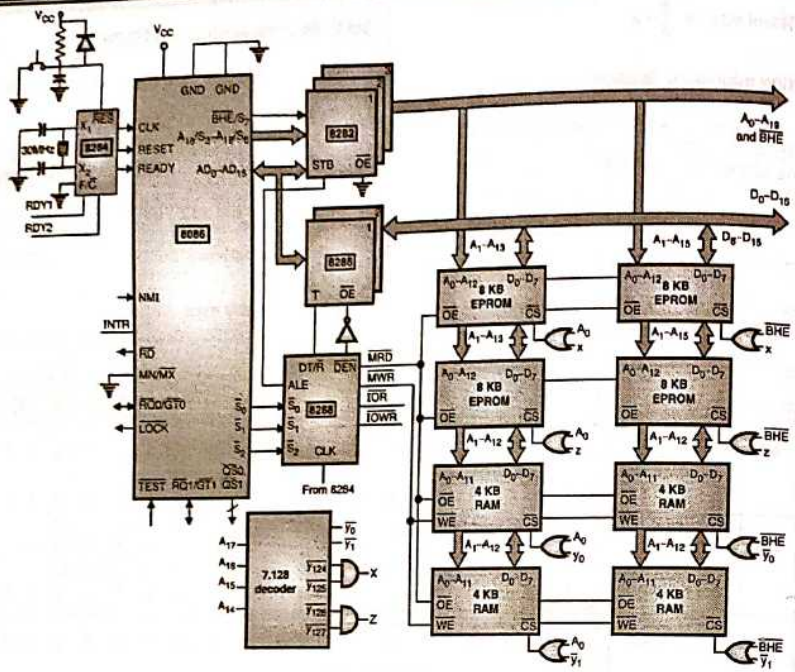


Fig. P. 11.5.9

Ex. 11.5.10 : Design 8086 microprocessor based system with following specification.

(a) Microprocessor 8086 working at 8 MHz in maximum mode

(b) 32 KB EPROM using 16 KB chips

(c) 16 KB SRAM using 8 KB chips

Explain the design along with memory address map.

MU - May 15, 10 Marks

Soln. :

Step 1 : Total EPROM required = 32 KB

Chip size available = 16 KB

∴ No. of chips required = 2 chips.

∴ No. of sets required = 1 set

Set 1 : Ending address = FFFFFH.

Set size = Chip size * 2 = 32 KB

⇒ 07FFFH

∴ Starting address = F8000H

	Even Bank	Odd Bank
Starting Address	F8000H	F8001H
Ending Address	FFFFEH	FFFFFH

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

⇒ 03FFFH.

∴ Ending address = 03FFFH

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

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 = 03FFEH	0	0	0	0	0	0	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	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EPROM set 1 $\overline{y_{62}} \cdot \overline{y_{63}}$	EB	SA = F8000H	1	1	1	1	1	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 = F8001H	1	1	1	1	1	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 : Final Implementation

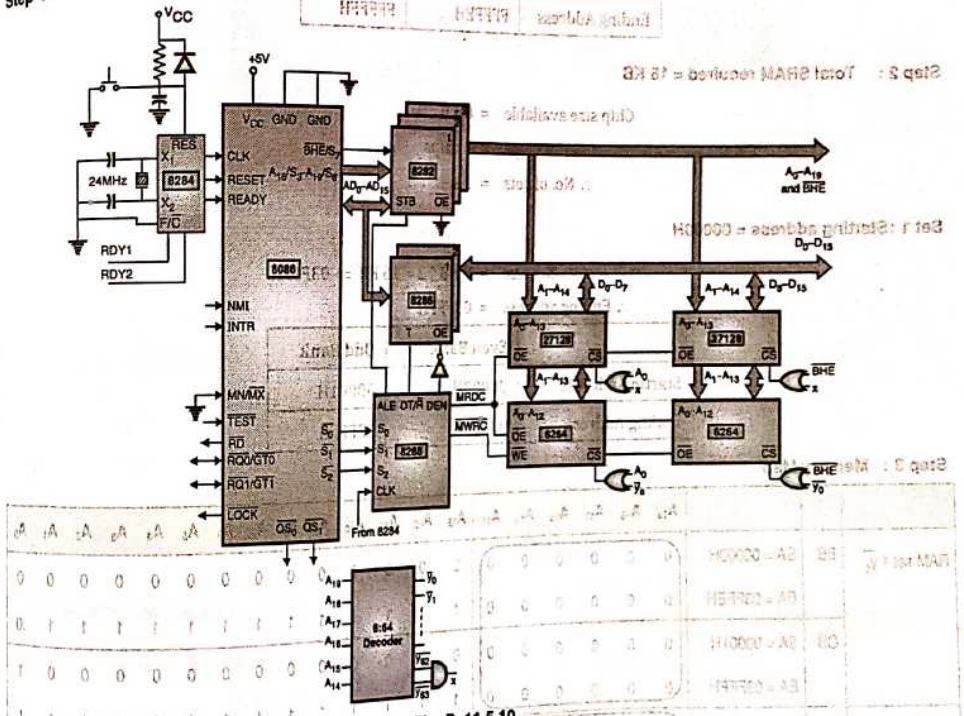


Fig. P. 11.5.10

Example 11.5.11 : Design 8086 microprocessor based system with following specification.

(a) Microprocessor 8086 working at 8 MHz in maximum mode

(b) 32 KB EPROM using 16 KB chips

(c) 16 KB SRAM using 8 KB chips

Explain the design along with memory address map.

MU - May 15, 10 Marks

Soln. :

Step 1 : Total EPROM required = 32 KB

Chip size available = 16 KB

∴ No. of chips required = 2 chips.

∴ No. of sets required = 1 set

Set 1 : Ending address = FFFFFH.

Set size = Chip size * 2 = 32 KB ⇒ 07FFFH

∴ Starting address = F8000H

	Even Bank	Odd Bank
Starting Address	F8000H	F8001H
Ending Address	FFFFEH	FFFFFH

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 ⇒ 03FFFH.

∴ Ending address = 03FFFH

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

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 = 03FFEH	0	0	0	0	0	0	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	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EPROM set 1 $\overline{y_{62}} \cdot \overline{y_{63}}$	EB	SA = F8000H	1	1	1	1	1	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 = F8001H	1	1	1	1	1	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 : Final Implementation

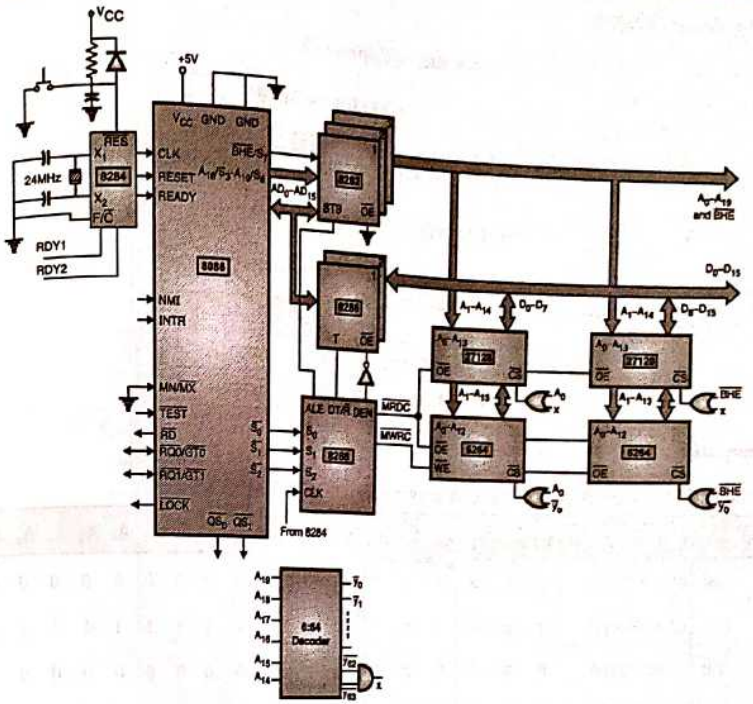


Fig. 11.5.11

Example 11.5.12 : Design 8086 microprocessor based system with following specification :

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

(b) 64 KB EPROM using 16 KB*4 chips

(c) 16 KB RAM using 8 KB*2 chips

MU - Dec. 18, 10 Marks

Soln. :

Step1 : Total EPROM required = 16 KB

Chip size available = 8 KB (IC 2764 i.e. 8 kB EPROM)

Number of chips required $= \frac{16\ KB}{8\ KB} = 2$

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

$= \frac{2}{2} = 1$

SET 1 : Ending address of SET 1 = FFFFFH

SET size = Chip size × 2

= 8 KB × 2 = 16 KB 16 KB ⇒ 2^{14}

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

Starting address = Ending address – SET size.

= FFFFFH – 03FFF = FC000H

		Even bank	Odd bank
ROM SET 1	Starting Address	FC000H	FC001H
	Ending Address	FFFFEH	FFFFFH

Step 2 : Total RAM required is 64 KB

Chip size available is 32 KB (IC 62256 i.e. 32 kB RAM)

∴ Number of chips required $= \frac{64\ KB}{32\ KB} = 2$

∴ Number of sets required $= \frac{no.\ of\ chips}{no.\ of\ banks}$

$= \frac{2}{2} = 1$

SET 1 : Starting address = 00000 H

Set size = Chip size × 2

= 32 KB × 2 = 64 KB

i.e. 16 KB = 2^{16} 0000 1111 1111 1111 1111

0 F F F F

Ending address = Starting address + SET Size

= 00000H + 0FFFFH = 0FFFFH

		Even bank	Odd bank
RAM SET 1	Starting Address	00000H	00001H
	Ending Address	0FFFEH	0FFFFH

Step 3 : Memory Map :

EA ⇒ Ending address, SA ⇒ Starting address, EB ⇒ Even Bank, OB ⇒ 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 $\bar{y}_6 \cdot \bar{y}_1 \cdot \bar{y}_2 \cdot \bar{y}_3$	EB	SA = 00000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		EA = 0FFFEH	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 = 0FFFFH	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ROM Set-1 $\bar{y}_7$	EB	SA = FC000H	1	1	1	1	1	1	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 = FC001H	1	1	1	1	1	1	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

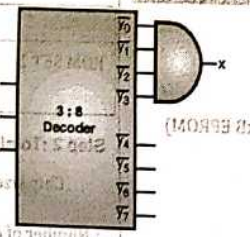


Fig. P. 11.5.12(a)

Step 4: Final Implementation :

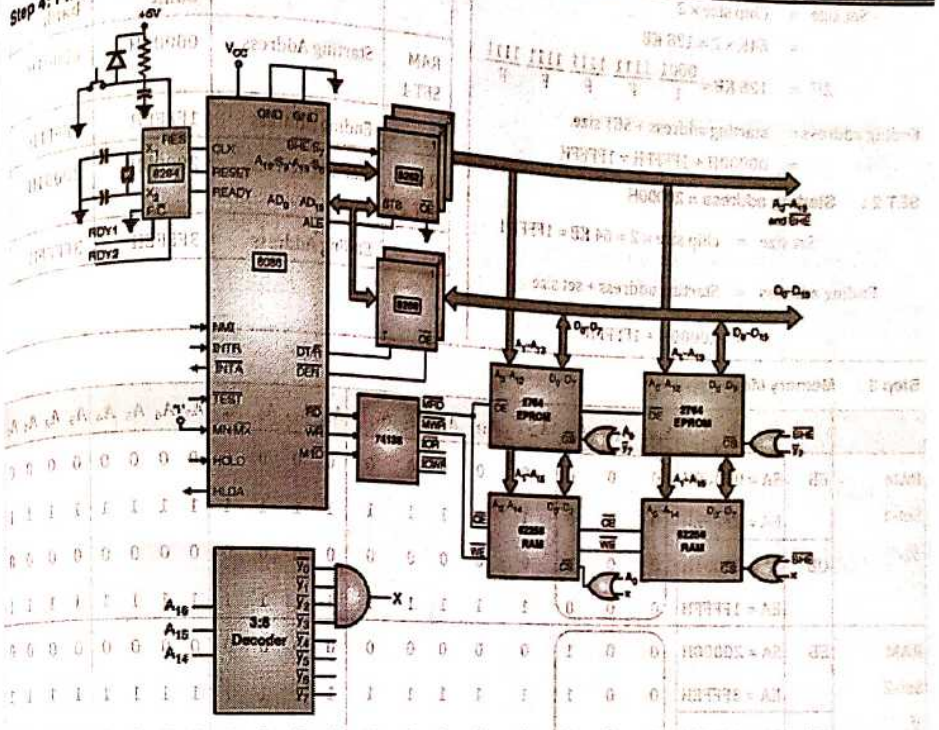


Fig. P. 11.5.12(b)

Ex. 11.5.13 : 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 ROM using 64 KB x 8-bit device

III. Three 8-bit parallel ports using 8255

IV. Support for 8 interrupts **MU - May 16, 12 Marks**

Soln. :

Step 1 : Total EPROM required is equal to 128 KB

Chip size available = 64 KB

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

∴ Number of sets required = $\frac{\text{No. of Chips}}{\text{No. of Banks}} = \frac{2}{2} = 1$

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

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

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

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_{16}	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 $\bar{y}_0$	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	1
	OB	SA = 00001H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		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 $\bar{y}_1$	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	1
	OB	SA = 20001H	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		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 $\bar{y}_7$	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	1
	OB	SA = E0001H	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		EA = FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Step 4 :

∴ 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

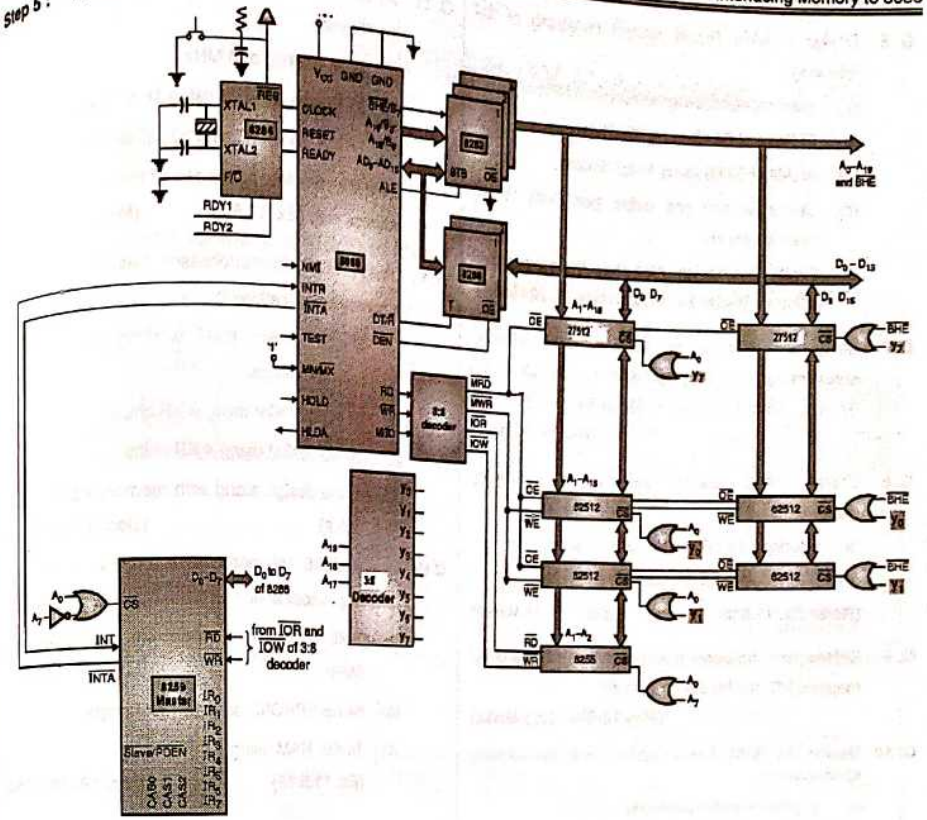


Fig. P. 11.5.13

11.6 Exam Pack (Review and University Questions)

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.

(Refer Ex. 11.2.2) **(Dec. 13, 20 Marks)**

Q. 2 (a) Design 8086 based system in minimum mode system for following requirements:

(i) 128 KB ROM using 32 KB × 8 memory device

(ii) 512 KB RAM using 64 KB × 8 memory device

(b) Interface following I/O devices to system designed in (a)

(i) Three 16 bit ports using 8255

(ii) 15 interrupt support using 8259

(Refer Ex. 11.2.3) **(May 14, 20 Marks)**

Q. 3 Explain in brief address decoding techniques **(Refer Section 11.3)** **(Dec. 11, 5 Marks)**

Q. 4 Explain the need of DRAM controller for interfacing DRAM with 8086. Draw and explain interfacing of DRAM controller with 8086. **(Refer Sections 11.4 and Fig. P. 11.5.1(a))** **(Dec. 16, 10 Marks)**

Q. 5 Differentiate between the Memory mapped I/O and I/O mapped I/O. **(Refer Section 11.5)** **(Dec. 11, May 12, Dec. 12, 5 Marks)**

Q. 6 Design a 8086 Based system consisting of the following :

(a) 8086 microprocessor working at 8 MHz.

(b) EPROM of 64 kB using 32 kB devices.

(c) SRAM of 64 kB using 16 kB devices.

(d) One input and one output port (both 16 bit) interrupt driven.

Explain the design. Also show the memory and I/O map. **(Refer Ex. 11.5.4) (May 11, 20 Marks)**

Q. 7 Design an interface for 8255 with the following requirements : 1, 16-bit input and 1, 16-bit output port. Starting address is 2000 H. **(Refer Ex. 11.5.8) (Dec. 11, 5 Marks)**

Q. 8 Design 8086 based system with following specifications :

(a) Interface 32 KB SRAM. Use IC 6264.

(b) Interface 16 KB DRAM. Use 8 KB chips.

(Refer Ex. 11.5.6) (Dec. 11, 10 Marks)

Q. 9 Differentiate between memory mapped I/O and I/O mapped I/O. **(Refer Section 11.5) (May 12, Dec. 12, 5 Marks)**

Q. 10 Design an 8086 based system with the following specifications.

(i) 8086 is in minimum mode

(ii) 64 Kbyte EPROM using 52 KB devices

(iii) 64 Kbyte RAM using 32KB devices.

Draw the complete schematic of the design indicating address map.

(Refer Ex. 11.5.7) (Dec. 12, 10 Marks)

Q. 11 Design an 8086 based system with following specification :

(i) 8086 working at 8 MHz

(ii) 32 Kbytes EPROM using 16 KB devices

(iii) 64 Kbytes RAM using 32 KB devices

Draw the complete schematic of the design indicating address map **(Ex. 11.5.5)** **(May 13, 10 Marks)**

Q. 12 Design 8086 microprocessor based system with following specifications

(a) Microprocessor 8086 working at 10 MHz in minimum mode

(b) 32 KB EPROM using 8 KB chips

(c) 16 KB SRAM using 4 KB chips

Explain the design along with memory address map. **(Ex. 11.5.9)** **(Dec. 14, 10 Marks)**

Q.13 Design 8086 microprocessor based system with following specification :

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

(b) 64 KB EPROM using 16 KB*4 chips

(c) 16 KB RAM using 8 KB*2 chips.

(Ex. 11.5.12) **(Dec. 18, 10 Marks)**

□□□

12 Intel 80386DX Processor

MODULE 4

12.1 Introduction to the Evolution of Intel Processors

Intel began with developing the first microprocessor of 4-bit i.e. 4004. The specialties of some of the processors of Intel in order of their evolution are listed below.

Processor	Prefetch Queue Size	Features			
		CLK speed	Bus width		Some special features
			Data	Address	
4004	-	740kHz	4 bits	4 bits	World's first microprocessor
4040	-		4 bits	4 bits	Interrupts were introduced
8008	-		8 bits	8 bits	
8080	-	2MHz	8 bits	16 bits	
8085	-	3MHz, 5MHz	8 bits	8 bits	Serial Communication was introduced
8086	6	5MHz, 8MHz, 10MHz	16 bits	20 bits	1. Multibus compatible (multiprocessor configuration) 2. Introduction of pipelining 3. Concept of Memory Segmentation.
8088	4	5MHz	8 bits	20 bits	
80186	6	5MHz			It has two on-chip Timers, DMA Controller and Interrupt Controller
80286	6	6MHz	16 bits	24 bits	Memory protection to support multi-tasking
80386	16	16, 20, 25, 33 MHz	32 bits	32 bits	Concept of Virtual Mode
80486	32	25, 33, 50 MHz			Level 1 cache on-chip

12.2 Detailed Study of the 80386DX Block Diagram

Features of 80386

1. Flexible 32-bit microprocessor i.e. 8, 16 or 32 bit data types.
2. 8 general purpose 32-bit registers available.
3. Large memory space.
   (a) 4 GB Physical Memory and
   (b) 64 TB Virtual Memory.
4. Integrated Memory Management Unit
   (a) Virtual Memory support.
   (b) Optional on-chip paging.
   (c) 4 levels of protection.
5. Virtual 8086 mode allows running of 8086 software in protected mode.
6. Object code compatible with 8086.
7. Hardware debugging support.
8. Optimized system performance
   (a) Pipelined Instruction execution.

(b) On chip Translational caches i.e. Translational Look-aside Buffers (TLB).

(c) 16 to 20 MHz operation. A 16 MHz 80386DX provides almost 10 times higher performance than that of a standard 5 MHz 8086 processor.

9. High speed Numeric support using 80387 Coprocessor

10. 132 pin grid array package

12.2.1 Architecture of 80386 DX

University Question

Q. Draw the block diagram of 80386 DX processor and explain each block in brief.

MU - May 13, 10 Marks

- The 80386 as shown in Fig. 12.2.1 consists of a Central Processing Unit (CPU), Memory Management Unit (MMU) and a Bus Interface Unit (BIU).
- The CPU consists of execution unit and instruction decoding unit. The execution unit consists of 8, 32-bit general purpose registers for both address and data.
- It also consists of a barrel shifter used to speed the shift, rotate, multiply and divide operations.
- The MMU consists of a segmentation unit and a paging unit. Segmentation unit allows the conversion of logical address to linear address.
- Paging mechanism allows conversion from linear address to physical address if paging is enabled.

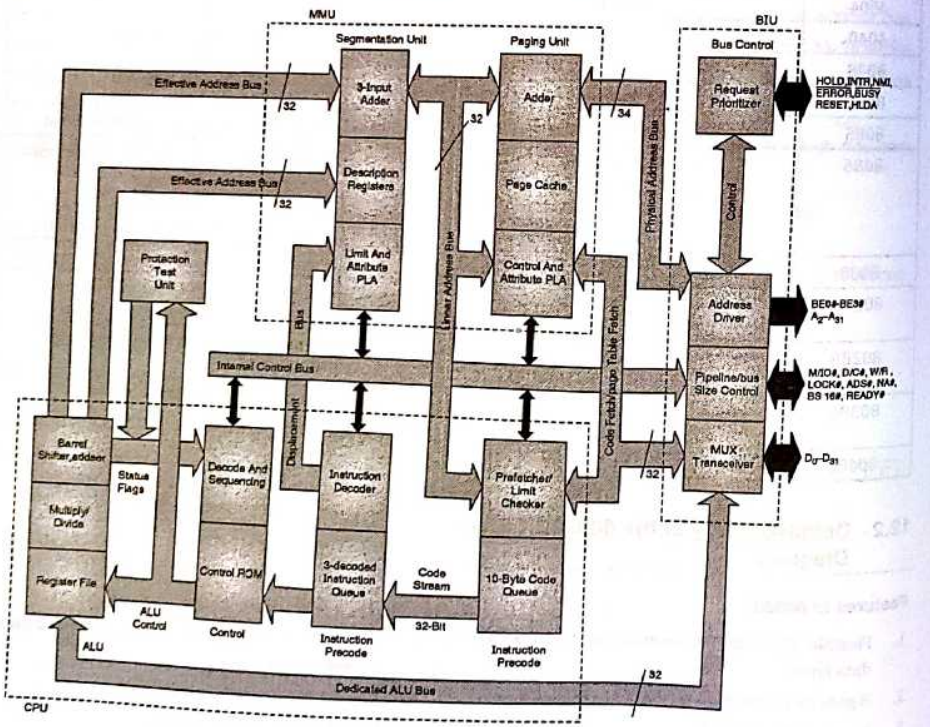


Fig. 12.2.1 : Block diagram of 80386DX

- The segmentation unit provides four level of protection to protect applications, operating system, system drivers and kernel from each other.
- The BIU handles all external accesses by providing the address and generating the respective control signals. The block diagram of 80386DX includes the following units as shown in Fig. 12.2.1.

1. Bus Interface Unit (BIU)

This unit includes the address drivers, transceivers for data bus and bus control signals as seen in the block diagram.

2. Prefetcher and the prefetch queue

The prefetcher fetches the instructions from the external memory and stores them in the prefetch queue to be executed further. The prefetch queue is 16-byte in size.

3. Instruction decoder and decoded instruction queue

The instruction decoder takes the instruction from the prefetch queue and after decoding it, stores them in the decoded instruction queue. The decoded instruction queue can store upto three decoded instructions.

4. Control ROM and the sequencing logic

The control ROM provides the control signals to be issued for the corresponding instruction, which are then sequenced by the sequencing logic

5. Execution unit

The execution unit includes a multiply unit, adder, barrel shifter and divide unit. It also includes the registers which are explained in detail in the next section i.e. programmer's model of 80386/Pentium

6. Protection unit

This unit is responsible for protected mode operation of 80386 which supports multi-tasking. This will be explained in detail in the subsequent sections of this chapter.

7. Segmentation unit

This unit is responsible for segmentation mechanism. It is also an important feature that supports multi-tasking in protected mode. This unit will be discussed in detail with the different modes of operation of 80386DX in this chapter.

8. Paging unit

This unit converts the linear address to physical address. This unit is also described in the further sections of this chapter.

12.3 Registers of 80386/ Pentium

University Questions

Q. Discuss the register set of 80386 processor.

MU - May 13, 5 Marks

- The Intel x86 architecture register set has six, 16-bit registers and twenty four, 32-bit registers as shown in Fig.12.3.1 and is subdivided into the following groups:

1. Base architecture register

(a) General purpose registers

(b) Instruction Pointer

(c) Flags register

(d) Segment registers

2. System registers

(a) Memory management registers

(b) Control registers

3. Floating-point registers (This set of registers are not present in 80386, they are in Pentium processor which has an on-chip Floating point unit)

(a) Data registers (These are 8, 80 bit registers to hold floating point data)

(b) Tag word (This word hold the information or tag of each register, indicating the status of data in that register)

(c) Status word (This register holds the status of the operation of Floating point operation's result i.e. similar to flag register)

(d) Control word (This register is used to control the different features of the floating point unit)

(e) Instruction and data pointers (these registers are used to point to the program and data when the operations are to be performed by the Floating point unit)

4. Debug and test registers

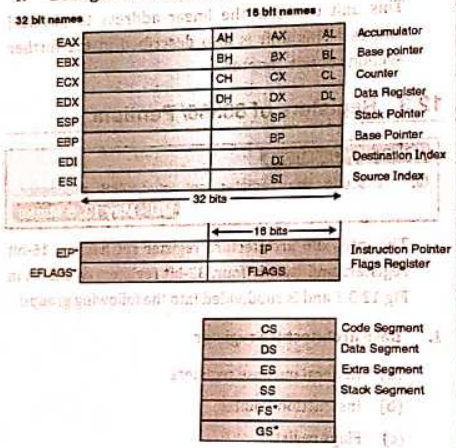


Fig.12.3.1 : Base architecture registers

12.3.1 General Purpose Registers

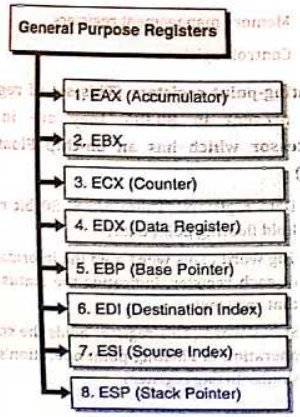


Fig. 12.3.2

1. EAX (Accumulator)

EAX	32 bit register
AX	16 bit register
AH and AL	8 bit registers

It usually accumulates the result of any ALU operation, but can also be used as General purpose. (In 386 andabove EAX may also hold an address to access a memory location.). It also works as register for I/O and string instructions.

2. EBX

EBX	32 bit register
BX	16 bit register
BH and BL	8 bit registers

It works as Base Index. (In 386 and above EBX may also hold a address to access a memory location)

3. ECX (Counter)

ECX	32 bit register
CX	16 bit register
CH and CL	8 bit registers

It is used for repeated string Instructions, shift, rotate and Loop Instructions. (In 386 and above ECX may also hold an address to access a memory location)

4. EDX (Data Register)

EDX	32 bit register
DX	16 bit register
DH and DL	8 bit registers

It holds result after multiplication or for division it holds the dividend. (In 386 and above EDX may also hold a address to access a memory location)

5. EBP (Base Pointer)

EBP	32 bit register
BP	16 bit register

It works as random pointer for stack segment.

6. EDI (Destination Index)

EDI	32 bit register
DI	16 bit register

It holds the destination data for string instructions. For other instructions it is used to select a location in data segment.

7. ESI (Source Index)

ESI	32 bit register
SI	16 bit register

It holds the source data for string instructions. For other instruction it is used to select a location in data segment.

8. ESP (Stack Pointer)

Used to address the memory location in the stack segment in association with stack segment register.

12.3.2 Instruction Pointer

1. The instruction pointer is a 32-bit register called EIP.
2. It holds the offset address within a segment of the next instruction to be executed. The offset is always relative to the base pointed by the code segment register.
3. The lower 16-bits of the EIP contain the 16-bit instruction pointer named IP, which is used for 16-bit offset addressing.

EIP : 80286 and above in Protected Mode

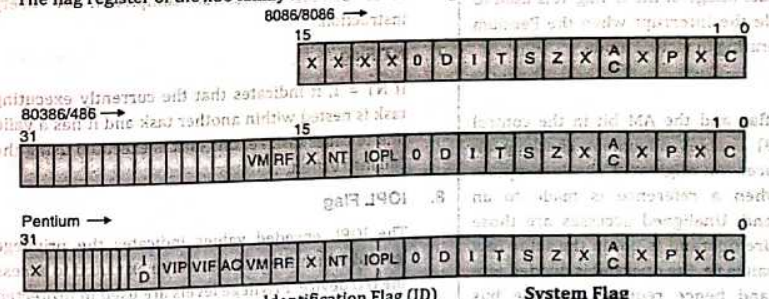
IP:8086/8088, Real Mode of 80286 and above.

12.3.3 EFLAGS Register

University Questions

Q. State the use of following X86 flags : RF, TF, VM, NT, IOPL.	MU - May 11, 10 Marks
Q. Explain EFLAGS bits of Pentium.	MU - May 13, 10 Marks
Q. Write short note on : State the use of RF, TF, VM, NT, IOPL flag bits.	MU - Dec. 14, 5 Marks
Q. Explain flag register of 80386DX.	MU - Dec. 16, 5 Marks
Q. Draw and explain EFLAG register format of 80386 DX.	MU - Dec. 17, 10 Marks
Q. Explain VM,RF,IOPL,TFand NT flags of 80386 microprocessor.	MU - May 19, Dec. 19, 5 Marks

The flag register of the x86 family is as shown in the Fig. 12.3.3.



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.12.3.3: 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.

12.3.4 Segment Registers

1. Six, 16-bit segment registers CS, SS, DS, ES, FS, and GS hold segment selector values identifying the currently addressable memory segments for the protected mode operation. While for the real or virtual modes (discussed in later sections of this chapter), when multiplied by 10H, provide the starting address of the corresponding segments.
2. The selector in CS indicates the current code segment, the selector in SS indicates the current stack segment, and the selectors in DS, ES, FS, and GS indicate the current four data segments.

12.3.5 Memory Management Registers

1. There are four memory management registers.
2. The Global Descriptor Table Register (GDTR) and the interrupt descriptor table register (IDTR) can be loaded with instructions, which get a 6-byte data item from memory and are used to point to the global descriptors (detailed study of this will be done in later sections of this chapter) and Interrupt descriptors that point to ISRs.
3. The Local Descriptor Table Register (LDTR) and task register (TR) can be loaded with instructions, which can take a 16-bit segment selector as an operandand are used to point to the local descriptors (detailed study of this will be done in later sections of this chapter) and current Task State Segment (TSS).

12.3.6 Control Registers

University Question

Q. Write short note on : Control registers of 80386 DX. **MU - May 17. 4 Marks**

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.

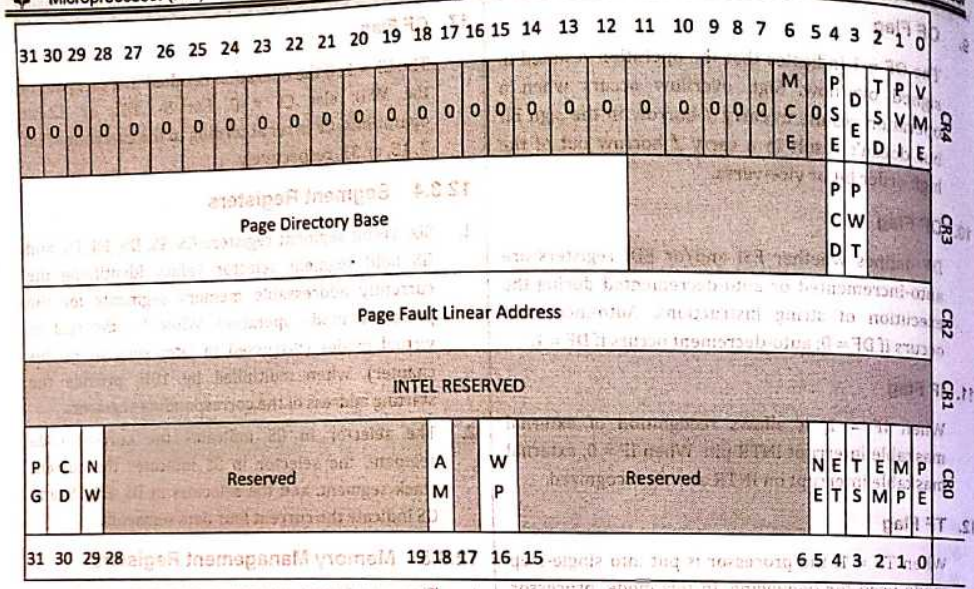


Fig. 12.3.4 : Control Registers

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 usewrite-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.

Note : The low-order 16-bits of the CR0 are also known as machine status word (MSW) for compatibility with i286.

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.

12.3.7 Debug Registers

University Questions

Q. Explain the Debug registers of 80386 DX processor. **MU - May 12, 10 Marks**

Q. Explain the debug registers of Pentium. **MU - Dec. 12, 10 Marks**

1. The x86 architecture features eight debug registers, DR0 to DR7.
2. Only programs executing at highest privilege level can access these registers.
3. DR0 to DR3 hold the linear address for breakpoints. DR6 and DR7 indicate the status of the processor at these breakpoint addresses.

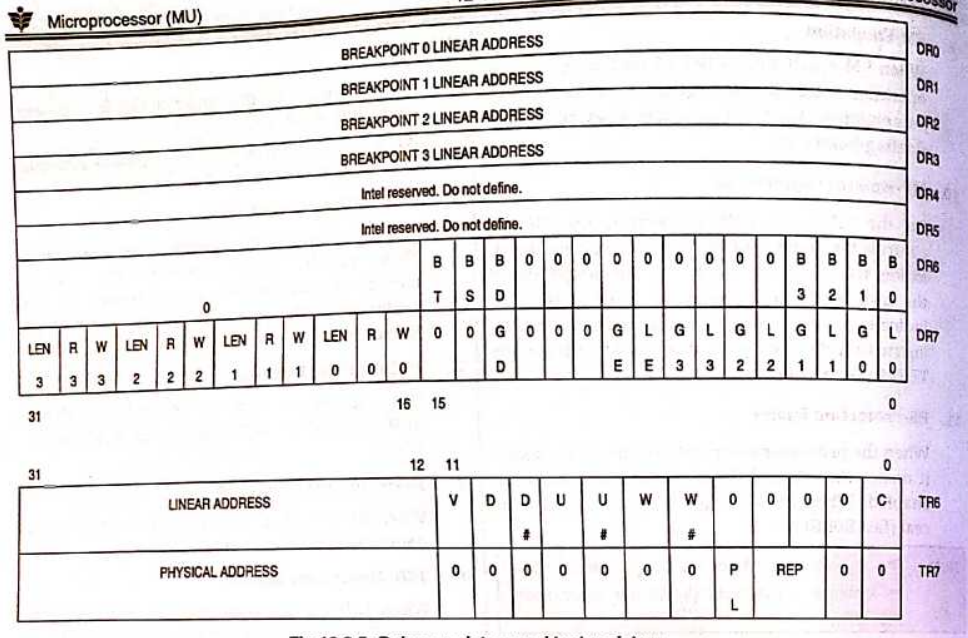


Fig.12.3.5: Debug registers and test registers

4. The functions of the bits in the debug registers is as given below. The DR7 register is used to control the breakpoint operations while the DR6 gives the status of the processor at those breakpoint addresses.

   BT : If '1', indicates the debug interrupt is caused by a task switch.

   BS : If '1' indicates the debug interrupt was caused by trap flag.

   BD : If '1' indicates that the interrupt was caused by an attempt to read the debug register with GD bit ='1'. The GD bit protects access to the debug registers.

   B3-B0: Indicates which of the four debug breakpoint address caused the debug interrupt.

   LEN : Each of the four length fields corresponds to each of the four breakpoint addresses stored in DR0 to DR3. These bits define the size of access at the breakpoint address.

   RW : The RW field selects the cause of action that enabled a breakpoint address as instruction access, data write or data read.

   GD : This bit protects access to the debug registers.

   GE : If '1', it selects a global breakpoint address for all the four breakpoint address registers.

   LE : If '1', it selects a local breakpoint address for all the four breakpoint address registers.

5. TR6 and TR7 that are test registers of 80386. TR6 is also called as Test control register as it gives the control for the test operation, while the TR7 is called as Test status register as it provides the status of the corresponding test.

6. They are used to test the TLBs (Translation Look-aside Buffers). The TLBs are used with the paging mechanism that will be studied in the later sections of this chapter.

7. The different bits of Test registers are as explained below

   Linear address : It holds the tag field of the TLB

   Physical address : It holds the physical address of the TLB

   V : Indicates if the entry in the TLB is valid or not.

   D : Indicates if the entry in the TLB is dirty (i.e. not updated in the next level of memory)

   U : This bit is assigned to the TLB

   W : This bit indicates if the area addressed by the TLB entry is writable or not.

   C : Selects between write and lookup of the TLB.

   PL : If this bit is '1', it indicates TLB hit.

   REP : Selects which block of TLB is written

12.4 Processing Modes of 80386

University Question

Q. Write short note on : Operating modes of 80386.

MU - May 14, 5 Marks

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

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

The Fig. 12.4.1 shows mode transition diagram.

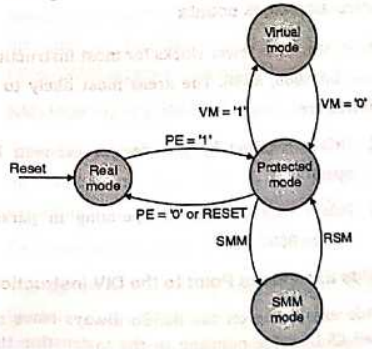


Fig. 12.4.1

- One purpose of protection is to prevent applications from interfering with the operating system; therefore, the processor makes certain registers and instructions inaccessible to applications programs.
- 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.

12.4.1 Difference between Real Mode, Protected Mode and Virtual 8086 Mode of Microprocessor

University Question

Q. Differentiate Real Mode ,Protected Mode and virtual 8086 mode of 80386 microprocessor.

MU - May 19, Dec. 19, 10 Marks

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

Sr. No.	Real Mode	Protected Mode	Virtual 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

12.5 Real and Virtual Mode

University Questions

Q. Write short note on : V-86 mode of operation. **MU - May 11, May 17, 5 Marks**

Q. Explain V86 mode of 80386DX. **MU - May 16, 5 Marks**

- 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, 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.

Comparison of real mode or virtual mode of 80386 and actual 8086

In general, the 80386 in real-address mode will correctly execute software designed for the 8086, 8088, 80186, and 80188. Following is a list of some of the minor differences between 8086 execution on the 80386 and on an 8086.

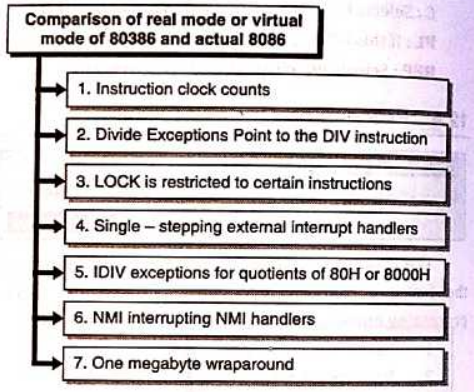


Fig. 12.5.1 : Comparison of real mode or virtual mode of 80386 and actual 8086

1. Instruction clock counts

The 80386 takes fewer clocks for most instructions than the 8086, 8088. The areas most likely to be affected are:

(a) Delays required by I/O devices between I/O operations.

(b) Delays with 8086/8088 operating in parallel with an 8087.

2. Divide Exceptions Point to the DIV Instruction

Divide exceptions on the 80386 always leave the saved CS:IP value pointing to the instruction that failed.

On the 8086/8088, the CS:IP value points to the next instruction.

3. LOCK is restricted to certain instructions

The LOCK prefix and its corresponding output signal should only be used to prevent other bus masters from interrupting a data movement operating.

The 80386 always asserts the LOCK signal during an XCHG instruction with memory (even if the LOCK prefix is not used). LOCK may only be used with the following 80386 instructions when they update memory: BTS, BTR, BTC, XCHG, ADD, ADC, XUB, SBB, INC, DEC, AND, OR, XOR, NOT, and NEG. An undefined – opcode exception (interrupt 6) results from using LOCK before any other instruction.

4. Single – stepping external interrupt handlers

The priority of the 80386 single-step exception is different from that of the 8086/8088. The change prevents an external interrupt handler from being single-stepped if the interrupt occurs while a program is being single-stepped. The 80386 will still single -step through an interrupt handler invoked by the INT instructions or by an exception.

5. IDIV exceptions for quotients of 80H or 8000H

The 80386 can generate the largest negative number as a quotient for the IDIV instruction when dividing a negative number by zero. The 8086/8088 causes exception zero instead.

6. NMI interrupting NMI handlers

After an NMI is recognized on the 80386, the NMI interrupt is masked until an IRET instruction is executed.

7. One megabyte wraparound

The 80386 does not wrap addresses at 1 megabyte in real – address mode. On members of the 8086 family, it possible to specify addresses greater than one megabyte.

For example, with a selector value 0FFFFH and an offset of 0FFFFH, the effective address would be 10FFEFH (1 Mbytes + 65535 - 16). The 8086, which can form addresses only up to 20 bits long, truncates the high – order bit, thereby "wrapping" this address.

12.5.1 Addressing in Real Mode

University Questions

Q. Differentiate segmentation in real mode. **MU - May 13, 5 Marks**

The memory is subdivided into parts called as segments. Segments can start at any base address (multiple of 10H) in memory, and storage overlapping between segments allowed. A virtual (logical) address in x86 is formed using two components:

1. A 16-bit segment register, used to determine the linear base address.
2. A 16-bit offset register inside the segment.

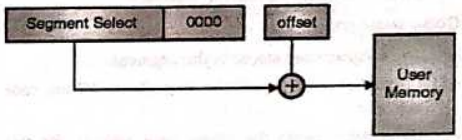


Fig. 12.5.2 : Addressing in Real mode

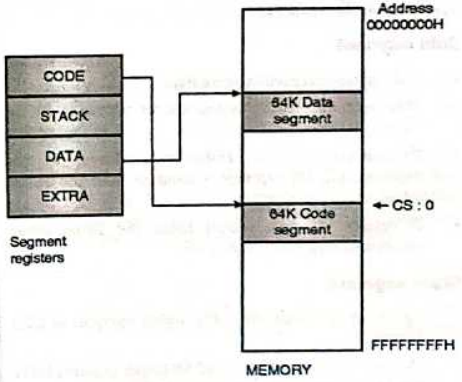


Fig. 12.5.3

- Although 80386 has 32 address lines, the addressing mechanism explained above, allows 80386 to access only 1 MB + 64 KB – 1 B in real mode.
- This is because the maximum value a segment register can have is FFFFH.
- The offset added to it can be again of a maximum value of FFFFH.
- Thus the last accessible location is FFFF0H + FFFFH = 0001FFEFH.
- This is 1 MB + 64 KB – 16 B starting from 00000000H. Although when switched on 80386 keeps the 12 MSBs (of the 32 bit address) at logic '1', allowing the processor to access the lower segment of the memory.
- But the first branching instruction it executes, makes the processor to access the top locations i.e. 00000000H to 0001FFEFH.
- 80386 can access only 4 segments of this at a time i.e. code segment, data segment, stack segment and extra segment in real or virtual mode.
- To access 4 segments 80386 has four 16-bit segment register CS, DS, SS and ES to hold the base address of the segments and 16-bit offset registers to hold the offset address for each of the segments.

Code segment

- The programs are stored in this segment.
- Instruction fetch operation is performed from code segment.
- CS register holds the 16-bit base address for this segment and IP register (Instruction pointer) holds the 16-bits offset address.

Data segment

- This segment is used to store data.
- This segment also holds the source operands during string operations.
- DS register holds the 16-bit base address for this segment and BX register is used to hold the 16-bit offset.
- SI register (Source Index) holds the 16-bit offset address during string operation.

Stack segment

- This segment holds the stack, which operates in LIFO manner.
- SS holds its base address and SP (stack pointer) holds the 16-bit offset address of the top of the stack.
- BP (base pointer) is used as offset register during random access of stack.

Extra segment

- This segment mainly stores destination operands during string operations.
- ES holds the base address and DI holds the offset address during string operations.

Address calculation

- The 20-bit physical address is calculated using the 16-bit base address and the 16-bit offset address as follows :
- Physical address = Segment address × 10 H + offset address

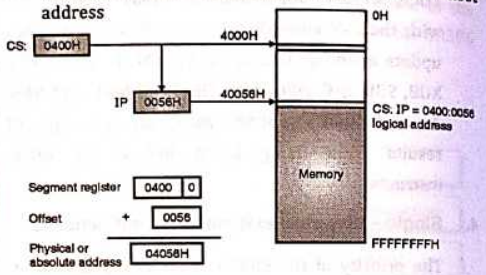


Fig. 12.5.4

- The offset is the distance in bytes from the start of the segment.
- The offset is given by the IP for the code segment.
- Instructions are always fetched with using the CS register
- The physical address is also called the absolute address.

Advantages of segmentation

1. Segmentation is very useful for multi-user environment, wherein sharing of data and protection of data is possible by restricting use of certain segments to the users.
2. It allows the programmer to access more memory using only smaller registers.
3. Multiple segments for code allows separate programs to be written in different segments and hence can be stored in modular structure. Similarly data can also be stored in modular structure.
4. It divides the memory logically to store instructions, data and stack separately.

Disadvantages of segmentation

1. At any given time the entire memory is not accessible.

2. To access any memory location 2 registers (Base and Offset) need to be initialized.

12.6 Protection Mode

This mode is mainly meant for multi tasking operations. Multiple tasks running simultaneously using separate code, data and stack segment.

Besides it also takes care of proper authentication of a task to access a particular segment.

12.6.1 Protection Mechanism

University Questions

- Q. Explain the protection mechanism in x86 DX via call gates, priviledge levels. MU - Dec. 12, 10 Marks
- Q. Explain the protection mechanism of x86 intel family microprocessor. MU - Dec. 13, 10 Marks
- Q. Write short notes on : CALL gate mechanism MU - Dec. 13, 10 Marks
- Q. Explain protection mechanism used in 80386. MU - May 15, 10 Marks
- Q. Explain in brief protection mechanism in 80386DX processor. MU - Dec. 15, May 16, 5 Marks

1. Most processors have only two protection levels (user and supervisor), but x86 architecture features four levels of protection, called **Privilege Levels (PL).**
2. They are designed to support the needs of multitasking OS to isolate and protect user programs from each other and the OS from unauthorized access.

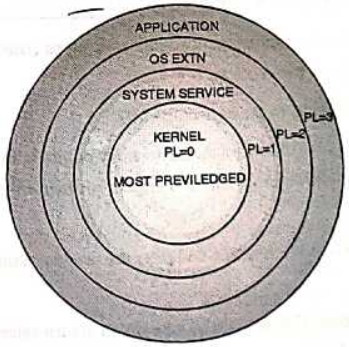


Fig. 12.6.1 : Privilege levels of the tasks in x86 architecture

3. The privilege levels control the use of privileged instructions, I/O instructions, and access to segments and segment descriptors. The x86 architecture offers an additional type of protection on a page basis, when paging is enabled.

   Requested PL ⇒ RPL
   Descriptor PL ⇒ DPL
   Current PL ⇒ CPL
   Effective PL ⇒ EPL
   EPL = max (RPL, CPL)

4. The PLs are numbered 0, 1, 2, and 3. Level 0 is the most privileged level. Level 3 is the least privileged. As shown in Fig. 12.6.1, level 3 is used for user application, level 2 is used for OS extensions, level 1 for system services, and the most privileged level 0 is used for kernel.
5. The x86 architecture controls access to both data and code between levels of task, according to the following rules of privilege:
   (a) Data stored in a segment with PL = p can be accessed only by code executing at a PL, numerically, at least as privileged as p.
   (b) A code segment (a procedure) with PL = p can be called only by a task executing at, numerically, the same or lower PL than p.
   (c) A stack segment with PL = p can be used only by a task executing at the same PL.
6. The following PLs are used to maintain privilege level check :
   (a) Requestor PL, RPL, the PL of the original task that supplies the selector. RPL is determined by the two LSBs of the selector.
   (b) Descriptor PL, DPL, the PL (according to the above rules) at which a task may access that descriptor and the segment associated with that descriptor. Bits 6 and 5 of the access rights byte (ARB) of a descriptor determine the DPL.
   (c) Current PL, CPL, the PL at which a task is currently executing, i.e. at which the code segment is being executed. CPL is stored in the processor, but not accessible to the programmer.
   (d) Effective PL, EPL, the least privileged of the RPL and the CPL. Since smaller PL values indicate greater privilege, EPL is the numerical maximum of RPL and CPL, i.e. EPL = max (RPL, CPL). EPL is not stored anywhere, but is immediately copied into CPL.

7. The bits of the access right byte of a descriptor are discussed the Table 12.6.1.

Table 12.6.1 : Access Right Byte (ARB) bits and their significance

Bit position	Name	Function	
7	Present (P)	P = 1	The entry of the descriptor is initialized.
		P = 0	The entry of the descriptor is not initialized.
6-5	Descriptor Privilege Level (DPL)	Segment privilege level, used in privilege tests.	
4	Segment Descriptor (S)	S = 1	Code or Data (includes stack) segment descriptor.
		S = 0	System segment descriptor or Gate descriptor. In case it is a system descriptor, the remaining four bits (bit 3 to bit 0) indicates the type of system descriptor.
3	Executable (E)	E = 0	Descriptor type is data (or stack) segment.
If E=0, bit 2 and 1 of ARB, have the following meaning			
2	Expansion Direction (ED)	ED = 0	Expand up segment. (data segment)
		ED = 1	Expand down segment. (Stack segment)
1	Writeable (W)	W = 0	Data segment cannot be written into i.e. read only.
		W = 1	Data segment may be written into.
3	Executable (E)	E = 1	Descriptor type is code segment.
If E=1, bit 2 and 1 of ARB, have the following meaning			
2	Conforming (C)	C = 1	Code segment may only be executed when CPL >= DPL. If C=0, no privilege check will be made.
1	Readable (R)	R = 0	Code segment cannot be read.
		R = 1	Code Segment may be read.
0	Accessed (A)	A = 0	Segment has not been accessed.
		A = 1	Segment has been accessed earlier.

8. System segments describe information about tasks, interrupts, subroutines etc. The different types (the lower four bits of ARB for a system descriptor) of system descriptor are as listed below:

TYPE 2 – It refers to LDT descriptor. It is located in the GDT and points to the base of the LDT.

TYPE 4, 7, C, E, F – These refer to gate descriptors. The gates are used to control access to entry points within the target code segment forcontrol transfer instructions. This also allows processor to perform protection checks automatically. There are four types of gates:

(a) **Call gates**that serve as an intermediary between code segments at different PLs. Call gates are used to change the PLs.

(b) **Task gates**that are used to perform task switch. The Task Register (TR) selects this gate which in turn selects the current Task State Segment (TSS). The TSS holds the context of the current task.

(c) **Interrupt gates**that are used to specify interrupt service routines (ISRs).

(d) **Trap gates**that are used to specify trap-handling routines.

9. Call gates are used to transfer program control to a more privileged level. The call gate descriptor consists of three fields.

(a) The access rights byte (ARB),

(b) The pointer (selector and offset), which points to the start of the target routine, and

(c) The word count (5 bit long), which specifies how many parameters are to be copied from the caller's stack to the stack of the called subroutine. This field is used by call gates only when there is a change in the PL. Other gates (Task gate, Interrupt gate and Trap gate) ignore this field.

10. Callgates are accessed via a CALL instruction and are syntactically identical to calling a normal subroutine. When an inter level (different privilege level) call gate is activated, the following actions occur:

(a) CS:EIP are loaded from the selector and offset fields of the gate, respectively, and are checked for validity.

(b) SS is pushed on stack with zero-extended to 32 bits.

(c) ESP is pushed on stack.

(d) The word count (32-bit parameters) is copied from the old stack to the new stack.

(e) The return address is pushed on stack.

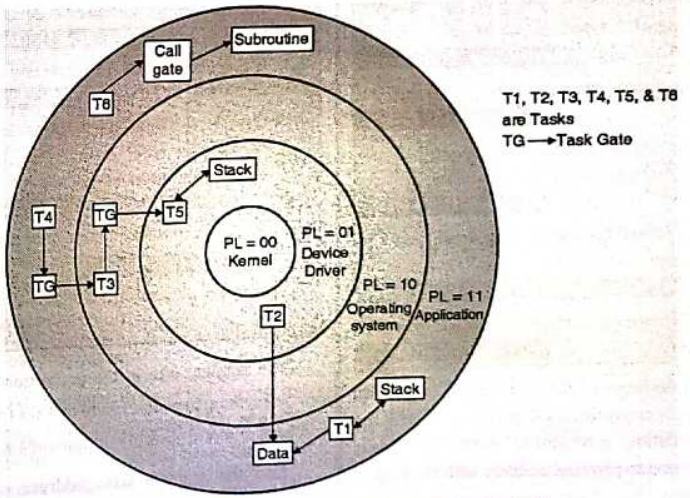


Fig. 12.6.2 : Rules for accessing data, code and stack

11. Accessing different types of programs using gates is shown in Fig. 12.6.2.
12. Gate descriptors follow all the access rules of privilege; i.e., gates can be accessed by a task if its EPL is equal to, or more privileged than the gate descriptor's DPL.
13. A task can access the code of higher PL using a gate (task gate, call gate, trap gate or interrupt gate)
12. A task can access data of lower or equal PL.
15. A task can access stack of same PL only.
16. The PL of the task (CPL) is compared with the PL of code /data /stack (DPL) for the above decisions.

12.6.2 Memory Management/Address Translation Mechanism in Protected Mode

University Questions

Q. Draw protected mode address translation mechanism in 80386DX with neat flow diagram. Explain segment translation in detail. **MU - May 11, Dec. 11, 10 Marks**

Q. What is segment descriptor? Draw and explain the structure. **MU - May 11, 5 Marks**

Q. Explain segment translation mechanism with flowchart. Also explain segment descriptor. **MU - Dec. 13, 10 Marks**

Q. Explain address translation mechanism used in protected mode of 80386DX. **MU - Dec. 14,Dec. 17, May 17, 10 Marks**

Q. Explain memory management in details in 80386DX processor. **MU - Dec. 15, 10 Marks**

As seen in Sections 12.6.3 and 12.6.4, the logical (virtual) address is converted to linear address with the help of segmentation mechanism; while the linear address is converted to physical address with the help of paging mechanism. This address conversion flow can be as shown in the Fig. 12.6.3.

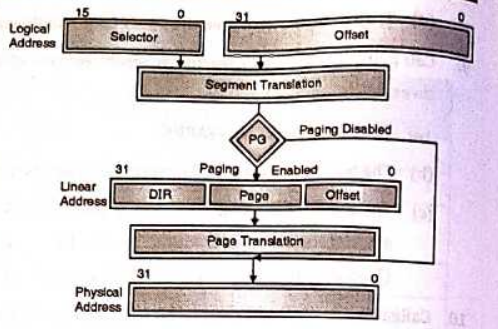


Fig. 12.6.3 : Virtual to physical address translation in 80386

Hence memory management includes segmentation and banking in 80386.

12.6.3 Segmentation in Protection Mode

University Questions

Q. Draw protected mode address translation mechanism in 80386DX with neat flow diagram. Explain segment translation in detail. **MU - May 11, Dec. 11, 10 Marks**

Q. Explain segmentation in protected mode of 80386 processor. **MU - May 12, 10 Marks**

Q. Explain segmentation mechanism in 80386 DX and hence draw and explain the protected mode address translation mechanism in 80386 DX. **MU - Dec. 12, 10 Marks**

Q. Differentiate segmentation in protected mode. **MU - May 13, 5 Marks**

Q. Draw format of selector and explain its field. **MU - Dec. 17,5 Marks**

The segment size in the protection mode for x86 can be anything between 1 byte to 4 GB (beyond 4 MB, it can be of different sizes in multiples of 4 KB).

Also the segment base address can start from anywhere removing the constraint (which was there in 8086) of last hex. digit to be '0'. Hence the entire memory is accessible in protection mode and the memory beyond 1MB is called as extended memory.

The logical address in protected mode architecture is formed out of two components:

(a) The 16-bit selector (segment register) that is used to determine the descriptor.

(b) 32-bit offset to give the internal address within the segment.

Selector

- Selects one of the 8192 descriptors from one of the tables viz. GDT and LDT (Global Descriptor Table and Local Descriptor Table).
- GDTR (Global Descriptor Table Register) points to segment (of 64KB in size) common for all programs while LDTR points to segment (of 64KB is size) containing programs that are unique to an application
- Each segment (GDT and LDT) consists of 8192 descriptors of 8 bytes each and hence a total of 16K segments in all can be addressed by any task.
- Hence to select a descriptor out of 8K descriptors, selector (segment register) uses 13 MSB as shown in the Fig. 12.6.4. One bit to select the descriptor table (GDT or LDT) called as table identity (TI) bit. Remaining two bits are used to pass on the Requested Privilege Level (RPL).

16 — 3	2	1 — 0
Selector	TI	RPL
Selects one of the 8192 Descriptors	TI=0 GTD TI=1 LDT	Requested Privilege level

Fig. 12.6.4 : Structure of the selector

Program Invisible Registers

- GDTR and IDTR contain address of the two descriptor table and their limits (that indicate the size of the segment). The limit is 16 bits and hence the segment length can be max 64KB.
- LDTR works as a segment register (selector) and by selecting a descriptor from GDT it selects the LDT.
- Task Register (TR) also holds a selector that accesses a descriptor that defines the current task (procedure or application program).
- Hence by changing the value of the TR, we can switch from one task to another. The task register allows task switching in about 17μs (for 386).This is how multitasking is implemented in the processor.

12.6.3(A) Structure of Descriptor

University Questions

Q. Write short note on: Structure of segment descriptor. **MU - Dec. 11, May 12, 5 Marks**

Q. State the use of following x86 flags : RF, TF, VM, NT, IOPL **MU - May 12, 5 Marks**

Q. What is descriptor ? Explain code and data segment descriptor with neat diagram. **MU - May 14, 10 Marks**

Q. Draw a segment descriptor format and explain different fields. **MU - Dec. 14, 10 Marks**

Q. Explain data segment descriptor with neat diagram. **MU - Dec. 16,10 Marks**

The descriptor gives the different details of the segment as shown in Fig. 12.6.5.

1. **Base address (B0 to B31) part indicates the starting address of the memory segment.**

Byte no.							Byte no.
7	Base (B31 – B24)	G	D	0	AV	Limit (L19 –L16)	6
5	Access Right Byte	Base address (B23 – B16)					4
3	Base address (B15 – B0)						2
1	Limit (L15 – L0)						0

Fig. 12.6.5 : Descriptor structure

2. **Limit address (L0 to L19)**

When added with the base address gives the last address of the segment. In case of 286 there is a 24 bit base address and 16-bit limit while in 386 we have 32 bits base address and 20 bit limit address.

3. **Granularity bit (G)**

When G = 0,20-bit Limit specifies the segment size from 1 byte to 1MB.

When G = 1,20-bit Limit is to be multiplied by 4K hence segment size varies from 4KB to 4GB.

4. **AV**

It indicates the availability of segment

When AV = 1,it indicates the corresponding segment is not used by any other task and hence is available.

When AV = 0, it indicates the corresponding segment is used by some other task and hence is not available.

5. **D**

It indicates data size.

When D = 0, it indicates 16-bit OS instructions (required in DOS, Windows).

When D = 1, it indicates 32-bit OS instructions (required in OS/2, Windows NT etc).

6. **Access Right byte (ARB)**

7	6	5	4	3	2	1	0
P	DPL		S	E	ED/C	RW	A

Fig. 12.6.6 : ARB Structure

These bits are already discussed in section 12.6.1. The bit description and their significance are given in Table 12.6.1.

12.6.3(B) Accessing Global Memory Location using Global Descriptor Table

University Question

Q. What is GDT? Explain structure of GDT.

MU - May 17, 5 Marks

- 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, as already discussed, 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. 12.6.7.

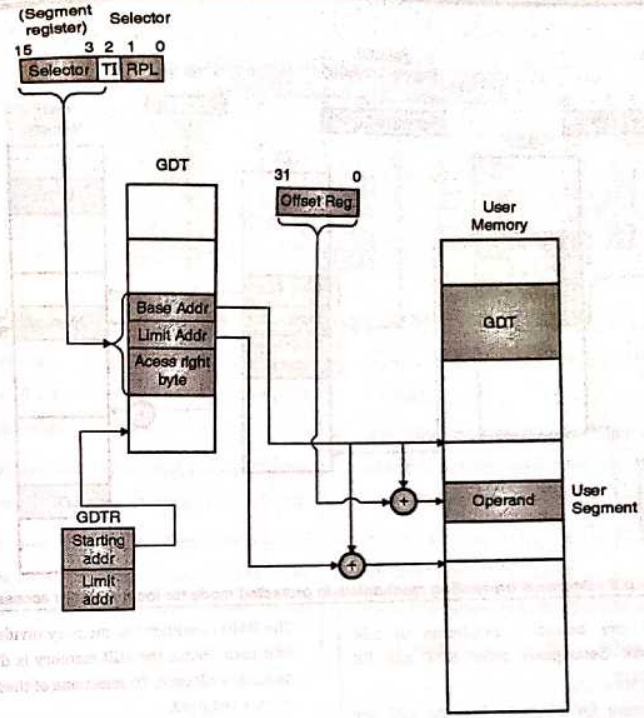


Fig. 12.6.7 : Segment translation mechanism in protected mode for global memory access

12.6.3(C) Accessing Local Memory Location using Local Descriptor Table

- Local memory location is one which is separately accessible to each task.
- The Local descriptor Table (LDT) is a separate for each task. The LDT is selected by a descriptor from the GDT, which in turn is selected by the LDT Register (LDTR). LDTR is 16-bit register that works as a selector similar to the segment register.
- This descriptor is called as a system descriptor as discussed in section 12.6.1. The maximum value of the 16-bit limit address can be 64 KB which means that the maximum size of the LDT can be 64 KB. Each descriptor, as already discussed, is of 8 bytes.
- Hence the total number of descriptors in LDT 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 local memory or segment translation mechanism for local data is as shown in Fig. 12.6.8.

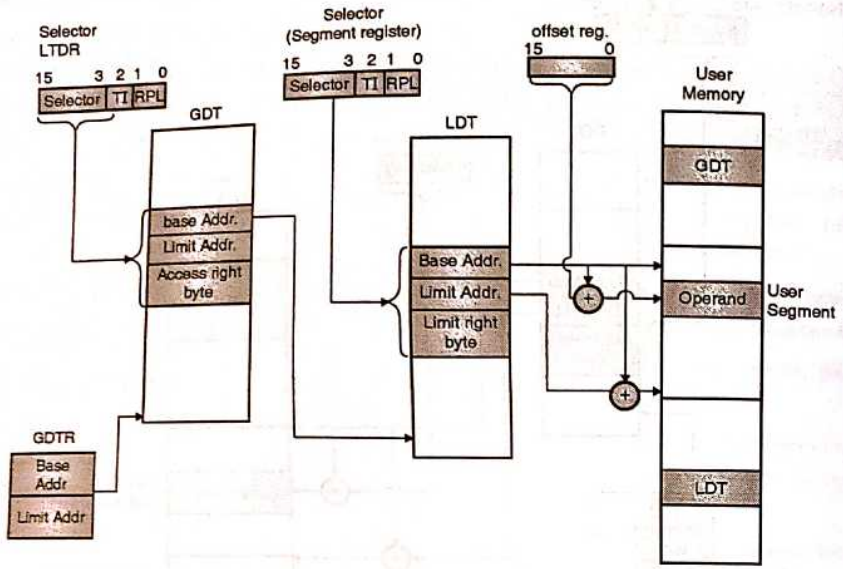


Fig. 12.6.8 : Segment translation mechanism in protected mode for local memory access

- Hence any task can access a maximum of 16K descriptors i.e. 8K descriptors from GDT and 8K descriptors from LDT.
- The GDT is common for all tasks but the LDT for individual task is different. Hence it helps in maintaining local and global data separately.
- The IDTR (Interrupt Descriptor Table Register) works similar to GDTR, but selects the descriptor that work as pointers to ISRs.
- While the TR (Task Register) works similar to LDTR, but selects the current Task State Segment (TSS).

12.6.4 Implementation of Paging in Protected Mode

University Question

Q. Explain paging in protected mode of 80386 processor **MU - May 12, 10 Marks**

- 80386 / Pentium processor use the virtual memory system with the help of on-chip Memory Management Unit (MMU) i.e. paging mechanism.
- The MMU considers the memory divided into pages of 4KB each. Hence the 4GB memory is divided into 1 M pages of 4 KB each. To select one of these 1 M pages, 20 bits are required.
- CR3 consists of 20-bit page directory base address that selects a page, called as page directory. This page is also of 4 KB that has 32-bit (i.e. 4 Byte) entry, and hence has 1 K entries.
- To select one of these 1 K entries, 10 bits are required. Hence the first 10-bits of the linear address i.e. bit 31 to bit 22, select one of the 1 K entries.
- This entry in page directory (32-bit entry), has 20-bit that selects another page, called as page table.
- Since there are 1 K entries in the page directory each of them selecting a page table, there can be 1 K page tables.
- The page table size is also 4 KB that has 32-bit (i.e. 4 Byte) entry, and hence has 1 K entries.

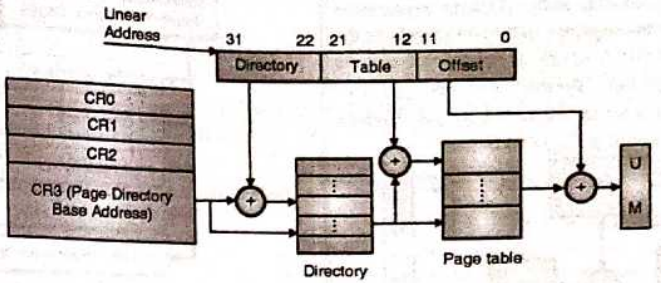


Fig. 12.6.9: Paging mechanism

- To select one of these 1 K entries, 10 bits are required. Hence the next 10-bits of the linear address i.e. bit 21 to bit 12, select one of the 1 K entries.
- This entry in a page table (32-bit entry), has 20-bit that selects the user required page. This page is also of 4 KB.
- The required memory operand from this page is selected by the last 12-bits of the linear address i.e. bit 11 to bit 0. This paging mechanism is shown in the Fig. 12.6.9.
- The page table or page directory entry is shown in Fig. 12.6.10. The different bits in these entries are discussed below
- Bit 31-12, Page frame address for page table entry, while Page table address for page directory entry.
- Bit 11-9, are reserved for Operating System (OS). The OS can use it for its purpose, storing some information related to this page.
- Bit 8 and 7, are zeroes, are unused.
- Bit 6, D (Dirty); It is set before anything is written page directory.
- It indicates that this page has some updated data in the physical memory, while the virtual memory has this page not updated or stale data.
- Bit 5, A (Accessed);It indicates that this page is accessed previously or not
- Bit 4, PCD (Page Cache Disable);As in CR0
- Bit 3, PWT (Page Write Through);As inCR0
- Bit 2, U/S (User/Supervisor);This bit differentiates between user and supervisor privileges.
- Bit 1, R/W (Read/Write); It indicates read or write protection for the page.
- Bit 0, P (Present); It indicates whether entry in the PDE and PTE (page Directory Entry and Page Table Entry) can be used. It indicates if the entry is initialized or not.

31 12	11 10 9	8	7	6	5	4	3	2	1	0
Page Frame/Table Add.	OS Reserved	0	0	D	A	PCD	PWT	U/S	R/W	P

Fig. 12.6.10 : Page table/ directory entry

- Translational Lookaside Buffer (TLB) is implemented in the memory management system, which reduces the memory access time, by giving the physical address without undergoing the paging mechanism.
- TLB is implemented in the 80386 processor whose structure is as shown in Fig. 12.6.11.

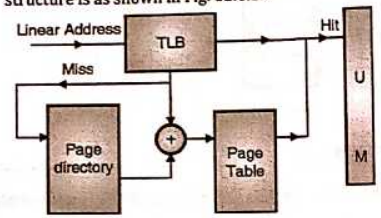


Fig. 12.6.11 :Translational lookaside buffer implementation.

12.6.5 Five Mechanisms of Protection Implementation in 80386

University Questions

Q. Explain protection mechanism implemented on 80386DX.

MU - May 14, 10 Marks

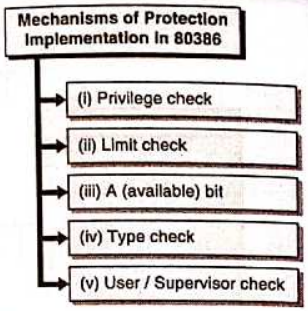


Fig. 12.6.12 : Mechanisms of protection implementation in 80386

(i) Privilege check

- There are PRIVILEGE LEVELS for each task i.e. 00 → Kernel, 01 → Device Drivers 10 → OS, 11 → Application.
- A TASK can access the **Code** of higher PL using a GATE (Task gate, Call gate, Trap gate, Interrupt gate).
- A TASK can access **Data** of lower or equal PL.
- A TASK can access **Stack** of the same PL only.
- The PL of the TASK (CPL) is compared with the PL of code data stack (DPL) for the above decisions.

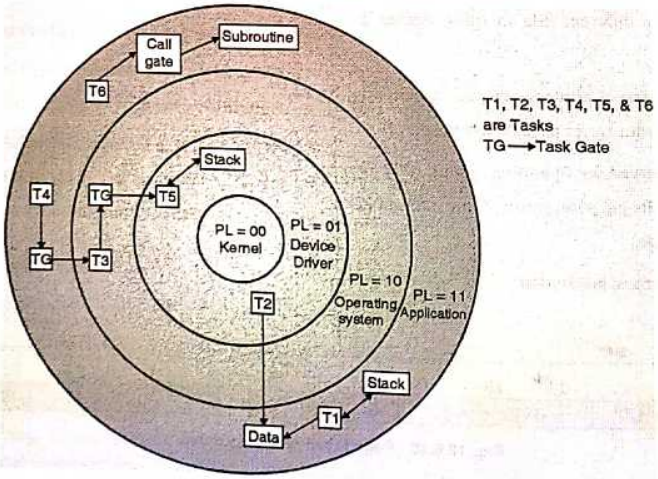


Fig. 12.6.13

(ii) Limit check

- Limit address in the Descriptor when added with the base address generates the last address of the segment.
- Since the offset register is 32-bit it can access the neighbouring segment and cause unauthenticated read / write operation.
- Hence to avoid it the limit address is checked before allowing the access.

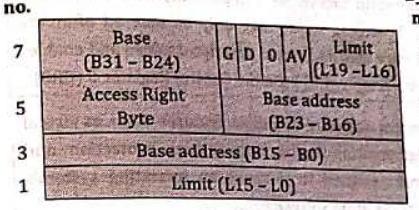


Fig. 12.6.14

(iii) A (available) bit

The Available bit in the Descriptor indicates whether this descriptor is available or already used by some other task so that two task should not use the same segment.

(iv) Type check

- A system segment access is possible only by a system selector (LDTR, TR and all gates).
- Executable Segment can be accessed only by code segment register as a selector.
- Non-executable stack segment can be accessed by stack segment register as a selector.
- Non-executable data segment can be accessed by data segment register as a selector.

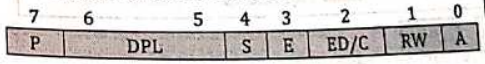


Fig. 12.6.15

(v) User / Supervisor check

- U/S bit in the page table entry.
- This bit indicates whether the page is a supervisory page or a general user page.

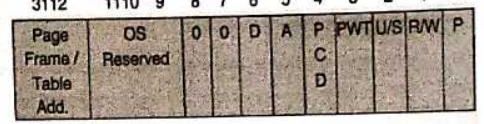


Fig. 12.6.16

12.6.6 Task Management

1. A **task** is an instance of the execution of a program.
2. A very important feature of any multitasking system is the ability to switch rapidly between tasks.
3. The task switch operation saves the entire state of the machine i.e. all registers, the memory variables and a link to the previous task, loads a new execution state, performs protection checks, and begins the execution of new task.
4. This context is stored in or retrieved from the associated Task State Segment (TSS).
5. The current TSS is identified by a special CPU register called TaskRegister (TR).
6. The task switch operation proceeds according to the following steps:
   (a) The entire task state, as discussed above, is saved in its TSS pointed to by TR.
   (b) The TR is loaded with a selector for the new TSS, and the user-invisible registers are loaded with the contents of the new TSS descriptor from the GDT. The CPU registers are loaded from this new TSS.
   (c) Protection checks are performed.
   (d) The execution of the new task begins from the entry point.
7. A task gate descriptor acts as an interface point between the task register and the TSS. The task gate descriptor contains a selector that points to the TSS and an access right byte. The current program in execution should have its CPL and RPL privileged enough to invoke the task gate.
8. There must be a valid task state segment (TSS) for the called task. The stack pointers in the TSS for privilege levels numerically less than or equal to the initial CPL, must also point to valid stack segments.
9. The task register also must point to an area in which to save the current task state. After the first task switch, the information dumped in this area is not needed, and the area can be used for other purposes.

12.6.7 Difference between Real Mode and Protected Mode of X86 Family

University Questions

Q. Differentiate between real mode and protected mode of X86 family. **MU - May 12, 10 Marks**

Q. Differentiate between real mode and protected mode. **MU - Dec. 14, May 15, 10 Marks**

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) bitin 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. 12.6.17	Refer Fig. 12.6.18

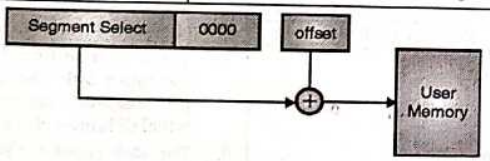


Fig. 12.6.17

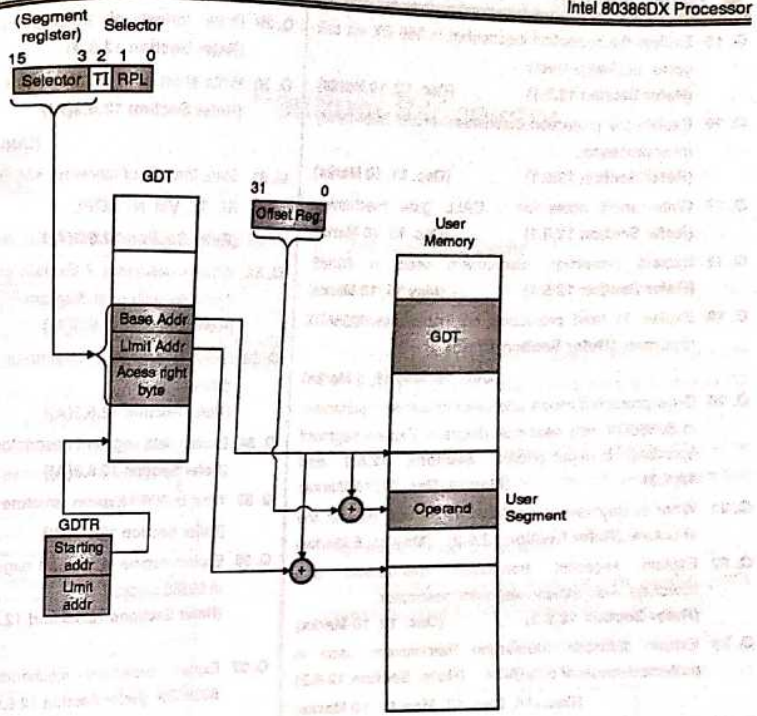


Fig. 12.6.18

12.7 Exam Pack (Review and University Questions)

Q. 1 Draw the block diagram of 80386 DX processor and explain each block in brief. **(Refer Section 12.2.1)** **(May 13, 10 Marks)**

Q. 2 Discuss the register set of 80386 processor. **(Refer Section 12.3)** **(May 13, 5 Marks)**

Q. 3 State the use of following X86 flags : RF, TF, VM, NT, IOPL. **(Refer Section 12.3.3)** **(May 11,10 Marks)**

Q. 4 Explain EFLAGS bits of Pentium. **(Refer Section 12.3.3)** **(May 13, 10 Marks)**

Q. 5 Write short note on : State the use of RF, TF, VM, NT, IOPL flag bits. **(Refer Section 12.3.3)** **(Dec. 14, 5 Marks)**

Q. 6 Explain flag register of 80386DX. **(Refer Section 12.3.3)** **(Dec. 16, 5 Marks)**

Q. 7 Draw and explain EFLAG register format of 80386 DX.**(Refer Section 12.3.3)** **(Dec. 17,10 Marks)**

Q. 8 Write short note on : Control registers of 80386 DX. **(Refer Section 12.3.6)** **(May 17, 4 Marks)**

Q. 9 Explain the Debug registers of 80386 DX processor. **(Refer Section 12.3.7)** **(May 12, 10 Marks)**

Q. 10 Explain the debug registers of Pentium. **(Refer Section 12.3.7)** **(Dec. 12, 10 Marks)**

Q. 11 Write short note on : Operating modes of 80386. **(Refer Section 12.4)** **(May 14, 5 Marks)**

Q. 12 Write short note on : V-86 mode of operation. **(Refer Section 12.5)** **(May 11, May 17, 5 Marks)**

Q. 13 Explain V86 mode of 80386DX. **(Refer Section 12.5)** **(May 16, 5 Marks)**

Q. 14 Differentiate segmentation in real mode and in protected mode.**(Refer Sections 12.5.1 and 12.6.3)** **(May 13, 5 Marks)**

Q. 15 Explain the protection mechanism in 386 DX via call gates, priviledge levels.
(Refer Section 12.6.1) **(Dec. 12, 10 Marks)**

Q. 16 Explain the protection mechanism of x86 intel family microprocessor.
(Refer Section 12.6.1) **(Dec. 13, 10 Marks)**

Q. 17 Write short notes on : CALL gate mechanism
(Refer Section 12.6.1) **(Dec. 13, 10 Marks)**

Q. 18 Explain protection mechanism used in 80386.
(Refer Section 12.6.1) **(May 15, 10 Marks)**

Q. 19 Explain in brief protection mechanism in 80386DX processor.**(Refer Section 12.6.1)**
(Dec. 15, May 16, 5 Marks)

Q. 20 Draw protected mode address translation mechanism in 80386DX with neat flow diagram. Explain segment translation in detail. **(Refer Sections 12.6.2 and 12.6.3)** **(May 11, Dec. 11, 10 Marks)**

Q. 21 What is segment descriptor? Draw and explain the structure. **(Refer Section 12.6.2)** **(May 11, 5 Marks)**

Q. 22 Explain segment translation mechanism with flowchart. Also explain segment descriptor.
(Refer Section 12.6.2) **(Dec. 13, 10 Marks)**

Q. 23 Explain address translation mechanism used in protected mode of 80386DX. **(Refer Section 12.6.2)**
(Dec.-14, Dec. 17, May 17, 10 Marks)

Q. 24 Explain memory management in details in 80386DX processor. **(Refer Section 12.6.2)(Dec.-15, 10 Marks)**

Q. 25 Draw protected mode address translation mechanism in 80386DX with neat flow diagram. Explain segment translation in detail.
(Refer Section 12.6.3) **(May 11, Dec. 11, 10 Marks)**

Q. 26 Explain segmentation and paging in protected mode of 80386 processor
(Refer Sections 12.6.3 and 12.6.4)
(May 12, 10 Marks)

Q. 27 Explain segmentation mechanism in 80386 DX and hence draw and explain the protected mode address translation mechanism in 80386 DX.
(Refer Section 12.6.3) **(Dec. 12, 10 Marks)**

Q. 28 Differentiate segmentation in real mode and in protected mode.**(Refer Sections 12.5.1 and 12.6.3)**
(May 13, 5 Marks)

Q. 29 Draw format of selector and explain its field.
(Refer Section 12.6.3) **(Dec. 17,5 Marks)**

Q. 30 Write short note on: Structure of segment descriptor.
(Refer Section 12.6.3(A))
(Dec. 11, May 12, 5 Marks)

Q. 31 State the use of following x86 flags :
RF, TF, VM, NT, IOPL.
(Refer Section 12.6.3(A)) **(May 12, 5 Marks)**

Q. 32 What is descriptor ? Explain code and data segment descriptor with neat diagram.
(Refer Section 12.6.3(A)) **(May 14, 10 Marks)**

Q. 33 Draw a segment descriptor format and explain different fields.
(Refer Section 12.6.3(A)) **(Dec. 14, 10 Marks)**

Q. 34 Explain data segment descriptor with neat diagram.
(Refer Section 12.6.3(A)) **(Dec. 16,10 Marks)**

Q. 35 What is GDT? Explain structure of GDT.
(Refer Section 12.6.3(B)) **(May 17, 5 Marks)**

Q. 36 Explain segmentation and paging in protected mode of 80386 processor.
(Refer Sections 12.6.3 and 12.6.4)
(May 12, 10 Marks)

Q. 37 Explain protection mechanism implemented on 80386DX. **(Refer Section 12.6.5)**
(May 14, 10 Marks)

Q. 38 Differentiate between real mode and protected mode of X86 family. **(Refer Section 12.6.7)**
(May 12, 10 Marks)

Q. 39 Differentiate between real mode and protected mode.
(Refer Section 12.6.7) **(Dec. 14, May 15, 10 Marks)**

Q. 40 Explain flag register of 80386 microprocessor.
(Refer Section 12.3.3) **(Dec. 14, May 15, 10 Marks)**

Q. 41 Explain VM , RF , IOPL,TF and NT flags of 80386 microprocessor.
(Refer Section 12.3.3) **(May 19, Dec. 19, 5 Marks)**

Q. 42 Differentiate Real Mode, Protected Mode and virtual 8086 mode of 80386 microprocessor.
(Refer Section 12.4) **(May 19, Dec. 19,10 Marks)**

□□□

MODULE 5

13 Pentium Processor

13.1 Specialties of Pentium Processors

University Questions
Q. Write down features of Pentium processor.
MU - May 15, 5 Marks

The Pentium processors had changes from its predecessors in the following areas.

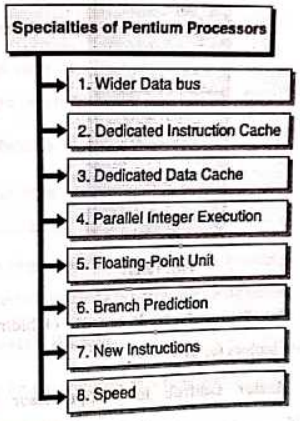


Fig. 13.1.1 :Specialties of Pentium processors

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.

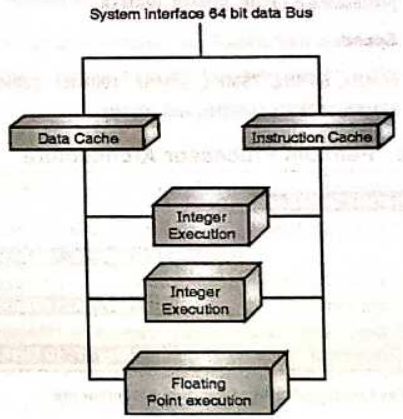


Fig. 13.1.2

- 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.
- Brach 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.

13.2 Pentium Processor Architecture

University Questions

Q. Draw and explain pentium processor architecture. **MU - May 13, May 16, 10 Marks**

Q. Explain pentium processor architecture with block diagram. **MU - May 14, 10 Marks**

Q. Draw and explain block diagram of Pentium processor. **MU - May 17, 10 Marks**

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.

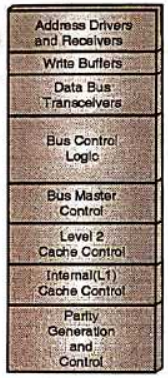


Fig. 13.2.1

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 Prefetcherand 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 instructionis possible. Pairing of FP instruction is possible with XCHG instruction of FPU.
- 3 operations can be done simultaneously for e.g. 3, FADD instructions.

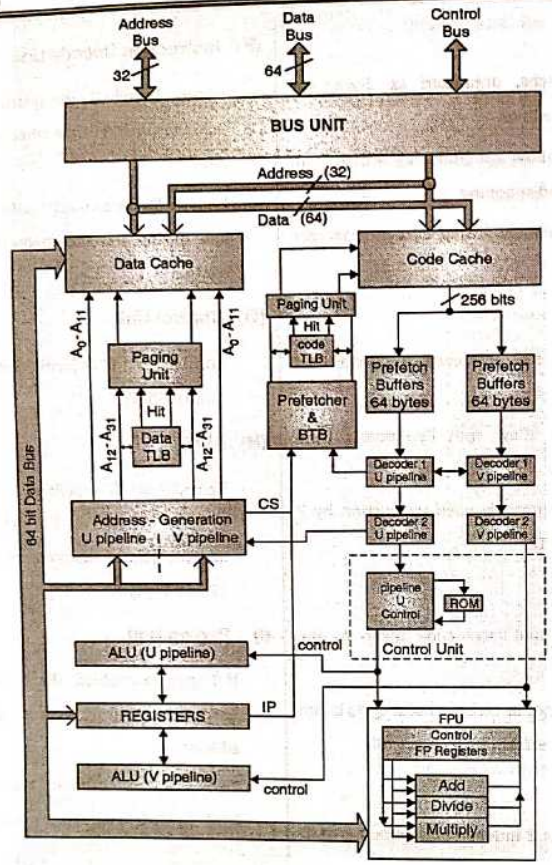


Fig. 13.2.2

13.3 Cache Basics

Before going to the cache of Pentium processor, we will see some basics of the cache like its operation, advantage, principles of locality of reference, cache architectures, write policies etc.

13.3.1 Cache Operation

1. Implementation of cache memory subsystem is an attempt to achieve almost all accesses with zero wait state while accessing memory, but with an acceptable system cost.
2. The cache controller maintains a directory to keep a track of the information it has copied into the cache memory.
3. When the processor initiates a memory read bus cycle, the cache controller checks the directory to determine if it has a copy of the requested information in cache memory.

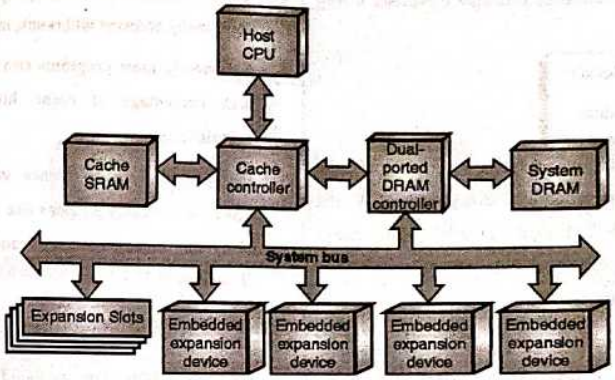


Fig. 13.3.1

4. If the copy is present, the cache controller reads the information from the cache, sends it to the processors data bus, and asserts the processor's ready signal. This is called as **READ HIT**.
5. If the cache controller determines that it does not have a copy of the requested information in its cache, the information is now read form main memory (DRAM). This is known as **READ MISS** and causes wait states due to slow access time of DRAM.
6. The requested information is from the DRAM given to the processor. The information is also copied into the cache memory by cache controller andit updates its directory to track the information stored in cache memory.

Assume the cache memory is empty, in the beginning (after reset). The following sequence takes place:

1. The processor performs a memory read cycle to fetch the first instruction from memory.
2. The cache controller uses the address issued by the processor to determine if a copy of the requested information is already in the cache memory. But a cache miss occurs as the cache memory is empty.
3. The cache controller initiates a memory read cycle to fetch the requested information from DRAM memory. This will consume some wait states.
4. The information from DRAM memory is sent to the processor. It is also copied into the cache memory and the cache controller updates its directory to reflect the presence of the new information. The information being sent is not just the required instruction, but a block (line) of data is sent to the cache. No performance gain is achieved due to absence of information in cache memory.
5. After executing the first instruction, the processor's prefetcher requests a series of memory read bus cycles to fetch the remaining instructions in program loop. But since an entire line was brought earlier, most of the required instructions will be in cache and hence resulting in faster access and performance gain. If the cache is sufficiently large, all instructions in the program loop can become resident in cache memory.
6. The program has loop instruction to jump to the beginning of the loop start over again. The processor then requires the same program again.
7. When the processor requests for the first instruction in the loop, cache controller detects the presence of the instruction in the cache memory and hence provides it to the processor with zero wait states.

13.3.2 Principles of Locality

1. **Locality of reference** is the term used to explain the characteristics of programs that run in relatively small loops in consecutive memory locations.

2. The locality of reference principle comprises of two components:

   1) Temporal locality
   2) Spatial locality

3. The performance gains are realized by the use of cache memory subsystem are because of most of the memory accesses that require zero wait states due to principles of locality.

1. **Temporal Locality**

   (a) Since the programs have loops, the same instructions are required frequently, i.e. the programs tend to use the most recently used information again and again.

   (b) If for a long time a information in cache is not used, it is less likely to be used again.

   (c) This is known as the principle of temporal locality.

2. **Spatial Locality**

   (a) Programs and the data accessed by the processor mostly reside in consecutive memory locations.

   (b) This means that processor is likely to need code or data that are close to locations already accessed.

   (c) This is known as the Principle of Spatial Locality.

13.3.3 Cache Performance

1. Performance of cache subsystems depends on the frequency of cache hits, usually termed as **hit rate.**

$$\%\ \text{HIT RATE} = \frac{\text{Cache Hits}}{\text{Total Memory Accesses}} \times 100\%$$

2. If a program requires a small area of memory and consists of loops, then maximum cache hits are possible.

3. On the other hand, if the program has non-looping code, many accesses will result in cache misses.

4. Fortunately, most programs run in loops and hence a high percentage of cache hits (85 – 95%) is experienced.

5. Besides locality of reference various other factors contribute to cache hit rates like Cache's Architecture, Size of cache Memory and Cache Memory Organization.

13.3.4 Cache Architectures

Two basic architectures are found in today's systems:

1. Look-through cache design
2. Look-aside cache design

1. **Look-through cache designs**

   (a) The performance of systems incorporating Look Through Cache is typically higher than that of systems incorporating Look Aside Cache.

   (b) Data from main memory (DRAM) is not transferred to the processor using system bus hence system bus is free for other bus masters (like DMAC) to access the main memory.

   (c) This system isolates the processor's local bus from the system bus hence achieving bus concurrency.

   (d) The major advantage is that two bus masters can operate simultaneously. One processor accesses look through cache while another bus master such as DMA can access the system bus. Hence concurrency.

   (e) To expansion devices, a look-through cache controller is like a system processor.

   (f) During memory writes, look-through cache provides zero wait state operation (using posted writes) for write misses.

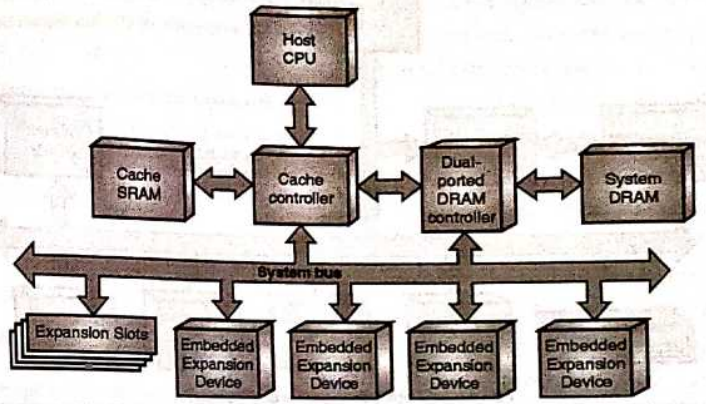


Fig. 13.3.2 :Look Through Cache architecture

Advantages

a) It reduces the system and memory bus utilization, leaving them available for use by other bus master.

b) It allows bus concurrency, where both the processor and another bus master can perform bus cycles at the same time.

c) It also completes write operations in zero wait states using posted writes.

Disadvantages

a) In the event that the memory request is a cache miss, the lookup process delays the request to memory. This delay is called as lookup penalty.

b) It is more complex, costly and difficult to design and implement.

2. **Look-aside cache designs**

   (a) In this case the processor is directly connected to the system bus or memory bus.

   (b) When the processor initiates a bus access, cache controller as well as main memory detects the bus access address.

   (c) The cache controller sits aside and monitors each processor memory request to determine if the cache contains the copy of the requested information.

   (d) If it is a cache hit, the cache controller terminates the bus cycle by instructing memory subsystem to ignore the request. If it is a cache miss, the bus cycle completes in normal fashion from memory (and wait states are required).

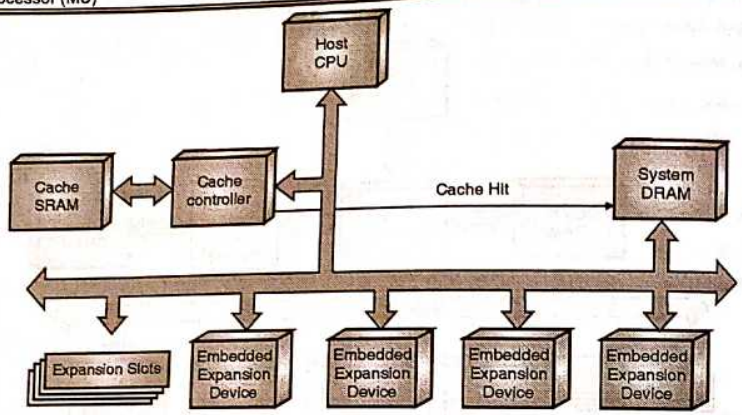


Fig. 13.3.3 Look aside cache architecture

Advantages

(a) Cache miss cycles complete faster in Look Aside Cache as the bus cycle is already in progress to memory and hence no look up penalty is incurred.

(b) Simplicity of designs because only one address is to be monitored by cache controller form processor and not from I/O devices.

9c) Lower cost of implementation due to their simplicity.

Disadvantages

(a) The processor requires system bus utilization for its every access, to access both cache subsystem and memory.

(b) Concurrent operations are not possible as all masters reside on the same bus.

13.3.5 Cache Consistency (also known as cache coherency)

1. In order for cache subsystems to work properly, the CPU and the other bus masters must be getting the most updated copy of the requested information.
2. There are several cases wherein the data stored in cache or in main memory may be altered whereas the duplicate copy remains unchanged.

Causes of cache consistency problems

1. When the copy of line in cache no longer matches the contents of line stored in memory, there is loss of cache consistency. It can be either due to cache line being updated while the memory line is not, or the memory line being updated while the cache line is not.
2. In each of these instances the stale data must be updated. It can be a result of cache write hit and hence the caches write policy has to handle this problem for the first case.
3. For the second case the coherency problem is due to some other bus master changing the data in memory. This change is to be updated in cache line by the cache controller, hence the cache controller has to monitor the system bus.

13.3.6 Write Policy

1. When the write hit occurs, the cache memory is updated and it contains the latest data while memory contains stale data.

2. Such a cache line is called as dirty or 'modified' because it no longer mirrors its corresponding line in memory.
3. In order to correct this cache consistency problem, the corresponding memory line must be updated to reflect the change made in the cache, else another bus master may get stale data if it reads from these lines.
4. Three write policies are used to prevent this type of consistency problem :Write-through, Buffered or posted Write-through and Write-back.

13.3.6(A) Write-Through Cache Designs

1. In this write policy, the data is passed to the memory immediately, so that the memory has the updated data.
2. Even on write hit operation, the cache controller updates the line in the cache and the corresponding line in memory, and hence ensuring that consistency is maintained between cache and memory.
3. Very simple and effective implementaion
4. But poor performance due to slow main memory writes operation.
5. Also it doesn't allows bus concurrency.

13.3.6(B) Buffered or Posted Write-Through Designs

1. It has an advantage of providing zero wait state write operation for cache hits as well as cache misses.
2. When a write occurs, buffered write through caches tricks the processor into thinking that the information was written to memory in zero wait states, when, in fact, the write to main memory has not been performed yet.
3. The look-through cache controller stores the entire write operation in a buffer, and writes to the main memory later.

Hence the processor need not perform slow write operation with wait states and hence doesn't impact processor's performance. This is assuming that the posted write buffer is only one transaction deep.

4. But, if there are two back-to-back memory write bus cycles, the cache controller will insert wait states into second bus cycle until the first write to memory has actually been completed. The bus controller will then post the second bus cycle and assert the processor's ready line. But since processor typically writes only one write operation, the memory writes are completed in zero wait states.
5. With this policy, another bus master is not permitted to use the bus until the write-through is completed, thereby ensuring that the bus master will receive the latest information from memory.

- The write-through operations use either system or memory bus. Hence when write-through to memory is in progress, bus masters are prevented from accessing memory.
- But actual cache consistency problem occurs only when the bus master reads from a location in memory that is stale. The frequency of this type of occurrence is very less. In fact, the memory line is likely to be updated many times by the processor before another bus master reads from that particular line.
- As a result the write-through and buffered write-through designs, update memory each time a memory write is performed, although the need for such action may not be required immediately.

13.3.6(C) Write-Back Cache Designs

1. Write-back designs improve the overall system performance by updating a line in main memory only when necessary, thereby keeping the system bus free for use by other processors and bus masters and hence ensuring bus concurrency.
2. The Memory is updated only when:
   (a) Another bus master initiates a read operation from a memory line that contains stale data.
   (b) Another bus master initiates a writeoperation from a memory line that contains stale data.
   (c) The cache line that contains modified information is about to be overwritten in order to store a line newly acquired from memory i.e. during line replacement.
3. Cache controller marks the cache lines as 'modified' in the cache directory when the processor updates them. Hence when read by another master or written into the memory, the cache subsystem checks whither it is marked as 'modified' in cache.
4. They design of such cache controller is COMPLICATED to implement because they must MAKE DECISIONS on when to write 'modified' lines back to memory to ensure consistency.

13.4 The MESI Model

University Question

Q. Draw and explain the state diagram of MESI transitions that occur within the Pentium's data Cache memory. MU - May 13, 10 Marks

Based on the discussion in the previous sections we can show the state transition diagram of a line in cache as shown in Fig. 13.4.1.

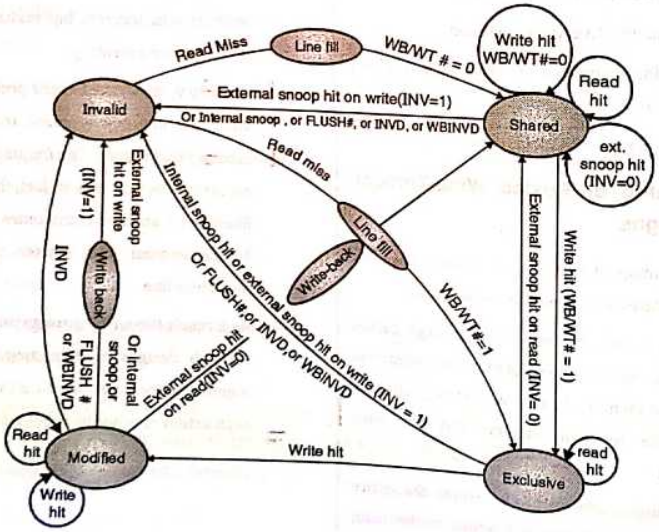


Fig. 13.4.1 : State diagram of MESI transition states as in Pentium's Data cache

Besides the basic Pentium read/write and another bus master read/write operations, there are some more cases shown in the MESI model.

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.

13.5 Code Cache Organization and Operation

University Questions

Q. Explain the cache organization of Pentium. MU - Dec. 11, May 12, May 13, 10 Marks

Q. Write short note on : Cache memory organization. MU - May 14, Dec. 14, 5 Marks

Q. Explain, in brief, cache organization of pentium processor. MU - May 16, 8 Marks

Q. Explain, with neat diagram, cache memory organization is supported by Pentium processor. MU - Dec. 15, 10 Marks

Q. Explain, with neat diagram, cache memory organization is supported by Pentium processor. MU - May 17, 10 Marks

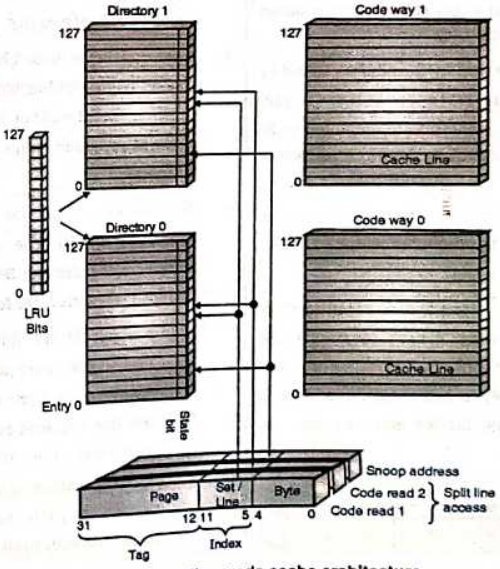


Fig. 13.5.1 : Pentium code cache architecture

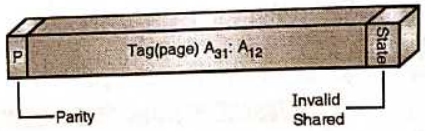


Fig. 13.5.2 : Code cache directory entry

- 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. 13.5.2.
- 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. 13.5.3.

Fig. 13.5.3 : 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.

13.5.1 Line Storage Algorithm

1. 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.
2. 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.
3. 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.
4. 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 8th 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.

13.5.2 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:

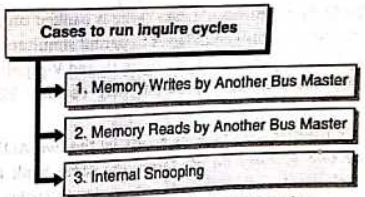


Fig. 13.5.4 : Cases to inquire cycles

1. **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.
2. **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.
3. **Internal Snooping**

- 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.
- 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.

13.5.3 Split Line Access

1. 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.
2. 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.
3. 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.
4. But, when the split line is read from the cache, the information is not properly aligned.
5. 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. 13.5.5.

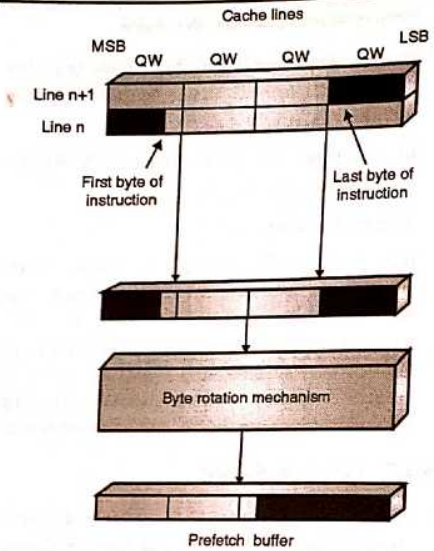


Fig. 13.5.5 : Split line access

6. 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.

7. When an instruction is decoded for the first time, the length of the instruction is fedback by the D1 unit to the code cache.

13.6 Data Cache

University Questions

Q. Explain the data cache organization of pentium. **MU - Dec. 12, 10 Marks**

Q. Explain data cache organization of pentium and give emphasis on triple ported access of data cache. **MU - Dec. 13, 10 Marks**

Q. Discuss data cache organization of Pentium. **MU - Dec. 17, 10 Marks**

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. 13.6.1.

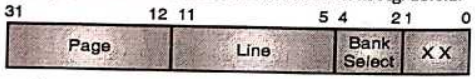


Fig. 13.6.1 : Address as viewed by L1 data cache

13.6.1 L1 Data Cache Structure

University Questions

Q. Write short note on : Data cache organization of Pentium. **MU - May 15, 5 Marks**

1. The 8KB internal data cache contains triple-ported high speed SRAM. It is organized into two 4KB ways referred to as way zero and way one.
2. Each way consists of 128 lines numbered 0 through 127 with a line size of 32 bytes.
3. Each data cache line consists of 8 doublewords and the cache ways are banked on doubleword boundaries.
4. A parity is generated for each byte within a line that is placed in the internal cache. When a byte of information is read from cache, parity is checked and in case of parity error being detected, an internal parity error is signaled to external logic through the IERR (Internal ERRor) output. Also, the processor generates a special shutdown bus cycle and stops execution.
5. When a line of information is read from a page of external memory, the cache controller stores the line in one of the two internal cache memory ways using the line storage algorithm.
6. The Fig. 13.6.2 shows the internal structure of L1 data cache.

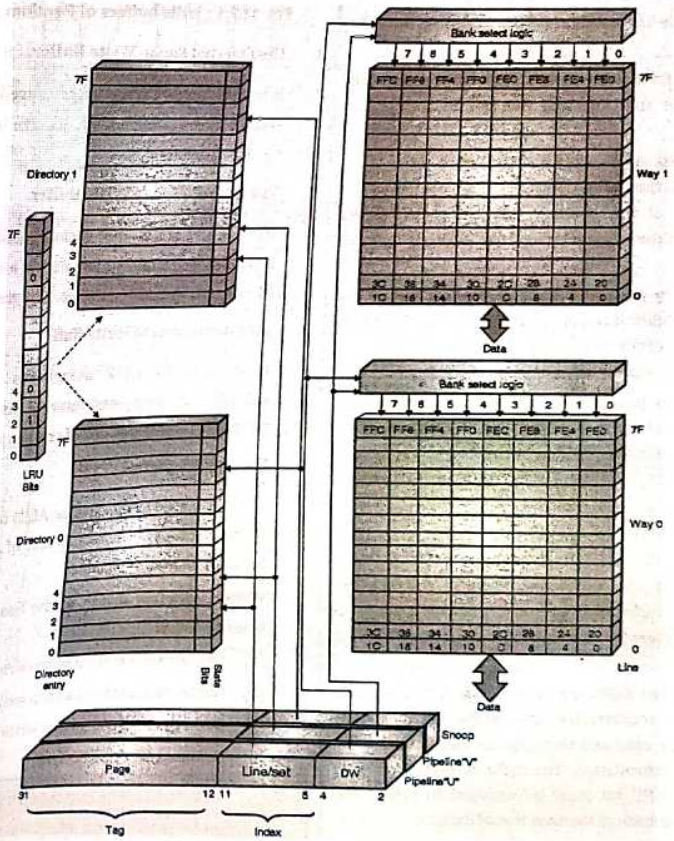


Fig. 13.6.2 : L1 Data cache structure

7. Each directory entry has a tag (or page) field used to record the page number of the memory page where the line of information came from. This directory entry is as shown in the Fig. 13.6.3

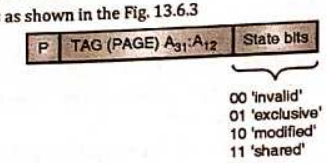


Fig. 13.6.3:Data cache Directory entry structure

8. When a line of information is read from memory, the cache controller selects the directory entry of way zero using the line number. It then examines the state bits.
9. If the state bits indicate invalid, no further checking is necessary for that line and the line in way one is checked.
10. If the state bits are checked and the line in either of the way is valid, the cache controller then compares the tag portion of the memory address given by the processor to the tag address stored in the entry's tag field.
11. If the memory page address requested is same as the tag field of either directory, it indicates a cache hit i.e. it has a copy of the addressed line stored within cache in the line associated with the entry.
12. If the memory page address requested is not same as the tag field of either directory i.e. it is a cache miss, the cache controller issues a cache line-fill request to the bus unit to fetch the desired information from L2 cache memory.
13. When the requested line of information is fetched from external memory, the cache controller places a copy of it in the cache and updates the affected directory to reflect its presence and MESI state in which it is stored. It also updates the corresponding LRU bit.
14. If both the Tag fields are currently in use, the cache controller must determine which of the two lines is the least recently used and then replace that line with the line of new information. The cache directory entry as well as the LRU bit must be updated to reflect the presence and state of the new line of data.

13.7 Write Buffers

Pentium processor has the following write buffers:

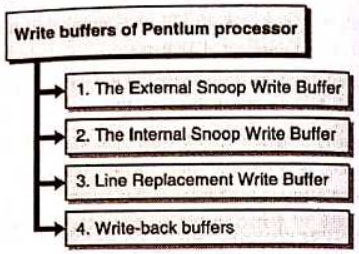


Fig. 13.7.1 : Write buffers of Pentium processor

1. **The External Snoop Write Buffer**

   It is used to store write-backs caused by an external snoop hit to a modified line. Its size is as that of one line i.e. 32-bytes.

2. **The Internal Snoop Write Buffer**

   It is used to store write-backs caused by an internal snoop hit to a modified line during a code cache miss. Its size is as that of one line i.e. 32-bytes.

3. **Line Replacement Write Buffer**

   It is used to store write-backs caused by the replacement of a least-recently used line that is modified. Its size is as that of one line i.e. 32-bytes.

4. **Write-back buffers**

   These are used by the pipeline ALUs to write the result to the memory. Its size is as that of ALU i.e. 32-bytes for each 'U' and 'V' pipelines.

   Write cycles are driven to the bus in the following priority order:

   (a) Contents of external snoop write buffer.
   (b) Contents of internal snoop write buffer.
   (c) Contents of replacement write buffer.
   (d) Contents of write buffers.

Note : To maintain cache consistency all write buffers are also snooped during external snoop operations.

13.8 Superscalar Operation

Pentium processor is a superscalar processor i.e. there are two pipelines and two execution units. The pipelining of superscalar is shown in Fig. 13.8.1. The Pentium processor yields its best performance when

1) Instruction Prefetch results in internal cache hit on the code cache, thereby reducing the chances for execution units to stall while waiting for the next instruction.
2) Sequential instructions are paired and preceded in parallel through the two instruction pipelines.
3) Branch instructions are correctly predicted by the branch prediction logic.

The pipelining of integer unit of Pentium processor is shown in the Fig. 13.8.1.

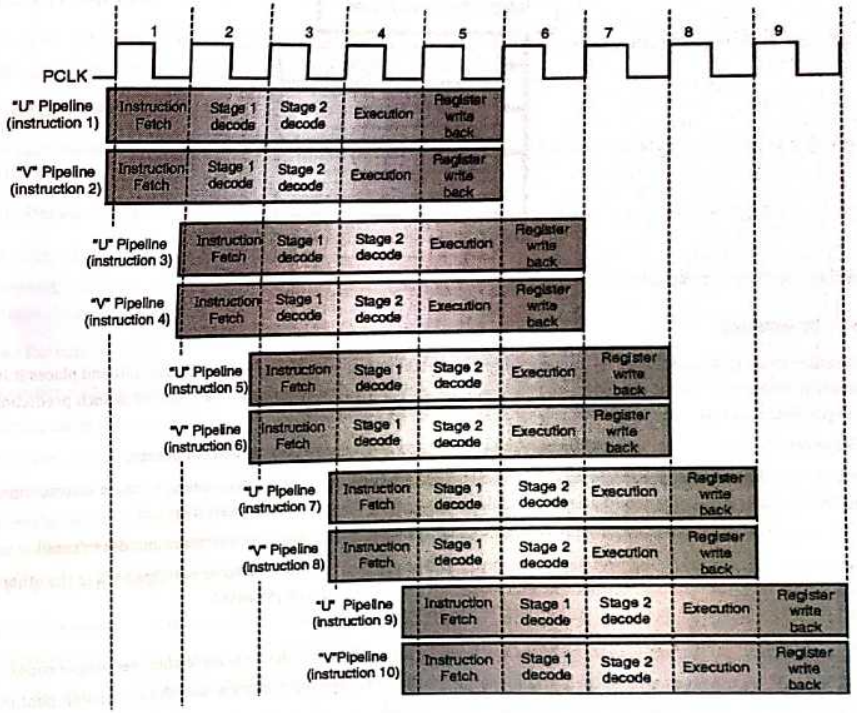


Fig. 13.8.1 : Integer pipelining in Pentium

13.9 Integer Pipelining Stages

Q. Explain in brief integer instruction pipeline stages of Pentium processor. List the steps in instruction issue algorithm. MU - May 12, 10 Marks

Q. Explain different stages of Integer pipeline and floating point pipeline of Pentium processor. MU - May 13, Dec. 13, 10 Marks

Q. Explain integer pipeline of pentium processor. MU - May 14, 10 Marks

Q. Explain in brief, pipeline stages on Pentium processor. MU - May 16, 5 Marks

The pipelining of superscalar is shown in Fig. 13.9.1. The pipeline stages of Pentium processor and their functions are listed below :

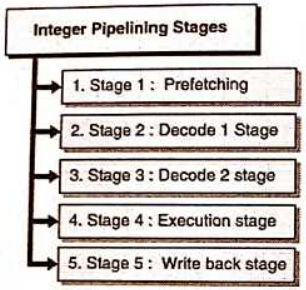


Fig. 13.9.1 : Integer pipelining stages

Stage 1 : Prefetching

- During normal operation with the cache enabled, Prefetcher receives a line of code from bus unit and places it into the active prefetch (i.e. the queue which is active at a time).And a queue remains active until the branch prediction logic has predicted a branch to be taken.
- The prefetch queue switch takes place only in case the branch prediction logic predicts a branch.
- In case of branch being predicted by the branch prediction logic, the newly active queue supplies instructions to the two instruction pipelines immediately behind the jump instruction predicted to cause a branch.
- If the branch is taken the instruction in the next queue are right instructions. The execution unit doesn't stall.
- But, if branch is not taken the prefetch queue and pipeline are flushed. The prefetcher switches back to the other queue which then begins to supply instructions that were fetched prior to branch predicted.
- The flush and switch causes the execution units to stall for 3-4 clocks.
- If the prediction in D1 stage is that the branch will not be taken the prefetch continues fetching sequential codes.
- But, if prediction proves wrong then when the instruction reaches the execution unit the instruction behind it are incorrect and hence the entire pipeline and queue are flushed.
- The branch target location is then fetched into the prefetcher queue and the target location instructions are fetched to the two pipelines.

Stage 2 : Decode 1 Stage

The basic functions of Decode 1 or D1 stage

(1) Checking for pairability of instructions
(2) Branch Prediction
(3) Instruction Boundary Information to Code cache

1. Checking for pairability of instructions

If pairable (rules for pairability are discussed in Section 13.7.1, both the instruction move in together, i.e. if an instruction takesmore time in a particular stage the other will wait in corresponding stage.

If not pairable the instruction in D1 stage of V pipeline is deleted andPrefetcher will shift it to D1 stage of U pipeline when the earlier moves to D2 stage of 'U' pipeline.

2. Branch Prediction

If either instruction is branching instruction, the branch prediction logic makes a prediction whether branching will take place. If predicted to be taken, it will indicate about it to the prefetcher.

3. Instruction Boundary Information to Code cache

D1 stage informs about the instruction boundaries to the code cache to help the prefetcher to fetch proper instructions to the two pipelines.

Stage 3 : Decode 2 stage

The basic functions of D2 stage

1) Operands calculated for operands residing in memory. It can handle instruction containing both displacement and immediate values or using base and index addressing in a single clock cycle.

Stage 4 : Execution stage

- Execution unit comprises of ALU for 'U' pipeline (with barrel shifter) and for V pipeline.
- Data accesses (form cache or main memory) are done by this stage.
- Accesses to data cache can be made simultaneously by 'U' pipeline and 'V' pipeline.
- Both instruction enter execution stage at the same time, but here if 'U' pipeline execution is over it moves on to write backstage while 'V' pipeline instruction is under execution.But the reverse i.e. if 'V' pipeline instruction finishes first it is not allowed to move forward instead it waits there for 'U' pipeline instruction.

Stage 5 : Write back stage

Target registers or memory locations are updated during write back stage. This also includes updating EFLAGS register.

13.9.1 Instruction Pairability Mechanism or Instruction Pairing Rules

University Questions

Q. Enlist the instruction pairing rules of 'U' and 'V' pipeline in Pentium. MU - May 11, May 12, Dec. 14, 5 Marks

Q. Explain instruction pairing rules of Pentium. Also explain the "Instruction Issue" algorithm in detail. MU - Dec. 11, 10 Marks

Q. Explain in brief integer instruction pipeline stages of Pentium processor. List the steps in instruction issue algorithm. MU - May 12, 10 Marks

Q. Explain the floating point pipeline stages. Also explain the list of steps in instruction issue algorithm. MU - Dec. 12, 10 Marks

Q. Explain how flushing of pipeline can be minimized in Pentium architecture. Also explain the instruction pairing rules for Pentium. MU - Dec. 12, 10 Marks

Q. Explain instruction pairing on pentium processor. MU - May 14, 5 Marks

Q. Write the instruction issue algorithm used in Pentium. MU - May 15, Dec. 17, May 19, 5 Marks

Q. Explain, in brief, cache organization of pentium processor. MU - May 16, 8 Marks

- 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:

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.

13.9.1(A) Register Contention

University Questions

Q. Explain instruction pairing rules of Pentium. Also explain the "Instruction Issue" algorithm in detail. MU - Dec. 11, 10 Marks

Q. Explain the floating point pipeline stages. Also explain the list of steps in instruction issue algorithm. MU - Dec. 12, 10 Marks

Q. Explain an instruction issue algorithm of Pentium processor. MU - Dec. 18, 5 Marks

Register contention results if 2 instructions attempt to access the same register during parallel execution. Register contentions are classified as:

(A) Explicit Register Contention
(B) Implicit Register Contention

(A) Explicit Register Contention

The different cases of Explicit register contention are as follows

1. RAW (Read after Write)

   MOV AX, 004BH

   MOV [BP], AX

   i.e. Register write followed by same Register read.

   If both instruction are paired then we get ERRATIC result. Since AX will be updated by 'U' pipeline in write back stage, while 'V' pipeline will read it in ALU stage; 'V' pipeline will take old data from AX.

2. WAW (Write after Write)

   MOV AX, 004BH

   XLAT

   i.e. Register write followed by same register write.

   If paired then we get ERRATIC result.

3. Writing to different parts of the same register

   MOV AL, 05H

   MOV AH, 0AH

   i.e. both instructions write to different parts of same register.

These explicit register contentions are solved by allowing only first instruction in U pipeline. Besides that 'U' pipeline is allowed to write-back first while 'V' pipeline to stall.

(B) Implicit Register Contention

It occurs if two instruction imply reference to same register.

MOV AX, [SI]

MOV BX, [SI + 4]

There are certain exclusions to implicit register Contention. Two exceptions for implicit register contention are:

1. Flags Reference- Compare and Branch operations

   eg. CMP and JC or

   ADD and JNZ.

These pair of instructions require flags i.e. contention, but still are allowed to be used simultaneously.

2. Stack pointer Reference : Pushes and Pops

   eg. PUSH and PUSH or

   PUSH and CALL or

   POP and POP.

Instruction issue algorithm

Based on the above discussion we can formulate the instruction issue algorithm 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

13.10 Floating Point Pipeline Stages

University Questions

Q. Explain different stages of floating point pipeline of Pentium Processor. MU - May 11, 5 Marks

Q. Explain floating point pipeline stages of Pentium processor. MU - Dec. 11, 10 Marks

Q. Explain different stages of Integer pipeline and floating point pipeline of Pentium processor. MU - May 13, Dec. 13, 10 Marks

- Floating point unit is heavily pipelined hence allowing several instructions to be executed simultaneously under certain conditions.
- Most of the floating Point instructions have to be in the 'U' pipeline only and cannot be paired with integer instructions.
- The first four stage of the floating point pipeline are shared with the integer pipeline units. The 8 pipeline stages of the floating point unit are as follows:

1. Prefetch
2. Instruction Decode 1 (D1) } Same as Integer Execution
3. Instruction Decode 2 (D2) } Stage Functions
4. **EX (Execution stage) :** This stage performs register reads, memory reads or memory writes as required
5. **FP E1 (Floating Point Execution 1) :** This stage reads the information from Register / memory and moving them into FP register. It is also converted to FP format.
6. **FP E2 (Floating Point Execution 2) :** In this stage the FP operation is performed.
7. **Write FP result :** In this stage the result is rounded off and written in target FP register.
8. **Error reports :** If error detected, report.

13.10.1 Floating Point Instruction Pairing

- FP simple instruction if given to 'U' pipeline can be paired with FXCH in 'V' pipeline.
- FP Simple Instructions are:

1. FLD 2. FLD ST(i) 3. FADD
4. FSUB 5. FMUL 6. FDIV
7. FCOM 8. FUCOM 9. FTST
10. FABS 11. FCHS

- Because most of the operations require operand to be on top of stack register, FXCH is required many times and hence it is kept pairable.
- FPU also permits pipelining instruction in itself except

1. FDIV which requires all FPU resources and hence no simultaneous instruction can be executed.
2. Two FMUL cannot be executed simultaneous due to resource limitations
3. One FMUL can be executed in parallel will one or two FADD
4. Three FADD instructions can be executed simultaneously.

13.11 Branch Prediction Logic

University Questions

Q. Explain how the flushing of pipeline problem is minimized in Pentium architecture. MU - May 11, Dec. 11, May 12, Dec. 14, Dec. 17, 10 Marks

Q. Write short note on following : Branch prediction logic. MU - May 11, 5 Marks

Q. Explain how flushing of pipeline can be minimized in Pentium architecture. Also explain the instruction pairing rules for Pentium. MU - Dec. 12, Dec. 19, 10 Marks

Q. Explain the dynamic branch prediction logic in Pentium. MU - Dec. 12, May 13, 10 Marks

Q. Explain branch prediction logic implemented on pentium processor. MU - May 14, 10 Marks

Q. Explain the branch prediction logic used in Pentium processor. MU - Dec. 18, 10 Marks

- 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 andEU indicates that the branching is taken then it remains here.
- If the state is here andEU indicates that the branching is not taken then is degraded to weakly taken.

2. **Weakly taken**

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

3. **Weakly not taken**

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

4. **Strongly not taken**

- If the state is here andEU 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.
- Thus, 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.
- While, 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.
- And if the branch was correctly predicted not to be taken, history bits are downgraded and no action required.
- While, 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 in this case because there are two prefetch queues.

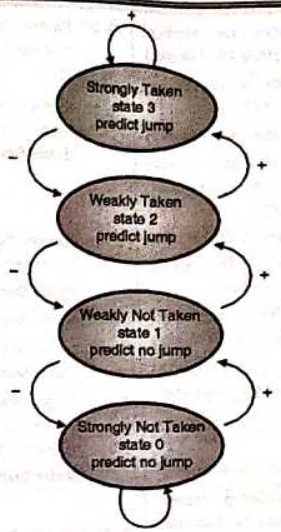


Fig. 13.11.1 : State transition diagram of BTB entry

13.12 Exam Pack (Review and University Questions)

Q. 1 Write down features of Pentium processor.
(Refer Section 13.1) **(May 15, 5 Marks)**

Q. 2 Draw and explain pentium processor architecture.
(Refer Section 13.2) **(May 13, May 16, 10 Marks)**

Q. 3 Explain pentium processor architecture with block diagram. **(Refer Section 13.2)** **(May 14, 10 Marks)**

Q. 4 Draw and explain block diagram of Pentium processor. **(Refer Section 13.2)** **(May 17, 10 Marks)**

Q. 5 Draw and explain the state diagram of MESI transitions that occur within the Pentium's data Cache memory.
(Refer Section 13.4) **(May 13, 10 Marks)**

Q. 6 Explain the cache organization of Pentium.
(Refer Section 13.5)
(Dec. 11, May 12, May 13, 10 Marks)

Q. 7 Write short note on : Cache memory organization.
(Refer Section 13.5) **(May 14, Dec. 14, 5 Marks)**

Q. 8 Explain, in brief, cache organization of pentium processor.
(Ref. Sections13.5 and 13.7.1) **(May 16, 8 Marks)**

Q. 9 Explain, with neat diagram, cache memory organization is supported by Pentium processor.
(Ref. Sections 13.5 and 13.6)
(Dec. 16, May 17, 10 Marks)

Q. 10 Explain the data cache organization of pentium.
(Refer Section 13.6) **(Dec. 12, 10 Marks)**

Q. 11 Explain data cache organization of pentium and give emphasis on triple ported access of data cache.
(Refer Section 13.6) **(Dec. 13, 10 Marks)**

Q. 12 Discuss data cache organization of Pentium.
(Refer Section 13.6) **(Dec. 17, 10 Marks)**

Q. 13 Write short note on : Data cache organization of Pentium. **(Refer Section 13.6.1)** **(May 15, 5 Marks)**

Q. 14 Explain in brief integer instruction pipeline stages of Pentium processor. List the steps in instruction issue algorithm.
(Refer Sections 13.9 and 13.7.1(A))
(May 12, 10 Marks)

Q. 15 Explain different stages of Integer pipeline and floating point pipeline of Pentium processor.
(Refer Sections 13.9)(May 13, Dec. 13, 10 Marks)

Q. 16 Explain integer pipeline of pentium processor.
(Refer Section 13.9) **(May 14, 10 Marks)**

Q. 17 Explain in brief, pipeline stages on Pentium processor. **(Refer Section 13.9) (May 16, 5 Marks)**

Q. 18 Enlist the instruction pairing rules of 'U' and 'V' pipeline in Pentium. **(Refer Section 13.9.1) (May 11, May 12, Dec. 14, 5 Marks)**

Q. 19 Explain instruction pairing rules of Pentium. Also explain the "Instruction Issue" algorithm in detail. **(Refer Sections 13.9.1 and 13.9.1(A)) (Dec. 11, 10 Marks)**

Q. 20 Explain the floating point pipeline stages. Also explain the list of steps in instruction issue algorithm. **(Refer Sections 13.9.1 and 13.9.1(A)) (Dec. 12, 10 Marks)**

Q. 21 Explain how flushing of pipeline can be minimized in Pentium architecture. Also explain the instruction pairing rules for Pentium. **(Refer Section Section 13.9.1) (Dec. 12, Dec. 19, 10 Marks)**

Q. 22 Explain instruction pairing on pentium processor. **(Refer Section 13.9.1) (May 14, 5 Marks)**

Q. 23 Write the instruction issue algorithm used in Pentium. **(Refer Section 13.9.1) (May 15, Dec. 17,May 19, 5 Marks)**

Q. 24 Explain, in brief, cache organization of pentium processor. **(Refer Sections 13.5 and 13.9.1) (May 16, 8 Marks)**

Q. 25 Explain the floating point pipeline stages. Also explain the list of steps in instruction issue algorithm. **(Refer Sections 13.9.1 and 13.9.1(A)) (Dec. 12, 10 Marks)**

Q. 26 Explain different stages of floating point pipeline of Pentium Processor. **(Refer Section 13.10) (May 11, 5 Marks)**

Q. 27 Explain floating point pipeline stages of Pentium processor. **(Refer Section 13.10) (Dec. 11, 10 Marks)**

Q. 28 Explain different stages of Integer pipeline and floating point pipeline of Pentium processor. **(Refer Sections 13.9and 13.10) (May 13, Dec. 13, 10 Marks)**

Q. 29 Explain how the flushing of pipeline problem is minimized in Pentium architecture. **(Refer Section 13.11) (May 11, Dec. 11, May 12, Dec. 14, Dec. 17, 10 Marks)**

Q. 30 Write short note on following : Branch prediction logic. **(Refer Section 13.11) (May 11, 5 Marks)**

Q. 31 Explain how flushing of pipeline can be minimized in Pentium architecture. Also explain the instruction pairing rules for Pentium. **(Refer Sections 13.11 and 5.7.1) (Dec. 12, 10 Marks)**

Q. 32 Explain the dynamic branch prediction logic in Pentium. **(Refer Section 13.11) (Dec. 12, May 13, 10 Marks)**

Q. 33 Explain branch prediction logic implemented on pentium processor. **(Refer Section 13.11) (May 14, 10 Marks)**

Q. 34 Explain branch prediction logic used in Pentium. **(Refer Section 13.11) (May 14, 10 Marks)**

Q. 35 Explain the branch prediction logic used in Pentium processor.**(Refer Section 13.11) (Dec. 18, 10 Marks)**

Q. 36 Explain an instruction issue algorithm of Pentium processor. **(Refer Section 13.9.1(A)) (Dec. 18, 5 Marks)**

❑❑❑

14 MODULE 6

Pentium 4 Processor

14.1 Pentium Pro Processor

- Pentium pro microprocessor is available in two versions: one that contains 256KB L2 cache, while another that contains 512KB L2 cache.
- Another notable difference between Pentium and Pentium Pro is that Pentium Pro has 36 address lines as against 32 in Pentium. Hence Pentium Pro can access up to 64GB physical memory. It requires 3.3 to 2.7 volts supply. It requires a maximum of 9.9A and hence a maximum power dissipation of 26.7W.
- Each Pentium Pro output pin is capable of providing an ample current of 48 mA, compared to 2 mA as on earlier processors. Hence only very large systems require external bus buffers.

14.1.1 Internal Structure of the Pentium Pro Processor

- Earlier microprocessors contained an execution unit and a bus interface unit and a cache buffering system. The structure is modified in later microprocessors, but the modifications were some additional stages as shown in the Fig. 14.1.1.
- The Pentium Pro processor has an additional 256 / 512 KB L2 cache memory on chip. The introduction of on-chip L2 cache speeds processing and reduces the number of components in a system.
- The L2 cache is connected to BIU, which in turn provides access to the L1 cache. The L1 code cache provides the instructions to the Instruction fetch and decode unit. The instructions are then put into the instruction pool as shown in the Fig. 14.1.1.

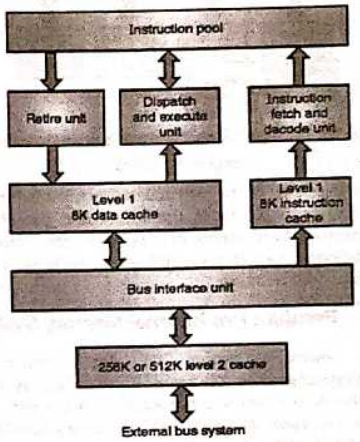


Fig. 14.1.1 : Pentium Pro internal structure

- The instruction pool is a memory accessible with its content i.e. content addressable or associative memory. The decoded instructions from this pool are dispatched to the execute unit by the dispatch unit.
- The Instruction Fetch and Decode Unit (IFDU), contains three separate instruction decoders that decode three instructions simultaneously. Also included in this IFDU is branch prediction logic. The branch prediction logic predicts if the branch will be taken or not for a conditional jump instruction. The branch prediction logic looks ahead in code sequence to determine the next instruction in the flow of program.
- The decoded instructions are retrieved by the execution unit and executed. The execute unit consists of three units namely two integer execution unit and one floating point unit.
- This means that the Pentium Pro processor like Pentium processor can execute two integer and one floating point instruction simultaneously.
- The Pentium Pro also consists of a jump execution unit or address generation unit. Hence the Pentium Pro processor takes up to five events and executes up to four of them based on their pairability. The execute unit can access the L1 Data cache and CPU registers.

– The instructions once executed retire and the result is written into the destination location by the retire unit. The scheduling is performed by the Reservation Station (RS) and its connections are as shown in the Fig. 14.1.2.

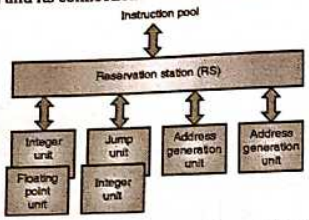


Fig. 14.1.2 : Connection of reservation station and the execution units

– The last stage of the instruction execution is the retire unit. This unit checks the instruction pool for the completed instructions and removes them from the instruction pool. It is capable of removing up to three instructions per cycle.

14.1.2 Pentium Pro Internal Memory System

– The memory system for the Pentium Pro microprocessor is 4G bytes in size, just as in the 80386DX-Pentium microprocessors, but access to an area between 4G and 64G is made possible by additional address lines A_{32}-A_{35}.

– The Pentium Pro uses a 64-bit data bus to address memory organized in eight banks.

– The additional memory is enabled with PAE (bit 5 of CR4), i.e. page address extension, and is accessible only when 2M paging is enabled. The 2M paging is new to the Pentium Pro to allow memory above 4G to be accessed.

– The Pentium Pro memory system is divided into eight banks that each stores a byte of data along with a parity bit. Most Pentium and Pentium Pro microprocessor-based systems do not use of the parity bit. The Pentium Pro, like the 80486 and Pentium, employs internal parity generation and checking unit for the memory system data bus information. The Fig. 14.1.3 shows the eight banks used in Pentium Pro processor

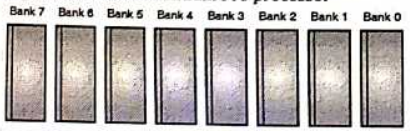


Fig. 14.1.3 : The eight banks in Pentium Pro processor memory system.

14.1.3 Special Pentium Pro Features

Besides the differences discussed in above sections, Pentium Pro processor has a difference in its control register CR4. There are two additional bits as shown in the Fig. 14.1.4.

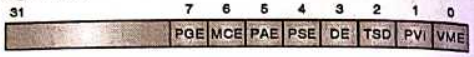


Fig. 14.1.4 : Control Register (CR) 4 of Pentium Pro processor

The two additional bits in the Fig. 14.1.4 as compared with the CR4 of Pentium, discussed in chapter 2, are bit 5 i.e. PAE and bit 7 i.e. PGE. These bits have the following functions :

1. PAE (Page Address Extension): This bit enables the higher four bits of paging i.e. A_{32} to A_{35}. Hence by enabling this bit the physical memory size increases from 4GB to 64GB, with the help of 36 address lines.
2. PGE (PaGe size Extension): This bit along with PSE (Page Size Extension i.e. bit 4), enables the 2 MB page size instead of 4MB in case of Pentium processor.

   If PGE bit is '1', the PSE bit = '1' indicates the page size is 2MB

   the PSE bit = '0' indicates the page size is 4KB

   If PGE bit is '0', the PSE bit = '1' indicates the page size is 4MB

   the PSE bit = '0' indicates the page size is 4KB

14.2 Pentium MMX

The Pentium MMX (MultiMedia eXtension) includes the following special features over Pentium processor:

1. Inclusion of 57 extra instructions named MMX instruction set.

– Helped chips perform multimedia operations at 10 – 20 times better over non – MMX chips.

– Allowed several tasks to be performed in single instruction.

2. 16 KB Data & Code Cache each.
3. 4.4 Million Transistors.
4. Pentium MMX allowed chips to reach speed upto 266 MHz.

14.2.1 Block Diagram of Pentium MMX

The Pentium MMX processor has 5 extra blocks as compared to the Pentium processor as shown in the Fig. 14.2.1. These five blocks are marked in the Fig. 14.2.1 and are explained below:

1. The split code and data cache are of 16KB in Pentium Pro as against 8 KB in Pentium processor
2. A MMX technology unit with SIMD (Single Instruction Multiple Data) instructions required for multimedia operations, is connected along with the floating point unit.
3. The bus connected from the Code cache to the prefetch queue is of 128 bits as against 256 bits in Pentium processor.
4. It includes a DPL (Data Prediction Logic) that predicts the further data required.
5. It also has an on-chip APIC (Advanced Programmable Interrupt Controller). Hence there is no interrupt controller required externally.

Note : All these features are marked with numbers from 1 to 5, to point their positions.

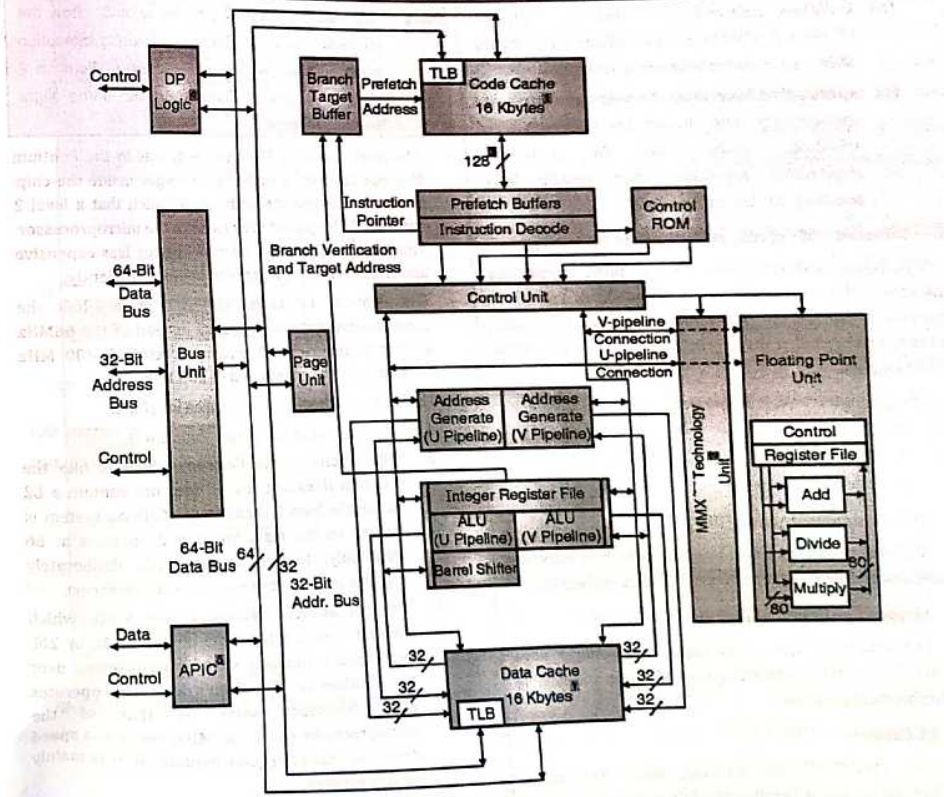


Fig. 14.2.1

14.3 Pentium 2 Processor

The Pentium 2 processor provides the following features:

1. Combination of Pentium and MMX
2. It consists of 7.5 million Transistors.
3. It is available in speeds from 233 MHz upto 450 MHz.
4. It has P6 Microarchitecture Dynamic Execution Technology :
   - (a) **Multiple Branch Prediction :** Predicts program execution through several branches, accelerating the flow of work to the processor.
   - (b) **Dataflow Analysis :** Creates an optimized, reordered schedule of instructions by analyzing data dependencies between instructions.
   - (c) **Speculative Execution :** Carries out instructions speculatively and based on this optimized schedule, ensures that the processor's superscalar execution units remain busy, boosting overall performance.
5. **Includes MMX media enhancement technology :**

Software designed for Intel's MMX technology unleashes the full multimedia capabilities of these processors including full-screen, full-motion video, enhanced color, and realistic graphics. The highlights of the technology are:

Single instruction, Multiple Data (SIMD) technique.
57 new instructions.
Eight 64-bit wide MMX technology registers.
Four new data types.

6. **Dual Independent Bus (DIB) :**

DIB architecture (system bus and cache bus) increases bandwidth and performance over single-bus processors.

7. **Single Edge Contact (SEC) :**

The processor core is package in the Single Edge Contact (SEC) cartridge enabling ease of design and flexible motherboard architecture.

8. **L1 Cache :**

32K (16K/16K) non-blocking, level- one cache provides fast access to heavily used data and code.

9. **L2 Cache :**

512 unified, non-blocking, and level-two cache.

10. **Error Correction Codes (ECC) :**

Includes data integrity and reliability features such as Error Correction Code (ECC), Fault Analysis, Recovery, and Functional Redundancy Checking for both system and L2 cache buses. Parity-protected a address/request and response system bus signals with a retry mechanism for high data integrity and reliability.

11. **Built-in Self Test (BIST) :**

BIST provides testing of the instruction cache, data cache, Translation Lookaside Buffers (TLBs) and ROMs.

Note : Since both Pentium 2 and Pentium 3 follow the same architecture called as "Dynamic execution micro architecture", the architecture of Pentium 2 is similar to that of Pentium 3, with some slight feature differences.

- The level1 cache is 32K-byte as it was in the Pentium Pro, but the level 2 cache is no longer inside the chip. Intel has changed the architecture such that a level 2 cache could be placed very close to the microprocessor. This change makes the microprocessor less expensive and still allows the L2 cache to operate efficiently.
- The Pentium L2 cache operates at one-half the microprocessor clock frequency instead of the 66MHz as that in the Pentium microprocessor. A 400 MHz Pentium II has a cache speed of 200 MHz.
- The Pentium II is available in three versions.
  1. The first is the full-blown Pentium II.
  2. The second is the Celeron, which is like the Pentium II, except that it does not contain a L2 cache; the level 2 cache in the Celeron system is located on the main board and operates at 66 MHz only. Certain features are deliberately disabled in this version and hence is low cost.
  3. The yet another version is the Xeon, which because it uses a L2 cache of 512K, 1M, or 2M, represents significant speed improvement over the Pentium II. The Xeon's L2 cache operates clock frequency same as that of the microprocessor i.e. at 400 MHz, hence at a speed double as that of regular Pentium II. It is mainly used in servers.

14.3.1 The Memory system of Pentium II Processor

The memory system of Pentium II processor is same as that of the Pentium Pro processor except that it doesn't employs parity checks although it is available. The Pentium II memory map is similar to other processors studied till now in this book. The only change is that the AGP area allows the OS to access the video information in the linear address space and hence making it faster. The memory map is shown in Fig. 14.3.1.

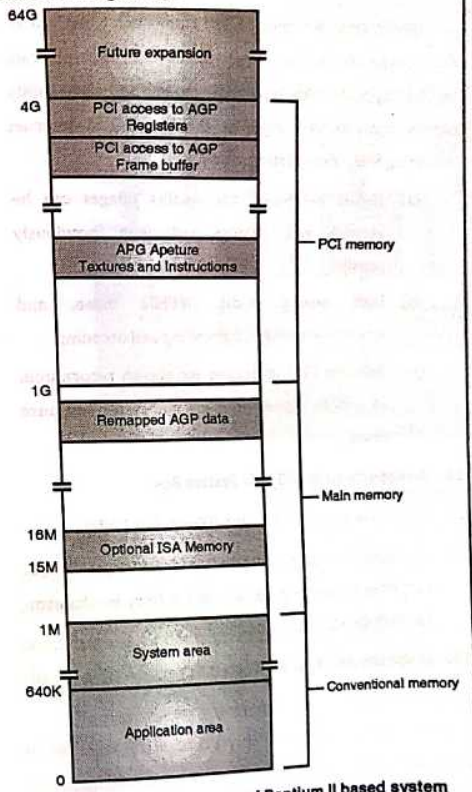


Fig. 14.3.1 : Memory map of Pentium II based system

14.3.2 Pentium II Software Changes

The only modifications in the instructions of Pentium II with respect to its predecessors are with the instructions CPUID, SYSENTER, SYSEXIT, FXSAVE and FXRSTOR. These are explained as below

1. **CPUID :** In Pentium II, CPUID have slight changes with respect to the feature information bit positions.
2. **SYSENTER and SYSEXIT :** These instructions are new to this Pentium architecture. They are accessible at highest privilege only i.e. PL=00. These instructions support fast call to a procedure. The difference in fast call with normal call is that here the stack is not used to store the context, instead the Model Specific Registers (MSR) are used to store the CS, ESP and EIP. SYSENTER instruction places these values in MSRs while the SYSEXIT instruction retrieves these values from the MSRs.
3. **FXSAVE and FXRSTOR :** FSAVE and FRSTOR instructions are used to save and restore the contents of the floating point coprocessor. The FXSAVE and FXRSTOR instructions are meant to do similar but are meant for storing the information properly for MMX machine.

14.4 Pentium 3

The special features of Pentium 3 processor are as listed below:

1. The Maximum Processor speed is 1 GHz.
2. The Maximum Bus speed is 133 MHz.
3. It has two Instruction fetch units.
4. It has three Decoders.
5. It has two Floating Point Units and two Integer Execution Units

6. It has a P6 Dynamic Execution Microarchitecture

   (a) **Multiple branch prediction**: predicts program execution through multiple branches, accelerating the flow of work to the processor.

   (b) **Dataflow analysis**: Creates an optimized, reordered schedule of instructions by analyzing data dependencies between instructions.

   (c) **Speculative execution**: carries out instruction execution speculatively and based upon this optimized schedule, ensures that the processor's superscalar execution units remain busy, boosting overall performance.

7. It has Dual Independent Bus (DIB)

8. It has Non-Blocking level 1 Cache

The Intel Pentium III Processor includes two separate 16-KB level 1 (L1) caches, one for instruction and one for data. The L1 cache provides fast access to the recently used data, increasing the overall performance of the system.

9. It also has 256-KB Level 2 Advanced Transfer Cache

The Advanced Transfer Cache (ATC) consists of microarchitectural improvements to provide a higher data bandwidth interface between the level 2 cache and the processor core that is completely scalable with the processor core frequency. Features of the ATC include:

Non-Blocking, full speed, on-die level 2 cache

8-way set associative.

256-bit data bus to the level 2 cache.

10. It has Advanced System Buffering

The advanced System Buffering consists of optimizations in the system bus buffer sizes and bus queue that result an increase in the utilization of the available bandwidth on the 100 and 133 MHz system bus.

4 write back buffers

6 fill buffers

8 bus queue entries.

11. Internet Streaming SIMD Extensions

The Internet Streaming SIMD Extensions consist of 70 instructions and includes single instruction, multiple data for floating-point, additional SIMD-Integer and cacheability control instructions. Some of the benefits of Internet Streaming SIMD Extensions include:

(a) Higher resolution and quality images can be viewed and manipulated than previously possible

(b) High quality audio, MPEG2 video, and simultaneous MPEG2 encoding and decoding.

(c) Reduced CPU utilization for speech recognition, as well as higher accuracy and faster response times.

12. It supports up to 133 MHz System Bus

13. It also has a pipelined Floating-Point Unit (FPU).

14. It supports parity-protected address/request and response system bus signals with a retry mechanism for high data integrity and reliability.

15. It also has a Built-in Self Test (BIST)

14.4.1 Internal Block Diagram of Pentium 2 and 3

The internal structure of Pentium 2 and Pentium 3 processors is shown in Fig. 14.4.1.

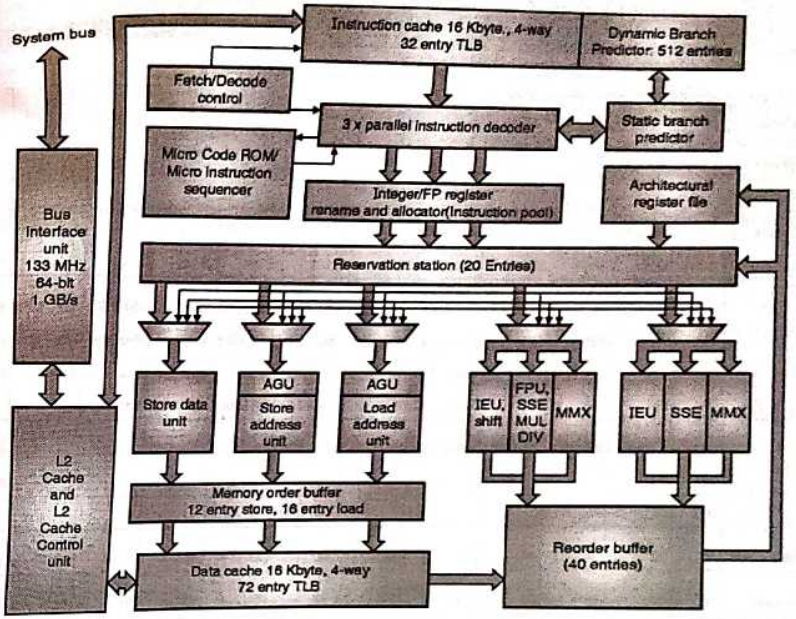


Fig. 14.4.1 : Internal block diagram of Pentium 2 and 3

The Bus Interface Unit (BIU) connects the unified L2 cache, external system bus and the internal blocks of the processor. The instructions are fetched by the prefetcher from the L1 instruction cache. These instructions are given to the decoders. There are three instruction decoders. Two of them are capable of decoding simple instructions while the third one is capable of decoding complex instructions. The instructions are decoded and placed in the instruction pool after checking the register alias table. Pentium 2 and 3 processors use register renaming to avoid contention.

The registers are assigned to an instruction by the reservation station. The reservation station along with Recorder Buffer takes the instructions in order from the out-of-order placed instructions in instruction pool. Pentium 2 and 3 have 5 execution units, namely, two Integer Execution units, Floating Point Unit, SIMD unit (MMX unit) and a Memory Management Unit (MMU). These units can access the registers and the data cache for the data. Once the instructions are executed the retirement unit writes back the result and the instruction is removed from the pool. The AGU (Address generation unit) for loading and storing the data are also available.

6. It has a P6 Dynamic Execution Microarchitecture

   (a) **Multiple branch prediction:** predicts program execution through multiple branches, accelerating the flow of work to the processor.

   (b) **Dataflow analysis:** Creates an optimized, reordered schedule of instructions by analyzing data dependencies between instructions.

   (c) **Speculative execution:** carries out instruction execution speculatively and based upon this optimized schedule, ensures that the processor's superscalar execution units remain busy, boosting overall performance.

7. It has Dual Independent Bus (DIB)

8. It has Non-Blocking level 1 Cache

The Intel Pentium III Processor includes two separate 16-KB level 1 (L1) caches, one for instruction and one for data. The L1 cache provides fast access to the recently used data, increasing the overall performance of the system.

9. It also has 256-KB Level 2 Advanced Transfer Cache

The Advanced Transfer Cache (ATC) consists of microarchitectural improvements to provide a higher data bandwidth interface between the level 2 cache and the processor core that is completely scalable with the processor core frequency. Features of the ATC include:

Non-Blocking, full speed, on-die level 2 cache

8-way set associative.

256-bit data bus to the level 2 cache.

10. It has Advanced System Buffering

The advanced System Buffering consists of optimizations in the system bus buffer sizes and bus queue that result an increase in the utilization of the available bandwidth on the 100 and 133 MHz system bus.

4 write back buffers

6 fill buffers

8 bus queue entries.

11. Internet Streaming SIMD Extensions

The Internet Streaming SIMD Extensions consist of 70 instructions and includes single instruction, multiple data for floating-point, additional SIMD-Integer and cacheability control instructions. Some of the benefits of Internet Streaming SIMD Extensions include:

   (a) Higher resolution and quality images can be viewed and manipulated than previously possible

   (b) High quality audio, MPEG2 video, and simultaneous MPEG2 encoding and decoding.

   (c) Reduced CPU utilization for speech recognition, as well as higher accuracy and faster response times.

12. It supports up to 133 MHz System Bus

13. It also has a pipelined Floating-Point Unit (FPU).

14. It supports parity-protected address/request and response system bus signals with a retry mechanism for high data integrity and reliability.

15. It also has a Built-in Self Test (BIST)

14.4.1 Internal Block Diagram of Pentium 2 and 3

The internal structure of Pentium 2 and Pentium 3 processors is shown in Fig. 14.4.1.

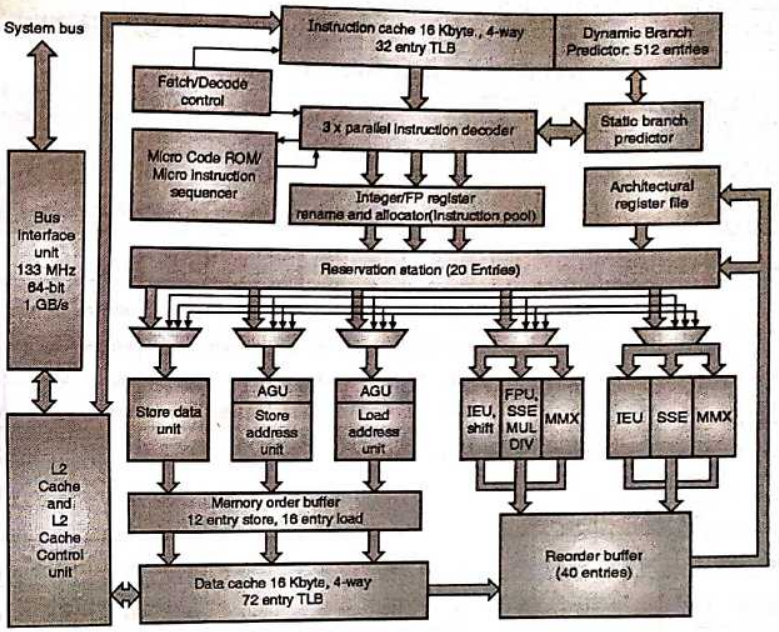


Fig. 14.4.1 : Internal block diagram of Pentium 2 and 3

The Bus Interface Unit (BIU) connects the unified L2 cache, external system bus and the internal blocks of the processor. The instructions are fetched by the prefetcher from the L1 instruction cache. These instructions are given to the decoders. There are three instruction decoders. Two of them are capable of decoding simple instructions while the third one is capable of decoding complex instructions. The instructions are decoded and placed in the instruction pool after checking the register alias table. Pentium 2 and 3 processors use register renaming to avoid contention.

The registers are assigned to an instruction by the reservation station. The reservation station along with Recorder Buffer takes the instructions in order from the out-of-order placed instructions in instruction pool. Pentium 2 and 3 have 5 execution units, namely, two integer Execution units, Floating Point Unit, SIMD unit (MMX unit) and a Memory Management Unit (MMU). These units can access the registers and the data cache for the data. Once the instructions are executed the retirement unit writes back the result and the instruction is removed from the pool. The AGU (Address generation unit) for loading and storing the data are also available.

14.5 Comparison Chart of Different Processors

Q. Differentiate between Pentium and Pentium pro-processor w.r.t generation, overclocking feature, core pipeline stages, number of transistors, address bits, main memory size, L2 cache, SMP support.

MU - May 12, Dec. 12, 10 Marks

Table 14.5.1 : Features of different INTEL processors

Sr. No.	Processors → Features ↓	8086	80386DX	80386SX	80486DX	Pentium	Pentium Pro	Pentium MMX	Pentium II	Pentium II based Celeron	Pentium III	Pentium II or III based XEON	Pentium 4
1	Processor Size	16 bits	32 bits	32 bits	32 bits	32 bits	32 bits	32 bits	32 bits	32 bits	32 bits	32 bits	32 bits
2	Speed	5 to 10 MHz	16 to 33 MHz	16 to 33 MHz	25 to 50 MHz	60 to 66 MHz (later upto 200 MHz)	60 MHz (later upto 200 MHz)	166 MHz (later upto 300 MHz)	233 to 300 MHz (later upto 450 MHz)	233 to 300 MHz (later upto 500 MHz)	450 to 500 MHz (later upto 1.4 GHz)	500 MHz (later upto 1 GHz)	400 MHz (later upto 2.26 GHz)
3	MIPS	0.33 to 0.75	5 -11.4	2.5 to 2.9	20 to 41	100 to 112	-	-	-	-	-	-	6500 to 10000
4	Address bus size	20	32	24	32	32	32	32	32	32	32	32	32
5	Data bus size	16	32	16	32	64	64	64	64	64	64	64	64
6	No. of transistors	29000	275000	275000	1.2	3.1 million	5.5 million	4.2 million	7.5 million	7.5 million	9.5 to 28	9.5 to 28	77 million
7	Addressable memory	1 MB	4 GB	16 MB	4 GB	4 GB	4 GB	4 GB	4 GB	4 GB	4 GB	4 GB	4 GB
8	Virtual memory	-	64 TB	32 GB	64 TB	64 TB	64 TB	64 TB	64 TB	64 TB	64 TB	64 TB	64 TB
9	L1 Cache	-	-	-	8KB Unified	16KB Split	16KB Split	32KB Split	32KB Split	32KB Split	32KB Split	32KB Split	12 KμCpcodes + 8KB data
10	L2 Cache	-	-	-	-	-	256KB to 1MB Unified	-	512 KB	-	256 KB ATC (or 512 KB)	256 KB ATC (or 1MB or 2MB)	1 MB ATC

Sr. No.	Processors → Features ↓	8086	80386DX	80386SX	80486DX	Pentium	Pentium Pro	Pentium MMX	Pentium II	Pentium II based Celeron	Pentium III	Pentium II or III based XEON	Pentium 4
11	L3 Cache	-	-	-	-	-	-	-	-	--	-	-	2 MB
12	On chip FPU	-	-	-	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
13	Superscalar	-	-	-	-	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
14	MMX Instruction set	-	-	-	-	-	-	SSE1	SSE1	SSE1	SSE2	SSE2	SSE2
15	Hyper threading support	-	No	No	No	No	No	No	No	No	No	No	Yes
16	No. of cores	1	1	1	1	1	1	1	1	1	1	1	1
17	Fabrication process	3 μm	1 μm	1 μm	1 μm	0.8 μm	0.35 μm	0.35 μm	0.35 μm	0.35 μm	0.18 μm	0.25 μm	0.13 μm
18	Generation	P1	P3	P3	P4	P5	P6	P6	P6	P6	P6	P6	NetBurst
19	SMP (multiprocessor) support	No	No	No	No	No	No	No	No	No	No	No	Yes
20	Integer pipeline stages	2	3	3	3	5	14	5	14	14	14	14	20
21	No. of Integer pipelines	1	1	1	1	2	2	2	2	2	2	2	4
22	Floating point pipeline stages	-	-	-	-	8	-	8	8	8	11	11	20
23	No. of floating point pipeline stages	-	-	-	1	1	1	1	1	1	1	1	2
24	Overclocking feature	No	No	No	No	No	No	No	No	No	No	No	Yes

14.6 Pentium-4 NetBurst Micro-Architecture

The Intel NetBurst micro-architecture is the foundation for the Intel Pentium 4 processor. Intel NetBurst micro architecture delivers a number of innovative features viz. Hyper-Thread Technology, hyper-pipelined technology, 800/533/400 MHz system bus, Execution Trace Cache, Advanced Transfer Cache, Advanced Dynamic Execute, enhanced floating-point and multimedia unit and Streaming SIMD Extensions 3 (SSE3).

14.6.1 Features of Pentium-4 processor

Q. Write the features of Pentium IV (10 Marks)
Q. List the features of Pentium 4 processor. (12 Marks)

1. Hyper-Threading Technology

- It allows software programs "see" two processors and hence work more efficiently. For this the processor has to provide two sets of certain features like registers.
- This new technology improves performance and system responsiveness. The Pentium 4 processor supports HT Technology and is designed specially to deliver immediate increases in performance and system responsiveness with existing applications in multitasking environments (that is, where two or more functions are running at the same time) and with many stand alone applications today.

2. Hyper-Pipelined Technology

- The hyper-pipelined technology of the NetBurst micro-architecture doubles the pipeline depth, compared to the P6 micro-architecture used in Pentium 2 and 3. One of the key features of this 20 stage pipeline, used in NetBurst microarchitecture, is the branch prediction/recovery pipeline.
- When compared to the equivalent pipeline in the P6 micro-architecture, which was implemented with a 10 stage pipeline, this technology significantly increases processor performance and frequency scalability of the base micro-architecture.

3. 800 MHz, 533 MHz or 400 MHz System Bus

- In the Pentium 4 processor with 800 MHz system bus, the bus supports high performance by delivering to 6.4 GB of data-per-second into and out of the processor.

4. Level 1 Execution Trace Cache

- The Pentium 4 processor includes an Execution stores up to 12K decoded micro-ops in the order of program execution. The Execution Trace Cache is an innovative way to implement a Level 1 instruction cache.
- It caches the decoded x86 instructions (i.e. micro-ops), thus removing the latency associated with the instruction decoder from the main execution loops and makes more efficient usage of the cache space.
- The Execution Trace Cache stores these micro-ops in the path of program execution flow, where the results of branches in the code are integrated into the same cache line.
- This increases the instruction flow from the cache and makes better use of the overall cache storage space (12K micro-ops) since the cache no longer stores instructions that are branched over and never executed.
- Hence it delivers a high volume of instructions to the processor's execution units and reduces the overall time required to recover from branches that have been miss-predicted.

5. 1 MB or 512 KB Level 2 Advanced Transfer Cache (ATC)

- The Level 2 ATC declares much higher data throughput channel between the Level 2 cache and the processor core.
- The Advanced Transfer Cache consists of a 256-bit (i.e. 32-byte) interface that transfers data on each clock. As a result, a 1.4-GHz Pentium 4 processor can deliver a data transfer rate of 44.8GB/s (i.e. 32 bytes × 1.4 GHz = 44.8GB/s). This compares to a transfer rate of 16GB/s on the Pentium III processor at 1 GHz and contributes to the Pentium 4 processor's ability to keep the high-frequency execution units executing instructions vs. sitting idle.

  Features of the ATC include :

  Non-Blocking, full speed, on-die level 2 cache

  8-way set associative

  256-bit data bus to the level 2 cache

  Data clocked into and out of the cache every clock cycle.

6. 2 MB L3 cache

- The 2-MB L3 cache is available only in the Pentium 4 processor Extreme Edition at 3.4 GHz.
- This additional third level of cache is located on the processor die and is designed specifically to meet the compute needs of high-end gamers and other power users.
- The integrated L3 cache is available in 2 MB and is connected with the 800 MHz system bus to provide a high bandwidth path to memory.
- This results in reduced average memory latency and increased throughput for larger workloads.

7. Advanced Dynamic Execution

- The Advance Dynamic Execution engine is very deep, out-of-order speculative execution engine and keeps the execution units executing instructions.
- It also includes an enhanced branch prediction algorithm that has the net effect of reducing the number of branch miss-predictions.

8. Minimizing the Penalty Associated with Branch Miss-predicts

- The NetBurst micro-architecture takes advantage of out-of-order, speculative execution, where the processor routinely uses an internal branch prediction algorithm to predict the result of branches in the program code and then speculatively executes instructions along the predicted code branch.
- Although branch prediction algorithms are highly accurate, they are not 100% perfect.
- If the processor miss-predicts a branch, all the instructions in the pipeline have to be flushed, and the execution will have to restart from the instruction along the correct program branch. On 20-stage pipeline of Pentium 4 more instructions have to be flushed from the pipeline, resulting in a longer recovery time from a branch miss-predict.
- The net result is that applications that have many, difficult to predict, branches will tend to have a lower average IPC (Instructions Per Clock). Hence, to minimize the branch miss-prediction penalty and maximize the average IPC, the deeply pipelined NetBurst micro-architecture greatly reduces the number of branch miss-predicts and provides a quick method of recovering from any branches that have been miss-predicted.
- To minimize this penalty, the NetBurst micro-architecture has implemented an Advanced Dynamic Execution engine and an Execution Trace Cache.

9. Enhanced Floating-Point and Multimedia Unit

The Pentium 4 processor expands the floating-point registers to a full 128-bit and adds an additional register for data movement which helps improve performance on both floating-point and multimedia applications.

10. Streaming SIMD Extensions 3 (SSE3) Instructions

The 13 new instructions in SSE3 are primarily designed to import thread synchronization and specific application areas such as media and gaming.

11. Built-Self Test (BIST)

It has a built in self test to test if all the attributes of the processor are working properly. This test is performed during the POST (Power On Self Test).

12. Rapid Execution Engine

- With a combination of architectural, physical and circuit designs, the simple Arithmetic Logic Units (ALUs) within the processor run at two times the frequency of the processor core.
- This allows the ALUs to execute certain instructions with a latency that is half the duration of the core clock and hence result in higher execution throughput as well as reduced latency of execution.

14.6.2 Pentium 4 Architecture

Q. Explain Intel's Net Burst Micro architecture with neat schematic. Also highlight on hyper-pipeline concept and rapid execution engine. (5 Marks)
Q. Write short note on Intel's Net burst micro-architecture (5 Marks)

The basic blocks of the Pentium 4 processor are shown in Fig. 14.6.1.

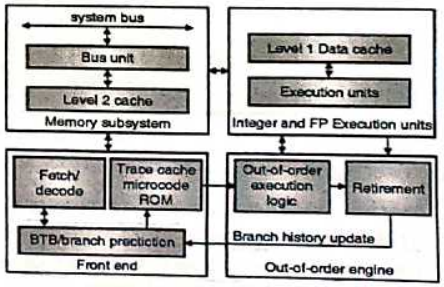


Fig. 14.6.1 : Basic Blocks of Pentium 4 processor

The basic blocks of Pentium 4, as shown in Fig. 14.6.1, consists of Memory subsystem, Front end, Out-of-order Engine and the Integer and Floating point execution units.

1. Memory Subsystem

- This unit handles the memory accesses and also consists of the L2 cache.
- The L2 cache is an Advanced Trace Cache as discussed in the features of the processor.

2. Front End

- This block consists of a Fetch / Decode unit Trace cache along with microcode ROM and BTB along with branch prediction logic.
- The fetch decode unit fetches the instructions and decodes them.
- The trace cache, as discussed already, stores the decoded instructions and hence reduces the time for decoding in loops.
- The branch prediction logic predicts the sequence of instructions during a branch instruction with the help of branch history stored in Branch Target Buffer (BTB).

3. Out-of-order Engine

- The out-of-order engine executes the instructions out of order in case of their dependencies and hence keeps the execution units continuously busy.
- But the retirement of the instructions are in order.

4. Execution units

- There are integer execution units to execute integer instructions, floating point unit to floating point instructions and MMX unit to execute SIMD vector instructions. The detailed block diagram of Pentium 4 is shown in Fig. 14.6.2.

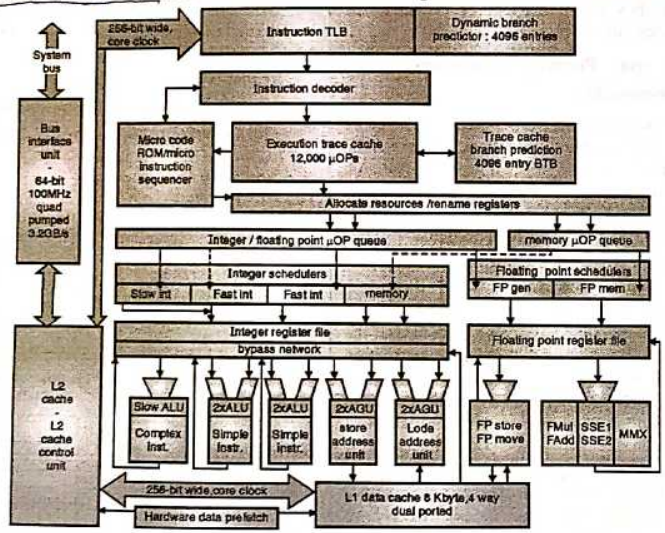


Fig. 14.6.2 : Detailed Block Diagram of Pentium 4

(a) AdvancedL2Cache

- The L2-cache also called as 'Advanced Transfer Cache' is 256 KB (or 512KB or 1 MB) and is 8-way associative. Pentium 4's L2-cache uses 128 byte cache lines, which are divided in two 64-byte pieces.
- The L1 data cache was reduced down to only 8 KB, to enable its extremely low latency of only 2 clock cycles and hence results in an overall improved read latency.

- The L1 data cache of Pentium 4 is four-way set associative and has 64 byte cache-lines. The dual-port architecture allows one load and one store operation simultaneously per clock.

(b) System Bus

- Pentium 4's system bus is clocked at 100 MHz and also 64 bit wide, but it is 'quad-pumped', using the principle as AGP4x.
- Quad-pumped is a technique wherein 4 accesses are performed in a single clock pulse. Thus it can transfer 8 byte * 100 million /s * 4 = 3,200 MB/s.

(c) Instruction Decoder and Trace Cache

- The Pentium 4 processor includes an Execution stores up to 12K decoded micro-ops in the order of program execution.
- The Execution Trace Cache is an innovative way to implement a Level 1 instruction cache.
- It caches the decoded x86 instructions (i.e. micro-ops), thus removing the latency associated with the instruction decoder from the main execution loops and makes more efficient usage of the cache space.
- The Execution Trace Cache stores these micro-ops in the path of program execution flow, where the results of branches in the code are integrated into the same cache line.
- This increases the instruction flow from the cache and makes better use of the overall cache storage space (12K micro-ops) since the cache no longer stores instructions that are branched over and never executed.
- Hence it delivers a high volume of instructions to the processor's execution units and reduces the overall time required to recover from branches that have been miss-predicted.

(d) Rapid Execution Engine

Fig. 14.6.3 : Rapid Execution Engine

The Fig. 14.6.3 shows all execution units of Pentium 4, including the "Rapid Execution Engine' as well as the 'not-so-rapid' execution units. While there four fast execution units, the other four are the actual units that are responsible for Pentium 4's peculiar behavior. The basic parts of 'Rapid Execution Engine' are the two 'double-pumped' ALUs (Arithmetic and Logic Units) and AGUs (Address Generation Units). Each of the four is clocked with double the processors clock, because they can receive microOPs every half clock. Simple microOPs that can be processed by the Rapid Execution Engine are executed in a half clock, which increases the performance of the processor. The instructions that cannot be processed by the rapid execution units or microOps need to use the one and only 'Slow ALU', which is not 'double pumped' the operation becomes slower. The specialties of the SIMD unit are described by SSE2 – The New Double Precision Streaming SIMD Extensions, which consists of

- - 144 new instructions
- - 4 single precision FP values (SSE)
- - 2 double precision FP values (SSE2)
- - 16 byte values (SSE2)
- - 8 word values (SSE2)
- - 4 double word values (SSE2)
- - 2 quad word values (SSE2)
- - 1, 128-bit integer value (SSE2)

14.6.3 Pipelining in Pentium 4

The 20-stage pipelining of Pentium 4 is shown in Fig. 14.6.4

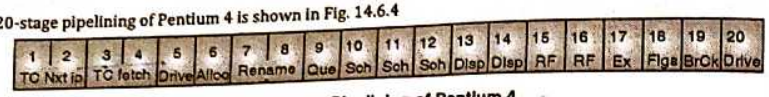


Fig. 14.6.4 : Pipelining of Pentium 4

One of the most well known features of the Pentium 4 is its extremely long pipeline. While the pipeline of Pentium III has 10 stages and the one of Athlon 11, Pentium 4 has 20 stages. The reason for the longer pipeline of Pentium 4 is to deliver highest clock rates. The smaller or shorter each pipeline stage, the fewer transistors or 'gates' it needs and hence the faster it is able to run. However, there is one big disadvantage to long pipelines and that is as soon as it turns out at the end of the pipeline that the software will branch to an address that was not predicted, the whole pipeline needs to be flushed. The longer the pipeline the more instructions will be lost and the longer it takes until the pipeline is filled again. The improved trace cache branch prediction unit described above ensures that flushes of this long pipeline are only rare occasions.

Here's a breakdown of the various stages:

- **Stages 1 and 2 - Trace Cache next Instruction Pointer :** In these stages, the Pentium 4's trace cache fetch logic gets a pointer to the next instruction in the trace cache.
- **Stages 3 and 4 - Trace Cache Fetch :** These two stages fetch an instruction from the trace cache to be sent to the Out of order (OOO) execution engine.
- **Stage 5 - Drive :** This is the first of two of Drive stages in the P4's pipeline, each of which is dedicated to driving signals from one part of the processor to the other. The P4 runs so fast that sometimes a signal can't make it all the way to where it needs to be in a single clock pulse, so the P4 dedicates some pipeline stages to let these signals propagate across the chip. It is definitely there because the P4's designers intend for it to reach such high clock speeds that stages like this are absolutely necessary.
- **Stages 6 through 8 - Allocate and Rename :** This group of stages handles the allocation of microarchitectural register resources to the instructions.
- Register renaming is a trick for improving register conflicts by having more registers in the microarchitecture than are specified in the instruction set architecture. These extra microarchitectural registers (the Pentium 4 has 128 of them) are allocated and put into use in these steps. The Allocator/Renamer can issue upto three µops per cycle to the next pipeline stage.
- **Stage 9 - Queue :** There are two queues that reside between the Allocator/Renamer and the scheduling logic, a memory µop queue and an arithmetic µop queue. These queues are where µops wait before being sent to one of the four "dispatch ports" that act as gateways to the execution engine's functional units.
- **Stages 10 through 12 - Schedule :** In these stages, instructions pass from the Allocator to one of four scheduling queues. These queues are where operations for each individual functional unit (or group of functional units) are scheduled to go onstage and be executed. The µop schedulers determine when a µop is ready to execute by tracking its input register operands. This is the heart of the out-of-order execution engine. The µop schedulers are what allow the instructions to be reordered to execute as soon as they are ready, while still maintaining the correct dependencies from the original program. The NetBurst microarchitecture has two sets of structures to aid in µop scheduling: the µop queues and the actual µop schedulers. Here's a breakdown of the four schedulers:
- **Memory Scheduler –** This scheduler schedules memory operations for the Load/Store Units (LSU) or Address Generation Units (AGU).

- **Fast ALU Scheduler –** This scheduler schedules Arithmetic-Logic Unit operations (simple integer and logical ops) for the Pentium 4's two double-pumped ALU units.
- **Slow ALU/General FPU Scheduler –** This scheduler schedules the rest of the ALU functions and most of the floating-point functions.
- **Simple FP Scheduler –** This scheduler schedules simple FP operations and FP memory operations.
- **Stages 13 and 14 - Dispatch :** In these two stages instructions travel through one of the four dispatch ports for actual execution. These ports act as a sort of gateways to the actual execution units. Up to 6 µops total per cycle can travel from the schedulers to the functional units through the dispatch ports. This is more µops per cycle than the front end can execute (3 per cycle) or the back end can retire (3 per cycle), but that's ok because it gives the machine some headroom in its middle so that it can have bursts (many) of activities.
- **Stages 15 and 16 - Register Files :** After traveling through the dispatch ports in the previous two stages, the instructions spend these two stages being loaded into the register files for execution.
- **Stage 17 - Execute :** In this stage, the instructions are actually executed by the execution engine's corresponding functional units.
- **Stage 18 - Flags :** If the instruction's result requires to set any flags, then it does so at this stage.
- **Stage 19 - Branch Check :** Here's where the Pentium 4 checks the result of a conditional branch to see if it has just wasted 19 cycles of its time executing some code that it'll have to throw away. By Stage 19, the condition has been evaluated and then the front end knows whether or not the branch predictor's guess was right or not.
- **Stage 20 - Drive:** We've already seen a the Drive stage. Again, this stage is dedicated to propagating signals across the chip.

14.7 Exam Pack (Review and University Questions)

Q. 1 Differentiate between Pentium and Pentium pro-processor w.r.t generation, overclocking feature, core pipeline stages, number of transistors, address bits, main memory size, L2 cache, SMP support. **(Section 14.5)** **(10 Marks)**

Q. 2 Write the features of Pentium IV **(Refer Section 14.6.1)** **(10 Marks)**

Q. 3 List the features of Pentium 4 processor. Explain Intel's Net Burst Micro architecture with neat schematic. Also highlight on hyper-pipeline concept and rapid execution engine. **(Refer Section 14.6.1 and 14.6.2)** **(12 Marks)**

Q. 4 Write short note on Intel's Net burst microarchitecture **(Refer Section 14.6.2)** **(5 Marks)**

□□□

Note

Lab Manual

Lab Manual

Experiment 1 :

Use of programming tools (Debug/TASM/MASM/8086kit) to perform basic arithmetic operations on 8-bit/16-bit data.

Program 1 : Add two 8 bit numbers

Ans. : Please refer Program 5.4.1 of Chapter 5 (Page No. 5-4)

Program 2 : Subtract two 8 bit numbers.

Ans. : Please refer Program 5.4.2 of Chapter 5 (Page No. 5-6)

Program 3 : To multiply two 8 bit numbers.

Ans. : Please refer Program 5.4.3 of Chapter 5 (Page No. 5-8)

Program 4 : Divide 16 bit number by an 8 bit number.

Ans. :Please refer Program 5.4.6 of Chapter 5 (Page No. 5-12)

Experiment 2 :

Code conversion (Hex to BCD and BCD to Hex)/ (ASCII to BCD and BCD to ASCII).

Ans. :

(A) BCD to HEX

Program statement

Write 8086 ALP to convert 5 digit BCD number to its equivalent Hex number.

Program

Label	Instruction	Comment
	.model small	
	.data	
	a dW 0506 H	Unpacked BCD data
	b dB ?	Variable to store result
	.code	
Start :	mov AX, @data	
	mov DS, AX	Initialize data segment
	mov AX, a	
	AAD	Convert unpacked BCD to Hex
	mov b, AL	Store result
	mov AH, 4CH	
	INT 21H	Terminate program execution
	end	

(B) Hex to BCD

Program

Label	Instruction	Comment
	.model small	
	.code	
	a dB 45 H	hex data
	b dW ?	Variable to store result
	.code	
Start :	mov AX, @ data	
	mov DS, AX	Initialize data segment
	mov AL, a	
	mov AH, 01 H	
	MUL AH	
	AAM	Convert Hex to unpacked BCD
	mov b, Ax	Store result
	mov AH, 4C H	
	int 21 H	Terminate program execution
	end	

Experiment 3 :

Assembly programming for 16-bit addition, subtraction, multiplication and division (menu based)

Program 1 : Add two 16 bit numbers.

Ans. : Please refer Program 5.4.7 of Chapter 5 (Page No. 5-14)

Program 2 : Subtract two 16 bit numbers.

Ans. : Please refer Program 5.4.8 of Chapter 5 (Page No. 5-15)

Program 3 : Multiply two 16 bit number.

Ans. :

Program statement

- Assuming that two words are available in registers AX and BX, write a program in the assembly language of 8086 to multiply two 16 bit numbers.

Explanation

- Consider that a word of data is present in the AX register and 2nd word of data is present in the BX register.
- We have to multiply the word in AX with the word in BX. Using MUL instruction, multiply the contents of the 2 registers.
- The multiplication of the two 16 bit numbers may result into a 32 bit number. So result is stored in the DX and AX register.
- The MSB of result is stored in the DX register and LSB of result in AX register.

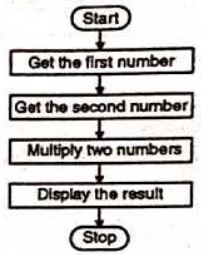
```
AX = 1234 H        1234 H
BX = 0100 H      × 0100 H
                 ---------
                 123400 H
```


Algorithm

Step I : Initialize the data segment.
Step II : Get the first number in AX register.
Step III : Get the second number in BX register.
Step IV : Multiply the two 16 bit numbers.
Step V : Display the result.
Step VI : Stop
Flowchart : Refer Flowchart 1.

Program

Label	Instruction	Comment
	.model small	
	.data	
	a dw 1234 h	
	b dw 0100 h	
	.code	
	mov ax, @data	Initialize data section
	mov ds, ax	
	mov ax, a	Load number 1 in ax
	mov bx, b	Load number 2 in bx
	mul bx	multiply numbers. Result in dx and ax
	mov si, ax	
	mov bx, dx	Result in reg bx
	mov dh, 2	
11:	mov ch, 04 h	Count of digits to be displayed
	mov cl, 04 h	Count to roll by 4 bits
12:	rol bx, cl	rotate left bl so that msb comes to lsb
	mov dl, bl	load dl with data to be displayed
	and dl, 0f h	get only lsb
	cmp dl, 09	check if digit is 0-9 or letter A-F
	jbe 14	
	add dl, 07	if letter add 37 h else only add 30 h
14:	add dl, 30 h	
	mov ah, 02	Function 2 under INT 21 h (Display character)



Flowchart 1

Label	Instruction	Comment
	int 21H	
	dec ch	Decrement Count
	jnz l2	
	dec dh	
	cmp dh, 0	
	mov bx, si	
	jnz l1	
	mov ah, 4cH	Terminate Program
	int 21H	
	end	

Result

```
C:\programs>tasm 16-mul.asm
Turbo Assembler  Version 3.0  Copyright (c) 1988, 1991 Borland International
Assembling file:   16-mul.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  438k
C:\programs>tlink 16-mul
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
Warning: No stack
C:\programs>16-mul
00123400
```

Experiment 4 :

Assembly program based on string instructions (overlapping/non-overlapping block transfer/ string search/ string length)

Program 1 : Transfer a block of N bytes from source to destination (Non overlapped block transfer)

Ans. : Please refer Program 5.4.13 of Chapter 5 (Page No. 5-19)

Program 2 : Write a program to check if the given string is palindrome.

Ans. : Please refer Program 5.4.15 of Chapter 5 (Page No. 5-23)

Experiment 5 :

Assembly program to display the contents of the flag register.

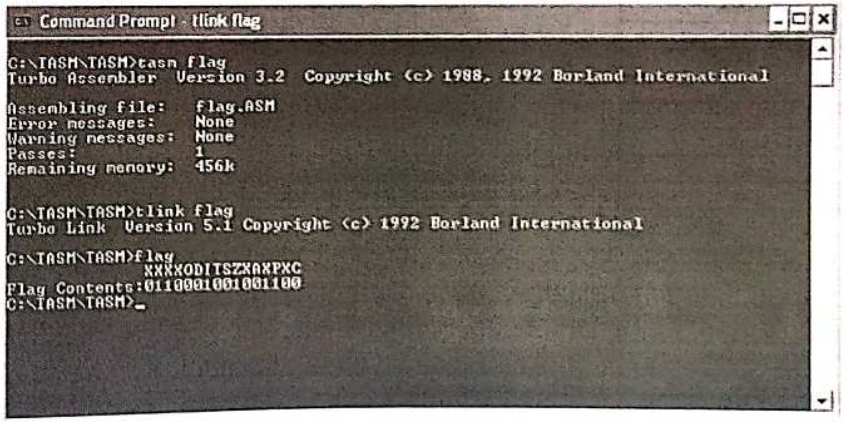
Ans. :

Program

Label	Instruction	Comments
	.model small	
	.data	
	a dw 2021	
	b dw 04H	
	msg db " XXXXODITSZXAXPXC",13,10,'$'	Message to be displayed above the flag bit values as the sequence of flag bit names

Label	Instruction	Comments
	msg1 db "Flag Contents:$"	
	.stack	
	.code	
Start :	mov ax, @data	Initialize data segment
	mov ds, ax	
	xor al,al	Perform some logical operation to update the flag bits
	lea dx,msg	Display the message of the flag bit format using INT 21H type 09H
	mov ah,09h	
	int 21h	
	lea dx,msg1	Display another message using INT 21H type 09H
	mov ah,09h	
	int 21h	
	pushf	Get the flag contents into register BX using stack
	pop bx	
	mov cl,15	Initialize counter to read the 16 bits of flag register
Again :	rcr bx,1	Rotate the value of register bx by 1, to shift the first bit in carry flag
	mov dl,30H	Convert the bit '0' or '1' to corresponding ASCII values 30H or 31H
	jnc over	
	mov dl,31H	
Over :	MOV ah,02h	Display the character using INT 21H type 02H
	int 21h	
	loop agn	Repeat for all 16 bits
	mov ah, 4cH	Terminate the program
	int 21H	
	end	

Output



Experiment 6 :

Any Mixed Language programs.

Ans. : Please refer Case 1 of Chapter 6 (Page No. 6-11)

Experiment 7 :

Assembly program to find the GCD/ LCM of two numbers

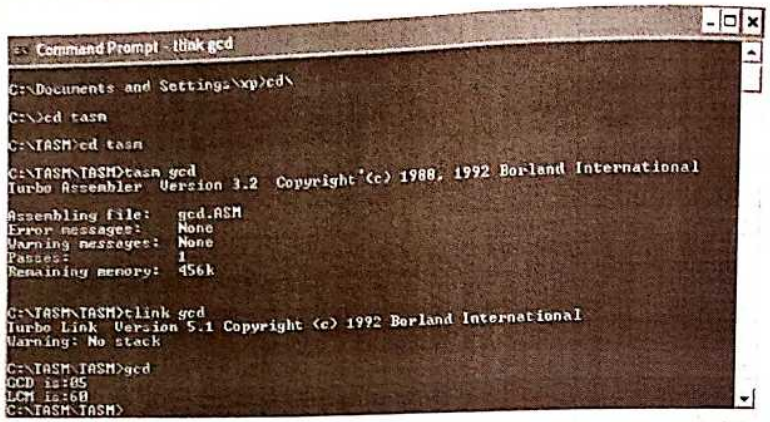
Ans. :

Program

Label	Instruction	Comments
	model small	
	.data	
	a db 20	
	b db 15	
	msg db "GCD is:$"	Initialize the message to display "GCD"
	msg1 db 10,13,"LCM is:$"	Initialize the message to display "LCM" along with a new line
	.code	
Start :	mov ax, @data	Initialize data segment
	mov ds, ax	
	mov dl,a	Load the two numbers in registers bl and dl also load the lower number into cl
	mov bl,b	
	mov cl,bl	
	cmp dl,bl	
	jc agn	
	mov cl,dl	
Again :	mov ah,00h	Check if the number in cl is divisible by both the given numbers
	mov al,bl	
	div cl	
	or ah,00h	
	jnz over	
	mov ah,00h	
	mov al,dl	
	div cl	
	or ah,00h	
	jz done	
Over :	loop agn	If not divisible decrement cl and check again
Done :	lea dx,msg	If cl is divisible by both, then cl is the gcd. Display the message using INT 21H type 09H
	mov ah,09h	
	int 21h	
	mov al,cl	Display the two digits of GCD by separating them. To separate, divide by the decimal number 10 or $(0A)_H$ and separate the quotient and remainder.
	mov ah,00h	
	mov bl,0ah	
	div bl	Display the quotient and remainder after converting them to ASCII form. To convert in ASCII form, add with $(30)_H$.
	mov dl,al	
	mov cl,ah	To display, use INT 21H and type 02H
	add dl,30h	

Label	Instruction	Comments
	MOV ah,02h	
	int 21h	
	mov al,cl	
	add al,30h	
	mov dl,al	
	MOV ah,02h	
	int 21h	
	mov dl,a	Load the two numbers in registers bl and dl also load the higher number into cl
	mov bl,b	
	mov cl,bl	
	cmp bl,dl	
	jc agn1	
	mov cl,dl	
Again :	mov ah,00h	Check if the number in cl is multiple of both the given numbers
	mov al,cl	
	div bl	
	or ah,00h	
	jnz over1	
	mov ah,00h	
	mov al,cl	
	div dl	
	or ah,00h	
	jz done1	
Over :	inc cl	
	jmp agn1	
Done :	lea dx,msg1	If cl is multiple of both, then cl is the lcm. Display the message using INT 21H type 09H
	mov ah,09h	
	int 21h	
	mov al,cl	Display the two digits of LCM by separating them. To separate, divide by the decimal number 10 or $(0A)_H$ and separate the quotient and remainder.
	mov ah,00h	
	mov bl,0ah	
	div bl	Display the quotient and remainder after converting them to ASCII form. To convert in ASCII form, add with $(30)_H$.
	mov dl,al	
	mov cl,ah	To display, use INT 21H and type 02H
	add dl,30h	
	MOV ah,02h	
	int 21h	
	mov al,cl	
	add al,30h	
	mov dl,al	
	MOV ah,02h	
	int 21h	
	mov ah, 4cH	Terminate the program
	int 21H	
	end	

Output



Experiment 8 :

Assembly program to sort numbers in ascending/descending order

Program 1 : Program to sort the numbers in ascending order.

Ans. : Please refer Program 5.4.11 of Chapter 5 (Page No. 5-18)

Program 2 : Sorting the numbers in descending order using 8086.

Ans. : Please refer Program 5.4.12 of Chapter 5 (Page No. 5-19)

Experiment 9 :

Any program using INT 10H

Ans. :

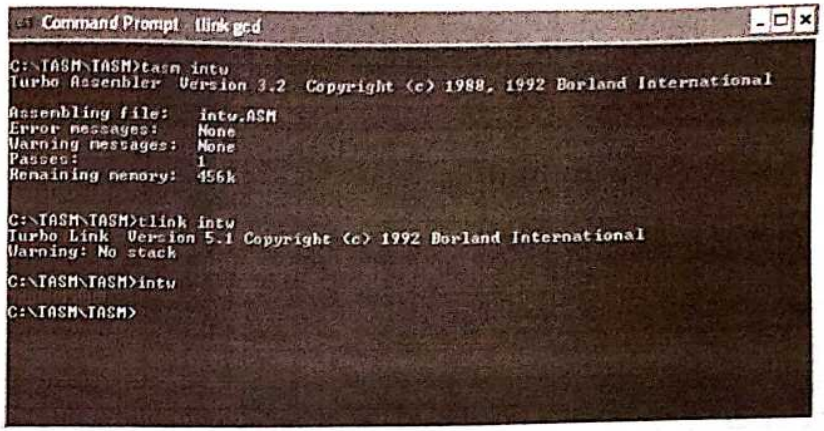
Let us write a program to display a character using Int 10H. The following are the information to be given for the same

Write character and attribute at cursor position	AH=09h	AL = Character, BH = Page Number, BL = Colour, CX = Number of times to print character

Program

Label	Instruction	Comments
	model small	
	.data	
	.code	
Start :	mov ax, @data	Initialize the data segment
	mov ds, ax	
	mov al,'H'	Initialize the character to be displayed in register al
	mov bh,00h	Select page 1 for display
	mov bl,04h	Select color "Red" in register bl
	mov cx,10	Initialise the counter 10H in register cx
	MOV ah,09h	Call interrupt 10H type 09H
	int 10h	
	mov ah, 4cH	Terminate the program
	int 21H	
	end	

Output



Experiment 10 :

Assembly program to find minimum/ maximum number from a given array

Program 1 : Program to find maximum number in the array.

Ans. :

Label	Instruction	Comments
	.model small	
	.data	
	array dB, 62H, 05H, 42H, 12H, 01H, 15H, 07H, 14H, 09H, 11H	Initialize array
	count dW OAH	Initialize count
	max db ?	Variable for resutl
	.code	
Start :	mov AX, @ data	
	mov DS, AX	Initialize data segment
	LEA SI, array	Initialize pointer
	mov CX, count	Initialize counter
	DEC CX	Number of comparisons are one less than count
	mov AL, [SI]	Load first data
Again :	INC SI	Increment pointer
	CMP AL, [SI[	Compare min number with next number
	JNC over	If no carry, jump to over
	MOV AL, [SI]	If carry, copy new number in AL
Over :	Loop again	
	MOV max, AL	Store result
	MOV AH, 4CH	
	INT 21 H	Terminate program execute
	End	

Program 2 : Program to find smallest number.

Ans. :

Program

Label	Instruction	Comments
	.model small	
	.data	
	array dB, 62H, 05H, 42H, 12H, 01H, 15H, 07H, 14H, 09H, 11H	Initialize array
	count dW OAH	Initialize count
	min db ?	Variable for resutl
	.code	
Start :	mov AX, @ data	
	mov DS, AX	Initialize data segment
	LEA SI, array	Initialize pointer
	mov CX, count	Initialize counter
	DEC CX	Number of comparisons are one less than count
	mov AL, [SI]	Load first data
Again :	INC SI	Increment pointer
	CMP AL, [SI[	Compare min number with next number
	JC over	If carry, jump to over
	MOV AL, [SI]	If no carry, copy new number in AL
Over :	Loop again	
	MOV min, AL	Store result
	MOV AH, 4CH	
	INT 21 H	Terminate program execute
	End	

Experiment 11 :

Assembly Program to display a message in different color with blinking.

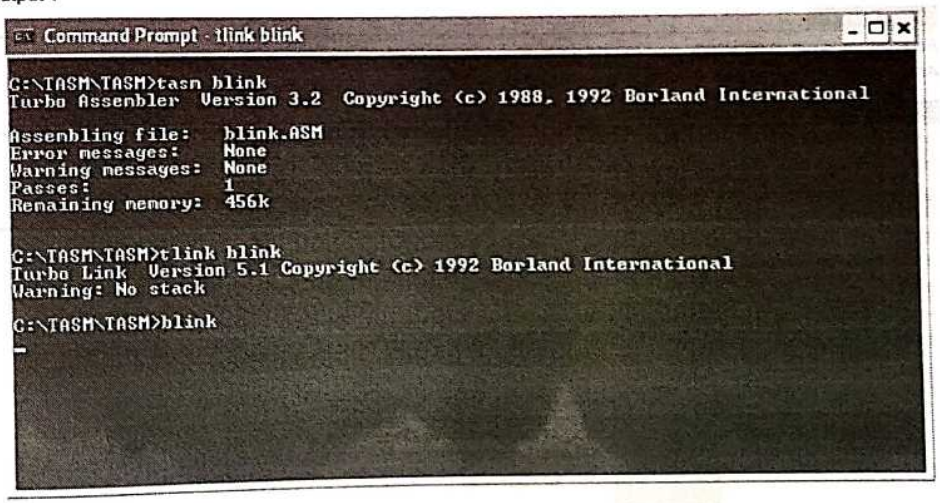
Ans. :

Program

Label	Instruction	Comments
	model small	
	.data	
	.code	
Start :	mov ax, @data	Initialize data segment
	mov ds, ax	
	mov dx,10	Initialize counter to blink for 10 times
	repeat: mov al,'H'	Initialize register al to display the character 'H'
	mov bh,00h	Initialize the page to be '0'
	mov bl,00h	Initialize the color to be black (00)
	mov cx,10	Initialise the counter to display for 10 times
	MOV ah,09h	Display using INT 21H type 09H
	int 10h	
	mov cx,05ffh	Generate a random delay before changing the display characters color to white.

Label	Instruction	Comments
Again 1	mov bx,0ffffh	
Again :	dec bx	
	jnz agn	
	loop agn1	
	mov al,'H'	Initialize register al to display the character 'H'
	mov bh,00h	Initialize the page to be '0'
	mov bl,0fh	Initialize the color to be black (00)
	mov cx,10	Initialise the counter to display for 10 times
	MOV ah,09h	Display using INT 21H type 09H
	int 10h	
	mov cx,05ffh	Generate a random delay before changing the display characters color to white.
Again 3 :	mov bx,0ffffh	
Again 2 :	dec bx	
	jnz agn2	
	loop agn3	
	dec dx	Repeat for dx times
	jnz repeat	
	mov ah, 4cH	Terminate the program
	int 21H	
	end	

Output 1



Output 2

```
Command Prompt - tlink blink

C:\TASM\TASM>tasm blink
Turbo Assembler  Version 3.2  Copyright (c) 1988, 1992 Borland International

Assembling file:   blink.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  456k

C:\TASM\TASM>tlink blink
Turbo Link  Version 5.1 Copyright (c) 1992 Borland International
Warning: No stack

C:\TASM\TASM>blink
HHHHHHHHHH
```

Experiment 12 :

Assembly program using proced ure.

Ans. : Program 6.2.1 : Addition of Two Arrays using Procedure.

Experiment 13 :

Assembly program using macro.

Ans. : Please refer Program 6.3.3 of Chapter 6 (Page No. 6-8)

Experiment 14 :

Program and interfacing using 8255.

Ans. : Please refer Program 9.8.2 of Chapter 9 (Page No. 9-13)

Experiment 15 :

Program and interfacing of ADC/ DAC/ Stepper motor.

Program 1 : Interfacing ADC with 8086.

Ans. :

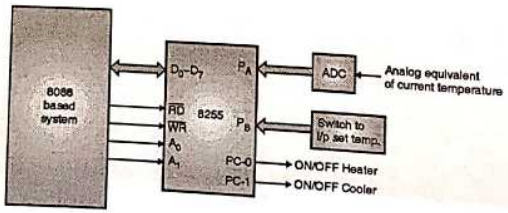


Fig. P. 1

I/O	GA mode		PA	PCU	GB mode		PB	PCL
1	0	0	1	0	0		1	0
	9			2				

Program for temperature controller

Label	Instruction
	MOV AL, 92H
	OUT 86H, AL
back :	IN AL, 80H
	MOV BL, AL
	IN AL, 82 H
	CMP AL, BL
	JC over
	MOV AL, 02 H
	OUT 84H, AL
	JMP back
over :	MOV AL, 01H
	OUT 84H, AL
	JMP back
	HLT

Program 2 : Interfacing DAC with 8086.

Ans. :

Write a program to generate a triangular waveform using DAC

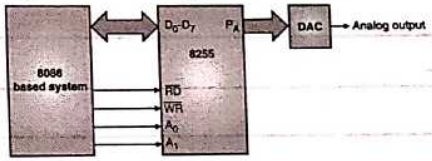


Fig. P. 2

I/O	GA mode		PA	PCU	GB mode		PB	PCL
1	0	0	0	0	0		0	0
	8			0				

Program for temperature controller

Label	Instruction
	MOV AL, 80H
	OUT 86H, AL
	MOV AL,00H
back :	OUT 82h,AL
	INC AL
	JMP back
	HLT