DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE MATERIALS



CST 307 MICROPROCESSOR AND MICROCONTROLLER

VISION OF THE INSTITUTION

To mould true citizens who are millennium leaders and catalysts of change through excellence in education.

MISSION OF THE INSTITUTION

NCERC is committed to transform itself into a center of excellence in Learning and Research in Engineering and Frontier Technology and to impart quality education to mould technically competent citizens with moral integrity, social commitment and ethical values.

We intend to facilitate our students to assimilate the latest technological know-how and to imbibe discipline, culture and spiritually, and to mould them in to technological giants, dedicated research scientists and intellectual leaders of the country who can spread the beams of light and happiness among the poor and the underprivileged.

ABOUT DEPARTMENT

- Established in: 2002
- Course offered : B.Tech in Computer Science and Engineering

  M.Tech in Computer Science and Engineering

  M.Tech in Cyber Security
- Approved by AICTE New Delhi and Accredited by NAAC
- Affiliated to the University of Dr. A P J Abdul Kalam Technological University.

DEPARTMENT VISION

Producing Highly Competent, Innovative and Ethical Computer Science and Engineering Professionals to facilitate continuous technological advancement.

DEPARTMENT MISSION

1. To Impart Quality Education by creative Teaching Learning Process
2. To Promote cutting-edge Research and Development Process to solve real world problems with emerging technologies.
3. To Inculcate Entrepreneurship Skills among Students.
4. To cultivate Moral and Ethical Values in their Profession.
5.

PROGRAMME EDUCATIONAL OBJECTIVES

PEO1: Graduates will be able to Work and Contribute in the domains of Computer Science and Engineering through lifelong learning.

PEO2: Graduates will be able to Analyse, design and development of novel Software Packages, Web Services, System Tools and Components as per needs and specifications.

PEO3: Graduates will be able to demonstrate their ability to adapt to a rapidly changing environment by learning and applying new technologies.

PEO4: Graduates will be able to adopt ethical attitudes, exhibit effective communication skills,Team work and leadership qualities.

PROGRAM OUTCOMES (POS)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSO)

PSO1: Ability to Formulate and Simulate Innovative Ideas to provide software solutions for Real-time Problems and to investigate for its future scope.

PSO2: Ability to learn and apply various methodologies for facilitating development of high quality Software Tools and Efficient Web Design Models with a focus on performance optimization.

PSO3: Ability to inculcate the Knowledge for developing Codes and integrating hardware/software products in the domains of Big Data Analytics, Web Applications and Mobile Apps to create innovative career path and for the socially relevant issues.

COURSE OUTCOMES

SUBJECT CODE: CST307		
COURSE OUTCOMES		
C304.1	K4	**Illustrate** the architecture, modes of operation and addressing modes of microprocessors
C304.2	K6	**Develop** 8086 assembly language programs
C304.3	K3	**Demonstrate** interrupts, its handling and programming in 8086.
C304.4	K4	**Illustrate** how different peripherals (8255,8254,8257) and memory are interfaced with microprocessors.
C304.5	K1	**Outline** features of microcontrollers and develop low level programs.

CO'S	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
C304.1	3	3	3	-	-	-	-	-	-	-	-	3
C304.2	3	3	3	3	-	-	-	-	-	-	-	3
C304.3	3	3	3	3	-	-	-	-	-	-	-	3
C304.4	3	3	3	3	-	-	-	-	-	-	-	3
C304.5	3	3	3	3	-	-	-	-	-	-	-	3
C304	3	3	3	3	-	-	-	-	-	-	-	3

CO'S	PSO1	PSO2	PSO3
C304.1	3	3	-
C304.2	3	3	-
C304.3	3	3	-
C304.4	3	3	-
C304.5	3	3	-
C304	3	3	-

Note: H-Highly correlated=3, M-Medium correlated=2, L-Less correlated=1

SYLLABUS

CST 307	MICROPROCESSORS AND MICROCONTROLLERS	Category	L	T	P	Credit	Year of Introduction
		PCC	3	1	0	4	2019

Preamble: The course enables the learners capable of understanding the fundamental architecture of microprocessors and micro controllers. This course focuses on the architecture, assembly language programming, interrupts, interfacing of microprocessors with peripheral devices and microcontrollers and its programming. It helps the learners to extend the study of latest processors and develop hardware based solutions.

Prerequisite : Sound knowledge in Logic System Design and Computer organization & architecture.

CO#	Course Outcomes
CO1	Illustrate the **architecture**, modes of operation and addressing modes of microprocessors **(Cognitive knowledge: Understand)**
CO2	Develop 8086 **assembly** language programs. **(Cognitive Knowledge Level: Apply)**
CO3	Demonstrate interrupts, its **handling** and programming in 8086. **(Cognitive Knowledge Level: Apply))**
CO4	Illustrate how different **peripherals** (8255,8254,8257) and memory are interfaced with microprocessors. **(Cognitive Knowledge Level: Understand)**
CO5	Outline features of **microcontrollers** and develop low level programs. **(Cognitive Knowledge Level: Understand)**

Continuous Internal Evaluation Pattern:

Attendance	:	**10 marks**
Continuous Assessment Tests	:	**25 marks**
Continuous Assessment Assignment	:	**15 marks**

Internal Examination Pattern:

Each of the two internal examinations has to be conducted out of 50 marks

First Internal Examination shall be preferably conducted after completing the first half of the syllabus and the Second Internal Examination shall be preferably conducted after completing remaining part of the syllabus.

There will be two parts: Part A and Part B. Part A contains 5 questions (preferably, 2 questions each from the completed modules and 1 question from the partly covered module), having 3 marks for each question adding up to 15 marks for part A. Students should answer all questions from Part A. Part B contains 7 questions (preferably, 3 questions each from the completed modules and 1 question from the partly covered module), each with 7 marks. Out of the 7 questions in Part B, a student should answer any 5.

End Semester Examination Pattern:

There will be two parts; Part A and Part B. Part A contains 10 questions with 2 questions from each module, having 3 marks for each question. Students should answer all questions. Part B contains 2 questions from each module of which a student should answer any one. Each question can have maximum 2 sub-divisions and carries 14 marks.

Syllabus

Module-1(Evolution of microprocessors):

8085 microprocessor (-Basic Architecture only). 8086 microprocessor – Architecture and signals, Physical Memory organization, Minimum and maximum mode of 8086 system and timings. Comparison of 8086 and 8088.Machine language Instruction format.

Module-2 (Addressing modes and instructions):

Addressing Modes of 8086. Instruction set – data copy /transfer instructions, arithmetic instructions, logical instructions, string manipulation instructions, branch instructions, unconditional and conditional branch instruction, flag manipulation and processor control instructions. Assembler Directives and operators. Assembly Language Programming with 8086.

Module- 3 (Stack and interrupts):

Stack structure of 8086, programming using stack- Interrupts - Types of Interrupts and Interrupt Service Routine- Handling Interrupts in 8086- Interrupt programming. -

COMPUTER SCIENCE AND ENGINEERIN

Programmable Interrupt Controller - 8259, Architecture (Just mention the control word, no need to memorize the control word)- Interfacing Memory with 8086.

Module- 4 (Interfacing chips):

Programmable Peripheral Input/output port 8255 - Architecture and modes of operation- Programmable interval timer 8254-Architecture and modes of operation- DMA controller 8257 Architecture (Just mention the control word, no need to memorize the control word of 8254 and 8257)

Module- 5 (Microcontrollers):

8051 Architecture- Register Organization- Memory and I/O addressing- Interrupts and Stack- 8051 Addressing Modes- Instruction Set- data transfer instructions, arithmetic instructions, logical instructions, Boolean instructions, control transfer instructions- Simple programs.

Text Books

1. Bhurchandi and Ray, Advanced Microprocessors and Peripherals, Third Edition McGraw Hill.
2. Raj Kamal, Microcontrollers: Architecture, Programming, Interfacing and System Design, Pearson Education.
3. Ramesh Gaonkar, Microprocessor Architecture, Programming, and Applications with the 8085, Penram International Publishing Pvt. Ltd.

Reference Books

1. Barry B. Brey, The Intel Microprocessors – Architecture, Programming and Interfacing, Eighth Edition, Pearson Education.
2. A. NagoorKani, Microprocessors and Microcontrollers, Second Edition, Tata McGraw Hill
3. Douglas V. Hall, SSSP Rao, Microprocessors and Interfacing, Third Edition, McGrawHill Education.

Sample Course Level Assessment Questions

Course Outcome1 (CO1):

1) Describe how pipelining is implemented in 8086 microprocessor
2) Illustrate maximum mode signals in 8086.

Course Outcome 2(CO2):

1) Write an 8086 assembly language program for sorting a sequence of N, 8 bit numbers. Describe the modifications that can be done on the above program so that it will sort N, 16 bit numbers. Rewrite the program with those modifications also.

COMPUTER SCIENCE AND ENGINEERIN

Course Outcome 3 (CO3):

1) Design an interface between 8086 CPU and two chips of 16 x 8 EPROM and

two chips of 32K x 8 RAM. Select the starting address of EPROM suitably.

The RAM address must start at 00000H.

2) Give the sequence of instructions for setting the IVT for interrupt type 23H. Assume the Interrupt Service Routine, is present in the code segment named CODE.

3) Describe the role of Interrupt Request register and In service register in 8259.

Course Outcome 4(CO4):

1) Show how to interface an 8255 with 8086 to work as an I/O port with the following specifications. Initialize port A as output, port B as input and port C as output. Port A address should be 05A0H. Write a program to sense switch positions SW 0 -SW 7 connected to port B. The sensed pattern is to be displayed on port A, to which 8 LED's are attached, while port C lower displays *number of off switches* out of total 8 switches.

2) Specify the importance of the DMA address register and Terminal count register in 8257.

Course Outcome 5(CO5):

1) Write an 8051 assembly language program to count the number of 1's and 0's in a given 8 bit number

2) Write an 8051 assembly language program for computing the square root of an 8 bit number.

QUESTION BANK

MODULE I				
Q:NO:	**QUESTIONS**	**CO**	**KL**	**PAGE NO:**
1	Briefly describe the evolution of microprocessor	CO1	K2	
2	Compare 8085 and 8086	CO1	K3	
3	List any six features of 8086 microprocessor.	CO1	K3	
4	Explain the basic functional block of 8086 with neat diagram	CO1	K2	
5	Define instruction pointer and Ax register in 8086 microprocessor	CO1	K2	
6	Draw and explain internal block diagram of 8086	CO1	K2	
7	Draw the Memory write timing diagram of 8086 in Minimum mode. Describe the status of the relevant signals during each clock period.	CO1	K2	
8	Explain the memory segmentation process and the generation of 20 bit physical address	CO1	K2	
9	Draw the Memory Read timing diagram of 8086 in Minimum mode. Describe the status of the relevant signals during each clock period	CO1	K2	
10	Explain the evolution of microprocessor.	CO1	K2	
11	Explain the maximum mode operation of 8086 with suitable diagrams.	CO1	K2	
MODULE II				
1	Discuss about 8086 PUSH & POP instruction with example.	CO2	K2	

2	List out the flag registers in 8086 microprocessor	CO2	K3	
3	Compare String manipulation and branch instruction..	CO2	K3	
4	Define the functions of the following 8086 assembler directives a) ASSUME b) EQU c) OFFSET	CO2	K2	
5	Describe any five addressing modes of 8086 with example	CO2	K2	
6	Explain unconditional and conditional branch instructions.	CO2	K2	
7	Explain the various directives of 8086?Explain each directive with syntax and examples	CO2	K2	
8	Discuss the different addressing modes supported by 8086.Give explanations with suitable example	CO2	K2	
9	Explain the significance of assembler directives in an assembly language program with suitable example.	CO2	K2	

10	Explain flag manipulation and process control instructions.	CO2	K2	71
	MODULE III			
1	With a neat diagram describe stack structure of 8086	CO3	K3	
2	Explain the response of 8086 to an interrupt and state the actions initiated on the occurrence of the interrupt	CO3	K2	
3	Explain the block diagram of 8259, priority interrupt controller.	CO3	K2	
4	Explain different types of interrupts in 8086 system.	CO3	K2	
5	Explain Interrupt Vector Table (IVT)? Explain IVT in 8086	CO3	K2	
6	Explain various types of interrupt in 8086. Write the condition(s) which cause 8086 to perform a Type 1, Type 2 and Type 3 interrupts	CO3	K2	
7	Discuss 8086 interrupt acknowledgement cycle	CO3	K2	
8	Explain the differences between 8086 procedure call and RET	CO3	K2	
9	Explain the basic categories of 8086 software interrupts?	CO3	K1	

MODULE IV

1	Briefly describe the control word format of 8255 PPI	CO4	K2	
2	Explain the features of 8257, DMA controller.	CO4	K1	
3	Write short notes on 8254.	CO4	K2	
4	Draw the architectural block diagram of 8255 and explain.	CO4	K2	
5	Design an interface between 8086 CPU and two chips of 16 x 8 EPROM and two chips of 32K x 8 RAM. Select the starting address of EPROM suitably. The RAM address must start at 00000H	CO4	K5	
6	Explain DMA controller with neat architecture	CO4	K2	
7	Interface 32Kx8 RAM using four numbers of 8Kx8 memory chips and 16Kx8 ROM using two numbers of 8Kx8 EPROM chips. The address map is given as RAM starts at 000000H and ROM ends at FFFFFH.	CO4	K5	
8	Draw the architectural block diagram of 8257 and explain the role of each functional part.	CO4	K2	
9	Draw the architectural block diagram of 8255 and explain the role of each functional part and control word format.	CO4	K2	
	MODULE V			
1	Discuss the selection criteria of a typical microcontroller.	CO5	K2	
2	Describe the program status word (PSW) of 8051.	CO5	K2	
3	Compare microprocessor and microcontroller	CO5	K3	
4	Explain different types of instructions used in a microcontroller with example	CO5	K2	
5	Explain the different operating modes of8051?	CO5	K2	

6	Describe different types of microcontrollers. What are the different criteria that should be considered while selecting a microcontroller?	CO5	K2	
7	Explain the logical instructions in8051	CO5	K2	
8	Discuss the arithmetic instructions in8051	CO5	K3	
9	Explain data transfer instructions in 8051	CO5	K3	
10	Explain different types of instructions used in a microcontroller with example	CO5	K2	
11	Explain about the programmable I/O ports of 8051 microcontroller	CO5	K5	
12	Draw the architecture of 8051and explain its	CO5	K2	
13	Describe control transfer instructions.	CO5	K2	
APPENDIX 1				
CONTENT BEYOND THE SYLLABUS				
1	PIC microcontroller	229		
2	IOT	235		

MODULE 1

Microprocessors and Microcontrollers

Module 1

Evolution of microprocessors, 8085 microprocessor- basic architecture only, 8086 Microprocessor - Architecture and signals, Physical Memory organisation, Minimum and maximum mode of 8086 system and timings. Comparison of 8086 and 8088.Machine language instruction format

1) Evolution of microprocessors

Microprocessors have undergone significant evolution over the past four decades. Development of some of the microprocessors can be given as follows.

- 4-bit Microprocessors

The first microprocessor was introduced in 1971 by Intel Corp.
It was named Intel 4004 as it was a 4 bit processor.
It could perform simple arithmetic and logic operations such as addition, subtraction, boolean AND and boolean OR.
The enhanced version of 4004 was Intel 4040. Some other 4 bit processors are International's PPS4 and Thoshiba's T3472.

- 8-bit Microprocessors

The first 8 bit microprocessor which could perform arithmetic and logic operations on 8 bit words was introduced in 1973 again by Intel.
This was Intel 8008 and was later followed by an improved version, Intel 8088. Some other 8 bit processors are Zilog-80 and Motorola M6800.

- 16-bit Microprocessors

The 8-bit processors were followed by 16 bit processors. They are Intel 8086 and 80286.

- 32-bit Microprocessors

The 32 bit microprocessors were introduced by several companies but the most popular one is Intel 80386.

- Pentium Series

Instead of 80586, Intel came out with a new processor namely Pentium processor.
Its performance is closer to RISC performance.
Pentium was followed by Pentium Pro CPU.
The low cost version of Pentium II is celeron.
The Pentium III provided high performance floating point operations for certain types of computations by using the SIMD extensions to the instruction set.
Pentium III was faster than high-end RISC CPUs.
Pentium IV could not execute code faster than the Pentium III when running at the same clock frequency.

First Generation Microprocessors

- The first generation microprocessors were introduced in the year 1971-1972.

- The instructions of these microprocessors were processed serially.

Second Generation Microprocessors

In the year 1970, small amount of transistors were available on the integrated circuit in the second generation microprocessors.

- Examples: 16-bit arithmetic 7 pipelined instruction processing, MC68000 Motorola microprocessor.
- These processors were introduced in the year 1979, and Intel 8080 processor is another example.

Third Generation Microprocessors: The third generation microprocessors were introduced in the year 1978, as denoted by Intel's 8086 and the Zilog Z8000.

These were 16-bit processors with a performance like mini computers.

Fourth Generation Microprocessors: Leading edge microprocessors like Motorola's 88100 and Intel's 80960CA could issue & retire more than one instruction per clock cycle.

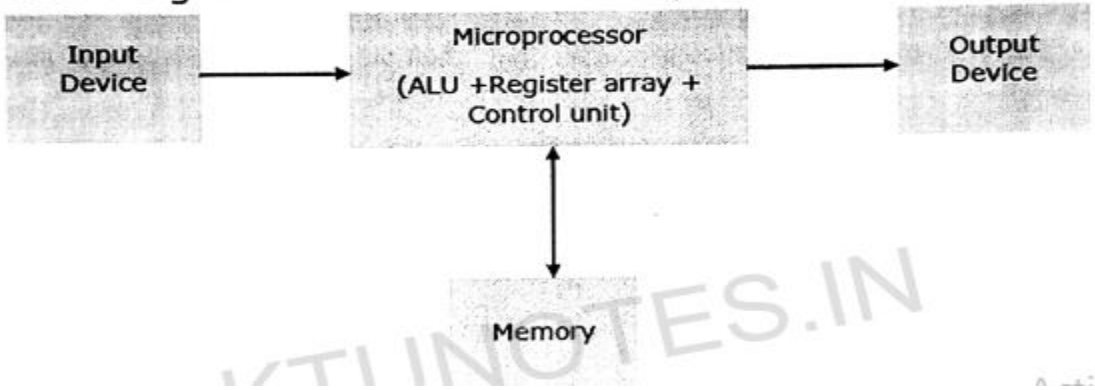
Fifth Generation Microprocessors: Fifth generation microprocessors employed decoupled super scalar processing, and their design soon exceeded 10 million transistors.

In fifth generation, PCs are a low-margin, high volume business conquered by a single microprocessor.

Microprocessor is a controlling unit of a micro-computer, fabricated on a small chip capable of performing ALU (Arithmetic Logical Unit) operations and communicating with the other devices connected to it.

Microprocessor consists of an ALU, register array, and a control unit. ALU performs arithmetical and logical operations on the data received from the memory or an input device. Register array consists of registers identified by letters like B, C, D, E, H, L and accumulator. The control unit controls the flow of data and instructions within the computer.

Block Diagram of a Basic Microcomputer



KTUNOTES.IN

2)8085 architecture

8085 is pronounced as "eighty-eighty-five" microprocessor. It is an 8-bit microprocessor designed by Intel in 1977 using NMOS technology.

It has the following configuration –

- 8-bit data bus
- 16-bit address bus, which can address upto 64KB
- A 16-bit program counter
- A 16-bit stack pointer
- Six 8-bit registers arranged in pairs: BC, DE, HL
- Requires +5V supply to operate at 3.2 MHZ single phase clock

It is used in washing machines, microwave ovens, mobile phones, etc.

8085 Microprocessor – Functional Units

8085 consists of the following functional units –

Accumulator

It is an 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal data bus & ALU.

KTUNOTES.IN

Arithmetic and logic unit

As the name suggests, it performs arithmetic and logical operations like Addition, Subtraction, AND, OR, etc. on 8-bit data.

Activate
Go to PC se

General purpose register

There are 6 general purpose registers in 8085 processor, i.e. B, C, D, E, H & L. Each register can hold 8-bit data.

These registers can work in pair to hold 16-bit data and their pairing combination is like B-C, D-E & H-L.

Program counter

It is a 16-bit register used to store the memory address location of the next instruction to be executed. Microprocessor increments the program whenever an instruction is being executed, so that the program counter points to the memory address of the next instruction that is going to be executed.

Activate
Go to PC s

Stack pointer

It is also a 16-bit register works like stack, which is always incremented/decremented by 2 during push & pop operations.

Temporary register

It is an 8-bit register, which holds the temporary data of arithmetic and logical operations.

Flag register

It is an 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending upon the result stored in the accumulator.

These are the set of 5 flip-flops –

- Sign (S)
- Zero (Z)
- Auxiliary Carry (AC)
- Parity (P)
- Carry (C)

Its bit position is shown in the following table –

KTUNOTES.IN

D7	D6	D5	D4	D3	D2	D1	D0
S	Z		AC		P		CY

Instruction register and decoder

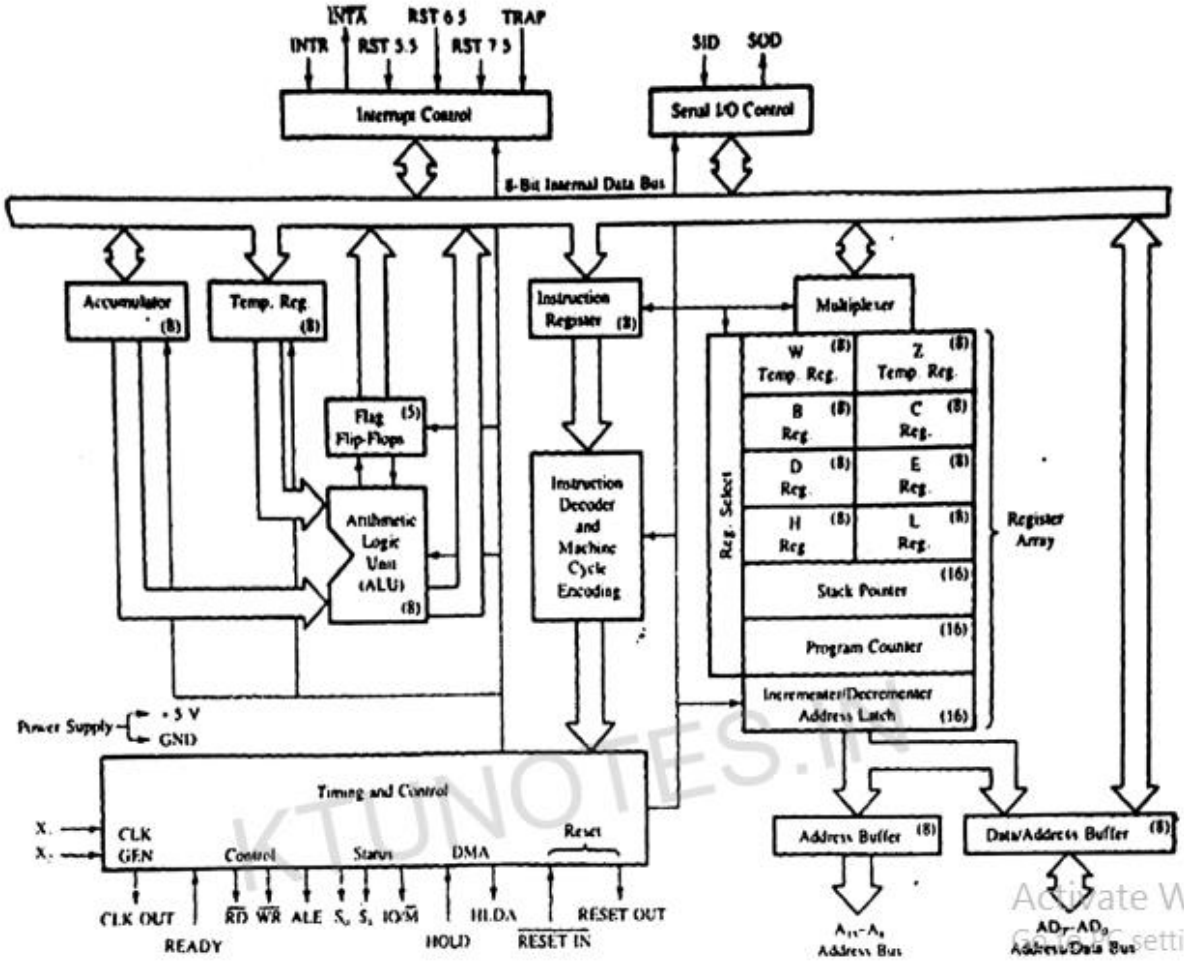
It is an 8-bit register. When an instruction is fetched from memory then it is stored in the Instruction register. Instruction decoder decodes the information present in the Instruction register.

Timing and control unit

It provides timing and control signal to the microprocessor to perform operations. Following are the timing and control signals, which control external and internal circuits –

- Control Signals: READY, RD', WR', ALE
- Status Signals: S0, S1, IO/M'

Figure below shows, 8085 architecture



- DMA Signals: HOLD, HLDA
- RESET Signals: RESET IN, RESET OUT

Interrupt control

As the name suggests it controls the interrupts during a process. When a microprocessor is executing a main program and whenever an interrupt occurs, the microprocessor shifts the control from the main program to process the incoming request. After the request is completed, the control goes back to the main program.

There are 5 interrupt signals in 8085 microprocessor: INTR, RST 7.5, RST 6.5, RST 5.5, TRAP.

Serial Input/output control

It controls the serial data communication by using these two instructions: SID (Serial input data) and SOD (Serial output data).

Address buffer and address-data buffer

The content stored in the stack pointer and program counter is loaded into the address buffer and address-data buffer to communicate with the CPU. The memory and I/O chips are connected to these buses; the CPU can exchange the desired data with the memory and I/O chips.

Address bus and data bus

Data bus carries the data to be stored. It is bidirectional, whereas address bus carries the location to where it should be stored and it is unidirectional. It is used to transfer the data & Address I/O devices.

KTUNOTES.IN
Activate Wir
Go to setting

The pins of a 8085 microprocessor can be classified into seven groups –

Address bus

A15-A8, it carries the most significant 8-bits of memory/IO address.

Data bus

AD7-AD0, it carries the least significant 8-bit address and data bus.

Control and status signals

These signals are used to identify the nature of operation. There are 3 control signal and 3 status signals.

Three control signals are RD, WR & ALE.

- **RD** – This signal indicates that the selected IO or memory device is to be read and is ready for accepting data available on the data bus.
- **WR** – This signal indicates that the data on the data bus is to be written into a selected memory or IO location.
- **ALE** – It is a positive going pulse generated when a new operation is started by the microprocessor. When the pulse goes high, it indicates address. When the pulse goes down it indicates data.

Three status signals are IO/M, S0 & S1.

IO/M

This signal is used to differentiate between IO and Memory operations, i.e. when it is high indicates IO operation and when it is low then it indicates memory operation.

S1 & S0

These signals are used to identify the type of current operation.

Power supply

There are 2 power supply signals – VCC & VSS. VCC indicates +5v power supply and VSS indicates ground signal.

KTUNOTES.IN

Clock signals

There are 3 clock signals, i.e. X1, X2, CLK OUT.

- **X1, X2** – A crystal (RC, LC N/W) is connected at these two pins and is used to set frequency of the internal clock generator. This frequency is internally divided by 2.
- **CLK OUT** – This signal is used as the system clock for devices connected with the microprocessor.

Interrupts & externally initiated signals

Interrupts are the signals generated by external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR. We will discuss interrupts in detail in interrupts section.

- **Vector interrupt** – In this type of interrupt, the interrupt address is known to the processor. **For example:** RST7.5, RST6.5, RST5.5, TRAP.
- **Non-Vector interrupt** – In this type of interrupt, the interrupt address is not known to the processor so, the interrupt address needs to be sent externally by the device to perform interrupts. **For example:** INTR.
- **Maskable interrupt** – In this type of interrupt, we can disable the interrupt by writing some instructions into the program. **For example:**RST7.5, RST6.5, RST5.5.
- **Non-Maskable interrupt** – In this type of interrupt, we cannot disable the interrupt by writing some instructions into the program. **For example:** TRAP.
- **Software interrupt** – In this type of interrupt, the programmer has to add the instructions into the program to execute the interrupt. There are 8 software interrupts in 8085, i.e. RST0, RST1, RST2, RST3, RST4, RST5, RST6, and RST7.
- **Hardware interrupt** – There are 5 interrupt pins in 8085 used as hardware interrupts, i.e. TRAP, RST7.5, RST6.5, RST5.5, INTA.

KTUNOTES.IN

Note – NTA is not an interrupt, it is used by the microprocessor for sending acknowledgement. TRAP has the highest priority, then RST7.5 and so on.

Interrupt Service Routine (ISR)

A small program or a routine that when executed, services the corresponding interrupting source is called an ISR.

Activ
Go to

TRAP

It is a non-maskable interrupt, having the highest priority among all interrupts. Bydefault, it is enabled until it gets acknowledged. In case of failure, it executes as ISR and sends the data to backup memory. This interrupt transfers the control to the location 0024H.

RST7.5

It is a maskable interrupt, having the second highest priority among all interrupts. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 003CH address.

RST 6.5

It is a maskable interrupt, having the third highest priority among all interrupts. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 0034H address.

RST 5.5

It is a maskable interrupt. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 002CH address.

INTR

It is a maskable interrupt, having the lowest priority among all interrupts. It can be disabled by resetting the microprocessor.

When **INTR signal goes high**, the following events can occur –

- The microprocessor checks the status of INTR signal during the execution of each instruction.
- When the INTR signal is high, then the microprocessor completes its current instruction and sends active low interrupt acknowledge signal.
- When instructions are received, then the microprocessor saves the address of the next instruction on stack and executes the received instruction.

KTUNOTES.IN

3) 8086 Microprocessor -Architecture and signals

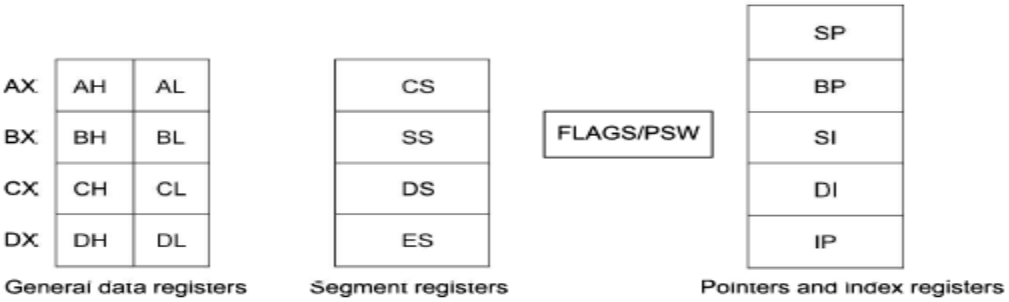


Fig: Register Organisation of 8086

Internal Registers of 8086

Registers of 8086 are classified as General Purpose registers and Special purpose registers**.**

These are 16 bit registers. The general purpose registers are used as 8 bit or 16 bit registers.

•There are four groups of the user accessible internal registers.
They are the instruction pointer, four data registers, four pointer and index register, four segment registers.
•The 8086 has a total of fourteen 16-bit registers including a 16 bit register called the **status register**, with 9 of bits implemented for status and control flags.
There are four different 64 KB segments for instructions, stack, data and extra data.
Four segment registers of 8086 are:
•**Code segment** (CS) is a 16-bit register containing address of 64 KB segment with processor instructions.
The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register.
•**Stack segment** (SS) is a 16-bit register containing address of 64KB segment with program stack.
The processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment.
•**Data segment** (DS) is a 16-bit register containing address of 64KB segment with program data.
The processor assumes that all data referenced by AX, BX, CX, DX and index register (SI, DI) is located in the data segment.
•**Accumulator** register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX.

AL in this case contains the low-order byte of the word, and AH contains the high-order byte.

•**Base** register consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX.
BL in this case contains the low-order byte of the word, and BH contains the high-order byte.
BX register usually contains a data pointer.

•**Count** register consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX.
CL register contains the low-order byte of the word, and CH contains the high-order byte.
Count register can be used as a counter in string manipulation.
•**Data** register consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX.
DL register contains the low-order byte of the word, and DH contains the high-order byte.
Data register can be used as a port number in I/O operations.
In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.
•**The following registers are both general and index registers:**
•**Stack Pointer** (SP) is a 16-bit register pointing to program stack.
•**Base Pointer** (BP) is a 16-bit register pointing to data in stack segment.
BP register is usually used for based, based indexed or register indirect addressing.
•**Source Index** (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data address in string manipulation instructions.
•**Destination Index** (DI) is a 16-bit register.
DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.
Other registers:
•**Instruction Pointer** (IP) is a 16-bit register.
•**Flags** is a 16-bit register containing 9 one bit flags.
•**Overflow Flag** (OF) - set if the result is too large positive number, or is too small negative number to fit into destination operand.
•**Direction Flag** (DF) - if set then string manipulation instructions will auto-decrement index registers.
•**Interrupt-enable Flag** (IF) - setting this bit enables maskable interrupts.
•**Single-step Flag** (TF) - if set then single-step interrupt will occur after the next instruction.
Sign Flag (SF) - set if the most significant bit of the result is set.
•**Zero Flag** (ZF) - set if the result is zero.
•**Auxiliary carry Flag** (AF) - set if there was a carry from or borrow to bits 0-3 in the AL register.
•**Parity Flag** (PF) - set if parity (the number of "1" bits) in the low-order byte of the result is even.
•**Carry Flag** (CF) - set if there was a carry from or borrow to the most significant bit during last result calculation.

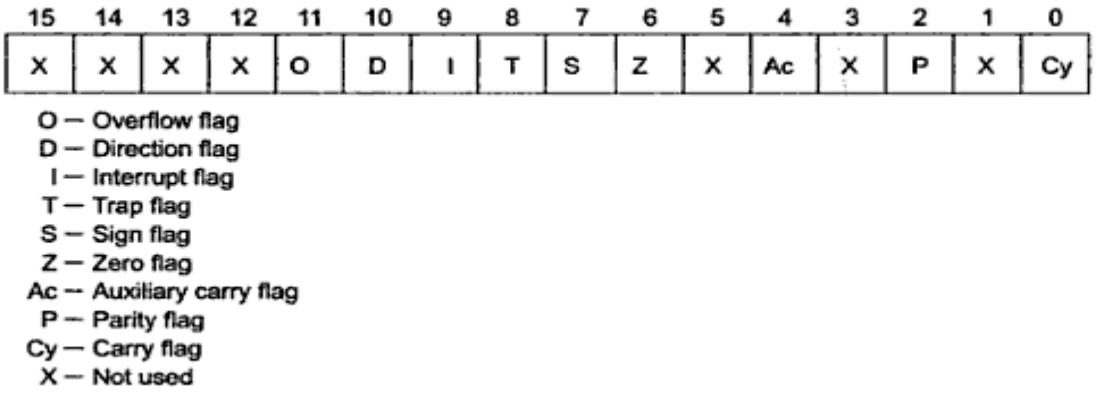


Fig: Flag register of 8086

8086 Microprocessor

•It is a 16-bit µp.

•8086 has a 20 bit address bus can access up to 220 memory locations (1 MB).

•It can support up to 64K I/O ports.

•It provides 14, 16 -bit registers.

•It has multiplexed address and data bus AD0- AD15 and A16 – A19.

•It can prefetches upto 6 instruction bytes from memory and queues them in order to speed up instruction execution.

•It requires +5V power supply.

•A 40 pin dual in line package

Minimum and Maximum Modes: The minimum mode is selected by applying logic 1 to the MN / MX input pin. This is a single microprocessor configuration.

•The maximum mode is selected by applying logic 0 to the MN / MX input pin. This is a multi micro processors configuration.

Internal Architecture of 8086

•8086 has two blocks BIU and EU.

•The BIU performs all bus operations such as instruction fetching, reading and writing operands for memory and calculating the addresses of the memory operands.

The instruction bytes are transferred to the instruction queue.

•EU executes instructions from the instruction system byte queue.

•EU contains Control circuitry, Instruction decoder, ALU, Pointer and Index register, Flag register.

•BIU contains Instruction queue, Segment registers, Instruction pointer, Address adder.

BUS INTERFACE UNIT:

•It provides a full 16 bit bidirectional data bus and 20 bit address bus.

•The bus interface unit is responsible for performing all external bus operations.

Specifically it has the following functions:

•Instruction fetch, Instruction queuing, Operand fetch and storage, Address relocation and Bus control.

The BIU uses a mechanism known as an instruction stream queue to implement a **pipeline architecture.**

•This queue permits prefetch of up to six bytes of instruction code.
Whenever the queue of the BIU is not full, it has room for at least two more bytes and at the same time the EU is not requesting it to read or write operands from memory, the BIU is free to look ahead in the program by prefetching the next sequential instruction.
•These prefetching instructions are held in its FIFO queue.

•The intervals of no bus activity, which may occur between bus cycles are known as **Idle state**.
•The BIU first completes the instruction fetch bus cycle before initiating the operand read / write cycle.
•The BIU also contains a dedicated adder which is used to generate the 20bit physical address that is output on the address bus.
This address is formed by adding an appended 16 bit segment address and a 16 bit offset address.

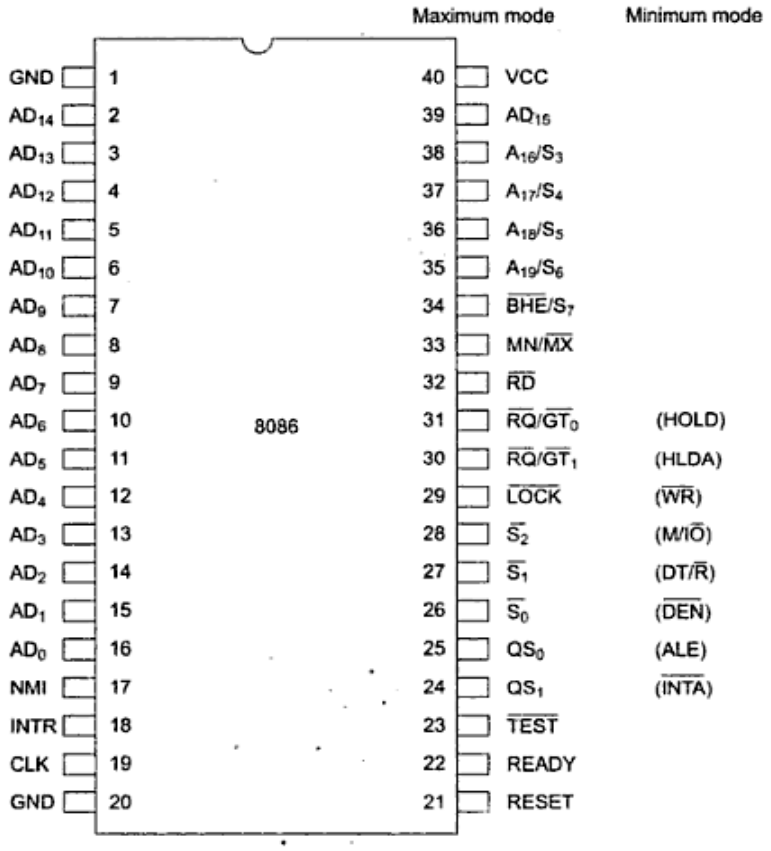
EXECUTION UNIT

The Execution unit is responsible for decoding and executing all instructions.
•The EU extracts instructions from the top of the queue in the BIU, decodes them, generates operands if necessary, passes them to the BIU and requests it to perform the read or write bys cycles to memory or I/O and perform the operation specified by the instruction on the operands.
•During the execution of the instruction, the EU tests the status and control flags and updates them based on the results of executing the instruction.
•If the queue is empty, the EU waits for the next instruction byte to be fetched and shifted to top of the queue.

Signal descriptions of 8086

The Microprocessor 8086 is a 16-bit CPU available in different clock rates and packaged in a 40 pin CERDIP or plastic package.
The pins serve a particular function in minimum mode (single processor mode) and other function in maximum mode configuration (multiprocessor mode).

•**The following signal descriptions are common for both modes.**
•**AD15-AD0**: These are the time multiplexed memory I/O address and data lines.
• Address remains on the lines during T1 state, while the data is available on the data bus during T2, T3, Tw and T4.



Pin Configuration of 8086

•**A19/S6,A18/S5,A17/S4,A16/S3:** These are the time multiplexed address and status lines.
•During T1 these are the most significant address lines for memory operations.
• During memory or I/O operations, status information is available on those lines for T2,T3,Tw and T4.
•The S4 and S3 combinedly indicate which segment register is presently being used for memory accesses as in fig.
The status line S6 is always low.
•The address bit are separated from the status bit using latches controlled by the ALE signal.

•Bit **S4 and S3** together from a 2 bit binary code that identifies which of the 8086 internal segment registers are used to generate the physical address.
•Code S4S3 = 00 identifies a register known as ***extra segment register*** as the source of the segment address.
•**BHE/S7:** The bus high enable is used to indicate the transfer of data over the higher order (D15-D8) data bus.

It goes low for the data transfer over D15-D8 and is used to derive chip selects of odd address memory bank or peripherals.

The status information is available during T2, T3 and T4.

It is low during T1 for the first pulse of the interrupt acknowledges cycle.

•**RDRead:** This signal on low indicates the peripheral that the processor is performing memory or I/O read operation.

RD is active low and shows the state for T2, T3, Tw of any read cycle. The signal remains tristated during the hold acknowledge.

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

S_4	S_3	Indication
0	0	Alternate Data
0	1	Stack
1	0	Code or none
1	1	Data

•**INTR-Interrupt Request**: This is a triggered input.
If any interrupt request is pending, the processor enters the interrupt acknowledge cycle.
•This can be internally masked by resulting the interrupt enable flag.
•**TEST** This input is examined by a 'WAIT' instruction.
If the TEST pin goes low, execution will continue, else the processor remains in an idle state.
The input is synchronized internally during each clock cycle on leading edge of clock.
•**CLK**- Clock Input: The clock input provides the basic timing for processor operation and bus control activity.
Its an asymmetric square wave with 33% duty cycle.
MN/MX: The logic level at this pin decides whether the processor is to operate in either minimum or maximum mode.
•**The following pin functions are for the minimum mode operation of 8086.**
•**M/IO – Memory/IO**: This signal is logically equivalent to S2 in maximum mode.
When it is low, it indicates the CPU is having an I/O operation, and when it is high, it indicates that the CPU is having a memory operation.
This line remains active till final T4 of the current cycle.
It is tristated during local bus "hold acknowledge".
•**INTA Interrupt Acknowledge**: This signal is used as a read strobe for interrupt acknowledge cycles. i.e. when it goes low, the processor has accepted the interrupt.
•**ALE – Address Latch Enable**: This output signal indicates the availability of the valid address on the address/data lines, and is connected to latch enable input of latches.
This signal is active high and is never tristated.
•**DT/R – Data Transmit/Receive**: This output is used to decide the direction of data flow through the transreceivers (bidirectional buffers).
When the processor sends out data, this signal is high and when the processor is receiving data, this signal is low.

•**DEN – Data Enable**: This signal indicates the availability of valid data over the address/data lines.
It is used to enable the transreceivers (bidirectional buffers) to separate the data from the multiplexed address/data signal.
•**HOLD, HLDA- Acknowledge**: When the HOLD line goes high, it indicates to the processor that another master is requesting the bus access.
•If the DMA request is made while the CPU is performing a memory or I/O cycle, it will release the local bus during T4 provided:
1. The request occurs on or before T2 state of the current cycle.
2.The current cycle is not operating over the lower byte of a word.
3.The current cycle is not the first acknowledge of an interrupt acknowledge sequence.

4. A Lock instruction is not being executed.

•**The following pin function are applicable for maximum mode operation of 8086**.
•**S2, S1, S0 – Status Lines**: These are the status lines which reflect the type of operation, being carried out by the processor.

These become activity during T4 of the previous cycle and active during T1 and T2 of the current bus cycles.

•**LOCK** When the CPU is executing a critical instruction which requires the system bus, the LOCK prefix instruction ensures that other processors connected in the system will not gain the control of the bus.
The LOCK signal is activated by the 'LOCK' prefix instruction and remains active until the completion of the next instruction.
•**QS1, QS0 – Queue Status:** These lines give information about the status of the code-prefetch queue.
These are active during the CLK cycle after while the queue operation is performed.

•The 8086 architecture has 6-byte instruction prefetch queue.
Even the largest (6-bytes) instruction can be prefetched from the memory and stored in the prefetch.
•By prefetching the instruction, there is a considerable speeding up in instruction execution in 8086.
This is known as ***instruction pipelining***.
•While the execution unit is busy in executing an instruction, after it is completely decoded, the bus interface unit may be fetching the bytes of the next instruction from memory, depending upon the queue status.

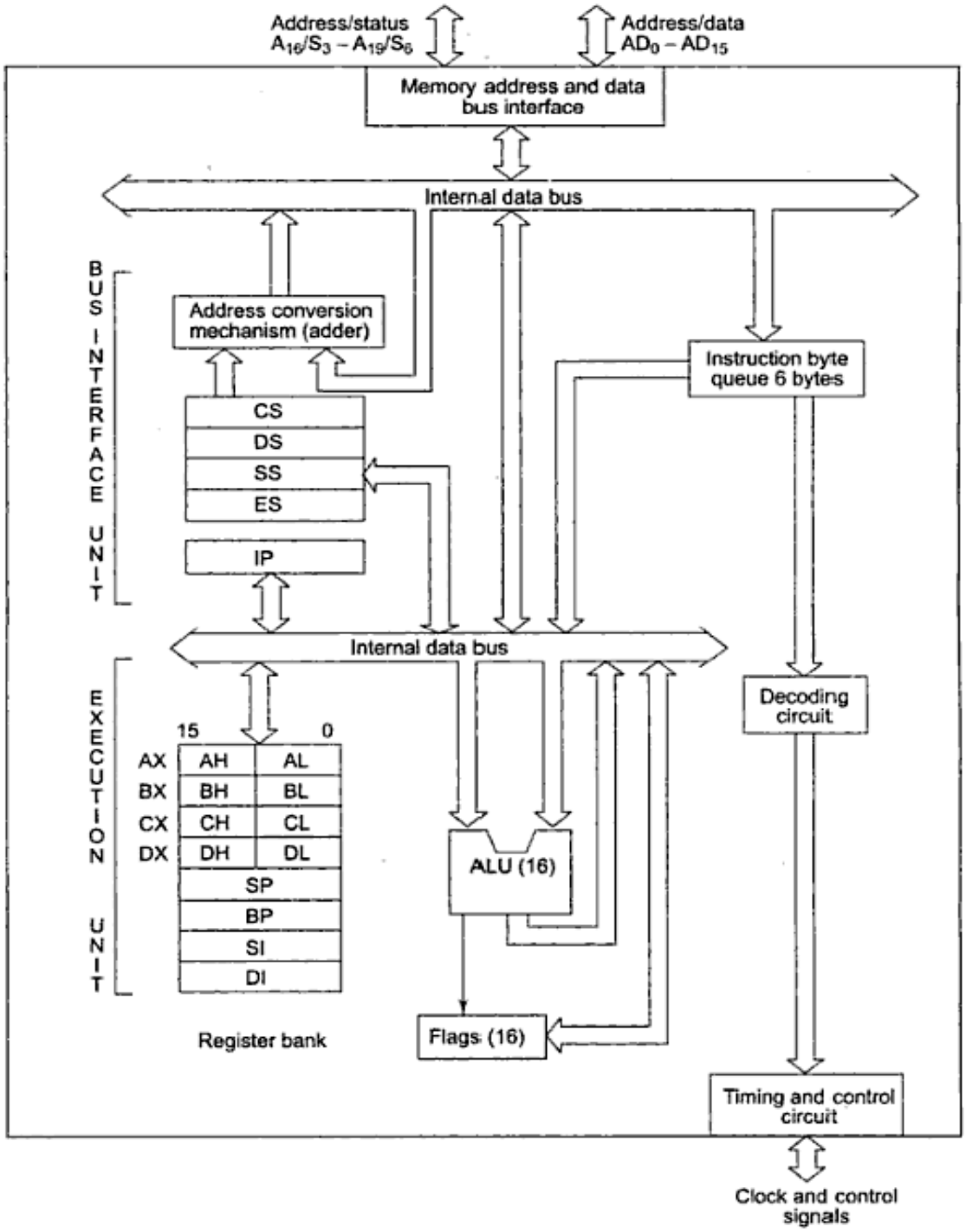


Fig: 8086 architecture

RQ/0GT,RQ/1GT – Request/Grant: These pins are used by the other local bus master in maximum mode, to force the processor to release the local bus at the end of the processor current bus cycle.

•Each of the pin is bidirectional with RQ/GT0 having higher priority than RQ/GT1.

RQ/GT pins have internal pull-up resistors and may be left unconnected.

BHE	A_0	Indication
0	0	Whole word
0	1	Upper byte from or to even address
1	0	Lower byte from or to even address
1	1	None

S_2	S_1	S_0	Indication
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code Access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

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

•Request/Grant sequence is as follows:

1.A pulse of one clock wide from another bus master requests the bus access to 8086.

2.During T4(current) or T1(next) clock cycle, a pulse one clock wide from 8086 to the requesting master, indicates that the 8086 will enter the 'hold acknowledge' state at next cycle.

3.A one clock wide pulse from the another master indicates to the 8086 that the 8086 may regain control of the local bus at the next clock cycle.

Thus each master to master exchange of the local bus is a sequence of 3 pulses.

3)Memory Organisation

Memory •Program, data and stack memories occupy the same memory space.

As the most of the processor instructions use 16-bit pointers the processor can effectively address only 64 KB of memory.

•The CPU uses special segment registers to specify where the code, stack and data 64 KB segments are positioned within 1 MB of memory.

•16-bit pointers and data are stored as: address: low-order byte address+1: high-order byte.
•**Program memory** - Program can be located anywhere in memory.
Jump and call instructions can be used for short jumps within currently selected 64 KB code segment, as well as for far jumps anywhere within 1 MB of memory.
•All conditional jump instructions can be used to jump within approximately +127 to -127 bytes from current instruction.
Data memory - The processor can access data in any one out of 4 available segments, which limits the size of accessible memory to 256 KB.
Accessing data from the Data, Code, Stack or Extra segments can be usually done by prefixing instructions with the DS:, CS:, SS: or ES:
•**Stack memory** can be placed anywhere in memory.
The stack can be located at odd memory addresses, but it is not recommended for performance reasons.
Reserved locations: 0000h - 03FFh are reserved for interrupt vectors. Each interrupt vector is a 32-bit pointer in format segment: offset.
•FFFF0h - FFFFFh - after RESET the processor always starts program execution at the FFFF0h address.
Memory Segmentation: The total memory size is divided into segments of various sizes.

A segment is just an area in memory. The 8086 can access memory with address ranging from 00000 H to FFFFFH.

Each Segment register stores the base address (starting address) of the corresponding segment.

- Segment registers only store the upper 16 bits. 20-bit address is obtained from 16- bit registers.
- The 20-bit address of a byte is called its Physical Address.
- Logical address is in the form of: Base Address : Offset
- Offset is the displacement of the memory location from the starting location of the segment.
- Eg: The value of Data Segment Register (DS) is 2222 H.
- The BIU appends 0H to the LSBs of the address.
- BIU uses the following formula: Effective Address = Starting Address of Segment + Offset.

All offsets are limited to 16-bits. The maximum size possible for segment is 216 = 65,535 bytes (64 KB).

The area of memory from the start of the second segment to the possible end of the first segment is called as overlapped segment.

The main advantages of the segmented memory scheme are as follows: a)Allows the memory capacity to be 1 MB although the actual addresses to be handled are of 16-bit size.

b) Allows the placing of code data and stack portions of the same program in different parts (segments) of memory, for data and code protection.

c) Permits a program and/ or its data to be put into different areas of memory each time program is executed, ie, relocation may be done.

4) Physical Memory Organization

The 1MB memory is physically organized as odd bank and even bank, each of 512kbytes, addressed in parallel by the processor.

Byte data with even address is transferred on $D_7 - D_0$ and byte data with odd address is transferred on $D_{15} - D_8$.

The processor provides two enable signals, $\overline{BHE}$ and A_0 for selecting of either even or odd or both the banks.

$D_{15} - D_0 / A_{15} - A_0$ are the common signal line in 8086 as $AD_{15} - AD_0$

FIG

$\overline{BHE}$	A_0	Function
0	0	Whole word
0	1	Upper byte/ odd address
1	0	Lower byte/even address
1	1	none

If the address bus contains $FFFFO_H$

If $\overline{BHE} = 0$, then it reads the 16 bits data from memory location $FFFFO_H$ and $FFFF1_H$.

If $\overline{BHE} = 1$, it reads the 8 bits data from memory location $FFFFO_H$.

Fig shows the physical memory organisation of 8086.

5)Minimum Mode 8086 System

•In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX pin to logic 1.

•All the control signals are given out by the microprocessor chip itself.

There is a single microprocessor in the minimum mode system.

The remaining components in the system are latches, transreceivers, clock generator, memory and I/O devices.

Chip selection logic may be required for selecting memory or I/O devices.

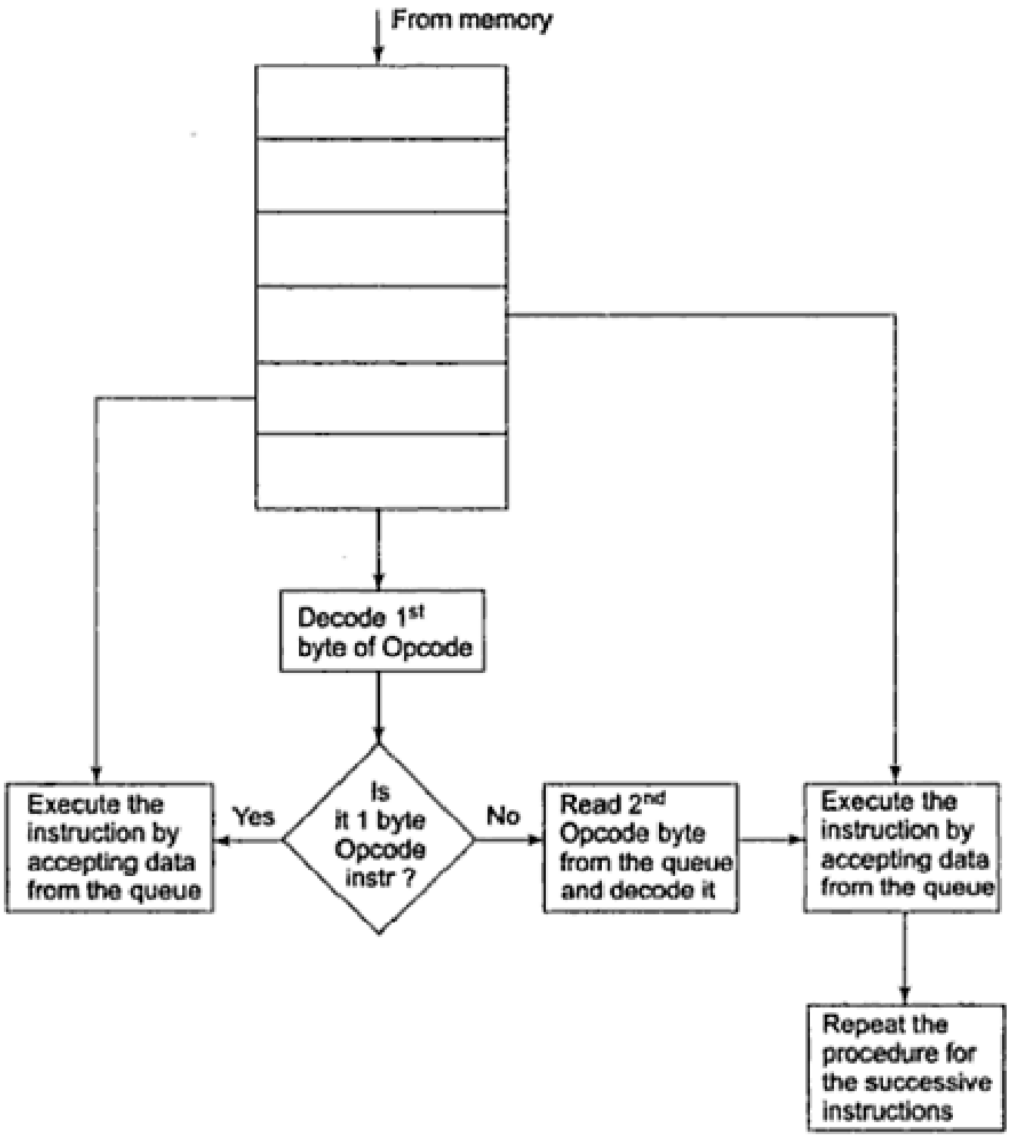
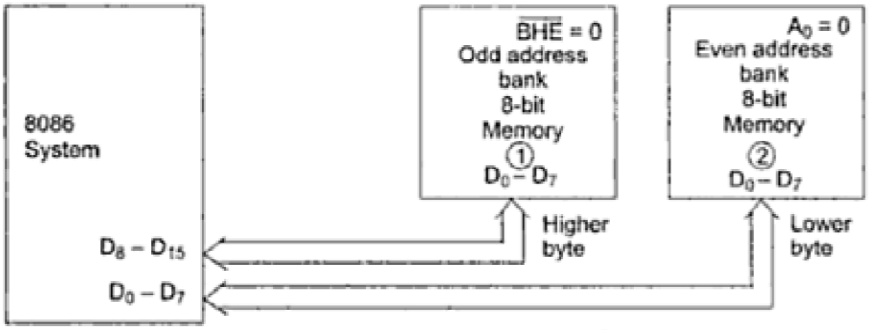


Fig: The Queue operation



Segment address ⟶ 1005H
Offset address ⟶ 5555H
Segment address ⟶ 1005H ⟶ 0001 0000 0000 0101
Shifted by 4 bit positions ⟶ 0001 0000 0000 0101 0000
+
Offset address ⟶ 0101 0101 0101 0101
Physical address ⟶ 0001 0101 0101 1010 0101
1 5 5 A 5

Fig: Memory segmentation

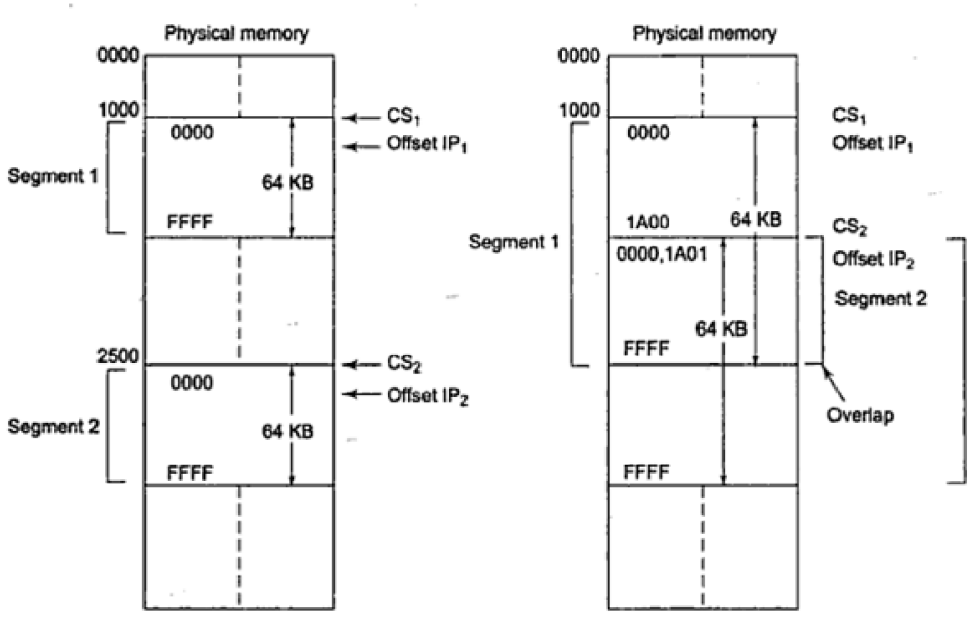


Fig: a) Non overlapping segments b) Overlapping segments

•Latches are used for separating the valid address from the multiplexed address/data signals and are controlled by the ALE signal generated by 8086.

•Transreceivers are the bidirectional buffers and some times they are called as data amplifiers. They are required to separate the valid data from the time multiplexed address/data signals.
•The DEN signal indicates the direction of data, i.e. from or to the processor.
The system contains memory for the monitor and users program storage.
•EPROM are used for monitor storage, while RAM for users program storage.
A system may contain I/O devices.

•In the Timing Diagram, the opcode fetch and read cycles are similar.
It can be categorized in two parts, the first is the timing diagram for read cycle and the second is the timing diagram for write cycle.
•The read cycle begins in T1 with the assertion of address latch enable (ALE) signal and also M / IO signal.
•The BHE and A0 signals address low, high or both bytes.
From T1 to T4 , the M/IO signal indicates a memory or I/O operation.
•At T2, the address is removed from the local bus and is sent to the output. The bus is then tristated.
The read (RD) control signal is also activated in T2.
•The read (RD) signal causes the address device to enable its data bus drivers. After RD goes low, the valid data is available on the data bus.
•The addressed device will drive the READY line high.
•A write cycle also begins with the assertion of ALE and the emission of the address.
The M/IO signal is again asserted to indicate a memory or I/O operation.
•The BHE and A0 signals are used to select the proper byte or bytes of memory or I/O word to be read or write.
•The M/IO, RD and WR signals indicate the type of data transfer.
Hold Response sequence: The HOLD pin is checked at leading edge of each clock pulse.
If it is received active, the CPU activates HLDA in the next clock cycle and for succeeding bus cycles, the bus will be given to another requesting master.
•The control of the bus is not regained by the processor until the requesting master does not drop the HOLD pin low.
When the request is dropped by the requesting master, the HLDA is dropped by the processor at the trailing edge of the next clock.
Minimum Mode Interface
•When the Minimum mode operation is selected, the 8086 provides all control signals needed to implement the memory and I/O interface.
The minimum mode signal can be divided into the following basic groups: address/data bus, status, control, interrupt and DMA.

$M/\overline{IO}$	$\overline{RD}$	$\overline{DEN}$	Transfer Type
0	0	1	I/O read
0	1	0	I/O write
1	0	1	Memory read
1	1	0	Memory write

Address/Data Bus: These lines serve two functions.
Address bus is 20 bits long and it consists of signal lines A0 through A19.

A19 represents the MSB and A0 LSB.
This gives the 8086 a 1Mbyte memory address space.
It has an independent I/O address space which is 64K bytes in length.

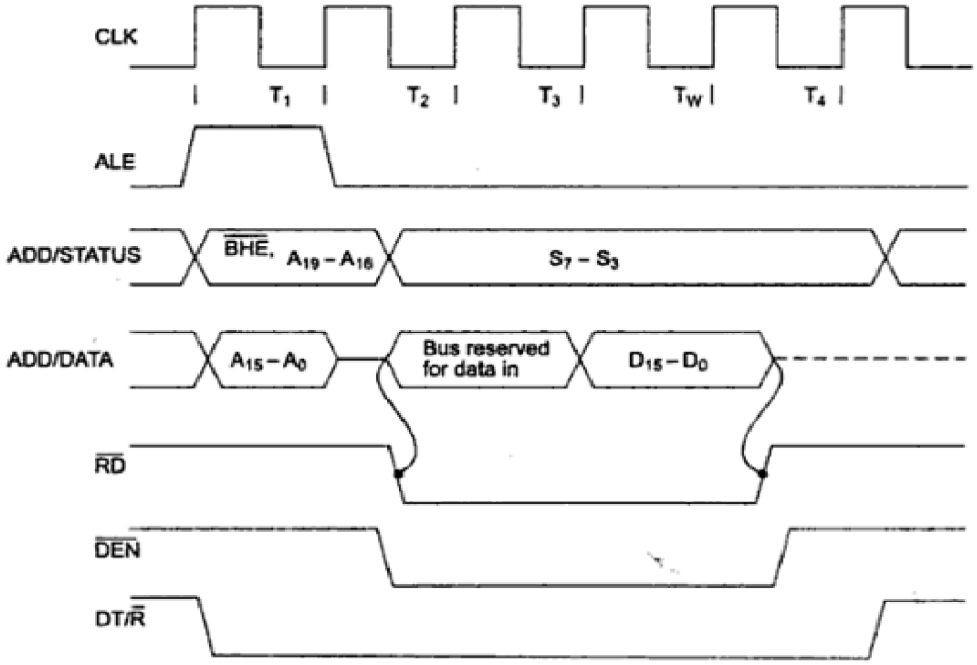
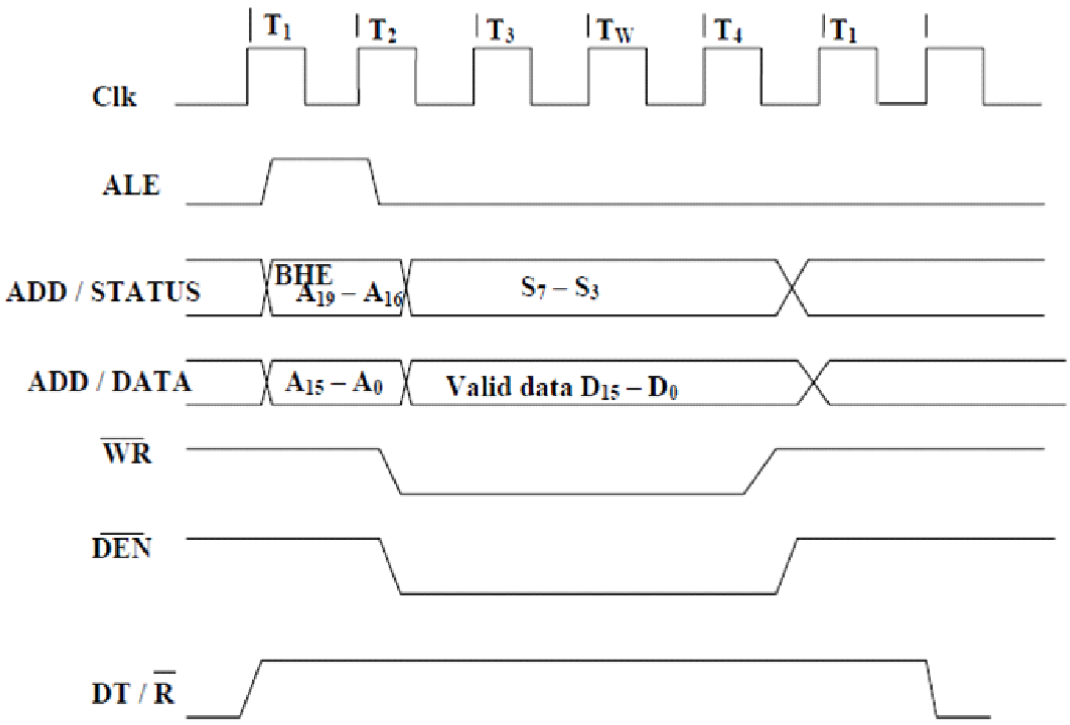


Fig: Read Cycle Timing diagram for Minimum mode

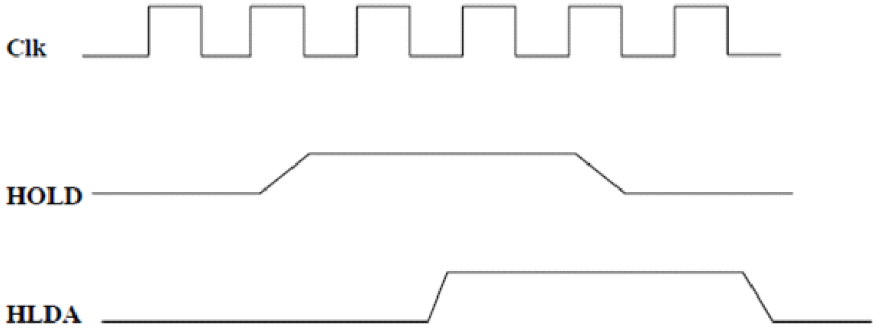
The 16 data bus lines D0 through D15 are actually multiplexed with address lines A0 through A15 respectively.
•Using Data bus system carries read/write data for memory, input/output data for I/O devices, and interrupt type codes from an interrupt controller.

•**Status signal:** The four most significant address lines A19 through A16 are also multiplexed but in this case with status signals S6 through S3.
These status bits are output on the bus at the same time that data are transferred over the other bus lines.
•Bit S4 and S3 together from a 2 bit binary code that identifies which of the 8086 internal segment registers are used to generate the physical address that was output on the address bus.
•Code S4S3 = 00 identifies a register known as **extra segment register** as the source of the segment address.
•Status line S5 reflects the status of another internal characteristic of the 8086.

The last status bit S6 is always at the logic 0 level.



Write Cycle Timing Diagram for Minimum Mode



Bus Request and Bus Grant Timings in Minimum Mode System

.

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

Fig: Memory segment status codes

Control Signals:

These signals control functions such as when the bus is to carry a valid address in which direction data are to be transferred over the bus, when valid write data are on the bus and when to put read data on the system bus.

•ALE is a pulse to logic 1 that signals external circuitry when a valid address word is on the bus.

• BHE is Bus high enable.

Logic 0 on this used as a memory enable signal for the most significant byte half of the data bus D8 through D1.

These lines also serves a second function, which is as the S_7 status line.

•Using the M/IO and DT/R lines, the 8086 signals which type of bus cycle is in progress and in which direction data are to be transferred over the bus.

•The logic level of M/IO tells external circuitry whether a memory or I/O transfer is taking place over the bus.

Logic 1 at this output signals a memory operation and logic 0 an I/O operation.

•The direction of data transfer over the bus is signaled by the logic level output at DT/R.

•The signal read RD and write WR indicates that a read bus cycle or a write bus cycle is in progress.

The 8086 switches WR to logic 0 to signal external device that valid write or output data are on the bus.

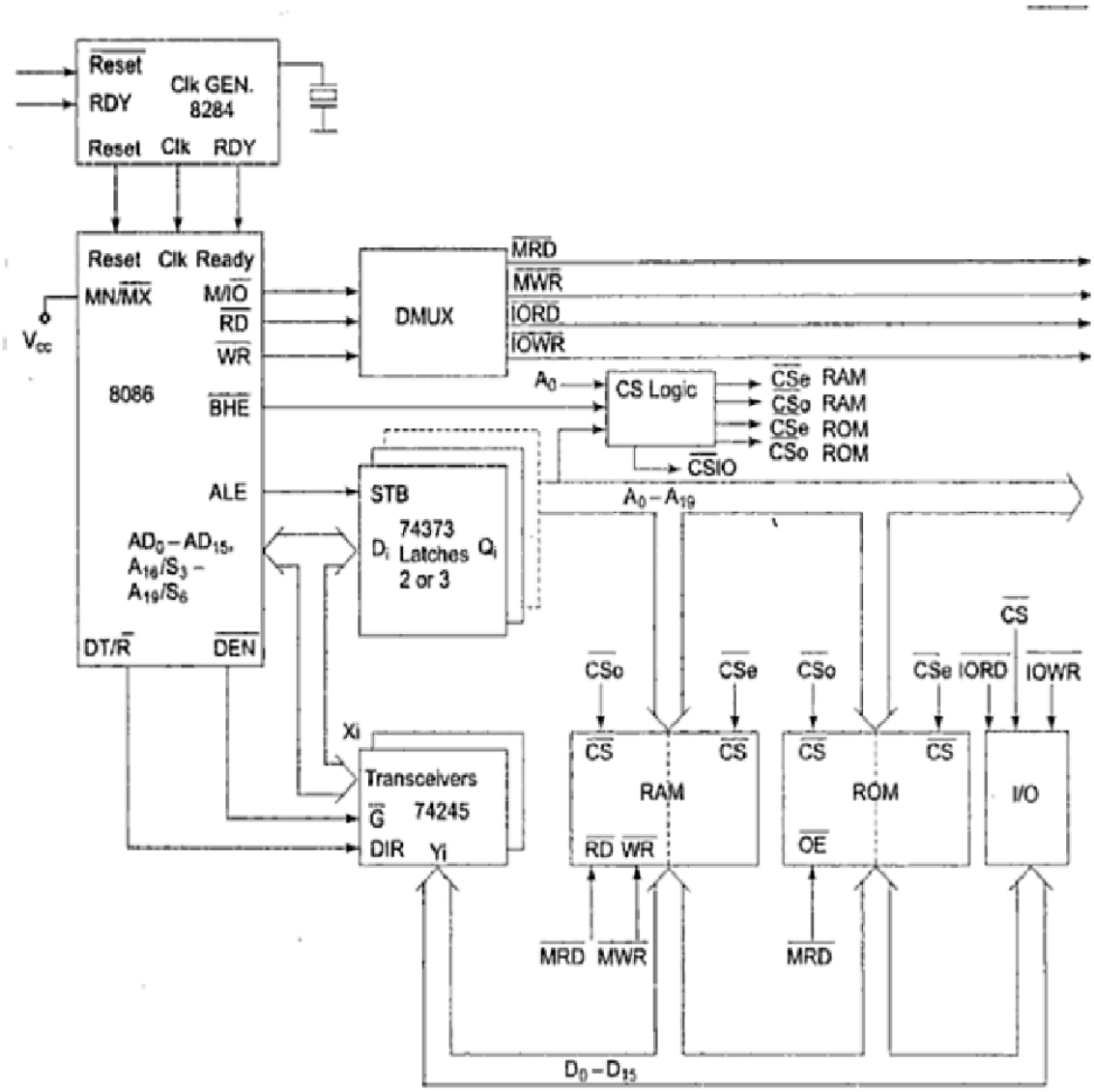


Fig: Minimum mode 8086 system

• RD indicates that the 8086 is performing a read of data of the bus.

During read operations, one other control signal is supplied i.e DEN (data enable) and it signals external devices when they should put data on the bus.

•There is one control signal that is involved with the memory and I/O interface. This is the READY signal.

•READY signal is used to insert wait states into the bus cycle such that it is extended by a number of clock periods.

This signal is provided by an external clock generator

•**Interrupt signals**: The key interrupt interface signals are interrupt request (INTR) and interrupt acknowledge (INTA).

•INTR is an input to the 8086 that can be used by an external device to signal that it need to be serviced.

•Logic 1 at INTR represents an active interrupt request.
When an interrupt request has been recognized by the 8086, it indicates this fact to external circuit with pulse to logic 0 at the INTA output.
•The TEST input is also related to the external interrupt interface.
Execution of a WAIT instruction causes the 8086 to check the logic level at the TEST input.
•If the logic 1 is found, the MPU suspend operation and goes into the idle state.
The 8086 no longer executes instructions, instead it repeatedly checks the logic level of the TEST input waiting for its transition back to logic 0.
•There are two more inputs in the interrupt interface: the nonmaskable interrupt NMI and the reset interrupt RESET.
•On the 0-to-1 transition of NMI control is passed to a nonmaskable interrupt service routine.
The RESET input is used to provide a hardware reset for the 8086.
•**DMA Interface signals**:The direct memory access DMA interface of the 8086 minimum mode consist of the HOLD and HLDA signals.
•When an external device wants to take control of the system bus, it signals to the 8086 by switching HOLD to the logic 1 level.
At the completion of the current bus cycle, the 8086 enters the hold state.
In the hold state, signal lines AD0 through AD15, A16/S3 through A19/S6, BHE, M/IO, DT/R, RD, WR, DEN and INTR are all in the high Z state.
The 8086 signals external device by switching its HLDA output to logic 1 level.
6)Maximum Mode 8086 System : In the maximum mode, the 8086 is operated by strapping the MN/MX pin to ground.
•In this mode, the processor derives the status signal S2, S1, S0.
Another chip called bus controller derives the control signal using this status information.
•In the maximum mode, there may be more than one microprocessor in the system configuration.
•The basic function of the bus controller chip IC8288, is to derive control signals like RD and WR,DEN, DT/R, ALE etc. using the information by the processor on the status lines.
•The bus controller chip has input lines S2, S1, S0 and CLK.
These inputs to 8288 are driven by CPU.
•It derives the outputs ALE, DEN, DT/R, MRDC, MWTC, AMWC, IORC, IOWC and AIOWC.
The AEN, IOB and CEN pins are specially useful for multiprocessor systems.
•AEN and IOB are generally grounded. CEN pin is usually tied to +5V.
The significance of the MCE/PDEN output depends upon the status of the IOB pin.
•INTA pin used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device.
•IORC, IOWC are I/O read command and I/O write command signals respectively.
These signals enable an IO interface to read or write the data from or to the address port.
•The MRDC, MWTC are memory read command and memory write command signals respectively and may be used as memory read or write signals.
•For both of these write command signals, the advanced signals namely AIOWC and AMWTC are available.
R0, S1, S2 are set at the beginning of bus cycle.

8288 bus controller will output a pulse as on the ALE and apply a required signal to its DT / R pin during T1.

•In T2, 8288 will set DEN=1 thus enabling transceivers, and for an input it will activate MRDC or IORC.

For an output, the AMWC or AIOWC is activated from T2 to T4 and MWTC or IOWC is activated from T3 to T4.

•The status bit S0 to S2 remains active until T3 and become passive during T3 and T4**.**

•If reader input is not activated before T3, wait state will be inserted between T3 and T4.

•**Timings for RQ/ GT Signals:**

The request/grant response sequence contains a series of three pulses.

The request/grant pins are checked at each rising pulse of clock input.

When a request is detected and if the condition for HOLD request are satisfied, the processor issues a grant pulse over the RQ/GT pin immediately during T4 (current) or T1 (next) state.

Maximum Mode Interface

•When the 8086 is set for the maximum-mode configuration, it provides signals for implementing a multiprocessor / coprocessor system environment.

•By multiprocessor environment we mean that one microprocessor exists in the system and that each processor is executing its own program.

•Usually in this type of system environment, there are some system resources that are common to all processors.

•They are called as ***global resources***. There are also other resources that are assigned to specific processors known as ***local or private resources***.

•Coprocessor also means that there is a second processor in the system.

In this two processors do not access the bus at the same time.

When the requesting master receives grant pulse, it accepts the control of the bus, it sends a release pulse to the processor using RQ/GT pin.

•**8288 Bus Controller – Bus Command and Control Signals**:

8086 does not directly provide all the signals that are required to control the memory, I/O and interrupt interfaces.

•Specially the WR, M/IO, DT/R, DEN, ALE and INTA, signals are no longer produced by the 8086.

Instead it outputs three status signals S0, S1, S2 prior to the initiation of each bus cycle.

This 3- bit bus status code identifies which type of bus cycle is to follow.

S2S1S0 are input to the external bus controller device, the bus controller generates the appropriately timed command and control signals.

The output of 8289 are bus arbitration signals: *Bus busy* (BUSY), *common bus request* (CBRQ), *bus priority out* (BPRO), *bus priority in* (BPRN), *bus request* (BREQ) and *bus clock* (BCLK).

•They correspond to the bus exchange signals of the Multibus and are used to lock other processor off the system bus during the execution of an instruction by the 8086.

QS_1	QS_0	Queue Status
0 (low)	0	No Operation. During the last clock cycle, nothing was taken from the queue.
0	1	First Byte. The byte taken from the queue was the first byte of the instruction.
1 (high)	0	Queue Empty. The queue has been reinitialized as a result of the execution of a transfer instruction.
1	1	Subsequent Byte. The byte taken from the queue was a subsequent byte of the instruction.

•**Queue Status Signals**: Two new signals that are produced by the 8086 in the maximum-mode system are queue status outputs QS0 and QS1.
Together they form a 2-bit queue status code, QS1QS0.
•**Local Bus Control Signal – Request / Grant Signals**: In a maximum mode configuration, the minimum mode HOLD, HLDA interface is also changed.
These two are replaced by request/grant lines RQ/ GT0 and RQ/ GT1, respectively.

7) Differences between 8086 and 8088 Microprocessor
The Intel 8088 microprocessor is a variant of the Intel **8086**.
Introduced on July 1, 1979, the 8088 had an **8**-bit external data bus instead of the **16**-bit bus of the **8086**.
The **16**-bit registers and the one megabyte address range were unchanged.
Comparison of 8086 v/s 8088

8086	**8088**
The instruction Queue is 6 byte long.	The instruction Queue is 4 byte long.
In 8086, memory divides into two banks -even or lower bank -odd or higher bank	The memory in 8088 does not divide into two banks.
The data bus of 8086 is 16-bit wide	The data bus of 8088 is 8-bit wide.
It has BHE (bar) signal on pin no. 34 &there is no SSO (bar) signal.	It does not have BHE (bar) signal on pin no. 34 &has only SSO (bar) signal. It has no S7 pin.

Control pin in 8086 is M/IO (bar).	Control pin in 8088 is IO/M(bar).
In 8086, all address &data Buses are multiplexed.	In 8088, address bus; AD_7- AD_0 buses are multiplexed.
Three clock speed: 5, 8, 10 MHz	Two clock speed: 5, 8 MHz

Status Inputs			CPU Cycles
$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$	
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	Instruction Fetch
1	0	1	Read Memory
1	1	0	Write Memory
1	1	1	Passive

Bus Status Codes

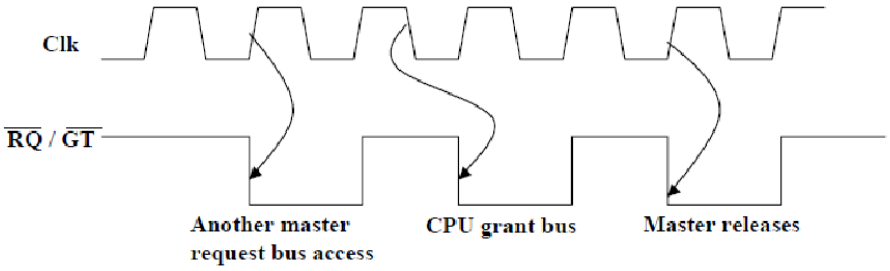
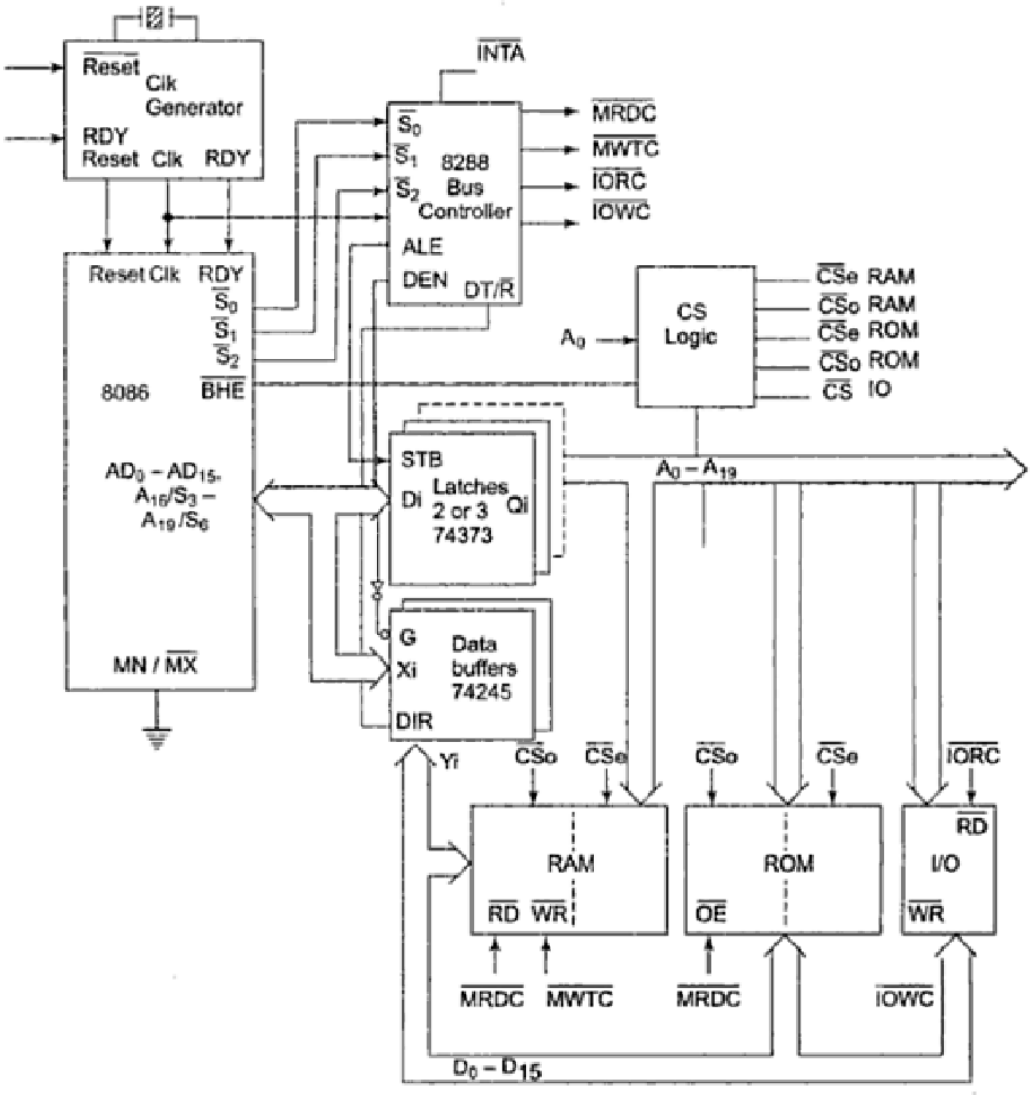


Fig: RQ/GT Timings in Maximum mode

Fig. shows the Maximum mode system



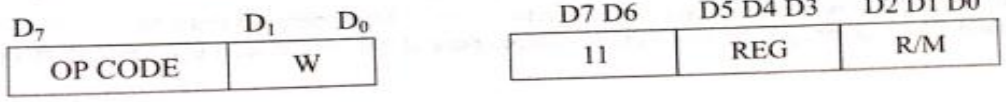
8) Machine Language instruction format

A machine language instruction format has one or more number of fields associated with it. The first field is called as *operation code field* or *opcode field*, which indicates the type of the operation to be performed by the CPU. The instruction format also contains other fields known as *operand fields*. The CPU executes the instruction using the information which reside in these fields.

There are six general formats of instructions in 8086 instruction set. The length of an instruction may vary from one byte to six bytes. The instruction formats are described as follows:

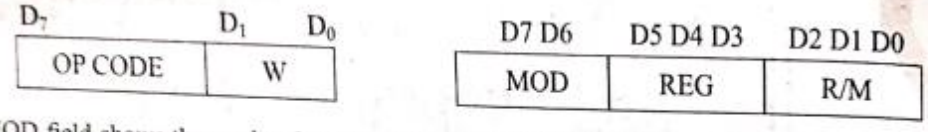
1. One byte Instruction This format is only one byte long and may have the implied data or register operands. The least significant 3-bits of the opcode are used for specifying the register operand, if any. Otherwise, all the 8-bits form an opcode and the operands are implied.

2. Register to Register This format is 2 bytes long. The first byte of the code specifies the operation code and width of the operand specified by w bit. The second byte of the code shows the register operands and R/M field, as shown below.



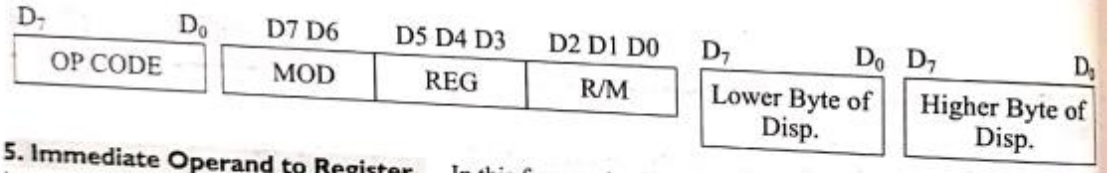
The register represented by the REG field is one of the operands. The R/M field specifies another register or memory location, i.e. the other operand.

3. Register to/from Memory with no Displacement This format is also 2 bytes long and similar to the register to register format except for the MOD field as shown.

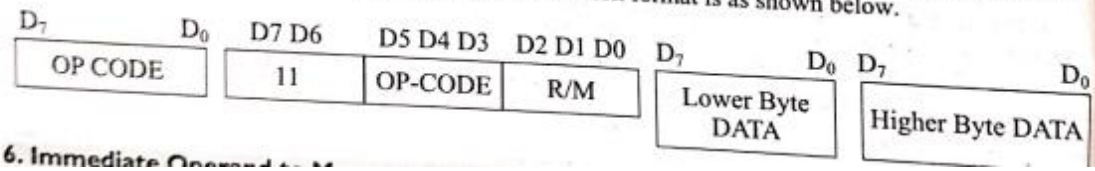


The MOD field shows the mode of addressing. The MOD, R/M, REG and the W fields are decided in Table 2.2.

4. Register to/from Memory with Displacement This type of instruction format contains one or two additional bytes for displacement along with 2-byte the format of the register to/from memory without displacement. The format is as shown below.

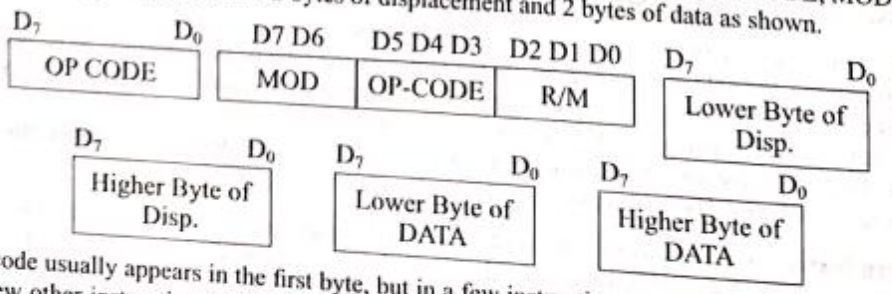


5. Immediate Operand to Register In this format, the first byte as well as the 3-bits from the second byte which are used for REG field in case of register to register format are used for opcode. It also contains one or two bytes of immediate data. The complete instruction format is as shown below.



6. Immediate Operand to M

6. Immediate Operand to Memory with 16-bit Displacement This type of instruction format re
quires 5 or 6 bytes for coding. The first 2 bytes contain the information regarding OPCODE, MOD, and R/M
fields. The remaining 4 bytes contain 2 bytes of displacement and 2 bytes of data as shown.



The opcode usually appears in the first byte, but in a few instructions, a register destination is in the fir
byte and few other instructions may have their 3-bits of opcode in the second byte. The opcodes have th
single bit indicators. Their definitions and significances are given as follows:

W-bit This indicates whether the instruction is to operate over an 8-bit or 16-bit data/operands. If W bit is 0
the operand is of 8-bits and if W is 1, the operand is of 16-bits.

D-bit This is valid in case of double operand instructions. One of the operands must be a register specified b
the REG field. The register specified by REG is source operand if D = 0, else, it is a destination operand.

S-bit This bit is called as sign extension bit. The S bit is used along with W-bit to show the type of the operation. For, example

8-bit operation with 8-bit immediate operand is indicated by S = 0, W = 0;
16-bit operation with 16-bit immediate operand is indicated by S = 0, W = 1 and
16-bit operation with a sign extended immediate data is given by S = 1, W = 1

V-bit This is used in case of shift and rotate instructions. This bit is set to 0, if shift count is 1 and is set to 1, if CL contains the shift count.

Z-bit This bit is used by REP instruction to control the loop. If Z bit is equal to 1, the instruction with REP prefix is executed until the zero flag matches the Z bit.

The REG code of the different registers (either as source or destination operands) in the opcode byte are assigned with binary codes. The segment registers are only 4 in number hence 2 binary bits will be sufficient to code them. The other registers are 8 in number, so at least 3-bits will be required for coding them. To allow the use of 16-bit registers as two 8-bit registers they are coded with W bit as shown in Table 2.1.

Table 2.1 *Assignment of Codes with Different Registers*

W	*Register Address (code)*	*Registers*	*Segment 2 bit bit Register (code)*	*Segment Register*
0	000	AL		
0	001	CL	00	ES
0	010	DL	01	CS
0	011	BL	10	SS
0	100	AH	11	DS
0	101	CH		
0	110	DH		
0	111	BH		
1	000	AX		
1	001	CX		
1	010	DX		
1	011	BX		
1	100	SP		
1	101	BP		
1	110	SI		
1	111	DI		

Activat
Go to PC

Please note that usually all the addressing modes have DS as the default data segment. However, the addressing modes using BP and SP have SS as the default segment register.

To find out the MOD and R/M fields of a particular instruction, one should first decide the addressing mode of the instruction. The addressing mode depends upon the operands and suggests how the effective address may be computed for locating the operand, if it lies in memory. The different addressing modes of the 8086 instructions are listed in Table 2.2. The R/M column and addressing mode row element specifies the R/M field, while the addressing mode column specifies the MOD field.

Evolution of microprocessors, 8085 microprocessor- basic architecture only, 8086 Microprocessor - Architecture and signals, Physical Memory organisation, Minimum and maximum mode of 8086 system and timings. Comparison of 8086 and 8088.Machine language instruction format

1) Evolution of microprocessors

Microprocessors have undergone significant evolution over the past four decades. Development of some of the microprocessors can be given as follows.

- 4-bit Microprocessors

The first microprocessor was introduced in 1971 by Intel Corp.
It was named Intel 4004 as it was a 4 bit processor.
It could perform simple arithmetic and logic operations such as addition, subtraction, boolean AND and boolean OR.
The enhanced version of 4004 was Intel 4040. Some other 4 bit processors are International's PPS4 and Thoshiba's T3472.

- 8-bit Microprocessors

The first 8 bit microprocessor which could perform arithmetic and logic operations on 8 bit words was introduced in 1973 again by Intel.
This was Intel 8008 and was later followed by an improved version, Intel 8088. Some other 8 bit processors are Zilog-80 and Motorola M6800.

- 16-bit Microprocessors

The 8-bit processors were followed by 16 bit processors. They are Intel 8086 and 80286.

- 32-bit Microprocessors

The 32 bit microprocessors were introduced by several companies but the most popular one is Intel 80386.

- Pentium Series

Instead of 80586, Intel came out with a new processor namely Pentium processor.
Its performance is closer to RISC performance.
Pentium was followed by Pentium Pro CPU.
The low cost version of Pentium II is celeron.
The Pentium III provided high performance floating point operations for certain types of computations by using the SIMD extensions to the instruction set.
Pentium III was faster than high-end RISC CPUs.
Pentium IV could not execute code faster than the Pentium III when running at the same clock frequency.

First Generation Microprocessors

- The first generation microprocessors were introduced in the year 1971-1972.
- The instructions of these microprocessors were processed serially.

Second Generation Microprocessors

In the year 1970, small amount of transistors were available on the integrated circuit in the second generation microprocessors.

- Examples: 16-bit arithmetic 7 pipelined instruction processing, MC68000 Motorola microprocessor.
- These processors were introduced in the year 1979, and Intel 8080 processor is another example.

Third Generation Microprocessors: The third generation microprocessors were introduced in the year 1978, as denoted by Intel's 8086 and the Zilog Z8000.

These were 16-bit processors with a performance like mini computers.

Fourth Generation Microprocessors: Leading edge microprocessors like Motorola's 88100 and Intel's 80960CA could issue & retire more than one instruction per clock cycle.

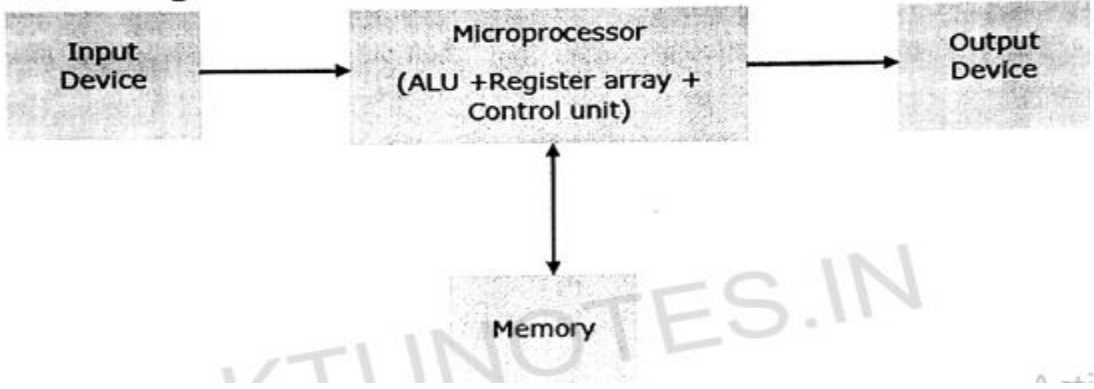
Fifth Generation Microprocessors: Fifth generation microprocessors employed decoupled super scalar processing, and their design soon exceeded 10 million transistors.

In fifth generation, PCs are a low-margin, high volume business conquered by a single microprocessor.

Microprocessor is a controlling unit of a micro-computer, fabricated on a small chip capable of performing ALU (Arithmetic Logical Unit) operations and communicating with the other devices connected to it.

Microprocessor consists of an ALU, register array, and a control unit. ALU performs arithmetical and logical operations on the data received from the memory or an input device. Register array consists of registers identified by letters like B, C, D, E, H, L and accumulator. The control unit controls the flow of data and instructions within the computer.

Block Diagram of a Basic Microcomputer



KTUNOTES.IN

2)8085 architecture

8085 is pronounced as "eighty-eighty-five" microprocessor. It is an 8-bit microprocessor designed by Intel in 1977 using NMOS technology.

It has the following configuration –

- 8-bit data bus
- 16-bit address bus, which can address upto 64KB
- A 16-bit program counter
- A 16-bit stack pointer
- Six 8-bit registers arranged in pairs: BC, DE, HL
- Requires +5V supply to operate at 3.2 MHZ single phase clock

It is used in washing machines, microwave ovens, mobile phones, etc.

8085 Microprocessor – Functional Units

8085 consists of the following functional units –

Accumulator

It is an 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal data bus & ALU.

Arithmetic and logic unit

As the name suggests, it performs arithmetic and logical operations like Addition, Subtraction, AND, OR, etc. on 8-bit data.

General purpose register

There are 6 general purpose registers in 8085 processor, i.e. B, C, D, E, H & L. Each register can hold 8-bit data.

These registers can work in pair to hold 16-bit data and their pairing combination is like B-C, D-E & H-L.

Program counter

It is a 16-bit register used to store the memory address location of the next instruction to be executed. Microprocessor increments the program whenever an instruction is being executed, so that the program counter points to the memory address of the next instruction that is going to be executed.

KTUNOTES.IN
Activate
Go to PC s
Activate
Go to PC s

Stack pointer

It is also a 16-bit register works like stack, which is always incremented/decremented by 2 during push & pop operations.

Temporary register

It is an 8-bit register, which holds the temporary data of arithmetic and logical operations.

Flag register

It is an 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending upon the result stored in the accumulator.

These are the set of 5 flip-flops –

- Sign (S)
- Zero (Z)
- Auxiliary Carry (AC)
- Parity (P)
- Carry (C)

Its bit position is shown in the following table –

KTUNOTES.IN

D7	D6	D5	D4	D3	D2	D1	D0
S	Z		AC		P		CY

Instruction register and decoder

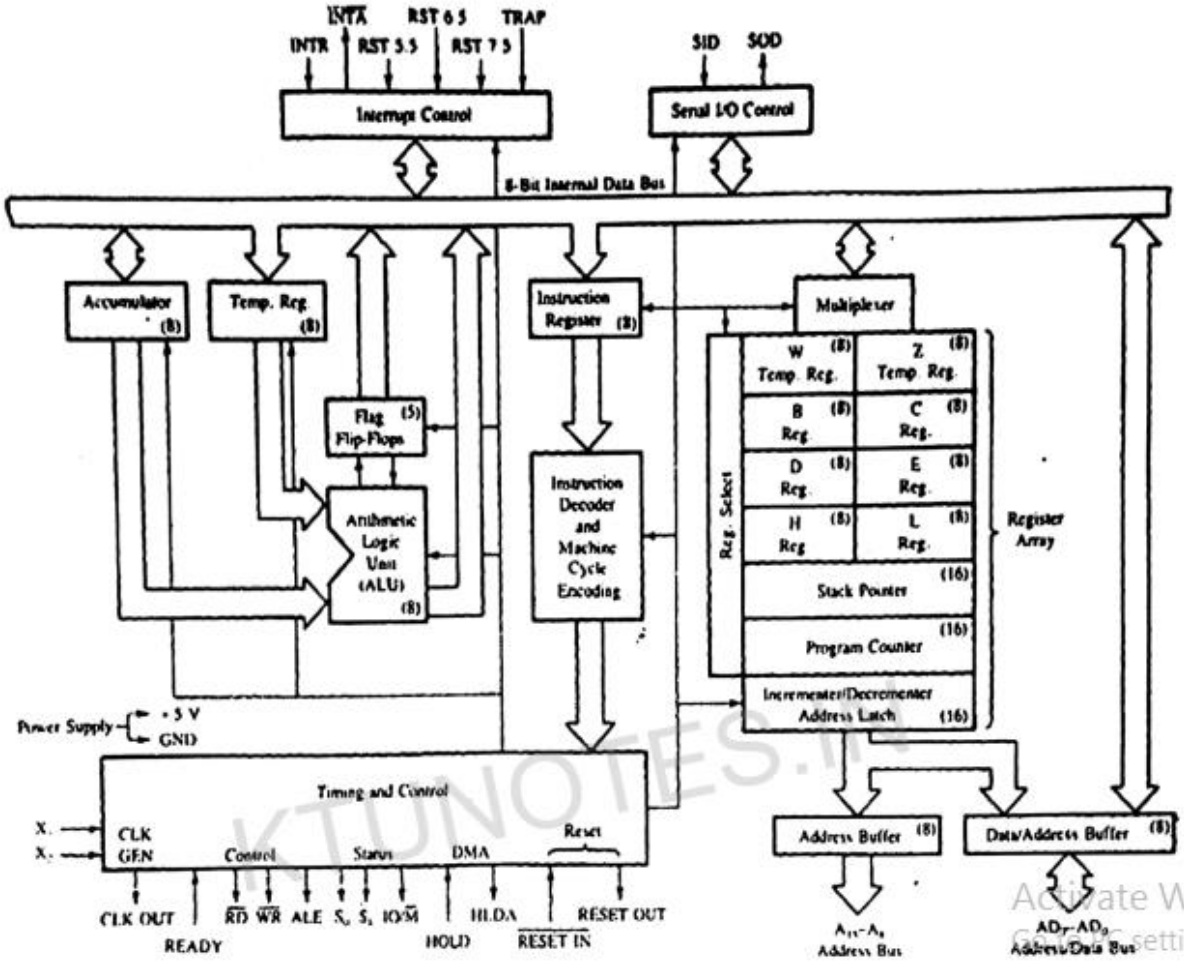
It is an 8-bit register. When an instruction is fetched from memory then it is stored in the Instruction register. Instruction decoder decodes the information present in the Instruction register.

Timing and control unit

It provides timing and control signal to the microprocessor to perform operations. Following are the timing and control signals, which control external and internal circuits –

- Control Signals: READY, RD', WR', ALE
- Status Signals: S0, S1, IO/M'

Figure below shows, 8085 architecture



- DMA Signals: HOLD, HLDA
- RESET Signals: RESET IN, RESET OUT

Interrupt control

As the name suggests it controls the interrupts during a process. When a microprocessor is executing a main program and whenever an interrupt occurs, the microprocessor shifts the control from the main program to process the incoming request. After the request is completed, the control goes back to the main program.

There are 5 interrupt signals in 8085 microprocessor: INTR, RST 7.5, RST 6.5, RST 5.5, TRAP.

Serial Input/output control

It controls the serial data communication by using these two instructions: SID (Serial input data) and SOD (Serial output data).

Address buffer and address-data buffer

The content stored in the stack pointer and program counter is loaded into the address buffer and address-data buffer to communicate with the CPU. The memory and I/O chips are connected to these buses; the CPU can exchange the desired data with the memory and I/O chips.

Address bus and data bus

Data bus carries the data to be stored. It is bidirectional, whereas address bus carries the location to where it should be stored and it is unidirectional. It is used to transfer the data & Address I/O devices.

The pins of a 8085 microprocessor can be classified into seven groups –

Address bus

A15-A8, it carries the most significant 8-bits of memory/IO address.

Data bus

AD7-AD0, it carries the least significant 8-bit address and data bus.

Control and status signals

These signals are used to identify the nature of operation. There are 3 control signal and 3 status signals.

Three control signals are RD, WR & ALE.

- **RD** – This signal indicates that the selected IO or memory device is to be read and is ready for accepting data available on the data bus.
- **WR** – This signal indicates that the data on the data bus is to be written into a selected memory or IO location.
- **ALE** – It is a positive going pulse generated when a new operation is started by the microprocessor. When the pulse goes high, it indicates address. When the pulse goes down it indicates data.

Three status signals are IO/M, S0 & S1.

IO/M

This signal is used to differentiate between IO and Memory operations, i.e. when it is high indicates IO operation and when it is low then it indicates memory operation.

S1 & S0

These signals are used to identify the type of current operation.

Power supply

There are 2 power supply signals – VCC & VSS. VCC indicates +5v power supply and VSS indicates ground signal.

KTUNOTES.IN

Clock signals

There are 3 clock signals, i.e. X1, X2, CLK OUT.

- **X1, X2** – A crystal (RC, LC N/W) is connected at these two pins and is used to set frequency of the internal clock generator. This frequency is internally divided by 2.
- **CLK OUT** – This signal is used as the system clock for devices connected with the microprocessor.

Interrupts & externally initiated signals

Interrupts are the signals generated by external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR. We will discuss interrupts in detail in interrupts section.

- **Vector interrupt** – In this type of interrupt, the interrupt address is known to the processor. **For example:** RST7.5, RST6.5, RST5.5, TRAP.
- **Non-Vector interrupt** – In this type of interrupt, the interrupt address is not known to the processor so, the interrupt address needs to be sent externally by the device to perform interrupts. **For example:** INTR.
- **Maskable interrupt** – In this type of interrupt, we can disable the interrupt by writing some instructions into the program. **For example:**RST7.5, RST6.5, RST5.5.
- **Non-Maskable interrupt** – In this type of interrupt, we cannot disable the interrupt by writing some instructions into the program. **For example:** TRAP.
- **Software interrupt** – In this type of interrupt, the programmer has to add the instructions into the program to execute the interrupt. There are 8 software interrupts in 8085, i.e. RST0, RST1, RST2, RST3, RST4, RST5, RST6, and RST7.
- **Hardware interrupt** – There are 5 interrupt pins in 8085 used as hardware interrupts, i.e. TRAP, RST7.5, RST6.5, RST5.5, INTA.

KTUNOTES.IN

Note – NTA is not an interrupt, it is used by the microprocessor for sending acknowledgement. TRAP has the highest priority, then RST7.5 and so on.

Interrupt Service Routine (ISR)

A small program or a routine that when executed, services the corresponding interrupting source is called an ISR.

Activ
Go to

TRAP

It is a non-maskable interrupt, having the highest priority among all interrupts. Bydefault, it is enabled until it gets acknowledged. In case of failure, it executes as ISR and sends the data to backup memory. This interrupt transfers the control to the location 0024H.

RST7.5

It is a maskable interrupt, having the second highest priority among all interrupts. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 003CH address.

RST 6.5

It is a maskable interrupt, having the third highest priority among all interrupts. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 0034H address.

RST 5.5

It is a maskable interrupt. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 002CH address.

INTR

It is a maskable interrupt, having the lowest priority among all interrupts. It can be disabled by resetting the microprocessor.

When **INTR signal goes high**, the following events can occur –

- The microprocessor checks the status of INTR signal during the execution of each instruction.
- When the INTR signal is high, then the microprocessor completes its current instruction and sends active low interrupt acknowledge signal.
- When instructions are received, then the microprocessor saves the address of the next instruction on stack and executes the received instruction.

KTUNOTES.IN

3) 8086 Microprocessor -Architecture and signals

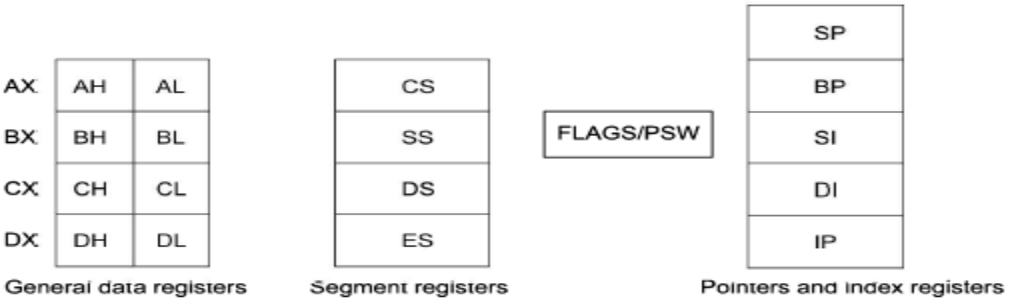


Fig: Register Organisation of 8086

Internal Registers of 8086

Registers of 8086 are classified as General Purpose registers and Special purpose registers**.**

These are 16 bit registers. The general purpose registers are used as 8 bit or 16 bit registers.

•There are four groups of the user accessible internal registers.
They are the instruction pointer, four data registers, four pointer and index register, four segment registers.
•The 8086 has a total of fourteen 16-bit registers including a 16 bit register called the **status register**, with 9 of bits implemented for status and control flags.
There are four different 64 KB segments for instructions, stack, data and extra data.
Four segment registers of 8086 are:
•**Code segment** (CS) is a 16-bit register containing address of 64 KB segment with processor instructions.
The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register.
•**Stack segment** (SS) is a 16-bit register containing address of 64KB segment with program stack.
The processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment.
•**Data segment** (DS) is a 16-bit register containing address of 64KB segment with program data.
The processor assumes that all data referenced by AX, BX, CX, DX and index register (SI, DI) is located in the data segment.
•**Accumulator** register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX.

AL in this case contains the low-order byte of the word, and AH contains the high-order byte.

•**Base** register consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX.
BL in this case contains the low-order byte of the word, and BH contains the high-order byte.
BX register usually contains a data pointer.

•**Count** register consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX.
CL register contains the low-order byte of the word, and CH contains the high-order byte.
Count register can be used as a counter in string manipulation.
•**Data** register consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX.
DL register contains the low-order byte of the word, and DH contains the high-order byte.
Data register can be used as a port number in I/O operations.
In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.
•**The following registers are both general and index registers:**
•**Stack Pointer** (SP) is a 16-bit register pointing to program stack.
•**Base Pointer** (BP) is a 16-bit register pointing to data in stack segment.
BP register is usually used for based, based indexed or register indirect addressing.
•**Source Index** (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data address in string manipulation instructions.
•**Destination Index** (DI) is a 16-bit register.
DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.
Other registers:
•**Instruction Pointer** (IP) is a 16-bit register.
•**Flags** is a 16-bit register containing 9 one bit flags.
•**Overflow Flag** (OF) - set if the result is too large positive number, or is too small negative number to fit into destination operand.
•**Direction Flag** (DF) - if set then string manipulation instructions will auto-decrement index registers.
•**Interrupt-enable Flag** (IF) - setting this bit enables maskable interrupts.
•**Single-step Flag** (TF) - if set then single-step interrupt will occur after the next instruction.
Sign Flag (SF) - set if the most significant bit of the result is set.
•**Zero Flag** (ZF) - set if the result is zero.
•**Auxiliary carry Flag** (AF) - set if there was a carry from or borrow to bits 0-3 in the AL register.
•**Parity Flag** (PF) - set if parity (the number of "1" bits) in the low-order byte of the result is even.
•**Carry Flag** (CF) - set if there was a carry from or borrow to the most significant bit during last result calculation.

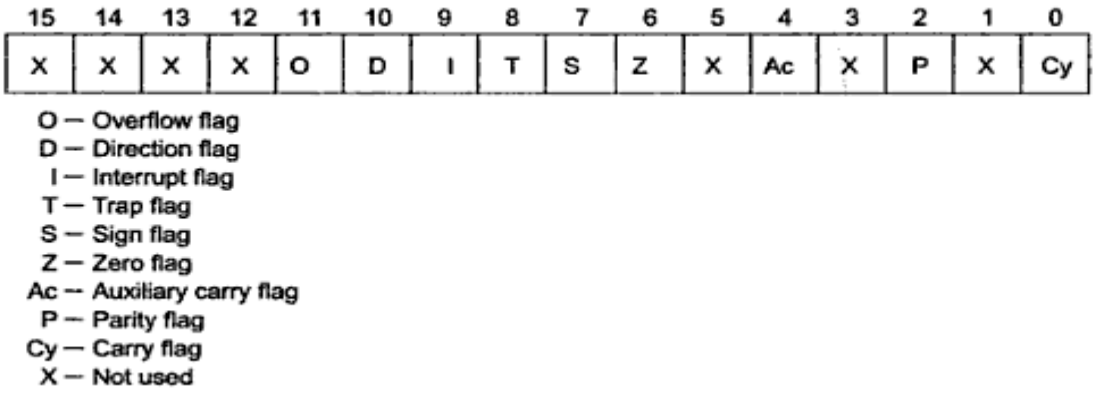


Fig: Flag register of 8086

8086 Microprocessor

•It is a 16-bit µp.

•8086 has a 20 bit address bus can access up to 220 memory locations (1 MB).

•It can support up to 64K I/O ports.

•It provides 14, 16 -bit registers.

•It has multiplexed address and data bus AD0- AD15 and A16 – A19.

•It can prefetches upto 6 instruction bytes from memory and queues them in order to speed up instruction execution.

•It requires +5V power supply.

•A 40 pin dual in line package

Minimum and Maximum Modes: The minimum mode is selected by applying logic 1 to the MN / MX input pin. This is a single microprocessor configuration.

•The maximum mode is selected by applying logic 0 to the MN / MX input pin. This is a multi micro processors configuration.

Internal Architecture of 8086

•8086 has two blocks BIU and EU.

•The BIU performs all bus operations such as instruction fetching, reading and writing operands for memory and calculating the addresses of the memory operands.

The instruction bytes are transferred to the instruction queue.

•EU executes instructions from the instruction system byte queue.

•EU contains Control circuitry, Instruction decoder, ALU, Pointer and Index register, Flag register.

•BIU contains Instruction queue, Segment registers, Instruction pointer, Address adder.

BUS INTERFACE UNIT:

•It provides a full 16 bit bidirectional data bus and 20 bit address bus.

•The bus interface unit is responsible for performing all external bus operations.

Specifically it has the following functions:

•Instruction fetch, Instruction queuing, Operand fetch and storage, Address relocation and Bus control.

The BIU uses a mechanism known as an instruction stream queue to implement a **pipeline architecture.**

•This queue permits prefetch of up to six bytes of instruction code.
Whenever the queue of the BIU is not full, it has room for at least two more bytes and at the same time the EU is not requesting it to read or write operands from memory, the BIU is free to look ahead in the program by prefetching the next sequential instruction.
•These prefetching instructions are held in its FIFO queue.

•The intervals of no bus activity, which may occur between bus cycles are known as **Idle state**.
•The BIU first completes the instruction fetch bus cycle before initiating the operand read / write cycle.
•The BIU also contains a dedicated adder which is used to generate the 20bit physical address that is output on the address bus.
This address is formed by adding an appended 16 bit segment address and a 16 bit offset address.

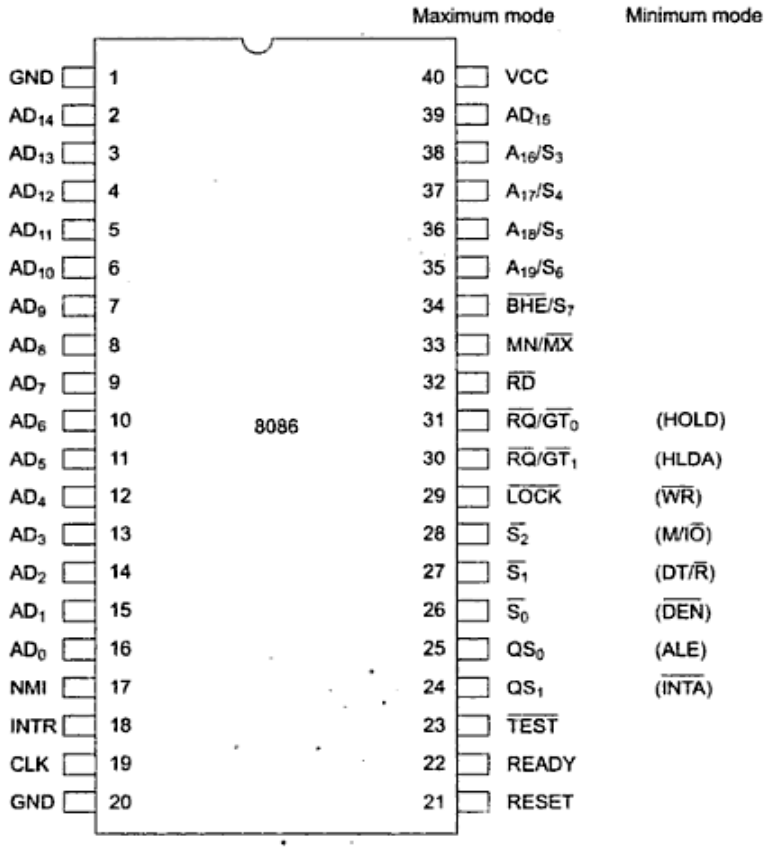
EXECUTION UNIT

The Execution unit is responsible for decoding and executing all instructions.
•The EU extracts instructions from the top of the queue in the BIU, decodes them, generates operands if necessary, passes them to the BIU and requests it to perform the read or write bys cycles to memory or I/O and perform the operation specified by the instruction on the operands.
•During the execution of the instruction, the EU tests the status and control flags and updates them based on the results of executing the instruction.
•If the queue is empty, the EU waits for the next instruction byte to be fetched and shifted to top of the queue.

Signal descriptions of 8086

The Microprocessor 8086 is a 16-bit CPU available in different clock rates and packaged in a 40 pin CERDIP or plastic package.
The pins serve a particular function in minimum mode (single processor mode) and other function in maximum mode configuration (multiprocessor mode).

•**The following signal descriptions are common for both modes.**
•**AD15-AD0**: These are the time multiplexed memory I/O address and data lines.
• Address remains on the lines during T1 state, while the data is available on the data bus during T2, T3, Tw and T4.



Pin Configuration of 8086

•**A19/S6,A18/S5,A17/S4,A16/S3:** These are the time multiplexed address and status lines.
•During T1 these are the most significant address lines for memory operations.
• During memory or I/O operations, status information is available on those lines for T2,T3,Tw and T4.
•The S4 and S3 combinedly indicate which segment register is presently being used for memory accesses as in fig.
The status line S6 is always low.
•The address bit are separated from the status bit using latches controlled by the ALE signal.

•Bit **S4 and S3** together from a 2 bit binary code that identifies which of the 8086 internal segment registers are used to generate the physical address.
•Code S4S3 = 00 identifies a register known as ***extra segment register*** as the source of the segment address.
•**BHE/S7:** The bus high enable is used to indicate the transfer of data over the higher order (D15-D8) data bus.

It goes low for the data transfer over D15-D8 and is used to derive chip selects of odd address memory bank or peripherals.

The status information is available during T2, T3 and T4.

It is low during T1 for the first pulse of the interrupt acknowledges cycle.

•**RDRead:** This signal on low indicates the peripheral that the processor is performing memory or I/O read operation.

RD is active low and shows the state for T2, T3, Tw of any read cycle. The signal remains tristated during the hold acknowledge.

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

S_4	S_3	Indication
0	0	Alternate Data
0	1	Stack
1	0	Code or none
1	1	Data

•**INTR-Interrupt Request**: This is a triggered input.
If any interrupt request is pending, the processor enters the interrupt acknowledge cycle.
•This can be internally masked by resulting the interrupt enable flag.
•**TEST** This input is examined by a 'WAIT' instruction.
If the TEST pin goes low, execution will continue, else the processor remains in an idle state.
The input is synchronized internally during each clock cycle on leading edge of clock.
•**CLK**- Clock Input: The clock input provides the basic timing for processor operation and bus control activity.
Its an asymmetric square wave with 33% duty cycle.
MN/MX: The logic level at this pin decides whether the processor is to operate in either minimum or maximum mode.
•**The following pin functions are for the minimum mode operation of 8086.**
•**M/IO – Memory/IO**: This signal is logically equivalent to S2 in maximum mode.
When it is low, it indicates the CPU is having an I/O operation, and when it is high, it indicates that the CPU is having a memory operation.
This line remains active till final T4 of the current cycle.
It is tristated during local bus "hold acknowledge".
•**INTA Interrupt Acknowledge**: This signal is used as a read strobe for interrupt acknowledge cycles. i.e. when it goes low, the processor has accepted the interrupt.
•**ALE – Address Latch Enable**: This output signal indicates the availability of the valid address on the address/data lines, and is connected to latch enable input of latches.
This signal is active high and is never tristated.
•**DT/R – Data Transmit/Receive**: This output is used to decide the direction of data flow through the transreceivers (bidirectional buffers).
When the processor sends out data, this signal is high and when the processor is receiving data, this signal is low.

•**DEN – Data Enable**: This signal indicates the availability of valid data over the address/data lines.
It is used to enable the transreceivers (bidirectional buffers) to separate the data from the multiplexed address/data signal.
•**HOLD, HLDA- Acknowledge**: When the HOLD line goes high, it indicates to the processor that another master is requesting the bus access.
•If the DMA request is made while the CPU is performing a memory or I/O cycle, it will release the local bus during T4 provided:
1. The request occurs on or before T2 state of the current cycle.
2.The current cycle is not operating over the lower byte of a word.
3.The current cycle is not the first acknowledge of an interrupt acknowledge sequence.

4. A Lock instruction is not being executed.

•**The following pin function are applicable for maximum mode operation of 8086**.
•**S2, S1, S0 – Status Lines**: These are the status lines which reflect the type of operation, being carried out by the processor.

These become activity during T4 of the previous cycle and active during T1 and T2 of the current bus cycles.

•**LOCK** When the CPU is executing a critical instruction which requires the system bus, the LOCK prefix instruction ensures that other processors connected in the system will not gain the control of the bus.
The LOCK signal is activated by the 'LOCK' prefix instruction and remains active until the completion of the next instruction.
•**QS1, QS0 – Queue Status:** These lines give information about the status of the code-prefetch queue.
These are active during the CLK cycle after while the queue operation is performed.

•The 8086 architecture has 6-byte instruction prefetch queue.
Even the largest (6-bytes) instruction can be prefetched from the memory and stored in the prefetch.
•By prefetching the instruction, there is a considerable speeding up in instruction execution in 8086.
This is known as ***instruction pipelining***.
•While the execution unit is busy in executing an instruction, after it is completely decoded, the bus interface unit may be fetching the bytes of the next instruction from memory, depending upon the queue status.

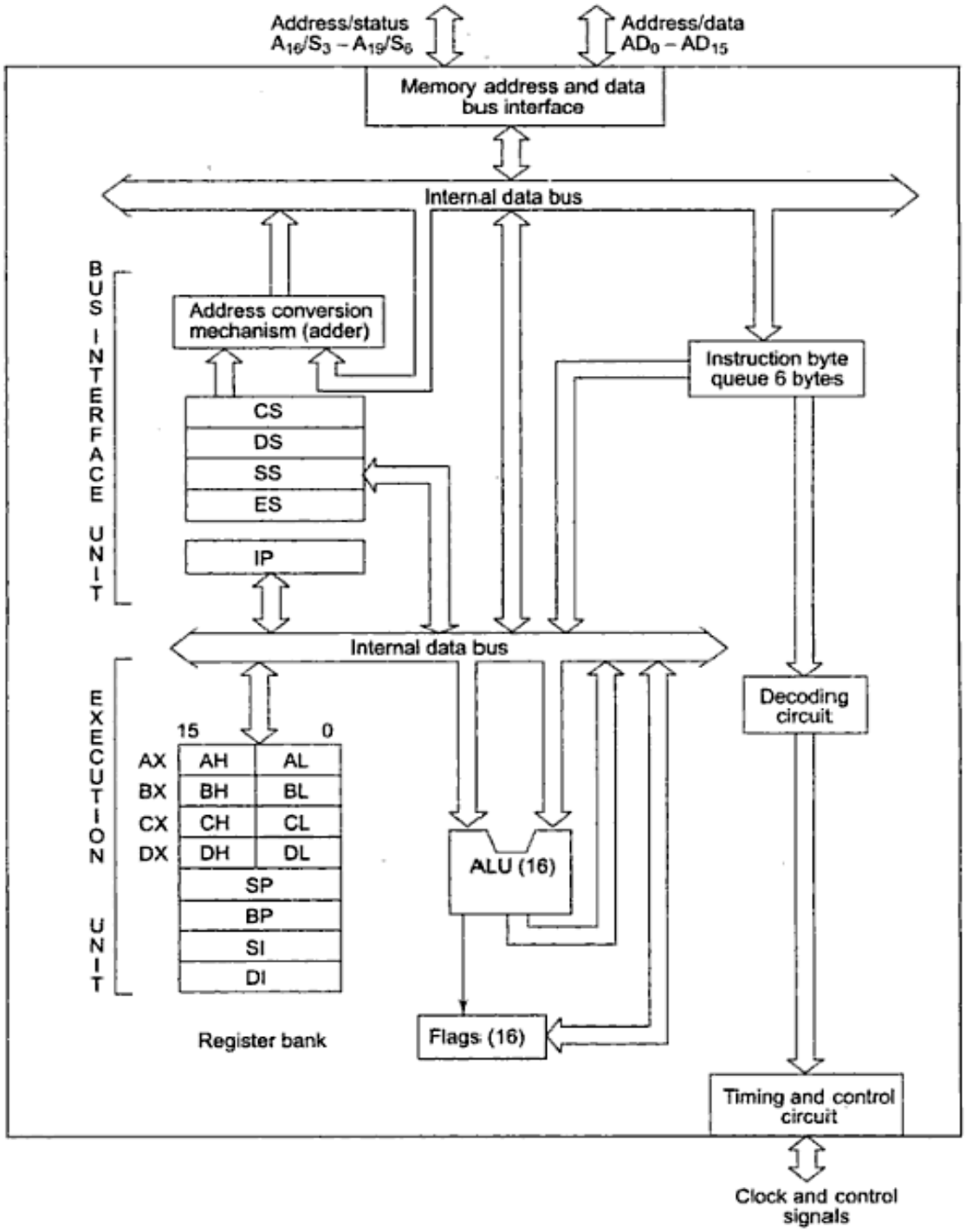


Fig: 8086 architecture

RQ/0GT,RQ/1GT – Request/Grant: These pins are used by the other local bus master in maximum mode, to force the processor to release the local bus at the end of the processor current bus cycle.

•Each of the pin is bidirectional with RQ/GT0 having higher priority than RQ/GT1.

RQ/GT pins have internal pull-up resistors and may be left unconnected.

BHE	A_0	Indication
0	0	Whole word
0	1	Upper byte from or to even address
1	0	Lower byte from or to even address
1	1	None

S_2	S_1	S_0	Indication
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code Access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

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

•Request/Grant sequence is as follows:

1.A pulse of one clock wide from another bus master requests the bus access to 8086.

2.During T4(current) or T1(next) clock cycle, a pulse one clock wide from 8086 to the requesting master, indicates that the 8086 will enter the 'hold acknowledge' state at next cycle.

3.A one clock wide pulse from the another master indicates to the 8086 that the 8086 may regain control of the local bus at the next clock cycle.

Thus each master to master exchange of the local bus is a sequence of 3 pulses.

3)Memory Organisation

Memory •Program, data and stack memories occupy the same memory space.

As the most of the processor instructions use 16-bit pointers the processor can effectively address only 64 KB of memory.

•The CPU uses special segment registers to specify where the code, stack and data 64 KB segments are positioned within 1 MB of memory.

•16-bit pointers and data are stored as: address: low-order byte address+1: high-order byte.
•**Program memory** - Program can be located anywhere in memory.
Jump and call instructions can be used for short jumps within currently selected 64 KB code segment, as well as for far jumps anywhere within 1 MB of memory.
•All conditional jump instructions can be used to jump within approximately +127 to -127 bytes from current instruction.
Data memory - The processor can access data in any one out of 4 available segments, which limits the size of accessible memory to 256 KB.
Accessing data from the Data, Code, Stack or Extra segments can be usually done by prefixing instructions with the DS:, CS:, SS: or ES:
•**Stack memory** can be placed anywhere in memory.
The stack can be located at odd memory addresses, but it is not recommended for performance reasons.
Reserved locations: 0000h - 03FFh are reserved for interrupt vectors. Each interrupt vector is a 32-bit pointer in format segment: offset.
•FFFF0h - FFFFFh - after RESET the processor always starts program execution at the FFFF0h address.
Memory Segmentation: The total memory size is divided into segments of various sizes.

A segment is just an area in memory. The 8086 can access memory with address ranging from 00000 H to FFFFFH.

Each Segment register stores the base address (starting address) of the corresponding segment.

- Segment registers only store the upper 16 bits. 20-bit address is obtained from 16- bit registers.
- The 20-bit address of a byte is called its Physical Address.
- Logical address is in the form of: Base Address : Offset
- Offset is the displacement of the memory location from the starting location of the segment.
- Eg: The value of Data Segment Register (DS) is 2222 H.
- The BIU appends 0H to the LSBs of the address.
- BIU uses the following formula: Effective Address = Starting Address of Segment + Offset.

All offsets are limited to 16-bits. The maximum size possible for segment is 216 = 65,535 bytes (64 KB).

The area of memory from the start of the second segment to the possible end of the first segment is called as overlapped segment.

The main advantages of the segmented memory scheme are as follows: a)Allows the memory capacity to be 1 MB although the actual addresses to be handled are of 16-bit size.

b) Allows the placing of code data and stack portions of the same program in different parts (segments) of memory, for data and code protection.

c) Permits a program and/ or its data to be put into different areas of memory each time program is executed, ie, relocation may be done.

4) Physical Memory Organization

The 1MB memory is physically organized as odd bank and even bank, each of 512kbytes, addressed in parallel by the processor.

Byte data with even address is transferred on $D_7 - D_0$ and byte data with odd address is transferred on $D_{15} - D_8$.

The processor provides two enable signals, $\overline{BHE}$ and A_0 for selecting of either even or odd or both the banks.

$D_{15} - D_0 / A_{15} - A_0$ are the common signal line in 8086 as $AD_{15} - AD_0$

FIG

$\overline{BHE}$	A_0	Function
0	0	Whole word
0	1	Upper byte/ odd address
1	0	Lower byte/even address
1	1	none

If the address bus contains $FFFFO_H$

If $\overline{BHE} = 0$, then it reads the 16 bits data from memory location $FFFFO_H$ and $FFFF1_H$.

If $\overline{BHE} = 1$, it reads the 8 bits data from memory location $FFFFO_H$.

Fig shows the physical memory organisation of 8086.

5)Minimum Mode 8086 System

•In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX pin to logic 1.

•All the control signals are given out by the microprocessor chip itself.

There is a single microprocessor in the minimum mode system.

The remaining components in the system are latches, transreceivers, clock generator, memory and I/O devices.

Chip selection logic may be required for selecting memory or I/O devices.

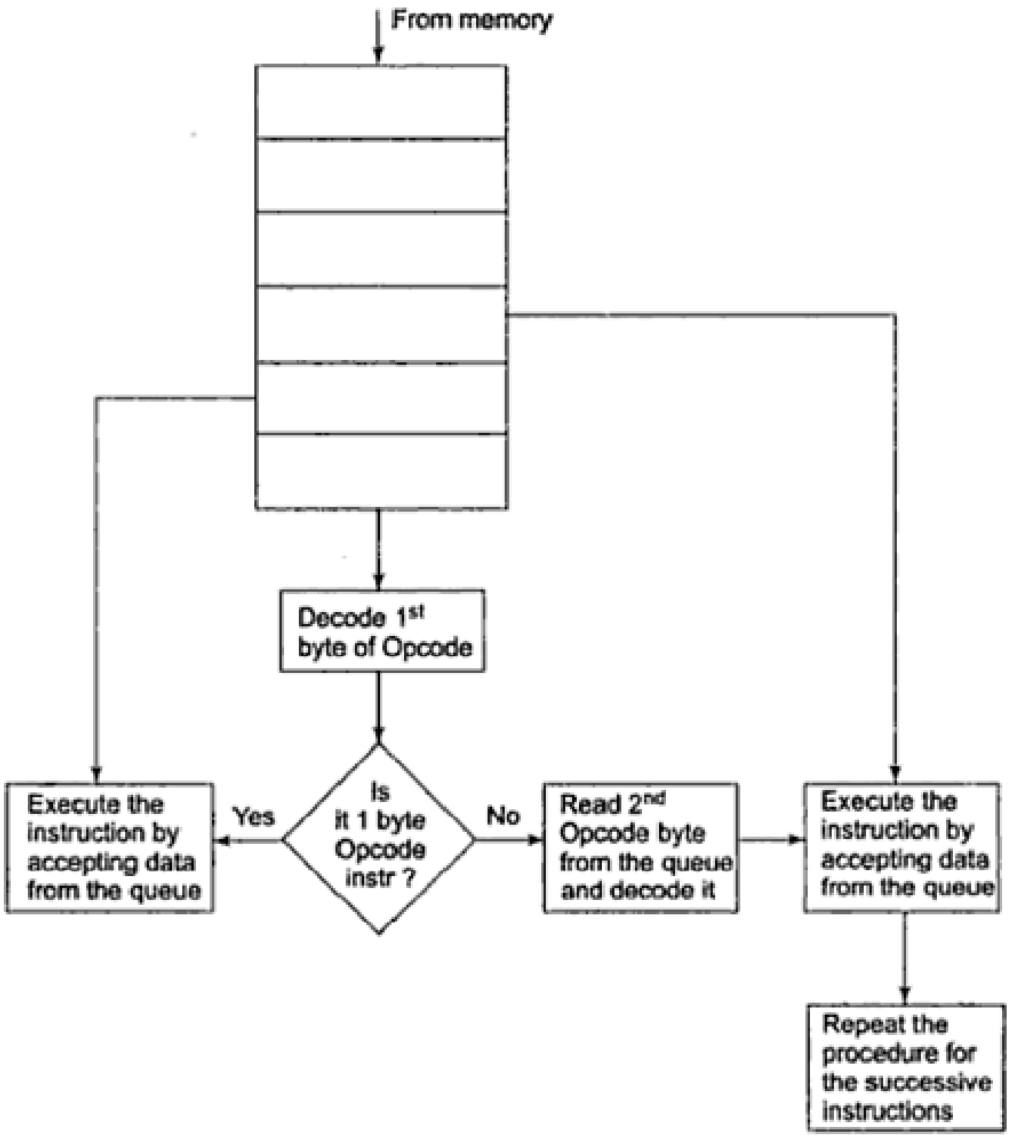
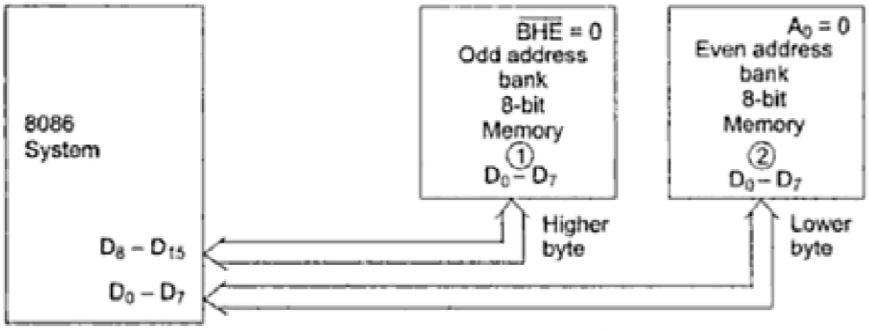


Fig: The Queue operation



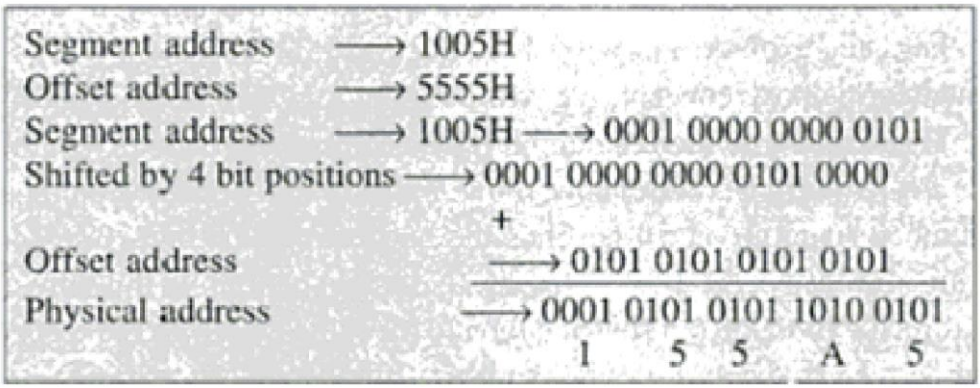


Fig: Memory segmentation

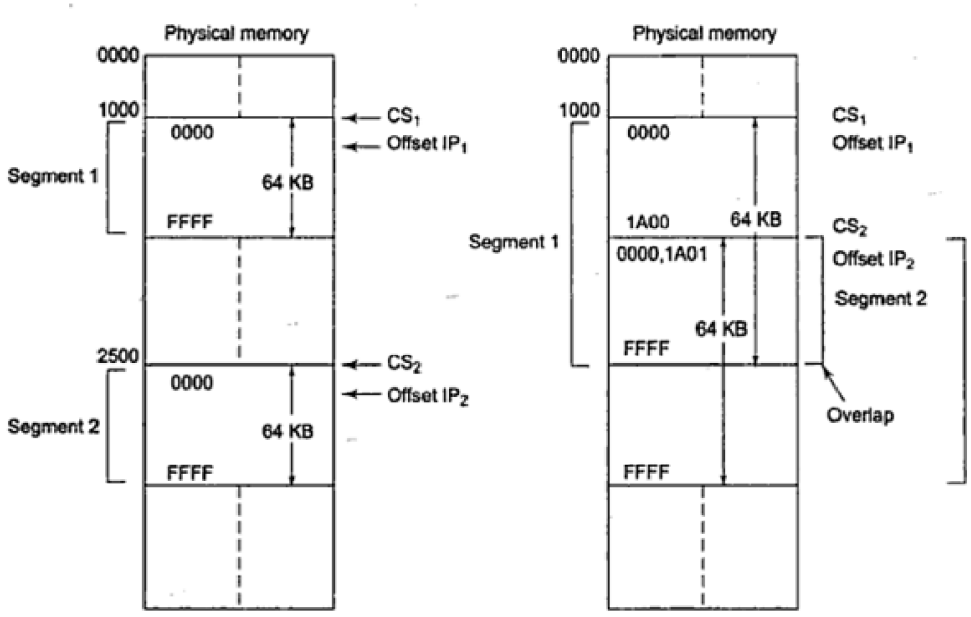


Fig: a) Non overlapping segments b) Overlapping segments

•Latches are used for separating the valid address from the multiplexed address/data signals and are controlled by the ALE signal generated by 8086.

•Transreceivers are the bidirectional buffers and some times they are called as data amplifiers. They are required to separate the valid data from the time multiplexed address/data signals.
•The DEN signal indicates the direction of data, i.e. from or to the processor.
The system contains memory for the monitor and users program storage.
•EPROM are used for monitor storage, while RAM for users program storage.
A system may contain I/O devices.

•In the Timing Diagram, the opcode fetch and read cycles are similar.
It can be categorized in two parts, the first is the timing diagram for read cycle and the second is the timing diagram for write cycle.
•The read cycle begins in T1 with the assertion of address latch enable (ALE) signal and also M / IO signal.
•The BHE and A0 signals address low, high or both bytes.
From T1 to T4 , the M/IO signal indicates a memory or I/O operation.
•At T2, the address is removed from the local bus and is sent to the output. The bus is then tristated.
The read (RD) control signal is also activated in T2.
•The read (RD) signal causes the address device to enable its data bus drivers. After RD goes low, the valid data is available on the data bus.
•The addressed device will drive the READY line high.
•A write cycle also begins with the assertion of ALE and the emission of the address.
The M/IO signal is again asserted to indicate a memory or I/O operation.
•The BHE and A0 signals are used to select the proper byte or bytes of memory or I/O word to be read or write.
•The M/IO, RD and WR signals indicate the type of data transfer.
Hold Response sequence: The HOLD pin is checked at leading edge of each clock pulse.
If it is received active, the CPU activates HLDA in the next clock cycle and for succeeding bus cycles, the bus will be given to another requesting master.
•The control of the bus is not regained by the processor until the requesting master does not drop the HOLD pin low.
When the request is dropped by the requesting master, the HLDA is dropped by the processor at the trailing edge of the next clock.
Minimum Mode Interface
•When the Minimum mode operation is selected, the 8086 provides all control signals needed to implement the memory and I/O interface.
The minimum mode signal can be divided into the following basic groups: address/data bus, status, control, interrupt and DMA.

$M/\overline{IO}$	$\overline{RD}$	$\overline{DEN}$	Transfer Type
0	0	1	I/O read
0	1	0	I/O write
1	0	1	Memory read
1	1	0	Memory write

Address/Data Bus: These lines serve two functions.
Address bus is 20 bits long and it consists of signal lines A0 through A19.

A19 represents the MSB and A0 LSB.
This gives the 8086 a 1Mbyte memory address space.
It has an independent I/O address space which is 64K bytes in length.

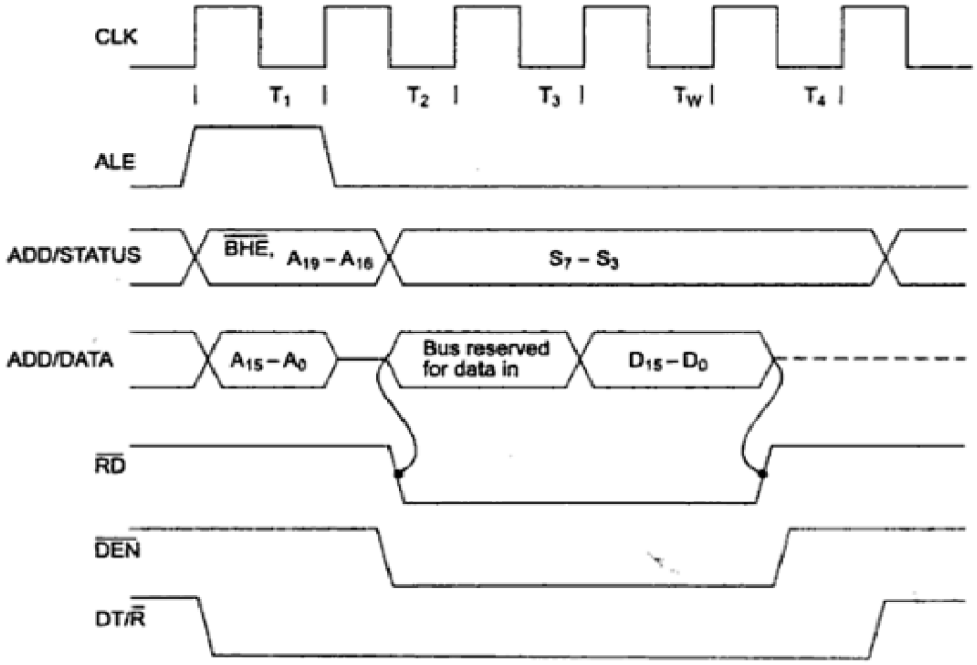
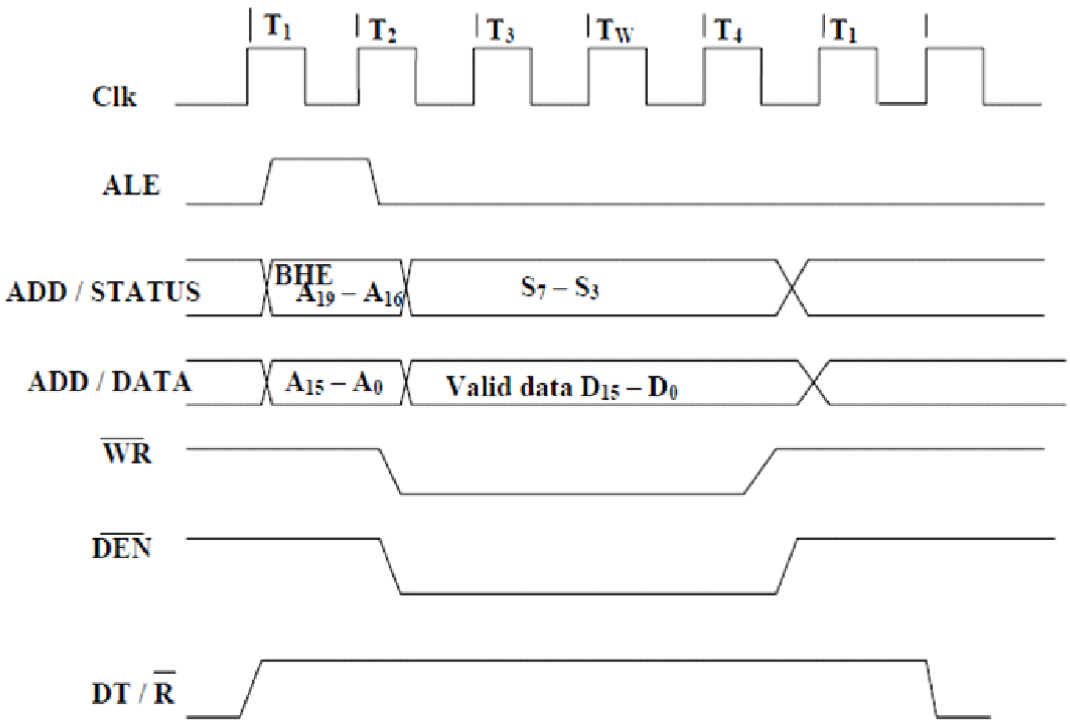


Fig: Read Cycle Timing diagram for Minimum mode

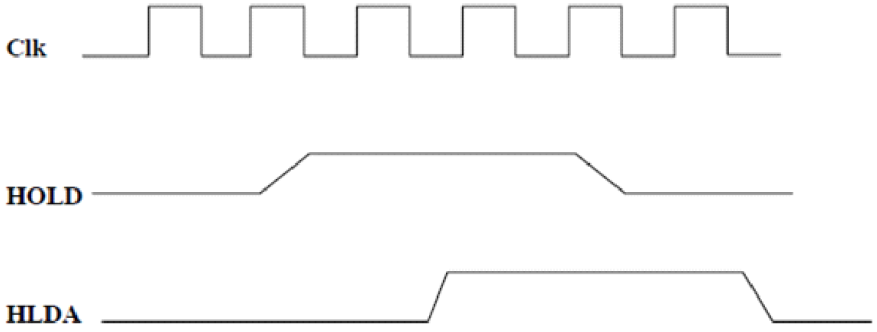
The 16 data bus lines D0 through D15 are actually multiplexed with address lines A0 through A15 respectively.
•Using Data bus system carries read/write data for memory, input/output data for I/O devices, and interrupt type codes from an interrupt controller.

•**Status signal:** The four most significant address lines A19 through A16 are also multiplexed but in this case with status signals S6 through S3.
These status bits are output on the bus at the same time that data are transferred over the other bus lines.
•Bit S4 and S3 together from a 2 bit binary code that identifies which of the 8086 internal segment registers are used to generate the physical address that was output on the address bus.
•Code S4S3 = 00 identifies a register known as **extra segment register** as the source of the segment address.
•Status line S5 reflects the status of another internal characteristic of the 8086.

The last status bit S6 is always at the logic 0 level.



Write Cycle Timing Diagram for Minimum Mode



Bus Request and Bus Grant Timings in Minimum Mode System

.

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

Fig: Memory segment status codes

Control Signals:

These signals control functions such as when the bus is to carry a valid address in which direction data are to be transferred over the bus, when valid write data are on the bus and when to put read data on the system bus.

•ALE is a pulse to logic 1 that signals external circuitry when a valid address word is on the bus.

• BHE is Bus high enable.

Logic 0 on this used as a memory enable signal for the most significant byte half of the data bus D8 through D1.

These lines also serves a second function, which is as the S_7 status line.

•Using the M/IO and DT/R lines, the 8086 signals which type of bus cycle is in progress and in which direction data are to be transferred over the bus.

•The logic level of M/IO tells external circuitry whether a memory or I/O transfer is taking place over the bus.

Logic 1 at this output signals a memory operation and logic 0 an I/O operation.

•The direction of data transfer over the bus is signaled by the logic level output at DT/R.

•The signal read RD and write WR indicates that a read bus cycle or a write bus cycle is in progress.

The 8086 switches WR to logic 0 to signal external device that valid write or output data are on the bus.

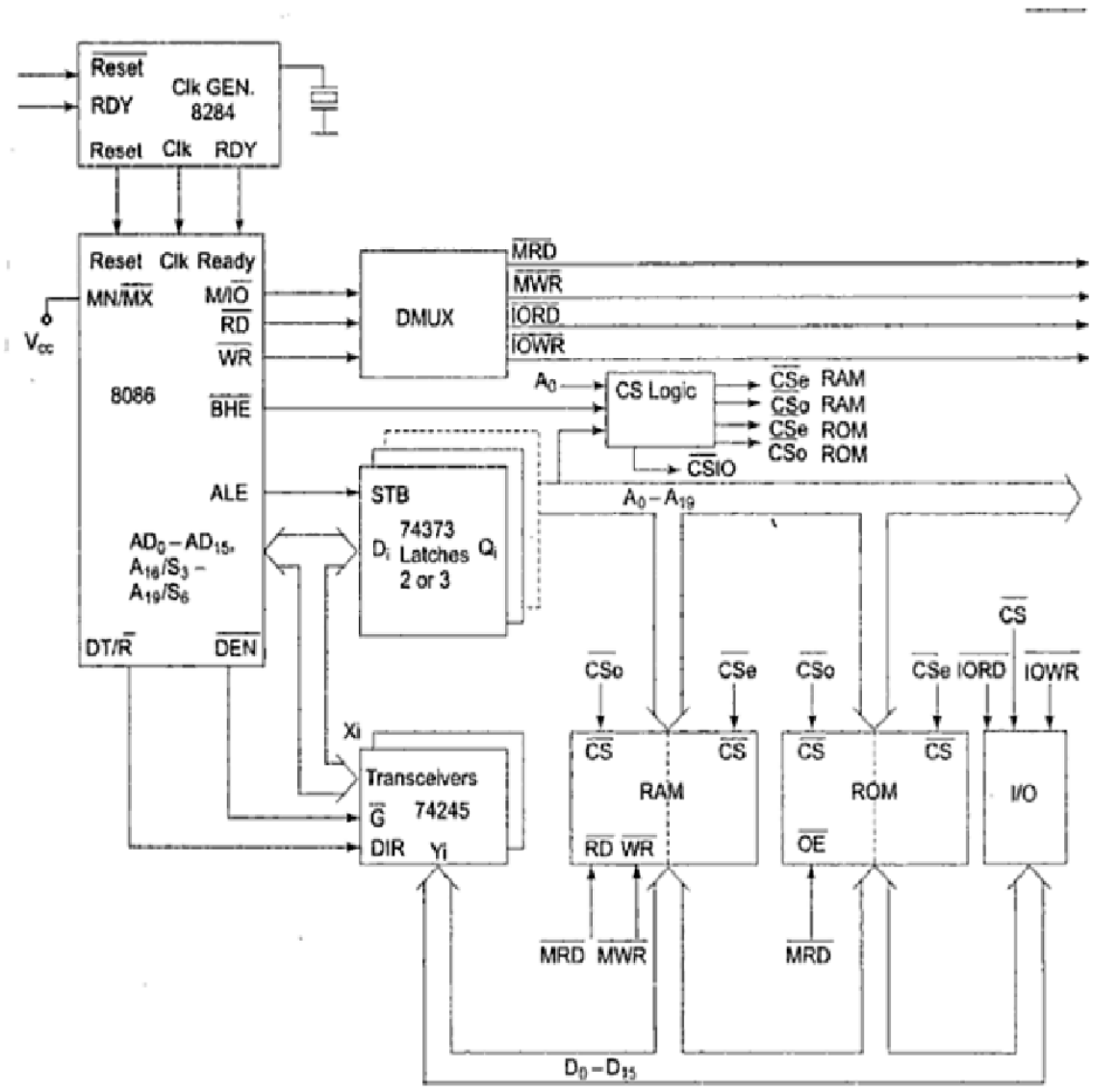


Fig: Minimum mode 8086 system

• RD indicates that the 8086 is performing a read of data of the bus.

During read operations, one other control signal is supplied i.e DEN (data enable) and it signals external devices when they should put data on the bus.

•There is one control signal that is involved with the memory and I/O interface. This is the READY signal.

•READY signal is used to insert wait states into the bus cycle such that it is extended by a number of clock periods.

This signal is provided by an external clock generator

•**Interrupt signals**: The key interrupt interface signals are interrupt request (INTR) and interrupt acknowledge (INTA).

•INTR is an input to the 8086 that can be used by an external device to signal that it need to be serviced.

•Logic 1 at INTR represents an active interrupt request.

When an interrupt request has been recognized by the 8086, it indicates this fact to external circuit with pulse to logic 0 at the INTA output.

•The TEST input is also related to the external interrupt interface.
Execution of a WAIT instruction causes the 8086 to check the logic level at the TEST input.
•If the logic 1 is found, the MPU suspend operation and goes into the idle state.
The 8086 no longer executes instructions, instead it repeatedly checks the logic level of the TEST input waiting for its transition back to logic 0.
•There are two more inputs in the interrupt interface: the nonmaskable interrupt NMI and the reset interrupt RESET.
•On the 0-to-1 transition of NMI control is passed to a nonmaskable interrupt service routine.
The RESET input is used to provide a hardware reset for the 8086.
•**DMA Interface signals**:The direct memory access DMA interface of the 8086 minimum mode consist of the HOLD and HLDA signals.
•When an external device wants to take control of the system bus, it signals to the 8086 by switching HOLD to the logic 1 level.
At the completion of the current bus cycle, the 8086 enters the hold state.
In the hold state, signal lines AD0 through AD15, A16/S3 through A19/S6, BHE, M/IO, DT/R, RD, WR, DEN and INTR are all in the high Z state.
The 8086 signals external device by switching its HLDA output to logic 1 level.
6)Maximum Mode 8086 System : In the maximum mode, the 8086 is operated by strapping the MN/MX pin to ground.
•In this mode, the processor derives the status signal S2, S1, S0.
Another chip called bus controller derives the control signal using this status information.
•In the maximum mode, there may be more than one microprocessor in the system configuration.
•The basic function of the bus controller chip IC8288, is to derive control signals like RD and WR,DEN, DT/R, ALE etc. using the information by the processor on the status lines.
•The bus controller chip has input lines S2, S1, S0 and CLK.
These inputs to 8288 are driven by CPU.
•It derives the outputs ALE, DEN, DT/R, MRDC, MWTC, AMWC, IORC, IOWC and AIOWC.
The AEN, IOB and CEN pins are specially useful for multiprocessor systems.
•AEN and IOB are generally grounded. CEN pin is usually tied to +5V.
The significance of the MCE/PDEN output depends upon the status of the IOB pin.
•INTA pin used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device.
•IORC, IOWC are I/O read command and I/O write command signals respectively.
These signals enable an IO interface to read or write the data from or to the address port.
•The MRDC, MWTC are memory read command and memory write command signals respectively and may be used as memory read or write signals.
•For both of these write command signals, the advanced signals namely AIOWC and AMWTC are available.
R0, S1, S2 are set at the beginning of bus cycle.
8288 bus controller will output a pulse as on the ALE and apply a required signal to its DT / R pin during T1.
•In T2, 8288 will set DEN=1 thus enabling transceivers, and for an input it will activate MRDC or IORC.
For an output, the AMWC or AIOWC is activated from T2 to T4 and MWTC or IOWC is activated from T3 to T4.
•The status bit S0 to S2 remains active until T3 and become passive during T3 and T4**.**
•If reader input is not activated before T3, wait state will be inserted between T3 and T4.
•**Timings for RQ/ GT Signals:**

The request/grant response sequence contains a series of three pulses.
The request/grant pins are checked at each rising pulse of clock input.
When a request is detected and if the condition for HOLD request are satisfied, the processor issues a grant pulse over the RQ/GT pin immediately during T4 (current) or T1 (next) state.

Maximum Mode Interface

•When the 8086 is set for the maximum-mode configuration, it provides signals for implementing a multiprocessor / coprocessor system environment.

•By multiprocessor environment we mean that one microprocessor exists in the system and that each processor is executing its own program.

•Usually in this type of system environment, there are some system resources that are common to all processors.
•They are called as ***global resources***. There are also other resources that are assigned to specific processors known as ***local or private resources***.
•Coprocessor also means that there is a second processor in the system.
In this two processors do not access the bus at the same time.
When the requesting master receives grant pulse, it accepts the control of the bus, it sends a release pulse to the processor using RQ/GT pin.

•**8288 Bus Controller – Bus Command and Control Signals**:
8086 does not directly provide all the signals that are required to control the memory, I/O and interrupt interfaces.
•Specially the WR, M/IO, DT/R, DEN, ALE and INTA, signals are no longer produced by the 8086. Instead it outputs three status signals S0, S1, S2 prior to the initiation of each bus cycle.
This 3- bit bus status code identifies which type of bus cycle is to follow.
S2S1S0 are input to the external bus controller device, the bus controller generates the appropriately timed command and control signals.
The output of 8289 are bus arbitration signals: *Bus busy* (BUSY), *common bus request* (CBRQ), *bus priority out* (BPRO), *bus priority in* (BPRN), *bus request* (BREQ) and *bus clock* (BCLK).
•They correspond to the bus exchange signals of the Multibus and are used to lock other processor off the system bus during the execution of an instruction by the 8086.

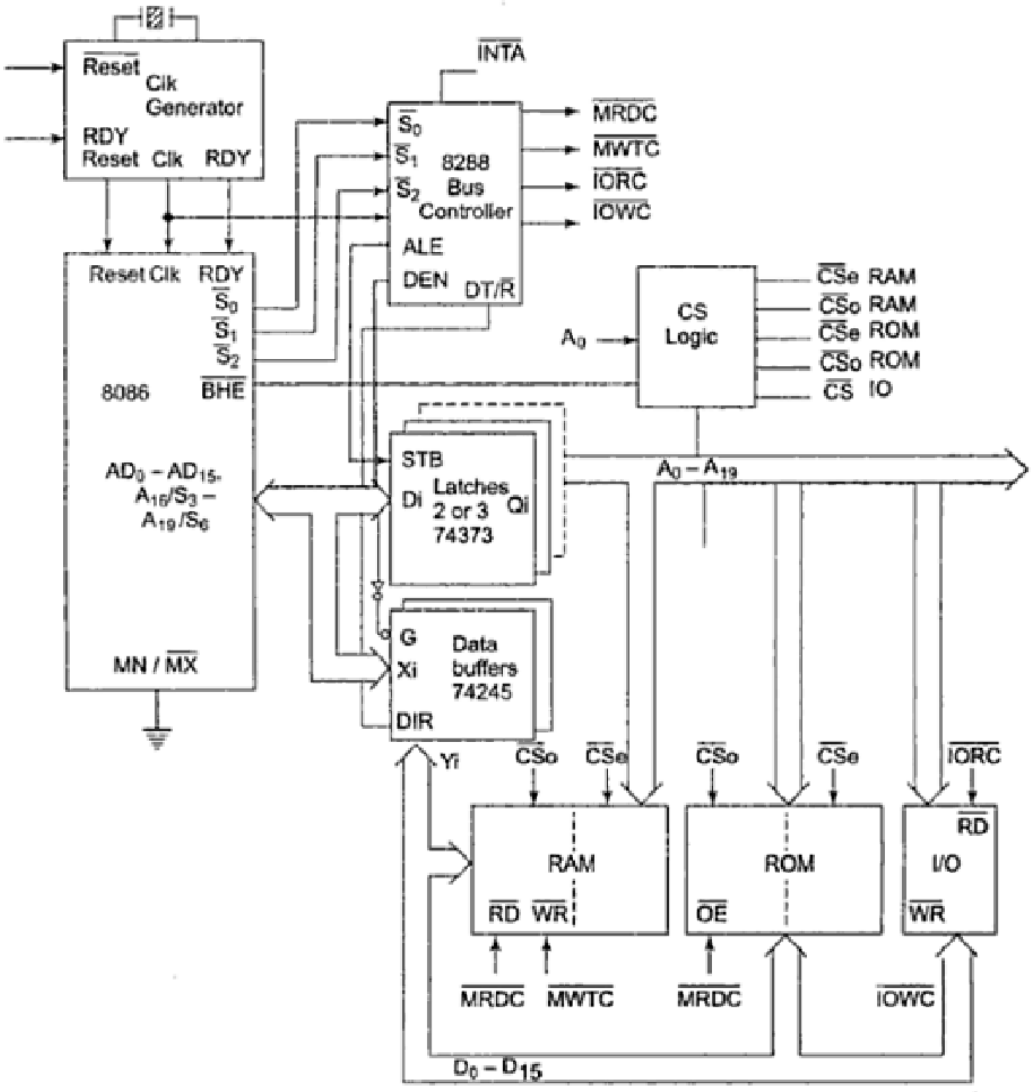
QS_1	QS_0	Queue Status
0 (low)	0	No Operation. During the last clock cycle, nothing was taken from the queue.
0	1	First Byte. The byte taken from the queue was the first byte of the instruction.
1 (high)	0	Queue Empty. The queue has been reinitialized as a result of the execution of a transfer instruction.
1	1	Subsequent Byte. The byte taken from the queue was a subsequent byte of the instruction.

•**Queue Status Signals**: Two new signals that are produced by the 8086 in the maximum-mode system are queue status outputs QS0 and QS1.
Together they form a 2-bit queue status code, QS1QS0.
•**Local Bus Control Signal – Request / Grant Signals**: In a maximum mode configuration, the minimum mode HOLD, HLDA interface is also changed.
These two are replaced by request/grant lines RQ/ GT0 and RQ/ GT1, respectively.

7) Differences between 8086 and 8088 Microprocessor
The Intel 8088 microprocessor is a variant of the Intel **8086**.
Introduced on July 1, 1979, the 8088 had an **8**-bit external data bus instead of the **16**-bit bus of the **8086**.
The **16**-bit registers and the one megabyte address range were unchanged.
Comparison of 8086 v/s 8088

8086	**8088**
The instruction Queue is 6 byte long.	The instruction Queue is 4 byte long.
In 8086, memory divides into two banks -even or lower bank -odd or higher bank	The memory in 8088 does not divide into two banks.
The data bus of 8086 is 16-bit wide	The data bus of 8088 is 8-bit wide.
It has BHE (bar) signal on pin no. 34 &there is no SSO (bar) signal.	It does not have BHE (bar) signal on pin no. 34 &has only SSO (bar) signal. It has no S7 pin.
Control pin in 8086 is M/IO (bar).	Control pin in 8088 is IO/M(bar).
In 8086, all address &data Buses are multiplexed.	In 8088, address bus; AD_7- AD_0 buses are multiplexed.
Three clock speed: 5, 8, 10 MHz	Two clock speed: 5, 8 MHz

Status Inputs			CPU Cycles
$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	
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	Instruction Fetch
1	0	1	Read Memory



8) Machine Language instruction format

A machine language instruction format has one or more number of fields associated with it. The first field is called as *operation code field* or *opcode field*, which indicates the type of the operation to be performed by the CPU. The instruction format also contains other fields known as *operand fields*. The CPU executes the instruction using the information which reside in these fields.

There are six general formats of instructions in 8086 instruction set. The length of an instruction may vary from one byte to six bytes. The instruction formats are described as follows:

1. One byte Instruction This format is only one byte long and may have the implied data or register operands. The least significant 3-bits of the opcode are used for specifying the register operand, if any. Otherwise, all the 8-bits form an opcode and the operands are implied.

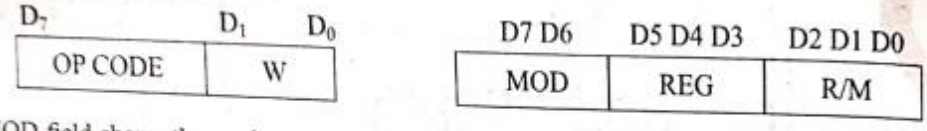
2. Register to Register This format is 2 bytes long. The first byte of the code specifies the operation code and width of the operand specified by w bit. The second byte of the code shows the register operands and R/M field, as shown below.

D_7	D_1	D_0
OP CODE	W	

D7 D6	D5 D4 D3	D2 D1 D0
11	REG	R/M

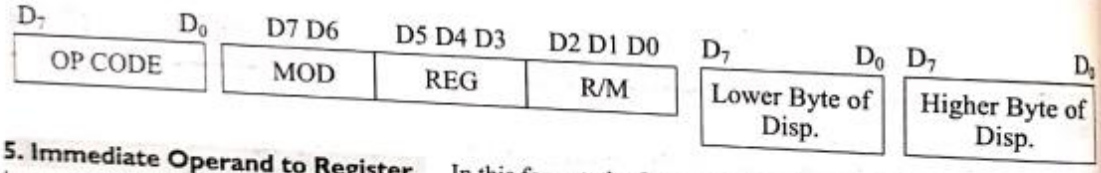
The register represented by the REG field is one of the operands. The R/M field specifies another register or memory location, i.e. the other operand.

3. Register to/from Memory with no Displacement This format is also 2 bytes long and similar to the register to register format except for the MOD field as shown.

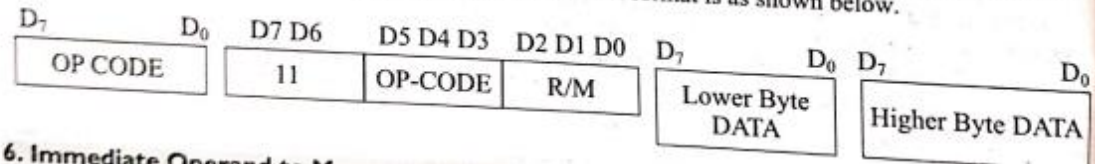


The MOD field shows the mode of addressing. The MOD, R/M, REG and the W fields are decided in Table 2.2.

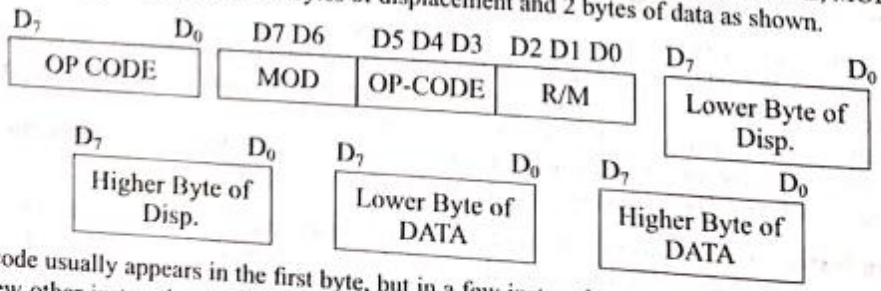
4. Register to/from Memory with Displacement This type of instruction format contains one or two additional bytes for displacement along with 2-byte the format of the register to/from memory without displacement. The format is as shown below.



5. Immediate Operand to Register In this format, the first byte as well as the 3-bits from the second byte which are used for REG field in case of register to register format are used for opcode. It also contains one or two bytes of immediate data. The complete instruction format is as shown below.



6. Immediate Operand to Memory with 16-bit Displacement This type of instruction format requires 5 or 6 bytes for coding. The first 2 bytes contain the information regarding OPCODE, MOD, and R/M fields. The remaining 4 bytes contain 2 bytes of displacement and 2 bytes of data as shown.



The opcode usually appears in the first byte, but in a few instructions, a register destination is in the first byte and few other instructions may have their 3-bits of opcode in the second byte. The opcodes have the single bit indicators. Their definitions and significances are given as follows:

W-bit This indicates whether the instruction is to operate over an 8-bit or 16-bit data/operands. If W bit is 0 the operand is of 8-bits and if W is 1, the operand is of 16-bits.

D-bit This is valid in case of double operand instructions. One of the operands must be a register specified by the REG field. The register specified by REG is source operand if D = 0, else, it is a destination operand.

S-bit This bit is called as sign extension bit. The S bit is used along with W-bit to show the type of the operation. For, example

8-bit operation with 8-bit immediate operand is indicated by S = 0, W = 0;
16-bit operation with 16-bit immediate operand is indicated by S = 0, W = 1 and
16-bit operation with a sign extended immediate data is given by S = 1, W = 1

V-bit This is used in case of shift and rotate instructions. This bit is set to 0, if shift count is 1 and is set to 1, if CL contains the shift count.

Z-bit This bit is used by REP instruction to control the loop. If Z bit is equal to 1, the instruction with REP prefix is executed until the zero flag matches the Z bit.

The REG code of the different registers (either as source or destination operands) in the opcode byte are assigned with binary codes. The segment registers are only 4 in number hence 2 binary bits will be sufficient to code them. The other registers are 8 in number, so at least 3-bits will be required for coding them. To allow the use of 16-bit registers as two 8-bit registers they are coded with W bit as shown in Table 2.1.

Table 2.1 *Assignment of Codes with Different Registers*

W	*Register Address (code)*	*Registers*	*Segment 2 bit bit Register (code)*	*Segment Register*
0	000	AL		
0	001	CL	00	ES
0	010	DL	01	CS
0	011	BL	10	SS
0	100	AH	11	DS
0	101	CH		
0	110	DH		
0	111	BH		
1	000	AX		
1	001	CX		
1	010	DX		
1	011	BX		
1	100	SP		
1	101	BP		
1	110	SI		
1	111	DI		

Activat
Go to PC

Please note that usually all the addressing modes have DS as the default data segment. However, the addressing modes using BP and SP have SS as the default segment register.

To find out the MOD and R/M fields of a particular instruction, one should first decide the addressing mode of the instruction. The addressing mode depends upon the operands and suggests how the effective address may be computed for locating the operand, if it lies in memory. The different addressing modes of the 8086 instructions are listed in Table 2.2. The R/M column and addressing mode row element specifies the R/M field, while the addressing mode column specifies the MOD field.

Table 2.2 *Addressing Modes and the Corresponding MOD, REG and R/M Fields*

Operands	*Memory Operands*			*Register Operands*	
	No Displacement	*Displacement 8-bit*	*Displacement 16-bit*		
MOD	*00*	*01*	*10*	*11*	
R/M				*W = 0*	*W = 1*
000	(BX) + (SI)	(BX) + (SI) + D8	(BX) + (SI) + D16	AL	AX
001	(BX) + (DI)	(BX) + (DI) + D8	(BX) + (DI) + D16	CL	CX
010	(BP) + (SI)	(BP) + (SI) + D8	(BP) + (SI) + D16	DL	DX
011	(BP) + (DI)	(BP) + (DI) + D8	(BP) + (DI) + D16	BL	BX
100	(SI)	(SI) + D8	(SI) + D16	AH	SP
101	(DI)	(DI) + D8	(DI) + D16	CH	BP
110	D16	(BP) + D8	(BP) + D16	DH	SI
111	(BX)	(BX) + D8	(BX) + D16	BH	DI

Note: 1. D8 and D16 represent 8 and 16 bit displacements respectively.
2. The default segment for the addressing modes using BP and SP is SS. For all other addressing modes the default segments are DS or ES.

DS is the default data segment register when a data is to be referred as an operand. CS is the default code segment register for storing program codes (executable codes). SS is the default segment register for the stack data accesses and operations. ES is the default segment register for the destination data storage. All the segments available (defined in a particular program) can be read or written as data segments by newly defining the data segment as required. There is no physical difference in the memory structure or no physical separation between the segment areas. They may or may not overlap with each other. Chapter 3 on 'Assembly

CS305-Microprocessors and Microcontrollers

S5 CSE

Module 2

- **Addressing modes of 8086**

Addressing mode indicates a way of locating data or operands. Depending upon the data types used in the instruction and the memory addressing modes, any instruction may belong to one or more addressing modes, or some instruction may not belong to any of the addressing modes. Thus the addressing modes describe the types of operands and the way they are accessed for executing an instruction.

According to the flow of instruction execution, the instructions may be categorised as (i) Sequential control flow instructions and (ii) Control transfer instructions.

Sequential control flow instructions are the instructions which after execution, transfer control to the next instruction appearing immediately after it (in the sequence) in the program. For example, the arithmetic, logical, data transfer and processor control instructions are sequential control flow instructions. *The control transfer instructions, on the other hand, transfer control to some predefined address or the address somehow specified in the instruction, after their execution.* For example, INT, CALL, RET and JUMP instructions fall under this category.

The addressing modes for sequential and control transfer instructions are explained as follows:

1. Immediate In this type of addressing, immediate data is a part of instruction, and appears in the form of successive byte or bytes.

Example

```
MOV AX, 0005H
MOV BL, 06H
```

In the above examples 0005H and 06H are the immediate data. The immediate data may be 8-bit or 16-bit in size.

---

2. Direct In the direct addressing mode, a 16-bit memory address (offset) or an IO address is directly specified in the instruction as a part of it.

Example

```
MOV AX, [5000H]
IN 80H
```

Here, data resides in a memory location in the data segment, whose effective address may be computed using 5000H as the offset address and content of DS as segment address. The effective address, here, is 10H*DS+5000H. In the second instruction 80H is IO address.

---

3. Register In the register addressing mode, the data is stored in a register and it is referred using the particular register. All the registers, except IP, may be used in this mode.

Example

```
MOV BX, AX.
ADC AL, BL
```

The operands in these instructions are provided in registers BX, AX and AL, BL respectively.

4. Register Indirect Sometimes, the address of the memory location which contains data or operand is determined in an indirect way, using the offset registers. This mode of addressing is known as register indirect mode. In this addressing mode, the offset address of data is in either BX or SI or DI register. The default segment is either DS or ES. The data is supposed to be available at the address pointed to by the content of any of the above registers in the default data segment.

Example

```
MOV AX, [BX]
```

Here, data is present in a memory location in DS whose offset address is in BX. The effective address of the data is given as 10H*DS+[BX].

5. Indexed In this addressing mode, offset of the operand is stored in one of the index registers. DS is the default segment for index registers SI and DI. In case of string instructions DS and ES are default segments for SI and DI respectively. This mode is a special case of register indirect addressing mode.

Example

```
MOV AX, [SI]
MOV CX, [DI]
```

Here, data is available at an offset address stored in SI in DS. The effective address, in this case, is computed as 10H*DS+[SI]. The content of address 10H*DS+[SI] will be transferred into register CX.

6. Register Relative In this addressing mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the registers BX, BP, SI and DI in the default (either DS or ES) segment.

Example

```
MOV AX, 50H[BX]
MOV 10H[SI], DX
```

Here, the effective address is given as 10H*DS+50H+[BX] and 10H∗DS+10H+[SI] respectively.

---

7. Based Indexed The effective address of data is formed, in this addressing mode, by adding content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI). The default segment register may be ES or DS.

Example

```
MOV AX, [BX] [SI]
MOV [BX] [DI], AX
```

Here, BX is the base register and SI is the index register. The effective address is computed as 10H*DS+[BX]+[SI].

---

8. Relative Based Indexed The effective address is formed by adding an 8 or 16-bit displacement with the sum of contents of any one of the base registers (BX or BP) and any one of the index registers, in a default segment.

Example

```
MOV AX,50H [BX][SI]
ADD 50H [BX] [SI], BP
```

Here, 50H is an immediate displacement, BX is a base register and SI is an index register. The effective address of data is computed as 10H*DS+[BX]+[SI]+50H. The second instruction adds content of B with memory location of which offset is given by adding 50H of content of BX and SI. The result is stored in the memory location.

For the control transfer instructions, the addressing modes depend upon whether the destination location is within the same segment or in a different one. It also depends upon the method of passing the destination address to the processor. Basically, there are two addressing modes for the control transfer instructions, viz. intersegment and intrasegment addressing modes.

If the location to which the control is to be transferred lies in a different segment other than the current one, the mode is called intersegment mode. If the destination location lies in the same segment, the mode is called intrasegment mode.

Figure shows the modes for control transfer instructions.

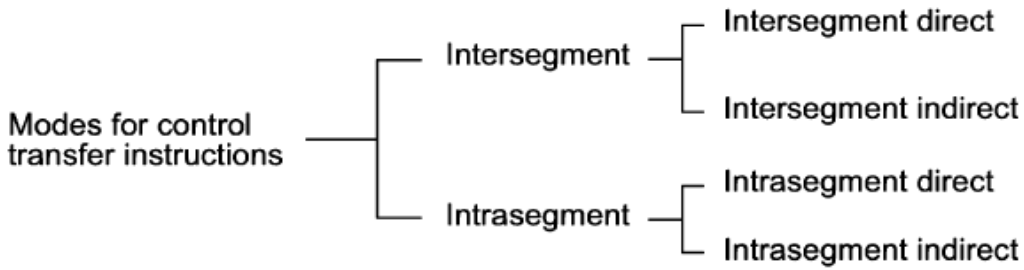


Fig. ***Addressing Modes for Control Transfer Instructions***

9. Intrasegment Direct Mode In this mode, the address to which the control is to be transferred lies in the same segment in which the control transfer instruction lies and appears directly in the instruction as an immediate displacement value. In this addressing mode, the displacement is computed relative to the content of the instruction pointer IP.

The effective address to which the control will be transferred is given by the sum of 8 or 16 bit displacement and current content of IP. In case of jump instruction, if the signed displacement (d) is of 8 bits (i.e. $-128 < d < +127$), we term it as *short jump* and if it is of 16 bits (i.e. $-32768 < d < +32767$), it is termed as *long jump*.

Example

```
JMP SHORT LABEL; LABEL lies within -128 TO +127 from the current IP
content.
```

Thus SHORT LABEL is 8-bit signed displacement.

A 16-bit target address of a label indicates that it lies within – 32768 to + 32767. But a problem arises when one requires a forward jump at a relative address greater than 32767 or backward jump at relative address – 32768; in the same segment. Suppose current contents of IP are 5000H then a forward jump may be allowed at all the displacement DISP so that IP + DISP = FFFFH or DISP = FFFF – 5000 = AFFFH. Thus forward jumps may be allowed for all 16-bit displacement values from 0000H to AFFFH. If displacement exceeds AFFFH i.e. from B000H to FFFFH, then all such jumps will be treated as backward jumps. All such jumps are called NEAR PTR jumps and coded as below.

```
JMP NEAR PTR LABEL
```

10. Intrasegment Indirect Mode In this mode, the displacement to which the control is to be transferred, is in the same segment in which the control transfer instruction lies, but it is passed to the instruction indirectly. Here, the branch address is found as the content of a register or a memory location. This addressing mode may be used in unconditional branch instructions.

Example

```
JMP [BX]; Jump to effective address stored in BX.
                          JMP [ BX + 5000H ]
```

---

11. Intersegment Direct In this mode, the address to which the control is to be transferred is in a different segment. This addressing mode provides a means of branching from one code segment to another code segment. Here, the CS and IP of the destination address are specified directly in the instruction.

Example

```
JPM 5000H : 2000H;
Jump to effective address 2000H in segment 5000H.
```

---

12. Intersegment Indirect In this mode, the address to which the control is to be transferred lies in a different segment and it is passed to the instruction indirectly, i.e. contents of a memory block containing four bytes, i.e. IP(LSB), IP(MSB), CS(LSB) and CS(MSB) sequentially. The starting address of the memory block may be referred using any of the addressing modes, except immediate mode.

Example

```
JMP [2000H];
```

Jump to an address in the other segment specified at effective address 2000H in DS, that points to the memory block as said above.

Forming the Effective Addresses The following examples explain forming of the effective addresses in the different modes.

Example

The contents of different registers are given below. Form effective addresses for different addressing modes.
Offset (displacement) = 5000H
[AX]-1000H, [BX]-2000H, [SI]-3000H, [DI]-4000H, [BP]-5000H,
[SP]-6000H, [CS]-0000H, [DS]-1000H, [SS]-2000H, [IP]-7000H.
Shifting a number four times is equivalent to multiplying it by 16_D or 10_H.

(i) Direct addressing mode

```
                              MOV AX, [5000H]
DS:OFFSET ⇔   1000H: 5000H
10H* DS ⇔      10000
Offset ⇔     + 5000
              -------
              15000H - Effective address
```

(ii) Register indirect

```
                              MOV AX, [BX]
DS:BX ⇔       1000H:2000H
10H*DS ⇔       10000
[BX] ⇔       + 2000
              -------
```

12000H - Effective address

(iii) Register relative

```
                                    MOV AX, 5000 [BX]

DS: [5000 + BX]
10H*DS ⇔   10000
Offset ⇔  + 5000
[BX] ⇔    + 2000
          ------
          17000H - Effective address
```

(iv) Based indexed

```
                                    MOV AX, [BX] [SI]

DS:[BX + SI]
10H*DS ⇔   10000
[BX] ⇔    + 2000
[SI] ⇔    + 3000
          ------
          15000H - Effective address
```

(v) Relative based indexed

```
                                MOV AX, 5000 [BX] [SI]

DS: [BX + SI + 5000]
10H*DS ⇔   10000
[BX] ⇔    + 2000
[SI] ⇔    + 3000
Offset ⇔  + 5000
          ------
          1A000 - effective address
```

we present examples of address formation in control transfer instructions.

Example

Suppose our main program resides in the code segment where CS = 1000H. The main program calls a subroutine which resides in the same code segment. The base register contains offset of the subroutine, i.e. BX = 0050H. Since the offset is specified indirectly, as the content of BX, this is indirect addressing. The instruction CALL [BX] calls the subroutine located at an address 10H*CS + [BX] = 10050H, i.e. in the same code segment. Since the control goes to the subroutine which resides in the same segment, this is an example of intrasegment indirect addressing mode.

Example

Let us now assume that the subroutine resides in another code segment, where CS = 2000H. Now CALL 2000H:0050H is an example of intersegment direct addressing mode, since the control now goes to different segment and the address is directly specified in the instruction. In this case, the address of the subroutine is 20050H.

8086 instruction set

The 8086/8088 instructions are categorised into the following main types.

(i) **Data Copy/Transfer Instructions** These types of instructions are used to transfer data from source operand to destination operand. All the store, move, load, exchange, input and output instructions belong to this category.

(ii) **Arithmetic and Logical Instructions** All the instructions performing arithmetic, logical, increment, decrement, compare and scan instructions belong to this category.

(iii) **Branch Instructions** These instructions transfer control of execution to the specified address. All the call, jump, interrupt and return instructions belong to this class.

(iv) **Loop Instructions** If these instructions have REP prefix with CX used as count register, they can be used to implement unconditional and conditional loops. The LOOP, LOOPNZ and LOOPZ instructions belong to this category. These are useful to implement different loop structures.

(v) **Machine Control Instructions** These instructions control the machine status. NOP, HLT, WAIT and LOCK instructions belong to this class.

(vi) **Flag Manipulation Instructions** All the instructions which directly affect the flag register, come under this group of instructions. Instructions like CLD,STD,CLI,STI, etc. belong to this category of instructions.

(vii) **Shift and Rotate Instructions** These instructions involve the bitwise shifting or rotation in either direction with or without a count in CX.

(viii) **String Instructions** These instructions involve various string manipulation operations like load, move, scan, compare, store, etc. These instructions are only to be operated upon the strings.

Data Copy/Transfer Instructions

MOV: Move This data transfer instruction transfers data from one register/memory location to another register/memory location. The source may be any one of the segment registers or other general or special purpose registers or a memory location and, another register or memory location may act as destination.

in case of immediate addressing mode, a segment register cannot be a destination register. In other words, direct loading of the segment registers with immediate data is not permitted. To load the segment registers with immediate data, one will have to load any general purpose register with the data and then it will have to be moved to that particular segment register.

Example

Load DS with 5000H.

1. MOV DS, 5000H; Not permitted (invalid)
   Thus to transfer an immediate data into the segment register, the correct procedure is given below.
2. MOV AX, 5000H
   MOV DS, AX
   It may be noted here that both the source and destination operands cannot be memory locations (except for string instructions). Other MOV instruction examples are given below with the corresponding addressing modes.
3. MOV AX, 5000H; Immediate
4. MOV AX, BX; Register
5. MOV AX, [SI]; Indirect

PUSH: Push to Stack This instruction pushes the contents of the specified register/memory location on to the stack. The stack pointer is decremented by 2, after each execution of the instruction. The actual current stack-top is always occupied by the previously pushed data. Hence, the push operation decrements SP by two and then stores the two byte contents of the operand onto the stack. The higher byte is pushed first and then the lower byte. Thus out of the two decremented stack addresses the higher byte occupies the higher address and the lower byte occupies the lower address.

The actual operation takes place as given below SS : SP points to the stack top of 8086 system as shown in Fig. and AH, AL contains data to be pushed.

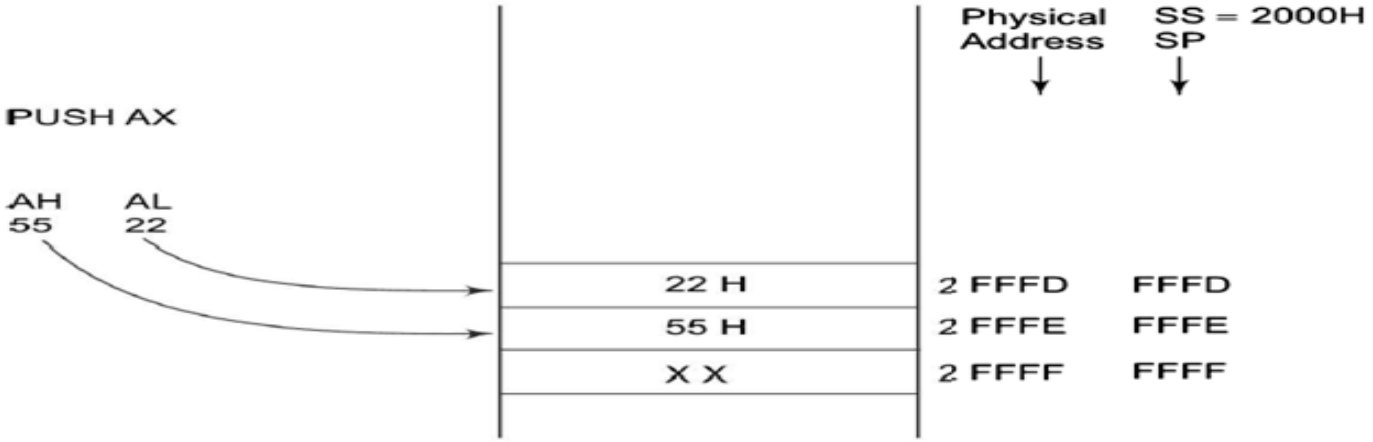


Fig. ***Pushing Data to Stack Memory***

The sequence of operation as below:

1. Current stack top is already occupied so decrement SP by one then store AH into the address pointed to by SP.
2. Further decrement SP by one and store AL into the location pointed to by SP.

Thus SP is decremented by 2 and AH–AL contents are stored in stack memory as shown in Fig. Contents of SP points to a new stack top.

The examples of these instructions are as follows:

Example

1. PUSH AX
2. PUSH DS
3. PUSH [5000H]; Content of location 5000H and 5001H in DS are pushed onto the stack

POP: Pop from Stack This instruction when executed, loads the specified register/memory location with the contents of the memory location of which the address is formed using the current stack segment and stack pointer as usual. The stack pointer is incremented by 2. The POP instruction serves exactly opposite to the PUSH instruction.

16-bit contents of current stack top are poped into the specified operand as follows.

The sequence of operation is as below.

1. Contents of stack top memory location is stored in AL and SP is incremented by one
2. Further contents of memory location pointed to by SP are copied to AH and SP is again incremented by 1

Effectively SP is incremented by 2 and points to next stack top.

The examples of these instructions are shown as follows:

Example

```
1.POP     AX
2.POP     DS
3.POP     [5000H]
```

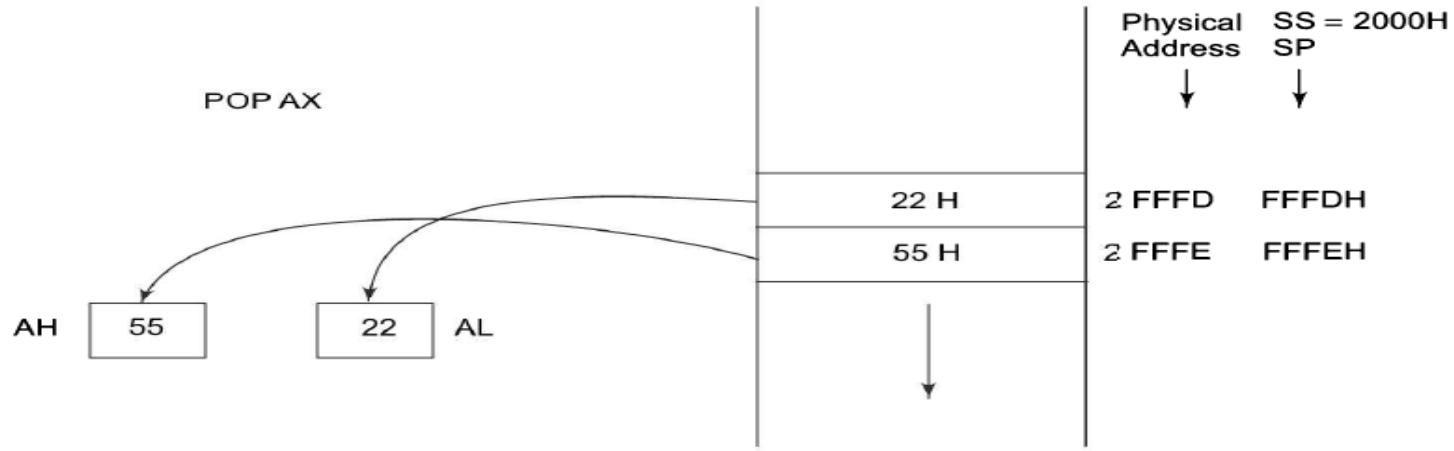


Fig. *Popping Register Contents from Stack Memory*

XCHG: Exchange This instruction exchanges the contents of the specified source and destination operands, which may be registers or one of them may be a memory location. However, exchange of contents of two memory locations is not permitted. Immediate data is also not allowed in these instructions. The examples are as follows:

Example

```
1. XCHG  [5000H], AX     ; This instruction exchanges data between AX and a
                         ; memory location [5000H] in the data segment.
2. XCHG  BX, AX          ; This instruction exchanges data between AX and BX.
```

IN: Input the Port This instruction is used for reading an input port. The address of the input port may be specified in the instruction directly or indirectly. AL and AX are the allowed destinations for 8 and 16-bit input operations. DX is the only register (implicit) which is allowed to carry the port address. If the port address is of 16 bits it must be in DX. The examples are given as shown:

Example

```
1. IN AL,03H        ; This instruction reads data from an 8-bit port whose address
                    ; is 03H and stores it in AL.
2. IN AX, DX        ; This instruction reads data from a 16-bit port whose
                    ; address is in DX (implicit) and stores it in AX.
3. MOV DX,0800H     ; The 16-bit address is taken in DX.
   IN AX, DX        ; Read the content of the port in AX.
```

OUT: Output to the Port This instruction is used for writing to an output port. The address of the output port may be specified in the instruction directly or implicitly in DX. Contents of AX or AL are transferred to a directly or indirectly addressed port after execution of this instruction. The data to an odd addressed port is transferred on D_8-D_{15} while that to an even addressed port is transferred on D_0-D_7. The registers AL and AX are the allowed source operands for 8-bit and 16-bit operations respectively. If the port address is of 16 bits it must be in DX. The examples are given as shown:

Example

```
1. OUT    03H,AL    ; This sends data available in AL to a port whose
                    ; address is 03H.
2. OUT    DX, AX    ; This sends data available in AX to a port whose
                    ; address is specified implicitly in DX.
```

```
3. MOV DX, 0300H    ; The 16-bit port address is taken in DX.
   OUT DX, AX       ; Write the content of AX to a port of which address is in DX.
```

XLAT: Translate The translate instruction is used for finding out the codes in case of code conversion problems, using look up table technique.

Suppose, a hexadecimal key pad having 16 keys from 0 to F is interfaced with 8086 using 8255. Whenever a key is pressed, the code of that key (0 to F) is returned in AL. For displaying the number corresponding to the pressed key on the 7-segment display device, it is required that the 7-segment code corresponding to the key pressed is found out and sent to the display port. This translation from the code of the key pressed to the corresponding 7-segment code is performed using XLAT instruction.

For this purpose, one is required to prepare a look up table of codes, starting from an offset say 2000H, and store the 7-segment codes for 0 to F at the locations 2000H to 200FH sequentially. For executing the XLAT instruction, the code of the pressed key obtained from the keyboard (i.e. the code to be translated) is moved in AL and the base address of the look up table containing the 7-segment codes is kept in BX. After the execution of the XLAT instruction, the 7-segment code corresponding to the pressed key is returned in AL, replacing the key code which was in AL prior to the execution of the XLAT instruction. To find out the exact address of the 7-segment code from the base address of look up table, the content of AL is added to BX internally, and the contents of the address pointed to by this new content of BX in DS are transferred to AL. The following sequence of instructions perform the task.

Example

```
MOV AX, SEG TABLE    ; Address of the segment containing look-up-table
MOV DS,AX            ; is transferred in DS
MOV AL, CODE         ; Code of the pressed key is transferred in AL
MOV BX, OFFSET TABLE; Offset of the code look-up-table in BX
XLAT                 ; Find the equivalent code and store in AL
```

LEA: Load Effective Address The load effective address instruction loads the effective address formed by destination operand into the specified source register. This instruction is more useful for assembly language rather than for machine language.

Example

```
LEA BX,ADR      ; Effective address of Label ADR i.e. offset of ADR will be
                  transferred to Reg ; BX.
LEA SI, ADR[Bx]; offsetofLabelADRwillbeaddedtocontentofBxtoformeffective
                ; address and it will be loaded in SI
```

LDS/LES: Load Pointer to DS/ES This instruction loads the DS or ES register and the specified destination register in the instruction with the content of memory location specified as source in the instruction. The example in Fig. : explains the operation.

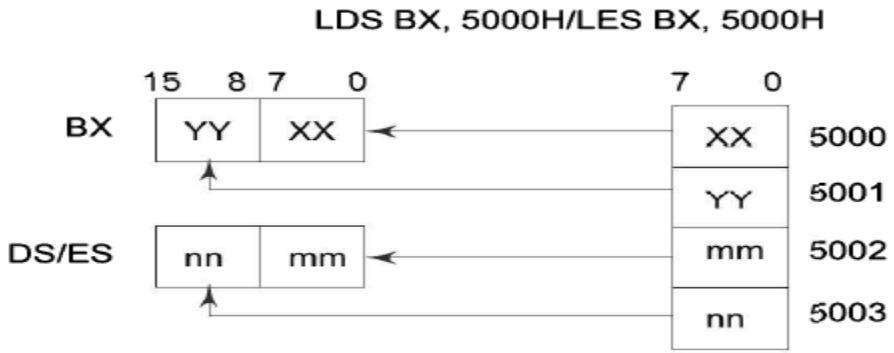


Fig. ***LDS/LES Instruction Execution***

LAHF : Load AH from Lower Byte of Flag This instruction loads the AH register with the lower byte of the flag register. This command may be used to observe the status of all the condition code flags (except over flow) at a time.

SAHF: Store AH to Lower Byte of Flag Register This instruction sets or resets the condition code flags (except overflow) in the lower byte of the flag register depending upon the corresponding bit positions in AH. If a bit in AH is 1, the flag corresponding to the bit position is set, else it is reset.

PUSHF: Push Flags to Stack The push flag instruction pushes the flag register on to the stack; first the upper byte and then the lower byte is pushed on to it. The SP is decremented by 2, for each push operation. The general operation of this instruction is similar to the PUSH operation.

POPF: Pop Flags from Stack The pop flags instruction loads the flag register completely (both bytes) from the word contents of the memory location currently addressed by SP and SS. The SP is incremented by 2 for each pop operation.

Arithmetic Instructions

These instructions perform the arithmetic operations, like addition, subtraction, multiplication and division along with the respective ASCII and decimal adjust instructions. The increment and decrement operations also belong to this type of instructions.

. The arithmetic instructions affect all the condition code flags. The operands are either the registers or memory locations or immediate data depending upon the addressing mode.

ADD: Add This instruction adds an immediate data or contents of a memory location specified in the instruction or a register (source) to the contents of another register (destination) or memory location. The result is in the destination operand. both the source and destination operands cannot be memory operands. That means memory to memory addition is not possible. Also the contents of the segment registers cannot be added using this instruction. All the condition code flags are affected, depending upon the result.

Example

```
1.ADD AX, 0100H       Immediate
2.ADD AX, BX          Register
3.ADD AX, [SI]        Register indirect
4.ADD AX, [5000H]     Direct
5.ADD [5000H], 0100H  Immediate
6.ADD 0100H           Destination AX (implicit)
```

ADC: Add with Carry This instruction performs the same operation as ADD instruction, but adds the carry flag bit (which may be set as a result of the previous calculations) to the result. All the condition code flags are affected by this instruction.

Example

```
1.ADC 0100H              Immediate (AX implicit)
2.ADC AX, BX             Register
3.ADC AX, [SI]           Register indirect
4.ADC AX, [5000H]        Direct
5.ADC [5000H], 0100H     Immediate
```

INC: Increment This instruction increases the contents of the specified register or memory location by 1. All the condition code flags are affected except the carry flag CF. This instruction adds 1 to the contents of the operand. Immediate data cannot be operand of this instruction.

Example

```
1.  INC     AX        Register
2.  INC     [BX]      Register indirect
3.  INC     [5000H]   Direct
```

DEC: Decrement The decrement instruction subtracts 1 from the contents of the specified register or memory location. All the condition code flags, except the carry flag, are affected depending upon the result. Immediate data cannot be operand of the instruction.

Example

```
1.DEC       AX        Register
2.DEC       [5000H]   Direct
```

SUB: Subtract The subtract instruction subtracts the source operand from the destination operand and the result is left in the destination operand. Source operand may be a register, memory location or immediate data and the destination operand may be a register or a memory location, but source and destination operands both must not be memory operands. Destination operand can not be an immediate data. All the condition code flags are affected by this instruction.

Example

```
1.SUB     AX, 0100H       Immediate [destination AX]
2.SUB     AX, BX          Register
3.SUB     AX, [5000H]     Direct
4.SUB     [5000H], 0100   Immediate
```

SBB: Subtract with Borrow The subtract with borrow instruction subtracts the source operand and the borrow flag (CF) which may reflect the result of the previous calculations, from the destination operand. Subtraction with borrow, here means subtracting 1 from the subtraction obtained by SUB, if carry (borrow) flag is set.

The result is stored in the destination operand. All the flags are affected (Condition code) by this instruction.

Example

```
1.SBB     AX, 0100H       Immediate [destination AX]
2.SBB     AX, BX          Register
3.SBB     AX, [5000H]     Direct
4.SBB     [5000H], 0100   Immediate
```

CMP: Compare This instruction compares the source operand, which may be a register or an immediate data or a memory location, with a destination operand that may be a register or a memory location.

For comparison, it subtracts the source operand from the destination operand but does not store the result anywhere. The flags are affected depending upon the result of the subtraction. If both of the operands are equal, zero flag is set. If the source operand is greater than the destination operand, carry flag is set or else, carry flag is reset.

Example

```
1.CMP BX, 0100H          Immediate
2.CMP AX, 0100H          Immediate
3.CMP [5000H], 0100H     Direct
4.CMP BX, [SI]           Register indirect
5.CMP BX, CX             Register
```

AAA: ASCII Adjust After Addition The AAA instruction is executed after an ADD instruction that adds two ASCII coded operands to give a byte of result in AL. The AAA instruction converts the resulting contents of AL to unpacked decimal digits. After the addition, the AAA instruction examines the lower 4 bits of AL to check whether it contains a valid BCD number in the range 0 to 9. If it is between 0 to 9 and AF is zero, AAA sets the 4 high order bits of AL to 0. The AH must be cleared before addition. If the lower digit of AL is between 0 to 9 and AF is set, 06 is added to AL.The upper 4 bits of AL are cleared and AH is incremented by one. If the value in the lower nibble of AL is greater than 9 then the AL is incremented by 06, AH is incremented by 1, the AF and CF flags are set to 1, and the higher 4 bits of AL are cleared to 0. The

remaining flags are unaffected. The AH is modified as sum of previous contents (usually 00) and the carry from the adjustment, as shown in Fig. 2.6. This instruction does not give exact ASCII codes of the sum, but they can be obtained by adding 3030H to AX.

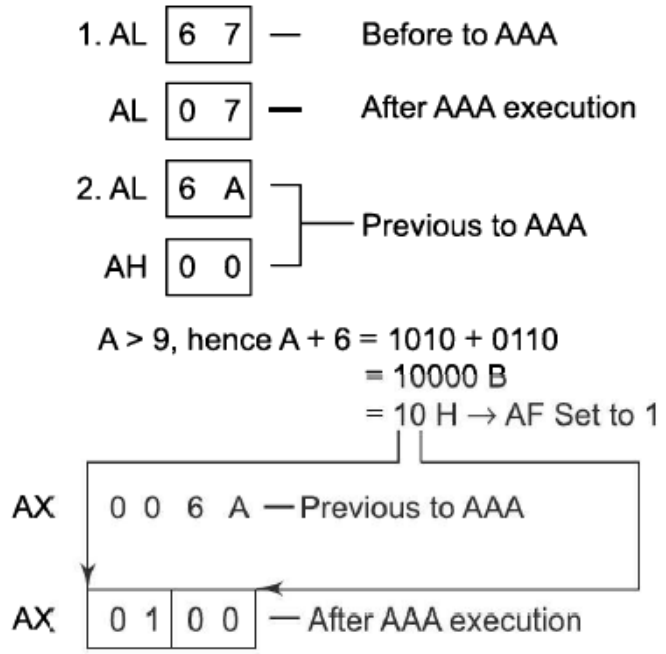


Fig. ***ASCII Adjust after Addition Instruction***

AAS: ASCII Adjust AL after Subtraction AAS instruction corrects the result in AL register after subtracting two unpacked ASCII operands. The result is in unpacked decimal format. If the lower 4 bits of AL register are greater than 9 or if the AF flag is 1, the AL is decremented by 6 and AH register is decremented by 1, the CF and AF are set to 1. Otherwise, the CF and AF are set to 0, the result needs no correction. As a result, the upper nibble of AL is 00 and the lower nibble may be any number from 0 to 9. The procedure is similar to the AAA instruction except for the subtraction of 06 from AL. AH is modified as difference of the previous contents (usually zero) of AH and the borrow for adjustment.

AAM : ASCII Adjust after Multiplication This instruction, after execution, converts the product available in AL into unpacked BCD format. The AAM—ASCII Adjust After Multiplication—instruction follows a multiplication instruction that multiplies two unpacked BCD operands, i.e. higher nibbles of the multiplication operands should be O. The multiplication of such operands is carried out using MUl instruction. Obviously the result of multiplication is available in AX.

Example

```
MOV AL, 04   ;  AL ← 04
MOV BL, 09   ;  BL ← 09
MUL BL       ;  AH-AL ← 24H (9 × 4)
AAM          ;  AH ← 03
             ;  AL ← 06
```

AAD: ASCII Adjust before Division Though the names of these two instructions (AAM and AAD) appear to be similar, there is a lot of difference between their functions. The AAD instruction converts two unpacked BCD digits in AH and AL to the equivalent binary number in AL. This adjustment must be made before dividing the two unpacked BCD digits in AX by an unpacked BCD byte. PF, SF, ZF are modified while AF, CF, OF are undefined, after the execution of the instruction AAD. The example explains the execution of the instruction. In the instruction sequence, this instruction appears before DIV instruction unlike AAM appears after MUL. Let AX contains 0508 unpacked BCD for 58 decimal, and DH contains 02H.

Example

```
AX  | 05 | 08 |
AAD result in AL | 00 | 3A |    58D = 3A H in AL
```

The result of AAD execution will give the hexadecimal number 3A in AL and 00 in AH. Note that 3A is the hexadecimal equivalent of 58 (decimal). Now, instruction DIV DH may be executed. So rather than ASCII adjust for division, it is ASCII adjust before division. All the ASCII adjust instructions are also called as unpacked BCD arithmetic instructions. Now, consider the two instructions related to packed BCD arithmetic.

DAA: Decimal Adjust Accumulator This instruction is used to convert the result of the addition of two packed BCD numbers to a valid BCD number. The result has to be only in AL. If the lower nibble is greater than 9, after addition or if AF is set, it will add 06 to the lower nibble in AL. After adding 06 in the lower nibble of AL, if the upper nibble of AL is greater than 9 or if carry flag is set, DAA instruction adds 60H to AL.

```
(ii) AL = 73        CL = 29
     ADD AL, CL     ; AL ← AL + CL
                    ; AL ← 73 + 29
                    ; AL ← 9C
     DAA            ; AL ← 02 and CF = 1
                      AL = 7 3
                         +
                    CL = 2 9
                         9 C
                       + 6
                         A 2
                     + 6 0
                    CF = 1  0 2  in AL
```

The instruction DAA affects AF, CF, PF, and ZF flags. The OF is undefined.

DAS: Decimal Adjust after Subtraction This instruction converts the result of subtraction of two packed BCD numbers to a valid BCD number. The subtraction has to be in AL only. If the lower nibble of AL is greater than 9, this instruction will subtract 06 from lower nibble of AL. If the result of subtraction sets

the carry flag or if upper nibble is greater than 9, it subtracts 60H from AL. This instruction modifies the AF, CF, SF, PF and ZF flags. The OF is undefined after DAS instruction.

```
(ii) AL = 38        CH = 6 1
     SUB AL, CH     ; AL ← D 7  CF = 1 (borrow)
     DAS            ; AL ← 7 7  (as D > 9, D - 6 = 7)
                    ; CF = 1 (borrow)
```

DAA and DAS instructions are also called packed BCD arithmetic instructions.

NEG: Negate The negate instruction forms 2's complement of the specified destination in the instruction. For obtaining 2's complement, it subtracts the contents of destination from zero. The result is stored back in the destination operand which may be a register or a memory location. If OF is set, it indicates that the operation could not be completed successfully. This instruction affects all the condition code flags.

MUL: Unsigned Multiplication Byte or Word This instruction multiplies an unsigned byte or word by the contents of AL. The unsigned byte or word may be in any one of the general purpose registers or memory locations. The most significant word of the result is stored in DX, while the least significant word of the result is stored in AX. All the flags are modified depending upon the result.

Immediate operand is not allowed in this instruction. If the most significant byte or word of the result is '0' CF and OF both will be set.

Example:

```
1. MUL BH              ; (AX) ← (AL) × (BH)
2. MUL CX              ; (DX) (AX) ← (AX) × (CX)
```

IMUL: Signed Multiplication This instruction multiplies a signed byte in source operand by a signed byte in AL or a signed word in source operand by a signed word in AX. The source can be a general purpose register, memory operand, index register or base register, but it cannot be an immediate data. In case of 32-bit results, the higher order word (MSW) is stored in DX and the lower order word is stored in AX. The AF, PF, SF, and ZF flags are undefined after IMUL. If AH and DX contain parts of 16 and 32-bit result respectively, CF and OF both will be set. The AL and AX are the implicit operands in case of 8 bits and 16 bits multiplications respectively. The unused higher bits of the result are filled by sign bit and CF, AF are cleared.

Example

```
1. IMUL     BH
2. IMUL     CX
3. IMUL     [SI]
```

CBW: Convert Signed Byte to Word This instruction converts a signed byte to a signed word. In other words, it copies the sign bit of a byte to be converted to all the bits in the higher byte of the result word. The byte to be converted must be in AL. The result will be in AX. It does not affect any flag.

CWD: Convert Signed Word to Double Word This instruction copies the sign bit of AX to all the bits of the DX register. This operation is to be done before signed division. It does not affect any flag.

DIV: Unsigned Division This instruction performs unsigned division. It divides an unsigned word or double word by a 16-bit or 8-bit operand. The dividend must be in AX for 16-bit operation and divisor may be specified using any one of the addressing modes except immediate. The result will be in AL (quotient) while AH will contain the remainder. If the result is too big to fit in AL, type 0 (divide by zero) and an interrupt is generated. In case of a double word dividend (32-bit), the higher word should be in DX and lower word should be in AX. The divisor may be specified as already explained. The quotient and the remainder, in this case, will be in AX and DX respectively. This instruction does not affect any flag.

IDIV: Signed Division This instruction performs the same operation as the DIV instruction, but with signed operands. The results are stored similarly as in case of DIV instruction in both cases of word and double word divisions. The results will also be signed numbers. The operands are also specified in the same way as DIV instruction. Divide by 0 interrupt is generated, if the result is too big to fit in AX (16-bit dividend operation) or AX and DX (32-bit dividend operation). All the flags are undefined after IDIV instruction.

Logical Instructions

These type of instructions are used for carrying out the bit by bit shift, rotate, or basic logical operations. All the condition code flags are affected depending upon the result. Basic logical operations available with 8086 instruction set are AND, OR, NOT, and XOR.

AND: Logical AND This instruction bit by bit ANDs the source operand that may be an immediate, a register or a memory location to the destination operand that may be a register or a memory location. The result is stored in the destination operand. At least one of the operands should be a register or a memory operand. Both the operands cannot be memory locations or immediate operands. An immediate operand cannot be a destination operand.

Example

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

If the content of AX is 3F0FH, the first example instruction will carry out the operation as given below. The result 3F9FH will be stored in the AX register.

```
0 0 1 1     1 1 1 1     0 0 0 0     1 1 1 1     = 3F0F H [AX]
↓ ↓ ↓ ↓     ↓ ↓ ↓ ↓     ↓ ↓ ↓ ↓     ↓ ↓ ↓ ↓       AND
0 0 0 0     0 0 0 0     0 0 0 0     1 0 0 0     = 0008 H
-----------------------------------------------------------
0 0 0 0     0 0 0 0     0 0 0 0     1 0 0 0     = 0008 H [AX]
```

The result 0008H will be in AX.

OR: Logical OR The OR instruction carries out the OR operation in the same way as described in case of the AND operation. The limitations on source and destination operands are also the same as in case of AND operation.

Example

```
1.OR AX, 0098H
2.OR AX, BX
3.OR AX, [5000H]
4.OR [5000H], 0008H
```

The contents of AX are say 3F0FH, then the first example instruction will be carried out as given below.

```
  0 0 1 1      1 1 1 1      0 0 0 0      1 1 1 1      = 3F0F H
  ↓ ↓ ↓ ↓      ↓ ↓ ↓ ↓      ↓ ↓ ↓ ↓      ↓ ↓ ↓ ↓        OR
  0 0 0 0      0 0 0 0      1 0 0 1      1 0 0 0      = 0098 H
-------------------------------------------------------------
  0 0 1 1      1 1 1 1      1 0 0 1      1 1 1 1      = 3F9F H
```

Thus the result 3F9FH will be stored in the AX register.

NOT: Logical Invert The NOT instruction complements (inverts) the contents of an operand register or a memory location, bit by bit.

Example

```
NOT AX
NOT [5000H]
```

If the content of AX is 200FH, the first example instruction will be executed as shown.

```
   AX     = 0 0 1 0    0 0 0 0    0 0 0 0    1 1 1 1
invert       ↓ ↓ ↓ ↓    ↓ ↓ ↓ ↓    ↓ ↓ ↓ ↓    ↓ ↓ ↓ ↓
             1 1 0 1    1 1 1 1    1 1 1 1    0 0 0 0
```

Result

in AX = D F F 0

The result DFF0H will be stored in the destination register AX.

XOR: Logical Exclusive OR The XOR operation is again carried out in a similar way to the AND and OR operation. The constraints on the operands are also similar. The XOR operation gives a high output, when the 2 input bits are dissimilar. Otherwise, the output is zero.

Example

```
1.XOR    AX, 0098H
```

If the content of AX is 3F0FH, then the first example instruction will be executed as explained. The result 3F97H will be stored in AX.

```
AX = 3F0FH =     0 0 1 1     1 1 1 1     0 0 0 0     1 1 1 1
       XOR       ↓ ↓ ↓ ↓     ↓ ↓ ↓ ↓     ↓ ↓ ↓ ↓     ↓ ↓ ↓ ↓
     0098H =     0 0 0 0     0 0 0 0     1 0 0 1     1 0 0 0
AX = Result =    0 0 1 1     1 1 1 1     1 0 0 1     0 1 1 1
           = 3F97H
```

TEST: Logical Compare Instruction The TEST instruction performs a bit by bit logical AND operation on the two operands. Each bit of the result is then set to 1, if the corresponding bits of both operands are 1, else the result bit is reset to 0. The result of this ANDing operation is not available for further use, but flags are affected. The affected flags are OF, CF, SF, ZF and PF. The operands may be registers, memory or immediate data.

Example

```
1.TEST     AX, BX
2.TEST     [0500], 06H
3.TEST     [BX] [DI], CX
```

SHL/SAL: Shift Logical/Arithmetic Left These instructions shift the operand word or byte bit by bit to the left and insert zeros in the newly introduced least significant bits. In case of all the SHIFT and ROTATE instructions, the count is either 1 or specified by register CL. The operand may reside in a register or a memory location but cannot be an immediate data. All flags are affected depending upon the result. Figure explains the execution of this instruction. It is to be noted here that the shift operation is through carry flag.

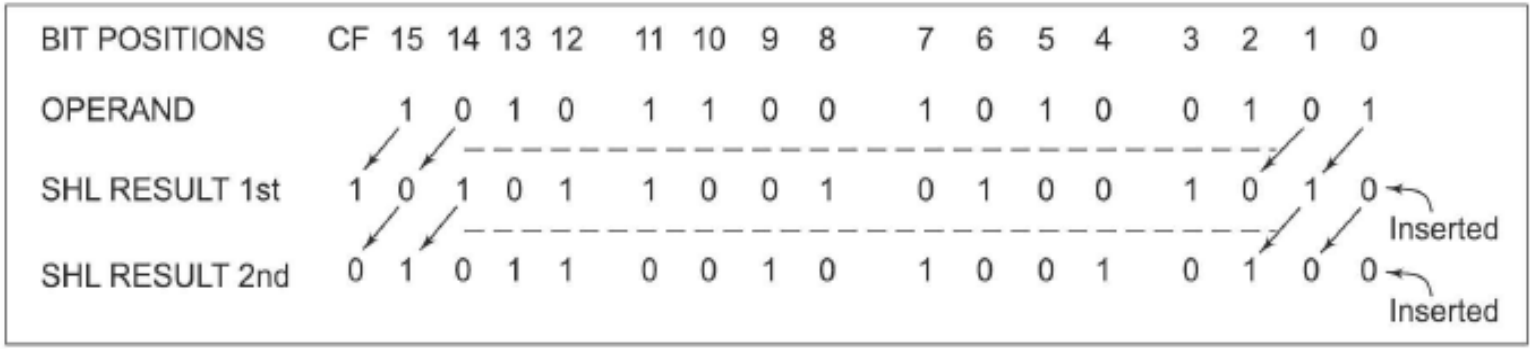


Fig. ***Execution of SHL/SAL Instruction***

SHR: Shift Logical Right This instruction performs bit-wise right shifts on the operand word or byte that may reside in a register or a memory location, by the specified count in the instruction and inserts zeros in the shifted positions. The result is stored in the destination operand. Figure explains execution of this instruction. This instruction shifts the operand through the carry flag.

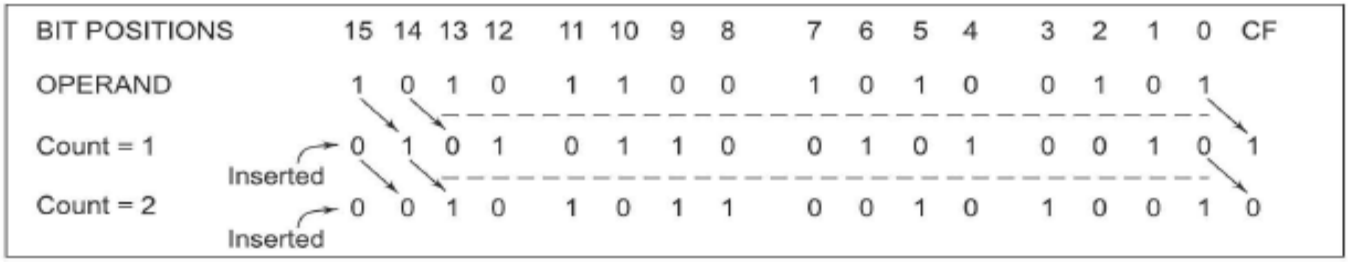


Fig. *Execution of SHR Instruction*

SAR: Shift Arithmetic Right This instruction performs right shifts on the operand word or byte, that may be a register or a memory location by the specified count in the instruction. It inserts the most significant bit of the operand in the newly inserted positions. The result is stored in the destination operand. Figure explains execution of the instruction. All the condition code flags are affected. This shift operation shifts the operand through the carry flag.

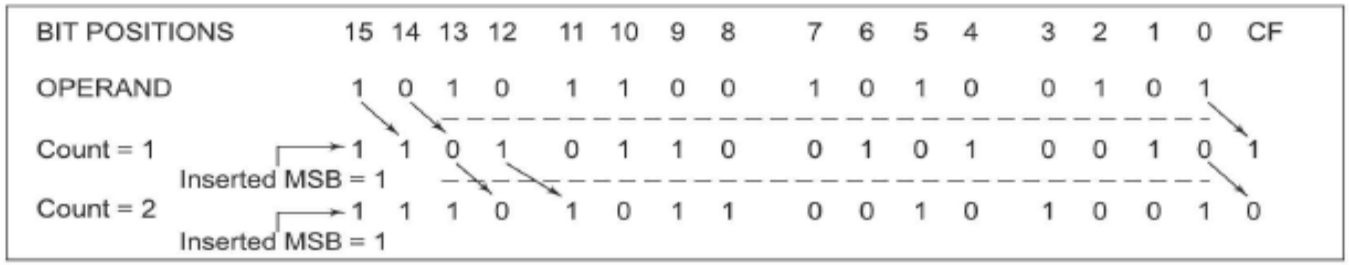


Fig. : *Execution of SAR Instruction*

Immediate operand is not allowed in any of the shift instructions.

ROR: Rotate Right without Carry This instruction rotates the contents of the destination operand to the right (bit-wise) either by one or by the count specified in CL, excluding carry. The least significant bit is pushed into the carry flag and simultaneously it is transferred into the most significant bit position at each operation. The remaining bits are shifted right by the specified positions. The PF, SF, and ZF flags are left unchanged by the rotate operation. The operand may be a register or a memory location but it cannot be an immediate operand. Figure explains the operation. The destination operand may be a register (except a segment register) or a memory location.

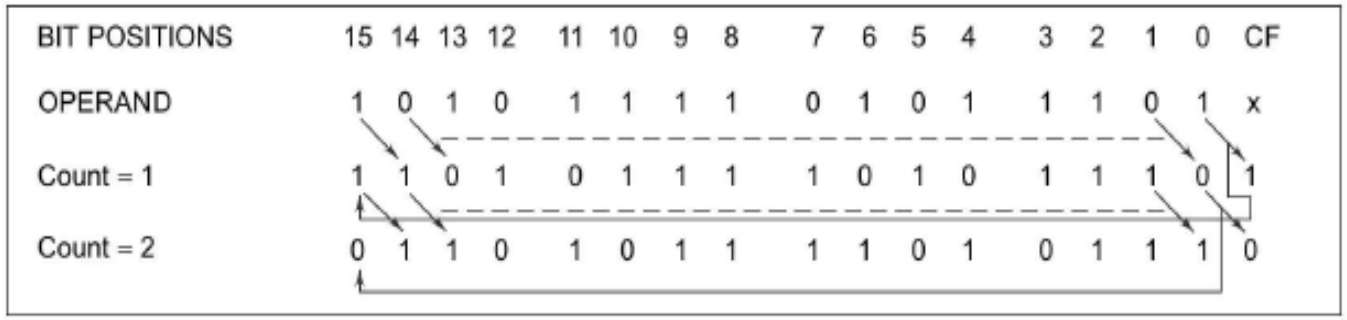


Fig. *Execution of ROR Instruction*

ROL: Rotate Left without Carry This instruction rotates the content of the destination operand to the left by the specified count (bit-wise) excluding carry. The most significant bit is pushed into the carry flag as well as the least significant bit position at each operation. The remaining bits are shifted left subsequently by the specified count positions. The PF, SF, and ZF flags are left unchanged in this rotate operation. The operand may be a register or a memory location. Figure explains the operation.

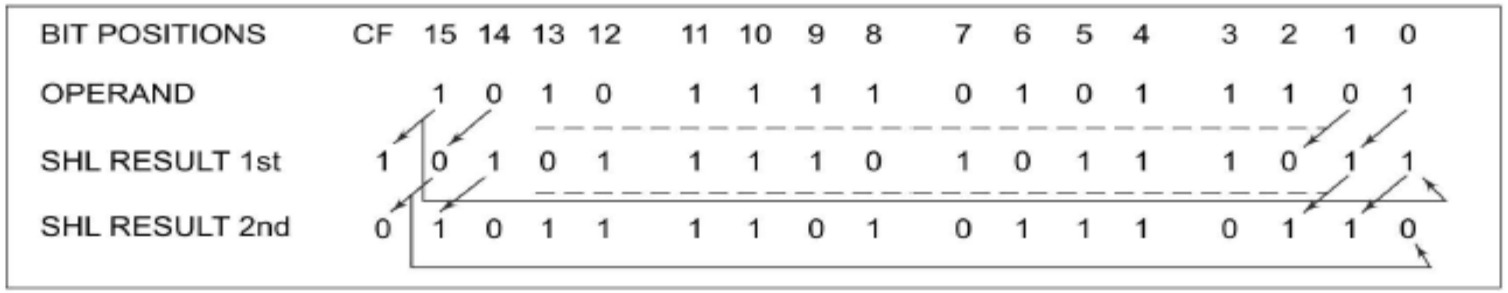


Fig. ***Execution of ROL Instruction***

RCR: Rotate Right through Carry This instruction rotates the contents (bit-wise) of the destination operand right by the specified count through carry flag (CF). For each operation, the carry flag is pushed into the MSB of the operand, and the LSB is pushed into carry flag. The remaining bits are shifted right by the specified count positions. The SF, PF, ZF are left unchanged. The operand may be a register or a memory location. Figure explains the operation.

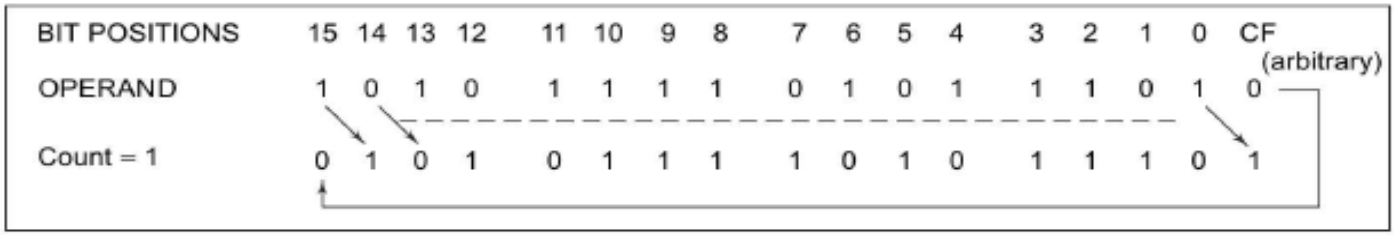


Fig. ***Execution of RCR Instruction***

RCL: Rotate Left through Carry This instruction rotates (bit-wise) the contents of the destination operand left by the specified count through the carry flag (CF). For each operation, the carry flag is pushed into LSB and the MSB of the operand is pushed into carry flag. The remaining bits are shifted left by the specified positions. The SF, PF, ZF are left unchanged. The operand may be a register or a memory location. Figure explains the operation.

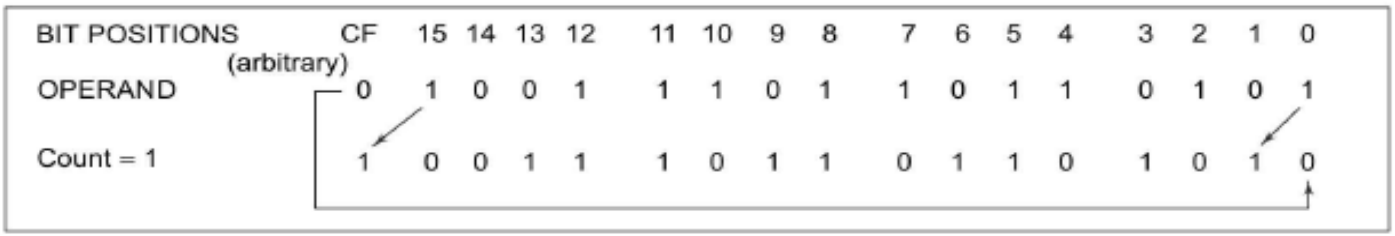


Fig. ***Execution of RCL Instruction***

The count for rotation or shifting is either 1 or is specified using register CL, in case of all the shift and rotate instructions.

String Manipulation Instructions

A series of data bytes or words available in memory at consecutive locations, to be referred to collectively or individually, are called as *byte strings* or *word strings*. For example, a string of characters may be located in consecutive memory locations, where each character may be represented by its ASCII equivalent. For referring to a string, two parameters are required, (a) starting or end address of the string and (b) length of the string. The length of a string is usually stored as count in the CX register. In case of 8085, similar structures can be set up by the pointer and counter arrangements which may be modified at each iteration, till the required condition for proceeding further is satisfied. On the other hand, the 8086 supports a set of more powerful instructions for string manipulations. The incrementing or decrementing of the pointer, in case of 8086 string instructions, depends upon the Direction Flag (DF) status. If it is a byte string operation, the index registers are updated by one. On the other hand, if it is a word string operation, the index registers are updated by two. The counter in both the cases, is decremented by one.

REP: Repeat Instruction Prefix This instruction is used as a prefix to other instructions. The instruction to which the REP prefix is provided, is executed repeatedly until the CX register becomes zero (at each iteration CX is automatically decremented by one). When CX becomes zero, the execution proceeds to the next instruction in sequence. There are two more options of the REP instruction. The first is REPE/REPZ, i.e. repeat operation while equal/zero. The second is REPNE/REPNZ allows for repeating the operation while not equal/not zero. These options are used for CMPS, SCAS instructions only, as instruction prefixes.

MOVSB/MOVSW: Move String Byte or String Word Suppose a string of bytes stored in a set of consecutive memory locations is to be moved to another set of destination locations. The starting byte of the source string is located in the memory location whose address may be computed using SI (Source Index) and DS (Data Segment) contents. The starting address of the destination locations where this string has to be relocated is given by DI (Destination Index) and ES (Extra Segment) contents. The starting address of the source string is 10H*DS+[SI], while the starting address of the destination string is 10H*ES+[DI]. The MOVSB/MOVSW instruction thus, moves a string of bytes/words pointed to by DS: SI pair (source) to the memory location pointed to by ES: DI pair (destination). The REP instruction prefix is used with MOVS instruction to repeat it by a value given in the counter (CX). The length of the byte string or word string must be stored in CX register. No flags are affected by this instruction.

After the MOVS instruction is executed once, the index registers are automatically updated and CX is decremented. The incrementing or decrementing of the pointers, i.e. SI and DI depend upon the direction flag DF. If DF is 0, the index registers are incremented, otherwise, they are decremented, in case of all the string manipulation instructions.

```
MOV CX, OFFH      ; Move length of the string to counter register CX
MOV SI, 1000H     ; Source index address 1000H is moved to SI
MOV DI, 2000H     ; Destination index address 2000H is moved to DI
CLD               ; Clear DF, i.e. set autoincrement mode
REP MOVSB         ; Move OFFH string bytes from source address to destination
```

CMPS: Compare String Byte or String Word The CMPS instruction can be used to compare two strings of bytes or words. The length of the string must be stored in the register CX. If both the byte or word strings are equal, zero flag is set. The flags are affected in the same way as CMP instruction. The DS:SI and ES:DI point to the two strings.The REP instruction prefix is used to repeat the operation till CX(counter) becomes zero or the condition specified by the REP prefix is false.

The comparison of the string starts from initial
byte or word of the string, after each comparison the index registers are updated depending upon the direction flag and the counter is decremented. This byte by byte or word by word comparison continues till a mismatch is found. When, a mismatch is found, the carry and zero flags are modified appropriately and the execution proceeds further.

```
MOV CX, 010H            ; Length of the string is moved to CX
CLD                     ; Clear DF, i.e. set autoincrement mode
REPE CMPSW              ; Compare 010H words of STRING1 and
                        ; STRING2, while they are equal, If a mismatch is found,
                        ; modify the flags and proceed with further execution
```

If both strings are completely equal, i.e. CX becomes zero, the ZF is set, otherwise, ZF is reset.

SCAS: Scan String Byte or String Word This instruction scans a string of bytes or words for an operand byte or word specified in the register AL or AX. The string is pointed to by ES:DI register pair. The length of the string is stored in CX. The DF controls the mode for scanning of the string, as stated in case of MOVSB instruction. Whenever a match to the specified operand, is found in the string, execution stops and the zero flag is set. If no match is found, the zero flag is reset. The REPNE prefix is used with the SCAS instruction. The pointers and counters are updated automatically, till a match is found.

```
  MOV AX,WORD    ; The word to be scanned for, i.e. WORD is in AL
  CLD            ; Clear DF
REPNE SCASW      ; Scan the 010H bytes of the string, till a match to
                 ; WORD is found
```

This string of instructions finds out, if it contains WORD. If the WORD is found in the word string, before CX becomes zero, the ZF is set, otherwise the ZF is reset. The scanning will continue till a match is found. Once a match is found the execution of the programme proceeds further.

LODS: Load String Byte or String Word The LODS instruction loads the AL/AX register by the content of a string pointed to by DS:SI register pair. The SI is modified automatically depending upon DF. The DF plays exactly the same role as in case of MOVSB/MOVSW instruction. If it is a byte transfer(LODSB), the SI is modified by one and if it is a word transfer(LODSW), the SI is modified by two. No other flags are affected by this instruction.

STOS: Store String Byte or String Word The STOS instruction stores the AL/AX register contents to a location in the string pointed by ES: DI register pair. The DI is modified accordingly. No flags are affected by this instruction.

The direction flag controls the string instruction execution. The source index SI and destination index DI are modified after each iteration automatically. If DF = 1, then the execution follows autodecrement mode. In this mode, SI and DI are decremented automatically after each iteration (by 1 or 2 depending upon byte or word operations). Hence, in autodecrementing mode, the strings are referred to by their ending addresses. If DF = 0, then the execution follows autoincrement mode. In this mode, SI and DI are incremented automatically (by 1 or 2 depending upon byte or word operation) after each iteration, hence the strings, in this case, are referred to by their starting addresses.

Control Transfer or Branching Instructions

The control transfer instructions transfer the flow of execution of the program to a new address specified in the instruction directly or indirectly. When this type of instruction is executed, the CS and IP registers get loaded with new values of CS and IP corresponding to the location where the flow of execution is going to be transferred. Depending upon the addressing modes specified the CS may or may not be modified. This type of instructions are classified in two types:

Unconditional Control Transfer (Branch) Instructions In case of unconditional control transfer instructions, the execution control is transferred to the specified location independent of any status or condition. The CS and IP are unconditionally modified to the new CS and IP.

Conditional Control Transfer (Branch) Instructions In the conditional control transfer instructions, the control is transferred to the specified location provided the result of the previous operation satisfies a particular condition, otherwise, the execution continues in normal flow sequence. The results of the previous operations are replicated by condition code flags. In other words, using this type of instruction the control will be transferred to a particular specified location, if a particular flag satisfies the condition.

Unconditional Branch Instructions

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

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

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

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

Example

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

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

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

INTO: Interrupt on Overflow This command is executed, when the overflow flag OF is set. The new contents of IP and CS are taken from the address 0000:0010 as explained in INT type instruction. This is equivalent to a Type 4 interrupt instruction.

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

JUMP	DISP 8-bit				Intrasegment, relative, short jump
JUMP	DISP.16-bit (LB)	DISP.16-bit (HB)			Intrasegment, relative, short jump
JUMP	IP (LB)	IP (HB)	CS (LB)	S (HB)	Intrasegment, direct, far jump

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

LOOP: Loop Unconditionally This instruction executes the part of the program from the label or address specified in the instruction up to the loop instruction, CX number of times.

At each iteration, CX is decremented automatically. In other words, this instruction implements DECREMENT COUNTER and JUMP IF NOT ZERO structure.

Example

```
        MOV   CX, 0005   ; Number of times in CX
        MOV   BX, 0FF7H  ; Data to BX
Label : MOV   AX, CODE1
        OR    BX, AX
        AND   DX, AX
        Loop  Label
```

The execution proceeds in sequence, after the loop is executed, CX number of times. If CX is already 00H, the execution continues sequentially. No flags are affected by this instruction.

Conditional Branch Instructions

When these instructions are executed, execution control is transferred to the address specified relatively in the instruction, provided the condition implicit in the opcode is satisfied. If not the execution continues sequentially. The conditions, here, means the status of condition code flags. These type of instructions do not affect any flag. The address has to be specified in the instruction relatively in terms of displacement which must lie within –80H to 7FH (or – 128 to 127) bytes from the address of the branch instruction. In other words, only short jumps can be implemented using conditional branch instructions. A label may represent the displacement, if it lies within the above specified range. The different 8086/8088 conditional branch instructions and their operations are listed in Table

Conditional Branch Instructions

	Mnemonic	*Displacement*	*Operation*
1.	JZ/JE	Label	Transfer execution control to address 'Label', if ZF=1.
2.	JNZ/JNE	Label	Transfer execution control to address 'Label', if ZF=0.
3.	JS	Label	Transfer execution control to address 'Label', if SF=1.
4.	JNS	Label	Transfer execution control to address 'Label', if SF=0.
5.	JO	Label	Transfer execution control to address 'Label', if OF=1.
6.	JNO	Label	Transfer execution control to address 'Label', if OF=0.
7.	JP/JPE	Label	Transfer execution control to address 'Label', if PF=1.
8.	JNP	Label	Transfer execution control to address 'Label', if PF=0.
9.	JB/JNAE/JC	Label	Transfer execution control to address 'Label', if CF=1.
10.	JNB/JAE/JNC	Label	Transfer execution control to address 'Label', if CF=0.
11.	JBE/JNA	Label	Transfer execution control to address 'Label', if CF=1 or ZF=1.
12.	JNBE/JA	Label	Transfer execution control to address 'Label', if CF=0 or ZF=0.
13.	JL/JNGE	Label	Transfer execution control to address 'Label', if neither SF=1 nor OF=1.
14.	JNL/JGE	Label	Transfer execution control to address 'Label', if neither SF=0 nor OF=0.
15.	JLE/JNC	Label	Transfer execution control to address 'Label', if ZF=1 or neither SF nor OF is 1.
16.	JNLE/JE	Label	Transfer execution control to address 'Label', if ZF=0 or at least any one of SF and OF is 1(Both SF and OF are not 0).

While the remaining instructions can be used for unsigned binary operations, the last four instructions are used in case of decisions based on signed binary number operations. The terms above and below are generally used for unsigned numbers, the terms less and greater are used for signed numbers. A conditional jump instruction, that does not check status flags for condition testing, is given as follows:

```
JCXZ    'Label'   Transfer execution control
                  to address 'Label', if CX=0.
```

The conditional LOOP instructions are given in Table with their meanings. These instructions may be used for implementing structures like DO_WHILE, REPEAT_UNTIL, etc.

Table ***Conditional Loop Instructions***

Mnemonic	*Displacement*	*Operation*
LOOPZ/LOOPE (Loop while ZF = 1; equal)	Label	Loop through a sequence of instructions from 'Label' while ZF=1 and CX ¹ 0.
LOOPNZ/LOOPENE (Loop while ZF = 0; not equal)	Label	Loop through a sequence of instructions from 'Label' while ZF=0 and CX ¹ 0.

Flag Manipulation and Processor Control Instructions

These instructions control the functioning of the available hardware inside the processor chip. These are categorized into two types; (a) flag manipulation instructions and (b) machine control instructions. *The flag manipulation instructions directly modify some of the flags of 8086. The machine control instructions control the bus usage and execution*. The flag manipulation instructions and their functions are listed in Table

Table *Flag Manipulation Instructions*

CLC	-	Clear carry flag
CMC	-	Complement carry flag
STC	-	Set carry flag
CLD	-	Clear direction flag
STD	-	Set direction flag
CLI	-	Clear interrupt flag
STI	-	Set interrupt flag

These instructions modify the Carry (CF), Direction (DF) and Interrupt (IF) flags directly. The DF and IF, which may be modified using the flag manipulation instructions, further control the processor operation; like interrupt responses and autoincrement or autodecrement modes. Thus, the respective instructions may also be called machine or processor control instructions. The other flags can be modified using POPF and SAHF instructions, which are termed as data transfer instructions, i No direct instructions, are available for modifying the status flags except carry flag.

The machine control instructions supported by 8086 and 8088 are listed in Table along with their functions. They do not require any operand.

		Machine Control Instructions
WAIT	-	Wait for Test input pin to go low
HLT	-	Halt the processor
NOP	-	No operation
ESC	-	Escape to external device like NDP (numeric co-processor)
LOCK	-	Bus lock instruction prefix.

after executing the HLT instruction, the processor enters the halt state. The two ways to pull it out of the halt state are to reset the processor or to interrupt it. When NOP instruction is executed, the processor does not perform any operation till 4 clock cycles, except for incrementing the IP by one. It then continues with further execution after 4 clock cycles. ESC instruction when executed, frees the bus for an external master like a coprocessor or peripheral devices. The LOCK prefix may appear with another instruction. When it is executed, the bus access is not allowed for another master till the lock prefixed instruction is executed completely. This instruction is used in case of programming for multiprocessor systems. The WAIT instruction when executed, holds the operation of processor with the current status till the logic level on the $\overline{\text{TEST}}$ pin goes low. The processor goes on inserting WAIT states in the instruction cycle, till the $\overline{\text{TEST}}$ pin goes low. Once the $\overline{\text{TEST}}$ pin goes low, it continues further execution.

3)ASSEMBLER DIRECTIVES

An assembler is a program used to convert an assembly language program into the equivalent machine code modules which may further be converted to executable codes. It decides the address of each label and substitutes the values for each of the constants and variables. It then forms the machine code for the mnemonics and data in the assembly language program. the assembler may find out syntax errors. For completing all these tasks, an assembler needs some hints from the programmer, i.e. the required storage for a particular constant or a variable, logical names of the segments, types of the different routines and modules, end of file, etc. These types of hints are given to the assembler using some predefined alphabetical strings called *assembler directives*, which help the assembler to correctly understand the assembly language programs to prepare the codes.

Another type of hint which helps the assembler to assign a particular constant with a label or initialise particular memory locations or labels with constants is an *operator*.

Eg of assembler — Microsoft Macro Assembler or Turbo Assembler.

DB: Define Byte The DB directive is used to reserve byte or bytes of memory locations in the available memory. While preparing the EXE file, this directive directs the assembler to allocate the specified number of memory bytes to the said data type that may be a constant, variable, string, etc.

Example

```
VALUE    DB  50H
```

This statement directs the assembler to reserve 50H memory bytes and leave them uninitialised for the variable named VALUE.

DW: Define Word The DW directive serves the same purposes as the DB directive, but it makes the assembler reserve the number of memory words (16-bit) instead of bytes.

Example

DW directive is explained with the DUP operator.

```
WDATA    DW 5 DUP (6666H)
```

This statement reserves five words, i.e. 10-bytes of memory for a word lable WDATA and initialises all the word locations with 6666H.

DQ: Define Quadword This directive is used to direct the assembler to reserve 4 words (8 bytes) of memory for the specified variable and may initialise it with the specified values.

DT: Define Ten Bytes The DT directive directs the assembler to define the specified variable requiring 10-bytes for its storage and initialise the 10-bytes with the specified values.

ASSUME: Assume Logical Segment Name The ASSUME directive is used to inform the assemble, the names of the logicals segments to be assumed for different segments used in the program.

. The statement ASSUME CS : CODE directs the assembler that the machine codes are available in a segment named CODE, and hence the CS register is to be loaded with the address (segment) allotted by the operating system for the label CODE,

ASSUME DS : DATA indicates to the assembler that the data items related to the program, are available in a logical segment named DATA, and the DS register is to be initialised by the segment address

value decided by the operating system for the data segment, It then considers the segment DATA as a default data segment for each memory operation, related to the data and the segment CODE as a source segment for the machine codes of the program. The ASSUME statement is a must at the starting of each assembly language program, ١

END: END of Program The END directive marks the end of an assembly language program. When the assembler comes across this END directive, it ignores the source lines available later on.

ENDP: END of Procedure , the subroutines are called proce-
dures. They may be independent program modules which return particular results or values to the calling programs. The ENDP directive is used to indicate the end of a procedure. A procedure is usually assigned a name, i.e. label. To mark the end of a particular procedure, the name of the procedure, i.e. label may appear as a prefix with the directive ENDP.

```
PROCEDURE STAR
      ⋮
    STAR ENDP
```

ENDS: END of Segment This directive marks the end of a logical segment. The logical segments are assigned with the names using the ASSUME directive.
. Whatever are the contents of the segments, they should appear in the program before ENDS.

```
DATA        SEGMENT
              ⋮
DATA        ENDS
ASSUME      CS : CODE, DS : DATA
```

data related to the program must lie between the DATA SEGMENT and DATA ENDS statements.
all the executable instructions must lie between CODE SEGMENT and CODE ENDS statements.

EVEN: Align on Even Memory Address The assembler, while starting the assembling procedure of any program, initialises a location counter and goes on updating it, as the assembly proceeds.

The EVEN directive updates the location counter to the next even address, if the current location counter contents are not even, and assigns the following routine or variable or constant to that address.

```
        EVEN
PROCEDURE     ROOT
                ⋮
        ROOT    ENDP
```

The above structure shows a procedure ROOT that is to be aligned at an even address.
When the assembler comes across the directive EVEN, it
checks the contents of the location counter. If it is odd, it is updated to the next even value and then the ROOT procedure is assigned to that address, i.e. the updated contents of the location counter. If the content of the location counter is already even, then the ROOT procedure will be assigned with the same address.

EQU: Equate The directive EQU is used to assign a label with a value or a symbol. The use of this directive is just to reduce the recurrence of the numerical values or constants in a program code. The recurring value is assigned with a label, and that label is used in place of that numerical value, throughout the program.

Using the EQU directive, even an instruction mnemonic can be assigned with a label, which can then be used in the program in place of that mnemonic.

Example

```
LABEL       EQU     0500H
```

The statement assigns the constant 500H with the label LABEL,

EXTRN: External and PUBLIC: Public The directive EXTRN informs the assembler that the names, procedures and labels declared after this directive have already been defined in some other assembly language modules. While in the other module, where the names, procedures and labels actually appear, they must be declared public, using the PUBLIC directive. If one wants to call a procedure FACTORIAL appearing in MODULE1 from MODULE 2; in MODULE1, it must be declared PUBLIC using the statement PUBLIC FACTORIAL and in module 2, it must be declared external using the declaration EXTRN FACTORIAL. ' Consider the following declarations.

```
MODULE1     SEGMENT
PUBLIC      FACTORIAL FAR
MODULE1     ENDS
MODULE2     SEGMENT
EXTRN       FACTORIAL FAR
MODULE2     ENDS
```

GROUP: Group the Related Segments

This directive is used to inform the assembler to form a logical group of the following segment names.

the group declared segments or operands must lie within a 64Kbyte memory segment.

Eg:

```
PROGRAM GROUP CODE, DATA, STACK
```

CODE, DATA and STACK

segment must lie within a 64kbyte memory segment that is named as PROGRAM. for the ASSUME statement, one can use the label PROGRAM rather than CODE, DATA and STACK

LABEL: Label The Label directive is used to assign a name to the current content of the location counter.

As the program assembly proceeds, the contents of the location counter are updated. The assembler assigns

the declared label with the current contents of the location counter. The type of the label must be specified, i.e. whether it is a NEAR or a FAR label, BYTE or WORD label, etc.

The label CONTINUE can be used for a FAR jump,

The LABEL directive can be used to refer to the data segment along with the data type, byte or word as shown.

```
DATA        SEGMENT
DATAS DB 50H DUP (?)
DATA-LAST LABEL BYTE FAR
DATA ENDS
```

After reserving 50H locations for DATAS, the next location will be assigned a label DATA-LAST and its type will be byte and far.

LENGTH: Byte Length of a Label . This is used to refer to the length of a data array or a string.

```
MOV CX, LENGTH ARRAY
```

This statement, when assembled, will substitute the length of the array ARRAY in bytes, in the instruction.

LOCAL The lables, variables, constants or procedures declared LOCAL in a module are to be used only by that particular module. Thus the same label may

serve different purposes for different modules of a program. With a single declaration statement, a number of variables can be declared local,

```
LOCAL a, b, DATA, ARRAY, ROUTINE
```

NAME: Logical Name of a Module The NAME directive is used to assign a name to an assembly language program module. The module, may now be referred to by its declared name. '

OFFSET: Offset of a Label When the assembler comes across the OFFSET operator along with a label, it first computes the 16-bit displacement of the particular label, and replaces the string 'OFFSET LABEL' by the computed displacement.

Example

```
CODE SEGMENT
MOV SI, OFFSET LIST
CODE ENDS
DATA SEGMENT
LIST DB 10H
DATA ENDS
```

ORG : Origin The ORG directive directs the assembler to start the memory allotment for the particular segment, block or code from the declared address in the ORG statement. If an ORG

200H statement is present at the starting of the code segment of that module, then the code will start from 200H address in code segment. In other words, the location counter will get initialised to the address 0200H instead of 0000H.

PROC: Procedure The PROC directive marks the start of a named procedure in the statement.

For example, the statement RESULT PROC NEAR marks the start of a routine RESULT, which is to be called by a program located in the same segment of memory. The FAR directive is used for the procedures to be called by the programs located in different segments of memory.

Example

```
RESULT    PROC   NEAR
ROUTINE   PROC   FAR
```

PTR: Pointer The POINTER operator is used to declare the type of a label, variable or memory operand. The operator PTR is prefixed by either BYTE or WORD. If the prefix is BYTE, then the particular label, variable or memory operand is treated as an 8-bit quantity, while if WORD is the prefix, then it is treated as a 16-bit quantity.

Example

```
INC BYTE PTR [BX]-          Increments byte contents of memory
                            location addressed by BX

INC WORD PTR [3000H] -      Increments word contents of memory
                            location 3000H considering contents
                            of 3000H (lower byte) and 3001H
                            (higher byte) as a 16-bit number
```

In case of JMP instructions, the PTR operator is used to specify the type of the jump, i.e. near or far,

```
JMP NEAR PTR [BX]-NEAR Jump
JMP FAR PTR [BX]-FAR Jump.
```

PUBLIC the PUBLIC directive is used along with the EXTRN directive. This informs the assembler that the labels, variables, constants, or procedures declared PUBLIC may be accessed by other assembly modules to form their codes, but while using the PUBLIC declared lables, variables, constants or procedures the user must declare them externals using the EXTRN directive.

SEG: Segment of a Label The SEG operator is used to decide the segment address of the label, variable, or procedure and substitutes the segment base address in place of ''SEG'' label.

Example

```
MOV AX, SEG ARRAY  ; This statement moves the segment address of ARRAY in
MOV DS, AX         ; which it is appearing, to register AX and then to DS.
```

SEGMENT: Logical Segment The SEGMENT directive marks the starting of a logical segment. The started segment is also assigned a name, i.e. label, by this statement.
In some cases, the segment may be assigned a type like PUBLIC or GLOBAL

```
EXE.CODE SEGMENT GLOBAL; Start of Segment named EXE.CODE,
                       ; that can be accessed by any other module.
EXE.CODE ENDS          ; END of EXE.CODE logical segment.
```

SHORT The SHORT operator indicates to the assembler that only one byte is required to code the displacement for a jump (i.e. displacement is within –128 to +127 bytes from the address of the byte next to the jump opcode).

```
JMP SHORT LABEL
```

TYPE The TYPE operator directs the assembler to decide the data type of the specified label and replaces the 'TYPE' label by the decided data type. For the word type variable, the data type is 2, for double word type, it is 4, and for byte type, it is 1. Suppose, the STRING is a word array. The instruction MOV AX, TYPE STRING moves the value 0002H in AX.

GLOBAL The labels, variables, constants or procedures declared GLOBAL may be used by other modules of the program.
following statement declares the procedure ROUTINE as a global label.

```
ROUTINE PROC     GLOBAL
```

'+ & –' Operators These operators represent arithmetic addition and subtraction respectively and are typically used to add or subtract displacements (8 or 16 bit) to base or index registers or stack or base pointers

Example

```
MOV AL, [ SI +2 ]
MOV BX, [ OFFSET LABEL + 10 H ]
```

FAR PTR This directive indicates the assembler that the label following FAR PTR is not available within the same segment and the address of the label is of 32-bits i.e. 2 bytes offset followed by 2 bytes segment address.

Example

```
JMP FAR PTR LABEL
CALL FAR PTR ROUTINE
```

Here, the target address is going to require four bytes;

Lower byte of offset, higher byte of offset, lower byte of segment and higher byte of segment; indicating intersegment addressing mode.

NEAR PTR This directive indicates that the label following NEAR PTR is in the same segment and needs only 16 bit i.e. 2 byte offset to address it.

Example

```
JMP NEAR PTR LABEL
CALL NEAR PTR ROUTINE
```

If a label is not preceded by NEAR PTR or FAR PTR, then it is by default considered a NEAR PTR label and two bytes are reserved by the assembler for its address during the process of assembling.

A FEW MACHINE LEVEL PROGRAMS

Example

Write a program to add a data byte located at offset 0500H in 2000H segment to another data byte available at 0600H in the same segment and store the result at 0700H in the same segment.

```
MOV   AX, 2000H    ; Initialising DS with value
MOV   DS, AX       ; 2000H
MOV   AX, [500H]   ; Get first data byte from 0500H
                     offset
ADD   AX, [600H]   ; Add this to the second byte
                     from 0600H
MOV   [700H], AX   ; Store AX in 0700H (result).
HLT                ; Stop
```



Fig. *Flow Chart for Example*

Example

Move a byte string, 16-bytes long, from the offset 0200H to 0300H in the segment 7000H.

Solution According to the program statement, a string that is 16-bytes long is available at the offset address 0200H in the segment 7000H. The required program should move this complete string at offset 0300H, in the same segment.

```
MOV AX,7000H
MOV DS,AX
MOV ES,AX
MOV CX,0010H
MOV SI,0200H
MOV DI,0300H
CLD
REP MOVSB
HLT
```

Example

Find out the largest number from an unordered array of sixteen 8-bit numbers stored sequentially in the memory locations starting at offset 0500H in the segment 2000H.

Solution . The first number of the array is taken in a register, say AL. The second number of the array is then compared with the first one. If the first one is greater than the second one, it is left unchanged. However, if the second one is greater than the first, the second number replaces the first one in the AL register. The procedure is repeated for every number in the array and thus it requires 15 iterations. At the end of 15th iteration the largest number will reside in the register AL.

```
        MOV  CX, OF H   ; Initialize counter for number of iterations
        MOV  AX, 2000H  ; Initialize data segment
        MOV  DS, AX     ;
        MOV  SI, 0500H  ; Initialize source pointer
        MOV  AL, [SI]   ; Take first number in AL
BACK :  INC  SI         ; Increment source pointer
        CMP  AL, [SI]   ; Compare next number with the previous
        JNC  NEXT       ; If the next number is larger
        MOV  AL, [SI]   ; replace the previous one with the next
NEXT :  LOOP BACK       ; Repeat the procedure 15 times
        HLT
```

- # Assembly language programs

Write a program for addition of two numbers.

Solution The following program adds two 16-bit operands. There are various methods of specifying operands depending upon the addressing modes

```
ASSUME CS:CODE, DS:DATA
DATA SEGMENT
OPR1 DW 1234H                    ; 1st operand
OPR2     DW 0002H                ; 2nd operand
RESULT   DW 01 DUP(?)            ; A word of memory reserved for re-
                                   sult
DATA     ENDS
CODE     SEGMENT
START:   MOV AX, DATA            ; Initialize data segment
         MOV DS, AX              ;
         MOV AX, OPR1            ; Take 1st operand in AX
         MOV BX, OPR2            ; Take 2nd operand in BX
         CLC                     ; Clear previous carry if any
         ADD AX, BX              ; Add BX to AX
         MOV DI, OFFSET RESULT   ; Take offset of RESULT in DI
         MOV [DI], AX            ; Store the result at memory address in DI
         MOV AH, 4CH             ; Return to DOS prompt
         INT 21H
CODE     ENDS                    ; CODE segment ends
         END START               ; Program ends
```

A program to find out the largest number from a given unordered array of 8-bit numbers, stored in the locations starting from a known address.

Solution Compare the *i*th number of the series with the (*i*+1)th number using CMP instruction. It will set the flags appropriately, depending upon whether the ith number or the (*i*+1)th number is greater. If the *i*th number is greater than (*i*+1)th, leave it in AX (any register may be used). Otherwise, load the (*i*+1)th number in AX, replacing the *i*th number in AX. The procedure is repeated till all the members in the array have been compared.

```
ASSUME CS:CODE, DS:DATA
DATA SEGMENT                          ; Data segment starts
LIST DB 52H, 23H, 56H, 45H, --        ; List of byte numbers
COUNT EQU OF                          ; Number of bytes in the list
LARGEST DB 01H DUP(?)                 ; One byte is reserved for the largest
                                        number.
DATA ENDS                             ; Data segment ends
CODE SEGMENT                          ; Code segment starts.
START:      MOV AX, DATA              ; Initialize data segment.
            MOV DS, AX
            MOV SI, OFFSET LIST
            MOV CL, COUNT             ; Number of bytes in CL.
            MOV AL, [SI]              ; Take the first number in AL
AGAIN:      CMP AL, [SI+1]            ; and compare it with the next number.
            JNL NEXT

            MOV AL, [SI+1]
NEXT:       INC SI                    ; Increment pointer to the byte list.
            DEC CL                    ; Decrement counter.
            JNZ AGAIN                 ; If all numbers are compared, point to
                                        result
            MOV SI, OFFSET LARGEST ; destination and store it.
            MOV [SI], AL
            MOV AH, 4CH               ; Return to DOS.
            INT 21H
            CODE ENDS
END         START
```

Program to find the largest of unordered array of series of words using subroutine

```
ASSUME CS:CODE, DS:DATA
DATA SEGMENT
LIST DW 1234H, 2354H, 0056H, 045AH, -
COUNT EQU OF
LARGEST DW 01H DUP(?)
DATA ENDS
CODE SEGMENT
START:      MOV AX, DATA
            MOV DS, AX
            MOV SI, OFFSET LIST

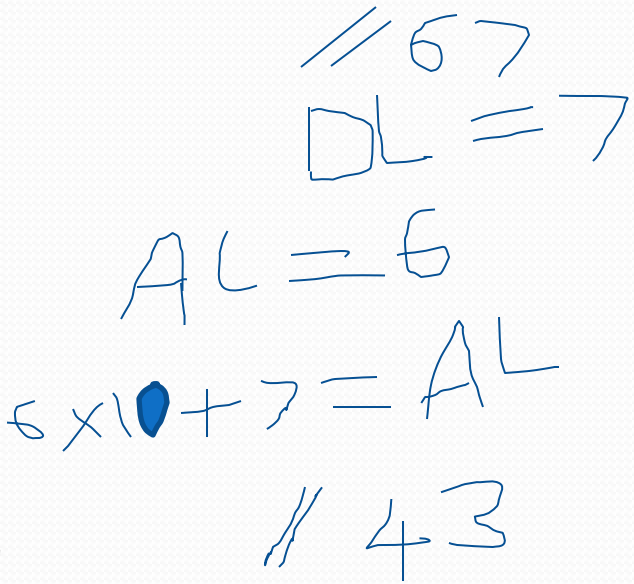
                MOV CL, COUNT
                MOV AX, [SI]

 CALL LARGE
 MOV SI,OFFSET LARGEST
 MOV [SI],AX
MOV AH,4CH
INT 21H
PROCEDURE   LARGE NEAR


 AGAIN:         CMP AX, [SI+2]
                JNL NEXT
                MOV AX, [SI+2]
 NEXT:          INC SI
                INC SI
                DEC CL
                JNZ AGAIN
RET
LARGE ENDP

 CODE           ENDS
                END      START
```

- Write a ALP to perform BCD to binary conversion
- ASSUME CS:CODE
- CODE SEGMENT
- ORG 1000H
- MOV BX,1100H
- MOV AL,[BX]
- MOV DL,AL
- AND DL, OFH
- MOV CL,4
- ROR AL,CL
- AND AL,OFH
- MOV DH,OAH
- MUL DH
- ADD AL,DL
- MOV [BX+1],AL
- HLT
- CODE ENDS
- END



Write a program for the addition of two 3 × 3 matrices. The matrices are stored in the form of lists (row wise). Store the result of addition in the third list.

Solution In the addition of two matrices, the corresponding elements are added to form the corresponding elements of the result matrix as shown:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11}+a_{12}+b_{12}+a_{13}+b_{13} \\ a_{21}+b_{21}+a_{22}+b_{22}+a_{23}+b_{23} \\ a_{31}+b_{31}+a_{32}+b_{32}+a_{33}+b_{33} \end{bmatrix}$$

$$[A] \quad + \quad [B] \quad = \quad [A + B]$$

The matrix *A* is stored in the memory at an offset MAT1, as given:

a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}, a_{31}, a_{32}, a_{33}, etc.

A total of 3 × 3 = 9 additions are to be done. The assembly language program is written as

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
            DIM EQU 09H
            MAT1 DB 01, 02, 03, 04, 05, 06, 07, 08, 09
            MAT2 DB 01, 02, 03, 04, 05, 06, 07, 08, 09
            RMAT3 DW 09H DUP(?)
DATA ENDS
CODE SEGMENT
START:      MOV AX, DATA
            MOV DS, AX
            MOV CX, DIM
            MOV SI, OFFSET MAT1
            MOV DI, OFFSET MAT2
            MOV BX, OFFSET RMAT3
```

```
NEXT:       XOR AX, AX
            MOV AL, [SI]
            ADD AL, [DI]
            MOV WORD PTR [BX], AX
            INC SI
            INC DI
            ADD BX, 02
            LOOP NEXT
            MOV AH, 4CH
            INT 21H
CODE        ENDS
            END START
```

Write an assembly language program to arrange a given series of hexadecimal bytes in ascending order.

Solution The algorithm used here is called *bubble sorting*. To start with, the first number of the series is compared with the second one. If the first number is greater than second, exchange their positions in the series otherwise leave the positions unchanged. Then, compare the second number in the recent form of the series with third and repeat the exchange part that you have carried out for the first and the second number, and for all the remaining numbers of the series. Repeat this procedure for the complete series $(n-1)$ times. After $(n-1)$ iterations, you will get the largest number at the end of the series, where n is the length of the series. Continue this till the complete series is arranged in ascending order. Let the series be as given:

```
53 , 25 , 19, 02          n = 4
25 , 53 , 19, 02          1st operation
25 , 19 , 53 , 02         2nd operation
25 , 19 , 02 , 53         3rd operation
largest no.          ⇒    4 – 1 = 3 operations
19 , 25 , 02 , 53         1st operation
19 , 02 , 25 , 53         2nd operation
2nd largest number   ⇒    4 – 2 = 2 operations
02 , 19 , 25 , 53         1st operation
3rd largest number   ⇒    4 – 3 = 1 operations
```

```
ASSUME CS:CODE, DS:DATA
DATA SEGMENT
LIST DW 53H, 25H, 19H, 02H
COUNT EQU 04
DATA ENDS
CODE SEGMENT
START:      MOV AX, DATA
            MOV DS, AX
            MOV DX, COUNT-1
AGAINO:     MOV CX, DX
            MOV SI, OFFSET LIST
AGAIN1:     MOV AX, [SI]
```

```
            CMP AX, [SI+2]
            JL  PR1
            XCHG [SI+2], AX
            XCHG [SI], AX
PR1:        ADD SI, 02
            LOOP AGAIN1
            DEC DX
            JNZ AGAINO
            MOV AH, 4CH
            INT 21H
CODE        ENDS
            END START
```

Display the message "The study of microprocessors is interesting." on the CRT screen of a micro-computer.

Solution

```
ASSUME    CS :CODE, DS :DATA
DATA      SEGMENT
MESSAGE   DB ODH, OAH,"  STUDY OF MICROPROCESSORS IS INTERESTING",
                         ODH, OAH, "$"
                         ;PREPARING STRING OF THE MESSAGE
DATA      ENDS
CODE      SEGMENT
START:    MOV AX, DATA    ;INITIALIZE DS
          MOV DS, AX
          MOV AH, 09H     ;SET FUNCTION VALUE  FOR DISPLAY
          MOV DX, OFFSET MESSAGE
          INT 21H         ;POINT TO MESSAGE AND RUN
          MOV AH, 4CH     ;THE INTERRUPT
          INT 21H         ;RETURN TO DOS
CODE      ENDS            ;STOP
          END START
```

A program to find out the number of even and odd numbers from a given series of 16-bit hexadecimal numbers.

Solution The simplest logic to decide whether a binary number is even or odd, is to check the least significant bit of the number. If the bit is zero, the number is even, otherwise it is odd. Check the LSB by rotating the number through carry flag, and increment even or odd number counter.

```
ASSUME CS:CODE, DS:DATA
DATA SEGMENT
LIST DW 2357H, 0A579H, 0C322H, 0C91EH, 0C000H, 0957H
COUNT EQU 006H
DATA ENDS
CODE SEGMENT
START:      XOR BX, BX
            XOR DX, DX
            MOV AX, DATA
            MOV DS, AX
            MOV CL, COUNT
            MOV SI, OFFSET LIST
AGAIN:      MOV AX, [SI]
            ROR AX, 01
            JC ODD
            INC BX
            JMP NEXT
ODD:        INC DX
NEXT:       ADD SI, 02
            DEC CL
            JNZ AGAIN
            MOV AH, 4CH
            INT 21H
            CODE ENDS
            END START
```

Write a program to find out whether a given byte is in the string or not. If it is in the string, find out the relative address of the byte from the starting location of the string.

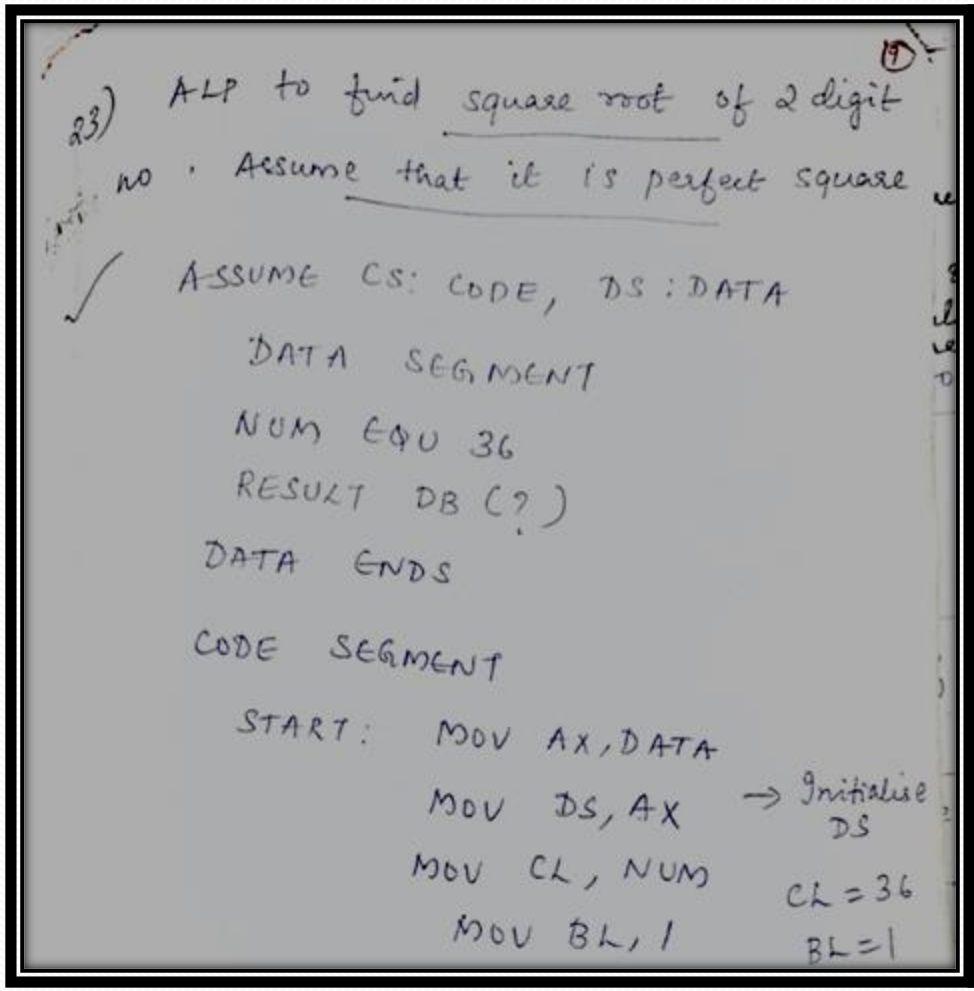
Solution The given string is scanned for the given byte. If it is found in the string, the zero flag is set; else, it is reset. Use of the SCASB instruction is there. A count should be maintained to find out the relative address of the byte found out. Note that, in this program, the code segment is written before the data segment.

```
        ASSUME CS:CODE, DS:DATA
        CODE SEGMENT
        START:      MOV AX, DATA
                    MOV DS, AX
                    MOV ES, AX
                    MOV CX, COUNT
                    MOV DI, OFFSET STRING
                    MOV BL, 00H
                    MOV AL, BYTE1
        SCAN1:      NOP
                    SCASB
                    JZ XXX
                    INC BL
                    LOOP SCAN1
        XXX:        MOV AH, 4CH
                    INT 21H
                    CODE ENDS
        DATA SEGMENT
        BYTE1 EQU 25H
        COUNT EQU 06H
        STRING DB 12H, 13H, 20H, 20H, 25H, 21H
        DATA ENDS
                    END START
```

Write a program to convert the BCD numbers 0 to 9 to their equivalent seven segment codes using the look-up table technique. Assume the codes [7-seg] are stored sequentially in CODELIST at the relative addresses from 0 to 9. The BCD number (CHAR) is taken in AL.

Solution Here we use the XLAT instruction.

```
ASSUME CS:CODE, DS:DATA
DATA SEGMENT
            CODELIST DB 34, 45, 56, 45, 23, 12, 19, 24, 21, 00
            CHAR EQU 05
            CODEC    DB 01H DUP(?)
DATA ENDS
CODE SEGMENT
START:      MOV AX, DATA
            MOV DS, AX
            MOV BX, OFFSET CODELIST
            MOV AL, CHAR
            XLAT
            MOV BYTE PTR CODEC,AL
            MOV AH, 4CH
            INT 21H
CODE        ENDS
            END START
```



23) ALP to find square root of 2 digit no. Assume that it is perfect square

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
NUM EQU 36
RESULT DB (?)
DATA ENDS
CODE SEGMENT
START: MOV AX, DATA
       MOV DS, AX      → Initialise DS
       MOV CL, NUM     CL = 36
       MOV BL, 1       BL = 1
       MOV AL, 0       AL = 0
UP:    CMP CL, 0       Check if CL is zero
       JLE RESULT      then
       SUB CL, BL      Store AL
       INC AL          result
       ADD BL, 02      if CL ≠ 0
       JMP UP          CL = CL - BL
RESULT: MOV RESULT, AL AL = AL + 1
       MOV AH, 4CH     BL = BL + 2
       INT 21H         Repeat till CL
CODE ENDS
```

22) Write an ALP to convert an 8 bit binary no into equivalent gray code

```
ASSUME CS: CODE, DS: DATA

DATA SEGMENT

NUM EQU 34H

RESULT DB (?)

DATA ENDS

START: MOV AX, DATA
       MOV DS, AX      - Initialise
       MOV AL, NUM     -> Bin no
       MOV BL, AL      -> BL = no
       CLC
       RCR AL, 1       -> Rotate AL through CF
       XOR BL, AL      -> XOR BL & AL
       MOV RESULT, AL
       MOV AH, 4CH     move result of A in memory
       INT 21H
CODE ENDS
```

20) Write an ALP to perform 1 byte BCD addition.

Stack
Square
Calculation
programs

```
ASSUME CS: CODE, DS: DATA

DATA SEGMENT

    OPR1 EQU 92H
    OPR2 EQU 59H        > 2 values

Store result ← RESULT DB 02 DUP(00)
DATA ENDS

CODE SEGMENT
START : MOV AX, DATA
        MOV DS, AX      → initialise DS

        MOV BL, OPR1    - 1st val
        XOR AL, AL      → clear AL, CF
        MOV AL, OPR2    → 2nd val
        ADD AL, BL      → add 2 val
        DAA
        MOV DI, OFFSET RESULT → result in decimal
        MOV [DI], AL
        JNC MSB0        store the result.
        MOV [DI+1], 01H ; if carry increment value
MSB0:   MOV AH, 4CH     Or store 1
        INT 21H
CODE ENDS
END START
```

17) Write an ALP for factorial of a number

H/W

```
ASSUME CS: CODE , DS: DATA
  DATA SEGMENT
FACT DB ?;
     DATA ENDS
          CODE SEGMENT
          MOV AX, DATA
            MOV DS, AX
          MOV CL, 05H
          MOV AX, 01H
AGAIN:    MUL CL
          DEC CL
  JNZ AGAIN
  MOV SI, OFFSET FACT
   MOV [SI], AL
  MOV AH, 4CH
   INT 21H
CODE ENDS
END START
```

18) Write an ALP to multiply 2 16 bit data

H/W

```
ASSUME CS: CODE
  CODE SEGMENT
  ORG 1000H
  MOV SI, 1100H
  MOV AX, [SI]
  MOV BX, [SI+2]
  MUL BX
  MOV [SI+4], AX
  MOV [SI+6], DX
   HLT
  CODE ENDS
```

> 16 bit input

15) Write an ALP for Binary to ASCII

Conversion

```
ASSUME CS: CODE
CODE SEGMENT
ORG 1000H
MOV SI, 1100H
MOV DI, 1400H
CLD
MOV BL, 09H
NEXT: LODSB
CMP AL, BL
JG L1
JLE L2
```

0 to 9
add 30 for ASCII
>9 add 37

AL=08
...
8 < 9

```
HLT
L2: ... ADD AL, 30
STOSB
HLT
L1: ADD AL, 37
STOSB
HLT
CODE ENDS
END
```

16) Write an ALP for average of set of numbers

H/W

```
DATA SEGMENT
NUM1  DB 5
NUM2  DB 9
NUM3  DB 7
AVG   DB ?
ENDS
CODE SEGMENT
ASSUME DS: DATA, CS: CODE
START:
    MOV AX, DATA
    MOV DS, AX
    MOV AL, NUM1
    ADD AL, NUM2
    ADD AL, NUM3
    MOV AH, 0
    MOV DL, 3
    DIV DL
    MOV SI, OFFSET AVG
    MOV [SI], AL
    MOV AH, 4CH
    INT 21H
CODE ENDS
    END START
```

DIV DL = AL/3 = AL

14) Write an ALP for generating fibonacci series

```
ASSUME CS: CODE
CODE SEGMENT
ORG 1000H
MOV SI, 1100H
MOV CL, [SI]
INC SI
MOV AL, 00H
MOV [SI], AL
INC SI
INC AL
MOV [SI], AL
SUB CL, 02H
```

Suppose CL=5
move count in CL

00 is 1st moved to SI after inc SI

SI ← 5
SI+1 — 00
SI+2 — 01
5-2 = 3 = CL

SI+1

```
GEN: DEC SI
     MOV AL, [SI]
     INC SI
     MOV BL, [SI]
     ADD AL, BL
     INC SI
     MOV [SI], AL
     LOOP GEN
     HLT
CODE ENDS
```

00

add previous 2 values in AL, BL

00 01
01 01
02 01

→ repeat till count=0

AL = 00
01 = BL
SI+2
00+01 = 01
SI+3 = 01
SI+2 = 01 = AL
SI+3 = 01
BL = 01
01+01 = 02
= SI+4

12) Write an ALP for addition of series of 8 bit nos. Series has 100 nos.

HW

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
NUMLIST DB 52H, 23H, ...      <- List of values
COUNT EQU 100D                -> 100 numbers
RESULT DW 01H DUP(?)          <- Store result in this loc
DATA ENDS
CODE SEGMENT
ORG 0200H                     -> start memory allocation for code
START: MOV AX, DATA
       MOV DS, AX             -> Initialise DS
       MOV CX, COUNT          -> move Count
       XOR AX, AX             -> clear AX, CF
```

S.I. to put

```
XOR BX, BX -> clear BX
MOV SI, OFFSET NUMLIST
```

point SI to

```
AGAIN: MOV BL, [SI]        <- first no
       ADD AX, BX          -> add 0 & first no.
       INC SI              -> inc pointer to get the next no
       DEC CX
       JNZ AGAIN           -> if non zero CX then repeat addition of next no & 1st no
                              Continue till CX = 0.
       MOV DI, OFFSET RESULT
                           -> point DI to destination
       MOV AH, 4CH
       INT 21H             -> Return to DOS
CODE   ENDS
       END START           -> End of execution
```

* Ax is the general purpose register used to initialise Data Segment register Code segment register gets automatically loaded by cs address of program.
* We cannot directly transfer data from memory source operand to memory destination operand except, string instructions.
* Result of Arithmetic instructions can be loaded into AX or DX.
* After segment initialisation, code starts at offset 0000_H.

Q) Add the contents of the memory location $2000_H : 0500_H$ to contents of $3000_H : 0600_H$ and store the result in $5000_H : 0700_H$

```
MOV CX: 2000H
MOV DS, CX      → initialise DS
MOV AX, [0500H]
MOV CX, 3000H
MOV DS, CX
MOV BX, [0600H]
ADD AX, BX
```

```
MOV CX, 5000H
MOV DS, CX
MOV [0700H], AX
HLT
```

The length is specified using CX register.

SI and DI point to source & destination locations.

The DS & ES registers can be initialised to source & destination segment addresses.

Machine level programming has disadvantages like:

- Process is complicated and time consuming
- Chances of error are more
- Debugging is difficult

Program called Assembler is used to convert the mnemonics of instructions & the data into object code modules.

These are converted into executable code using linker and loader programs.

This type of programming is called Assembly level programming.

Advantages of Assembly language are:-

- Programming is not complicated
- Chances of error are less
- Debugging is easier.

Assembly called input file source file is converted to object file which is further converted to executable codes / using a linker machine codes

In MASM (Microsoft Macro Assembler) Source file (input) is converted to object file which is converted by linker to an EXE file.

For MASM programming we need a text editor (Turbo C), Assembler (MASM), linker and DEBUG utility of DOS.

File typed in editor is saved as file name .asm

This .asm file is converted by assembler to .obj file, so type link file name .obj

For debugging type, debug file name .exe

This is useful for troubleshooting, executing

and observing results.

Some Debug commands

- -R - Display all registers and flags
- -D - Display 128 memory locations of RAM.
- -a - Assemble from the current CS:IP.
- -g - Execute from current CS:IP
- -q - Quit and return to DOS
- -T - Trace the program execution step by step.
- -l - Load the file filename.exe

DOS is the Disk Operating System which manages the hardware resources of system. It uses the instruction INT 21 for selecting specific resource.

If AH contains 09H then CRT display is used.

If it contains 0AH then keyboard is to be accessed. Value in AH is the function value. These interrupts are function calls.

Extra segment is used as additional destination data segment

Label STARTS marks the starting point of execution sequence.

Function value 4CH is used for returning to DOS prompt.

7) Stack

The stack provides sequential mechanism to store the partial results ie for temporary data storage

Microprocessor should know where the control is to be returned after execution of a subroutine.

Address of reentry into the main program may be stored onto the stack.

Stack is useful for storing the register status of the processor at the time of calling a subroutine, and getting it back at the time of returning, so that registers / memory locations can be reused.

the stack is a block of memory that may be used for temporarily storing the contents of registers inside CPU.

the memory locations of stack can be accessed using the SP and SS registers.

It is a top down structure whose elements

Scanned with CamScanner

are accessed using a pointer that is implemented using SP and SS registers.

As the data words are stored on to the stack the pointer goes on decrementing, and as we retrieve the word data, the pointer goes on incrementing.

SP is decremented / incremented by 2.

The process of storing the data in the stack is called pushing into the stack and reverse process of transferring the data back from the stack to the CPU register is known as popping off the stack.

The stack is a Last in First out (LIFO) data segment.

Data which is pushed into the stack last will be on top of stack up will be popped off the stack first.

Scanned with CamScanner

Stack structure of 8086/88

The stack contains a set of sequentially arranged data bytes with the last item appearing on top of the stack. This item will be popped off the stack first for use.

SP (stack pointer) register is a 16 bit register that contains the offset of the address that lies in the stack segment.

The SS (stack segment), have a memory block of a maximum of 64 KB locations and may overlap with other segments.

The SS register and SP register together address the stack top.

Let content of SS be 5000_H and the content of stack pointer be 2050_H.

Stack top address is calculated as,

SS register content → shifted left by 4 bit positions

(+)

16 bit content of stack pointer register

Scanned with CamScanner

or, 20 bit content of SS register (10_H x content of SS register) + 16 bit offset value.

Eg: $5000_H \times 10 + 2050_H = 50000 + 2050 = 52050_H$

(SS) (SP)

20 bit

(Stack top address).

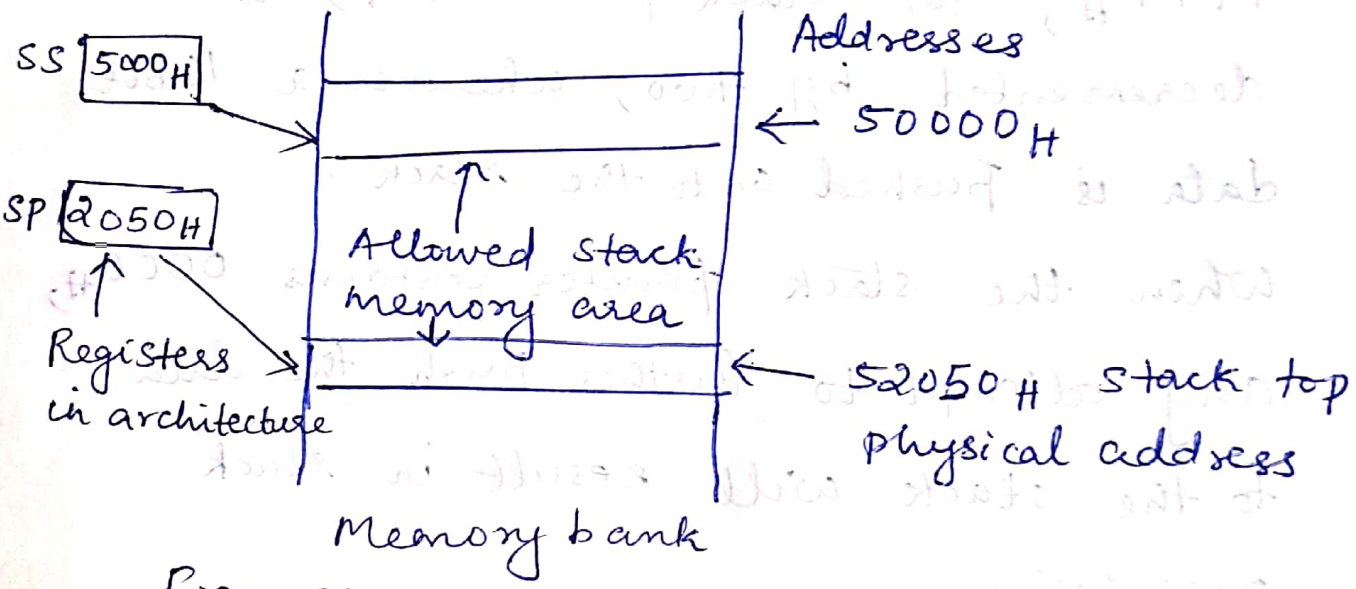


Fig: Stack top address calculation

If the stack top points to memory location 52050_H, previously pushed data is available at 52050_H.

The next 16 bit push operation will decrement the stack pointer by 2, so that it will point to new stack top $5204E_H$, contents of SP will be $204E_H$.

Scanned with CamScanner

Two locations will be required for a 2 byte (16 bit) data.

For a selected value of SS, the max. value of SP = $FFFF_H$ i.e the segment can have maximum of 64 K locations.

Starting with an initial value of $FFFF_H$, the stack pointer (SP) is decremented by two, whenever a 16 bit data is pushed on to the stack.

When the stack pointer contains 0000_H, any attempt to further push the data to the stack will result in stack overflow.

Each POP operation increments the SP.

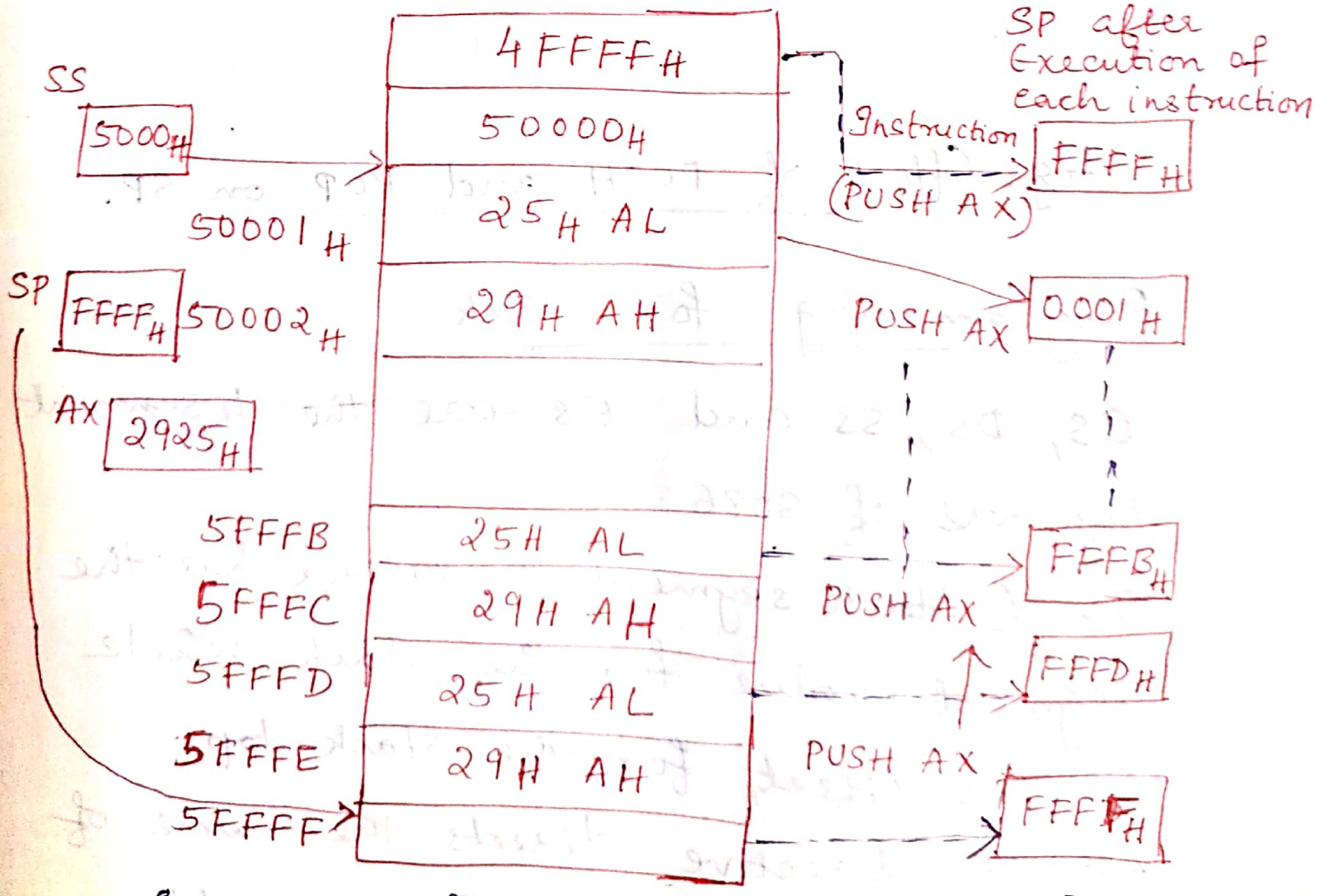
POP operation is used to retrieve the data stored onto the stack.

If there is a subroutine call instruction there is a possibility that all or some of the registers of the main program may be pushed onto the stack one by one.

Scanned with CamScanner

During each PUSH, SP will be modified. At the end of the execution of the subroutine, all the registers can get back the original contents by popping the data from the stack.

Register / memory location that is pushed into the stack at the end should be popped off first.



Execution of bracketed PUSH AX Results in Stack Overflow

Scanned with CamScanner

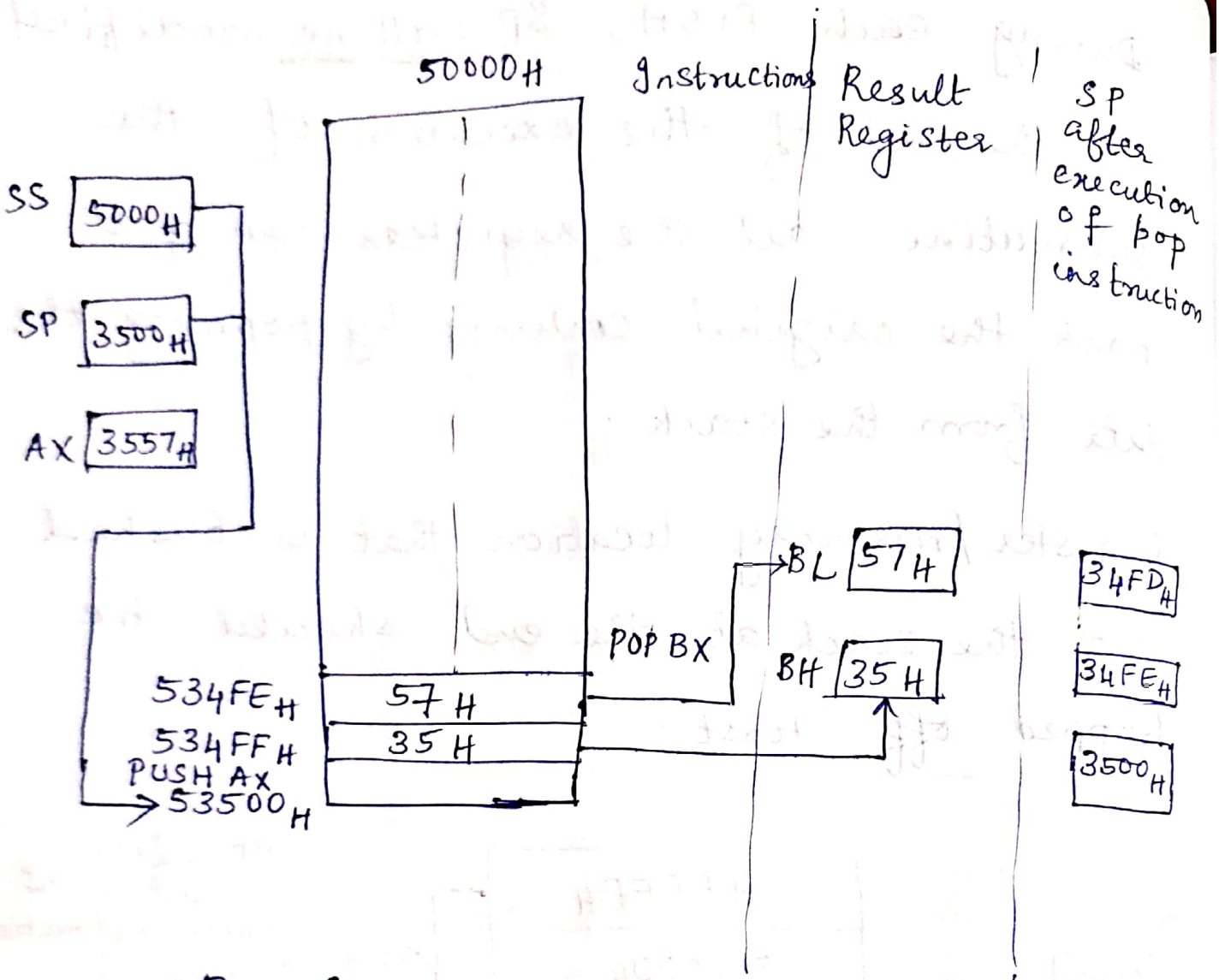


Fig: Effect of PUSH and POP on SP.

Programming for Stack

CS, DS, SS and ES are the 4 segment registers of 8086.

SS (stack segment) register has the segment value for the stack while SP has offset for the stack-top.

ASSUME directive directs the name of the stack segment to the assembler.

Scanned with CamScanner

SS register and SP register must be initialised in the program.

After a subroutine is called using the CALL instruction, the IP is incremented to the next instruction.

Contents of IP, CS and flag register are pushed automatically to the stack.

The control is then transferred to the specified address in the CALL instruction ie starting address of subroutine.

Subroutine is executed.

There should be an equal no. of PUSH & POP instructions ie SP contents at the time of calling subroutine must be equal to the contents of SP at the time of executing the RET instruction.

If not, control to be returned to next instruction after CALL instruction will not be returned back properly.

To calculate the squares, the multiplication

Scanned with CamScanner

is implemented as successive addition and the DAA instruction is used after each addition operation to convert the result in decimal format.

Push registers to stack at the start of subroutine.

Stack mechanism is also used in interrupt service routines to store IP & CS.

Stack size is set using DW / DB directive.

Eg

Write a program to calculate the squares of BCD nos 0 to 9 and store them sequentially from 2000_H offset onwards in the current data segment.

The numbers and their squares are in the BCD format.

Write a subroutine for the calculation of the squares of a number.

Scanned with CamScanner

The procedure of computing the square of a no. is to be repeated for all the nos. A subroutine for calculating the square is written which is called repetitively from the main program.

The 8086 does not have single instruction for calculation of the square of a number. Thus we may calculate the square of a no. using ADD and DAA instructions. The result of the ADD instruction is in hexadecimal format and it should be converted to decimal form; before it is stored into the memory.

The MUL instruction does not calculate the square of a decimal number, and the DAA instruction is to be used only after the ADD or ADC instructions.

Advantages of subroutine is that, a recurring sequence of instructions can be assigned a procedure name, called again & again

Scanned with CamScanner

```
ASSUME CS: CODE , DS: DATA , SS: STACK

DATA SEGMENT

    ORG 2000H

SQUARES DB 0FH DUP (?)        → 15 values for storing ie stack size

DATA ENDS

STACK SEGMENT

STAKDATA DB 100H DUP (?)      → Reserve 256 bytes for stack

STACK ENDS

CODE SEGMENT

START: MOV AX, DATA           → Initialise data segment

MOV DS, AX

MOV AX, STACK                 → Initialise stack segment

MOV SS, AX

MOV SP, OFFSET STAKDATA       → Initialise stack pointer

MOV CL, 0AH                   → Initialise counter for numbers
```

Scanned with CamScanner

```
MOV SI, OFFSET SQUARES      → Set pointer to array of squares.
MOV AL, 00                  → Start from 00
NEXTNUM: CALL SQUARE        // Calculate square
         MOV BYTE PTR [SI], AH   // Store square in the array
         INC AL             // Go for the next number
         INC SI             // Increment the array pointer
         DCR CL             // Decrement count
         JNZ NEXTNUM        // Stop if CL=0, else continue
         MOV AH, 4CH        // Return to DOS prompt
         INT 21H
PROCEDURE SQUARE NEAR       // square is a local procedure called by this segment only.
         MOV BH, AL
         MOV CH, AL         // Duplicate AL to CH & BH
         XOR AL, AL         // Clear flags and AL
```

Scanned with CamScanner

```
AGAIN:  ADD  AL, BH     // Successively add CH to AH
        DAA             // Get BCD equivalent
        DCR  CH         // Decrement Successive addition
        JNZ  AGAIN      // Counter till it becomes zero.
        MOV  AH, AL     // Store the square and
        MOV  AL, BH     // get back the original number
        RET             // After that return
SQUARE  ENDP
CODE    ENDS
        END  START
```

Scanned with CamScanner

interfacing with 8086 microprocessor

(1) Interrupts and Interrupt Service routines

The word 'interrupt' means to break the sequence of operation. While the CPU is executing a program, an interrupt breaks the normal sequence of execution of instructions, diverts its execution to some other program called Interrupt Service routine (ISR). After executing ISR, the control is transferred back again to the main program which was being executed at the time of interruption. The type of sequence of appearance of interrupts is called nested interrupt i.e interrupt within interrupt. Whenever a number of devices interrupt a CPU at a time, and if the processor is able to handle them properly it is said to have multiple interrupt processing

Scanned with CamScanner

capability. for eg: 8085 has five hardware interrupt pins and it is able to handle the interrupts simultaneously under the control of the software. In case of 8086 there are two interrupt pins ie, N.M.I and INTR. The NMI is a non maskable interrupt input pin which means that any interrupt request at NMI input cannot be masked or disabled by any means. The INTR interrupt may be masked using the Interrupt flag (IF). The INTR, is of 256 types. The INTR types may be from 00 to FFH (or 00 to 255). If more than one type of INTR interrupt occurs at a time, then an external chip called programmable interrupt controller is required to handle them. The same is the case for INTR interrupt input of 8085. Interrupt Service Routines (ISRs) are the programs to be executed by interrupting the main program

Scanned with CamScanner

execution of the CPU, after an interrupt request appears. After the execution of ISR, the main program continues its execution further from the point at which it was interrupted.

(2) Interrupt cycle of 8086/8088

There are two types of interrupts. The first out of them is external interrupt and the second is internal interrupt. In external interrupt an external device or a signal interrupts the processor from outside or in other words, the interrupt is generated outside the processor, for eg. a keyboard interrupt. The internal interrupt on the other hand, is generated internally by the processor circuit or by the execution of an interrupt instruction. The examples of this type are divide by zero interrupt, overflow interrupt, interrupts due to INT instructions etc.

Scanned with CamScanner

Suppose an external device interrupts the CPU at the interrupt pin, either NMI or INTR of the 8086 while the CPU is executing an instruction of a program. The CPU first completes the execution of the current instruction. The IP is then incremented to point to the next instruction. The CPU then acknowledges the requesting device on its $\overline{INTA}$ pin immediately if it is a NMI, TRAP or Divide by zero interrupt. If it is an INT request, the CPU checks the IF flag. If IF is set, the interrupt request is acknowledged using the $\overline{INTA}$ pin. If the IF is not set, the interrupt requests are ignored. Note that the responses to the NMI, TRAP and Divide by zero interrupt requests are independent of the IF flag.

Scanned with CamScanner

After an interrupt is acknowledged the CPU computes the vector address from the type of the interrupt that may be passed to the interrupt structure of the CPU internally (software interrupts, NMI, TRAP, Divide by zero interrupts) or externally ie from an interrupt controller in case of external interrupts. (The contents of IP and CS are next pushed to the stack. The contents of IP and CS now point to the address of the next instruction of the main program from which the execution is to be continued after executing the ISR. The PSW is also pushed to the stack). The Interrupt flag (IF) is cleared. The TF is also cleared, after every response to the single step interrupt. The control is then transferred to the interrupt service routine for serving the interrupting device. The new address of ISR is found out from the interrupt vector table. The execution of the ISR starts. If further interrupts are to be responded

Scanned with CamScanner

to during the time the first interrupt is being serviced, the IF should again be set to 1 by the ISR of the first interrupt.

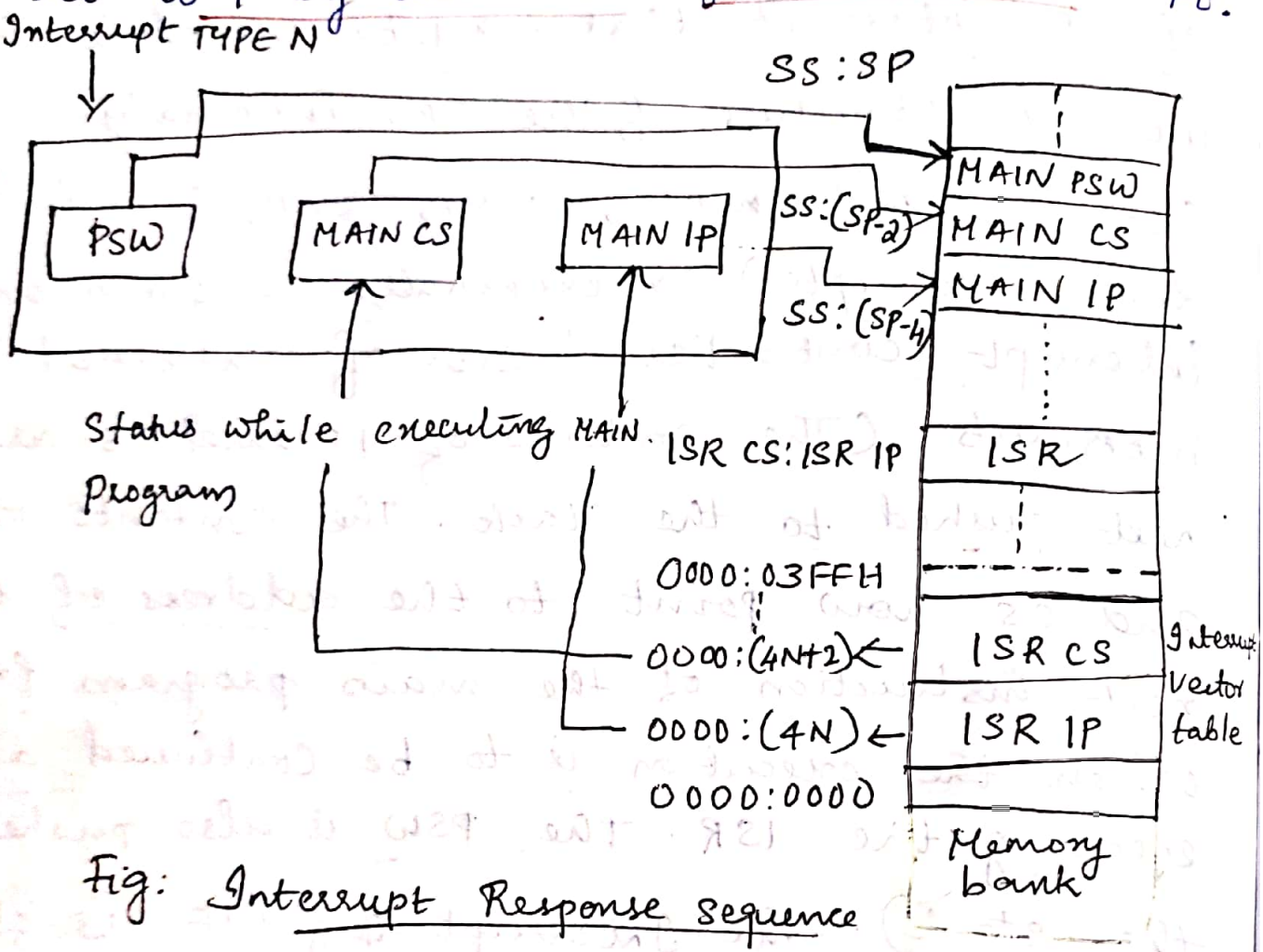


Fig: Interrupt Response sequence

If the interrupt flag is not set, the subsequent interrupt signals will not be acknowledged by the processor, till the current one is completed. The programmable interrupt controller is used for managing such multiple interrupts based on their priorities. At the end of the ISR the last instruction should be IRET, when the

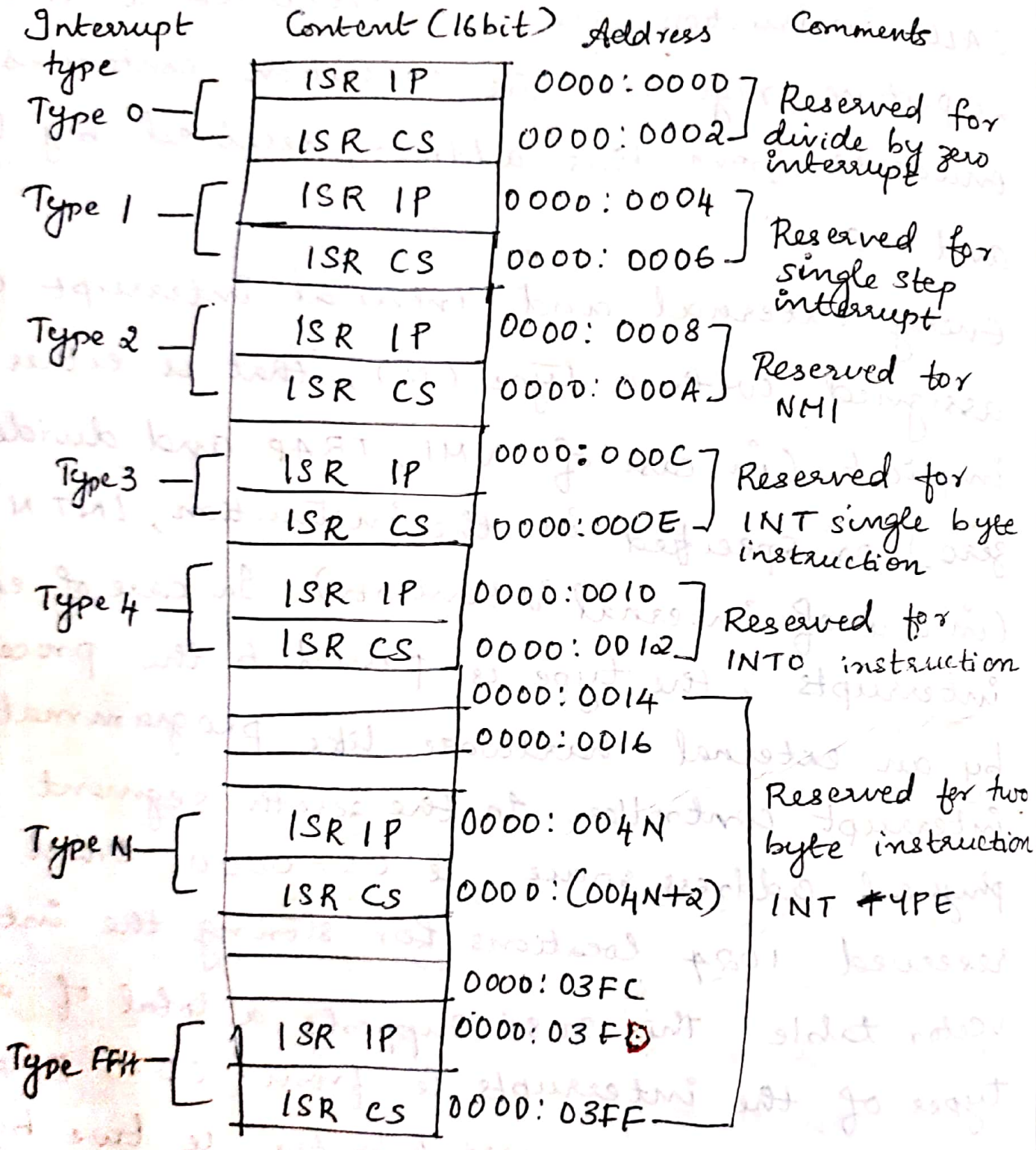
Scanned with CamScanner

CPU executes IRET, the contents of flags, IP and CS which were saved at the start by the CALL instruction are now retrieved to the respective registers. The execution continues onwards from this address, received by IP and CS.

Every external and internal interrupt is assigned with a type (N), that is either implicit (in case of NMI, TRAP and divide by zero) or specified in the instruction; INT N (in case of internal interrupts). In case of external interrupts, the type is passed to the processor by an external hardware like programmable interrupt controller. In the zeroth segment of physical address space, ie CS = 0000, Intel has reserved 1024 locations for storing the interrupt vector table. The 8086 supports a total of 256 types of the interrupts ie from 00 to FFH. Each interrupt requires 4 bytes, ie two bytes each for IP and CS of its ISR. A total of 1024 bytes are required for 256 interrupt types

Scanned with CamScanner

hence the interrupt vector table starts at location 0000:0000 and ends at 0000:03FF$_H$.



ISR: Interrupt Service Routine

Fig: Structure of Interrupt Vector Table of 8086/88

Scanned with CamScanner

The interrupt vector table contains the IP and CS of all the interrupt types stored sequentially from address 0000: 0000 to 0000: 03FFH. The interrupt type N is multiplied by 4 and the hexadecimal multiplication obtained gives the offset address in the zeroth code segment at which the IP and CS addresses of the interrupt service routine (ISR) are stored. The execution automatically starts from the new CS: IP.

(3) Types of interrupts

Non maskable interrupt

The processor 8086/86 has a Non Maskable Interrupt input pin (NMI), that has the highest priority among the external interrupts. TRAP (single step - Type 1) is an internal interrupt having the highest priority amongst all the interrupts except the Divide by zero (Type 0) exception. The NMI is activated on a positive

Scanned with CamScanner

transition (low to high voltage). The assertion of the NMI interrupt is equivalent to an execution of instruction INT 02, ie Type 2 INTR interrupt. The NMI pin should remain high for at least 2 clock cycles and need not synchronized with the clock for being sensed. When the NMI is activated, the current instruction being executed is completed and then the NMI is served. In the case of string type instructions this interrupt will be served only after the complete string has been manipulated. Another high going edge on the NMI pin of 8086, during the period in which the first NMI is served, triggers another response. The signal on the NMI pin must be free of logical bounces to avoid erratic NMI responses.

Scanned with CamScanner

Maskable interrupt (INTR)

The processor 8086/88 also provides a pin INTR that has lower priority as compared to NMI. The priorities within the INTR types are decided by the type of the INTR signal that is to be passed to the processor via data bus by some external device like the programmable interrupt controller. The INTR signal is level triggered and can be masked by resetting the interrupt flag. It is internally synchronized with the high transition of the CLK. For the INTR signal, to be responded to in the next instruction cycle, it must go high in the last clock cycle of the current instruction or before that. The INTR requests appearing after that the last clock cycle of the current instruction will be responded to after the execution of the next instruction. The status of the pending interrupts is checked at the

Scanned with CamScanner

end of each instruction cycle.

If the IF is set, the processor is ready to respond to any INTR interrupt, if the IF is reset the processor will not serve any interrupt appearing at this pin. Once the processor responds to any INTR signal, the IF should again be set. The interrupt acknowledge sequence is shown in figure.

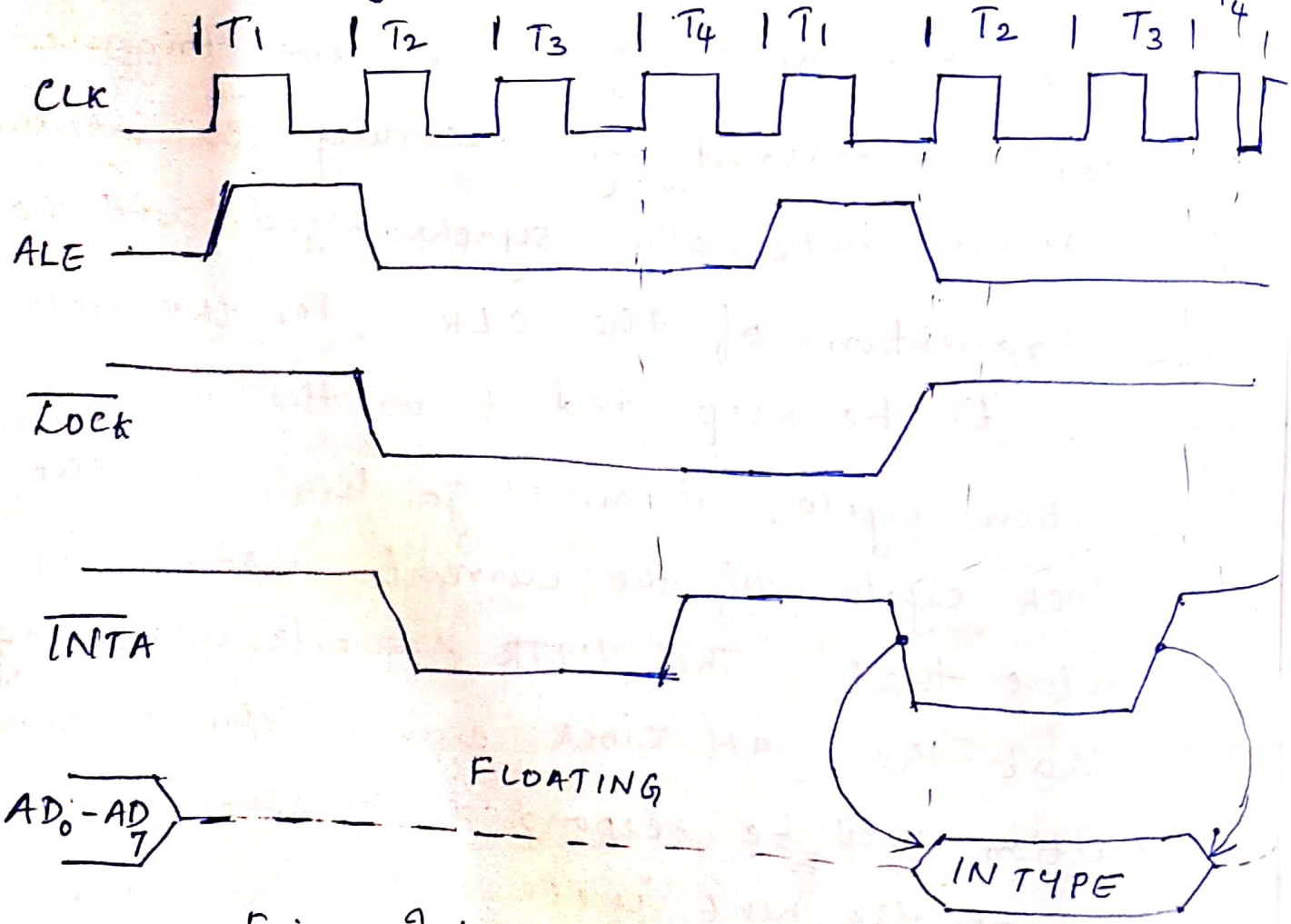


fig: Interrupt acknowledge sequence of 8086.

Scanned with CamScanner

Suppose an external signal interrupts the processor and pin $\overline{LOCK}$ goes low at the trailing edge of the first ALE pulse that appears after the interrupt signal preventing the use of bus for any other purpose. The pin $\overline{LOCK}$ remains low till the start of the next machine cycle. With the trailing edge of $\overline{LOCK}$, the $\overline{INTA}$ goes low and remains low for two clock states before returning back to the high state. It remains high till the start of the next machine cycle i.e. next trailing edge of ALE. Then $\overline{INTA}$ again goes low, remains low for two states before returning to the high state. The first trailing edge of ALE floats the bus $AD_0 - AD_7$, while the second trailing edge prepares the bus to accept the type of the interrupt. The type of the interrupt remains on the bus for a period of two cycles.

Scanned with CamScanner

4) Interrupt programming

While programming for any type of interrupt the programmer must either externally or through the program set the interrupt vector table for that type preferrably with the CS and IP addresses of the interrupt service routine. The method of defining the interrupt service routine for software (internal) as well as hardware interrupt is the same. (external)

Fig (4.7) shows the execution sequence in the case of software interrupt. It is assumed that the interrupt vector table is initialised suitably to point to the interrupt service routine. Fig. (4.8) shows the transfer of control for the nested interrupts.

```
ASSUME  CS: CODE, DS: DATA

DATA  SEGMENT
      :
      :
DATA  ENDS

CODE  SEGMENT
      :
```

Scanned with CamScanner

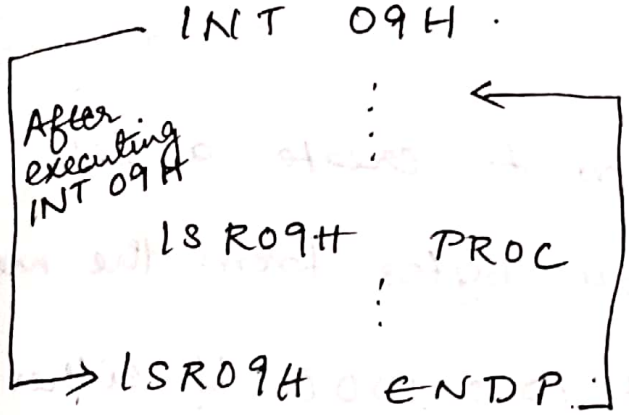


Fig: Transfer of control during execution of an Interrupt Service Routine
(4.7)

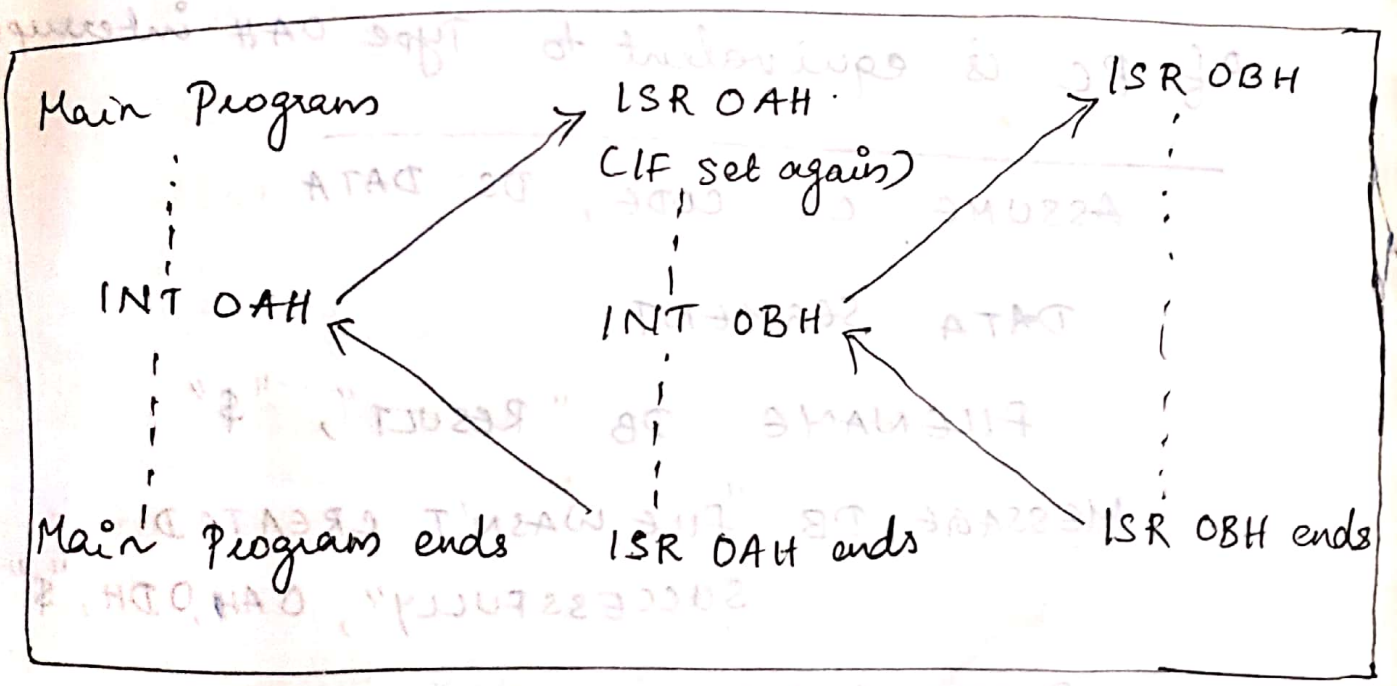


Fig: Transfer of control for Nested Interrupts
(4.8)

Scanned with CamScanner

Program

1) Write a program to create a file RESULT in and store it 500H bytes from the memory block starting at 1000:1000, if either an interrupt appears at INTR pin with Type OAH or an instruction equivalent to the above interrupt is executed.

Note: Pin IRQ_2 available at IO channel of PC is equivalent to Type OAH interrupt

---

```
ASSUME CS: CODE, DS: DATA

DATA SEGMENT
    FILENAME DB "RESULT", "$"
MESSAGE DB "FILE WASN'T CREATED
           SUCCESSFULLY", 0AH, 0DH, "$"

DATA ENDS
CODE SEGMENT
START : MOV AX, CODE
        MOV DS, AX ; Set DS at
                     CODE for
                     setting INT
```

Scanned with CamScanner

```
MOV DX, OFFSET ISR0A ; Set Dx at the offset of ISR0A
MOV AX, 250AH ; Set IVT using function value 250AH in AX.
INT 21H ; under INT21H.
MOV DX, OFFSET FILENAME ; set pointer to file name
MOV AX, DATA ; Set the DS at DATA for file name
MOV DS, AX
MOV CX, 00H
MOV AH, 3CH ; Create file with the file name 'RESULT'
INT 21H
JNC FURTHER ; If no carry, create operation is successful
MOV DX, OFFSET MESSAGE ; else display the MESSAGE
MOV AH, 09H ;
INT 21H.
JMP STOP.
FURTHER: INT 0AH ; If the file is created successfully, write in it and return to DOS prompt
STOP: MOV AH, 4CH
INT 21H ;
```

Scanned with CamScanner

```
                    ;  This interrupt service
                    ;  routine writes 500 bytes
ISR0A  PROC NEAR    ;  into file RESULT and
                    ;  returns to main program
       MOV BX, AX
       MOV CX, 500H ;  Take file handle in
       MOV DX, 1000H;  BX, byte count in
       MOV AX, 1000H;  CX,
       MOV DS, AX   ;  offset of block in
       MOV AH, 40H  ;  DX, segment value
                    ;  of block in DS.
       INT 21H      ;  Write in the file
       IRET         ;  and return
ISR0A  ENDP
CODE   ENDS
       END START
```

To execute the above program, first assemble it using MASM.EXE, link it using LINK.OBJ. Then execute the above program at a DOS prompt. After execution, we will find a new file RESULT in the directory. Then apply an external pulse to IRQ2 pin of the IBM PC IO channel. This will

Scanned with CamScanner

again cause the execution of ISR that writes 500H bytes into the file.

Program

2) Write a program that gives display 'IRT2 is OK', if a hardware signal appears on IRQ2 pin and 'IRT3 is OK' if it appears on IRQ3 pin of PC IO channel.

```
ASSUME CS: CODE , DS: DATA

DATA SEGMENT

MSG1 DB "IRT2 is OK", 0AH, 0DH,
                              "$"

MSG2 DB "IRT3 is OK", 0AH, 0DH,
                              "$"

DATA ENDS

CODE SEGMENT

START: MOV AX, CODE
       MOV DS, AX          ; Set IVT for
       MOV DX, OFFSET ISR1   Type 0AH
       MOV AX, 250AH       ; IRQ2 is
                             equivalent
```

Scanned with CamScanner

```
        INT 21H          ; to type DAH
HERE:   JUMP HERE

ISR1    PROC LOCAL       ; ISR1 and
        MOV AX, DATA       ISR2 display
        MOV DS, AX         the message

        MOV DX, OFFSET MSG1 ; Display
                            message
        MOV AH, 09H.        MSG1
        INT 21H
        IRET
ISR1    ENDP
ISR2    PROC LOCAL
        MOV AX, DATA
        MOV DS, AX
        MOV DX, OFFSET MSG2 ; Display
                              message
        MOV AH, 09H           MSG2
        INT 21H
        IRET
ISR2    ENDP
CODE    ENDS
        END START
```

Scanned with CamScanner

Prepare the EXE file of the above program as usual. Execute it at DOS prompt that will hang the system. Apply a pulse at IRQ_2 pin. The message : 'IRT2 is OK' is displayed on the screen. Then apply a pulse to IRQ_3 pin of IO channel. The message 'IRT3 is OK' is displayed on screen.

5) Programmable interrupt controller 8259A

The processor 8085 had five hardware interrupt pins. Out of these five interrupt pins, four pins were allotted fixed vector addresses but the pin INTR was not allotted any vector address rather an external device was supposed to hand over the type of interrupt, ie (Type 0 to 7 for RST0 to RST7), to the microprocessor. The microprocessor then gets this type and derives the interrupt

Scanned with CamScanner

vector address from that. Consider an application where a no. of I/o devices connected with a CPU desire to transfer data using interrupt driven data transfer mode. More no. of interrupt pins are required than available in a typical microprocessor. In these multiple interrupt systems, the processor will have to take care of the priorities for the interrupts, simultaneously occuring at the interrupt request pins. To overcome all these difficulties, we require a programmable interrupt controller which is able to handle a no. of interrupts at a time. This controller takes care of a no. of simultaneously appearing interrupt requests along with their types and priorities. This relieves the processor from all these tasks. The programmable interrupt controller 8259A from Intel is one such

Scanned with CamScanner

device. Its predecessor 8259 was designed to operate only with 8 bit processors like 8085. A modified version 8259A was later introduced that is compatible with 8 bit as well as 16 bit processors.

Architecture and signal descriptions of 8259A

The functional explaination of each block is given below:-

Interrupt Request Register (IRR)

The interrupts at IRQ input lines are handled by Interrupt Request Register internally. IRR stores all the interrupt requests inorder to serve them one by one on the priority basis

In Service Register (ISR)

This stores all the interrupt requests those are being served ie ISR keeps a track of the requests being served.

Scanned with CamScanner

Priority Resolver

This unit determines the priorities of the interrupt requests appearing simultaneously. The highest priority is selected and stored into the corresponding bit of ISR during $\overline{INTA}$ pulse. The IR_0 has the highest priority while the IR_7 has the lowest one normally in fixed priority mode. The priorities however may be altered by programming the 8259A in rotating priority mode.

Interrupt Mask register (IMR)

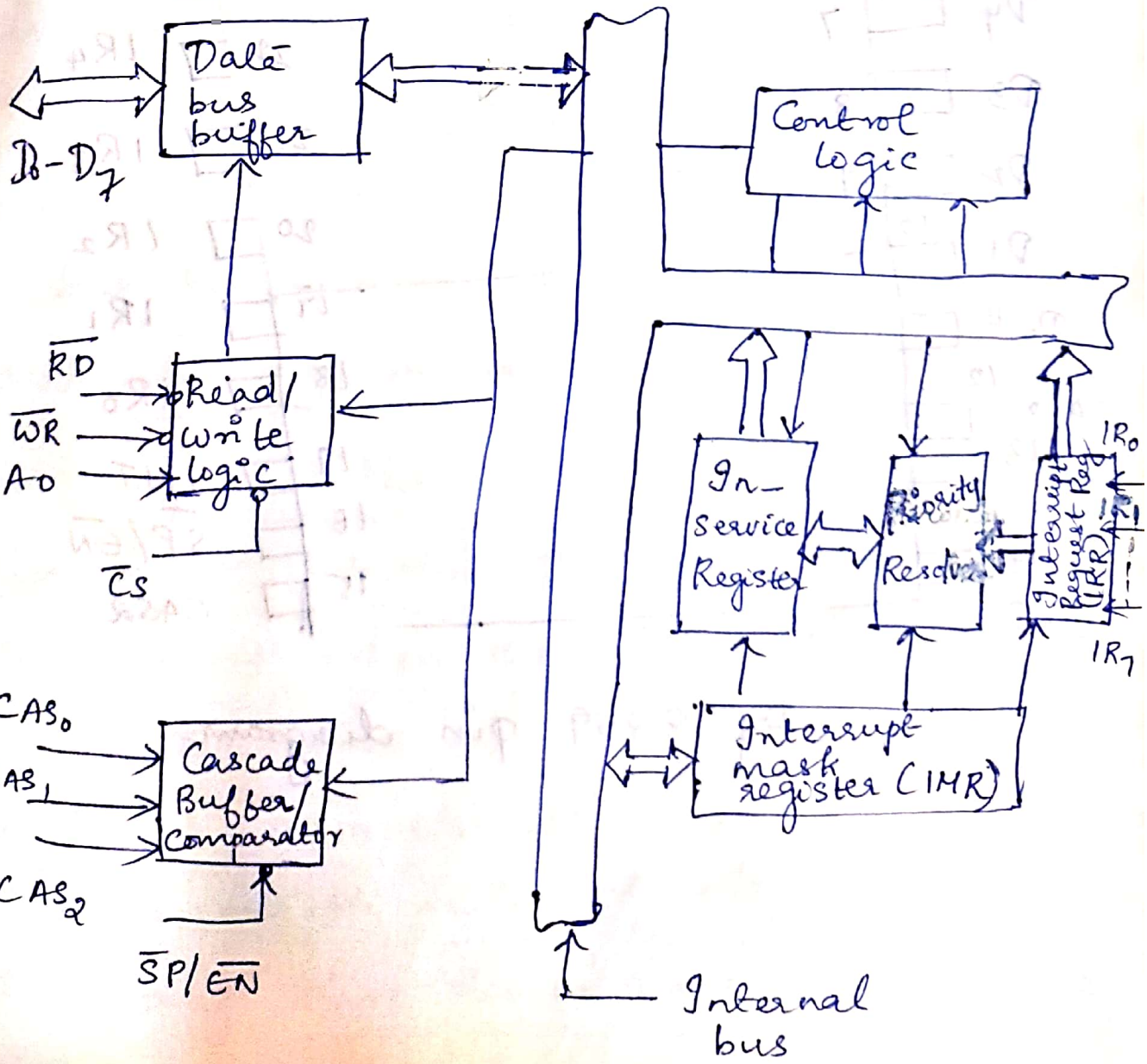
This register stores the bits required to mask the interrupt inputs. IMR operates on IRR at the direction of the priority resolver.

Interrupt control logic - This block manages the interrupt and the interru

Scanned with CamScanner

acknowledge signals to be sent to the CPU for serving one of the eight interrupt requests. This also accepts the interrupt acknowledge (INTA) signal from CPU that causes the 8259A to release vector address on to the data bus.

Fig: 8259A Block diagram



Scanned with CamScanner

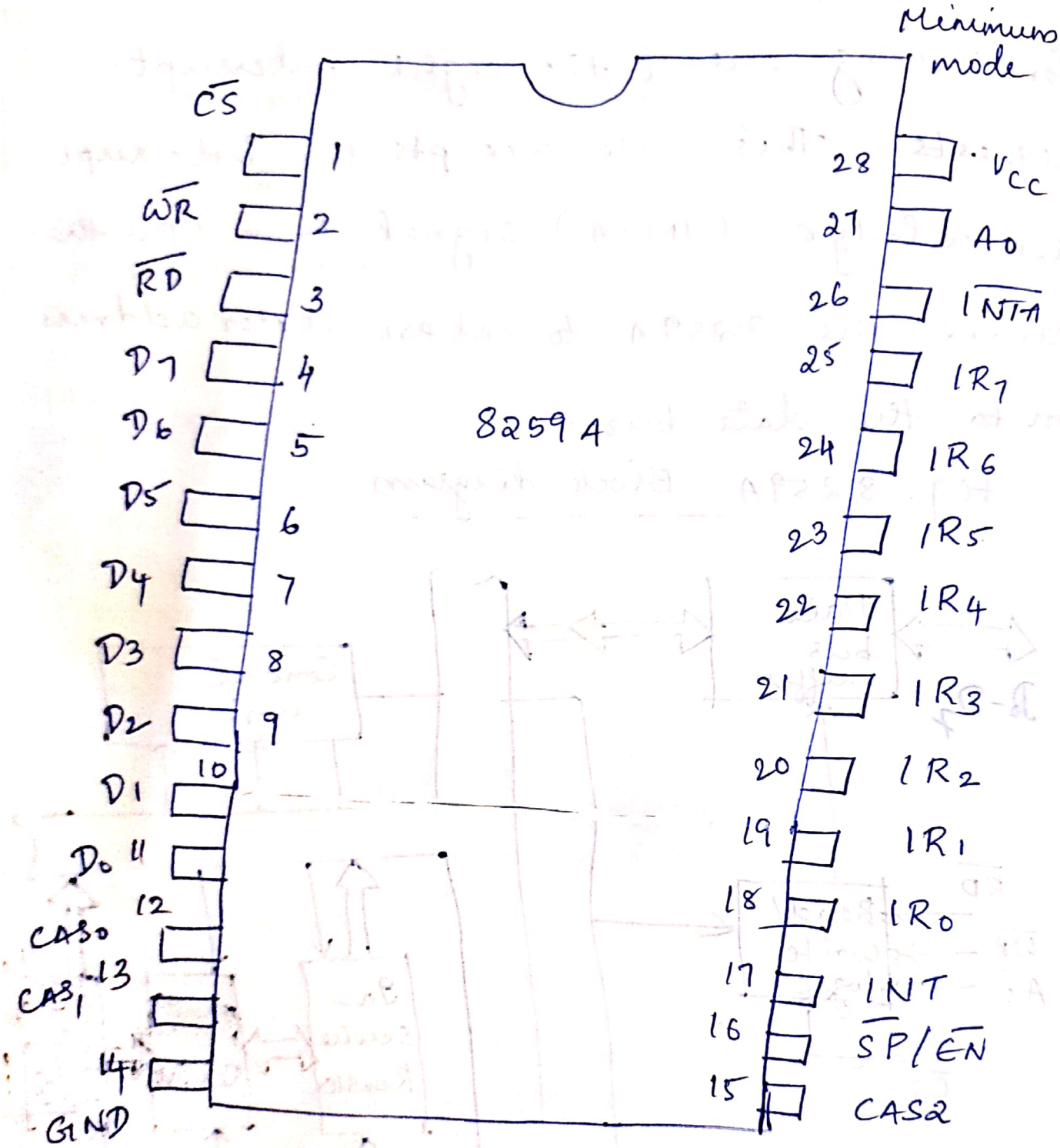


Fig: 8259 pin diagram

Scanned with CamScanner

Data bus buffer

The tristate bidirectional buffer interfaces internal 8259A bus to the microprocessor systems data bus. Control words, status and vector information pass through data buffer during read or write operations.

Read/write control logic

This circuit accepts and decodes commands from the CPU. This block also allows the status of the 8259A to be transferred onto the data bus.

Cascade buffer/comparator - This block stores and compares the IDs of all the 8259As used in the system. The three I/O pins CAS0-2 are outputs when the 8259A is used as a master. The same pins act as inputs when the 8259A is in the slave mode. The 8259A is in the master mode, sends the ID of the interrupting slave device on these lines. The

Scanned with CamScanner

slave thus selected will send its pre-programmed vector address on the data bus during the next $\overline{INTA}$ pulse.

Pin configuration of functional description of 8259

$\overline{CS}$ - This is an active low chip select signal for enabling $\overline{RD}$ and $\overline{WR}$ operations of 8259A. $\overline{INTA}$ function is independent of $\overline{CS}$.

$\overline{WR}$ - This pin is an active low write enable input to 8259A. This enables it to accept command words from CPU.

$\overline{RD}$ This is an active low read enable input to 8259A. A low on this line enables 8259A to release status onto the data bus of CPU.

$D_7 - D_0$ - These pins form a bidirectional data bus that carries 8 bit data either to control word or from status word registers. This also carries interrupt vector information

Scanned with CamScanner

CAS_0 - CAS_2 Cascade lines - A single 8259A provides eight vectored interrupts. If more interrupts are required, the 8259A is used in the cascade mode in which a master 8259A along with eight slaves 8259A can provide up to 64 vectored interrupt lines. Those three lines act as select lines for addressing the slaves 8259A

$\overline{SP}/\overline{EN}$ - This pin is a dual purpose pin. When the chip is used in buffered mode, it can be used as a buffer enable to control buffer transceivers. If this is not used in buffered mode then the pin is used as input to designate whether the chip is used as a master ($\overline{SP} = 1$) or a slave ($\overline{EN} = 0$).

$\overline{INT}$ - This pin goes high whenever a valid interrupt request is asserted. This is used to interrupt the CPU and is connected to the interrupt input of CPU.

Scanned with CamScanner

$IR_0 - IR_7$ (Interrupt requests) – These pins act as inputs to accept interrupt requests to the CPU. In the edge triggered mode, an interrupt service is requested by raising an IR pin from a low to a high state. It is held high until it is acknowledged and just by latching it to a high level, if used in the level triggered mode.

$\overline{INTA}$ (Interrupt acknowledge) – This pin is an input used to strobe-in 8259A interrupt vector data on to the data bus. In conjunction with $\overline{CS}$, $\overline{WR}$ and $\overline{RD}$ pins, this selects the different operations like, writing command words, reading status word etc.

The device 8259A can be interfaced with any CPU using either polling or interrupt. In polling, the CPU keeps on checking each

Scanned with CamScanner

peripheral device in sequence to ascertain if it requires any service from the CPU. If any such service request is noticed, the CPU serves the request and then goes on to the next device in sequence. After all the peripheral devices are scanned as above the CPU again starts from the first device. This type of systems operation results in the reduction of processing speed because most of the CPU time is consumed in polling the peripheral devices.

In the interrupt driven method, the CPU performs the main processing task til it is interrupted by a service requesting peripheral device. The net processing speed of these type of systems is high because the CPU serves the peripheral only if it receives the interrupt request. If more than one

Scanned with CamScanner

interrupt requests are received at a time, all the requesting peripherals are served one by one on priority basis. This method of interfacing may require additional hardware if number of peripherals to be interfaced is more than the interrupt pins available with the CPU.

Interrupt sequence in an 8086 system

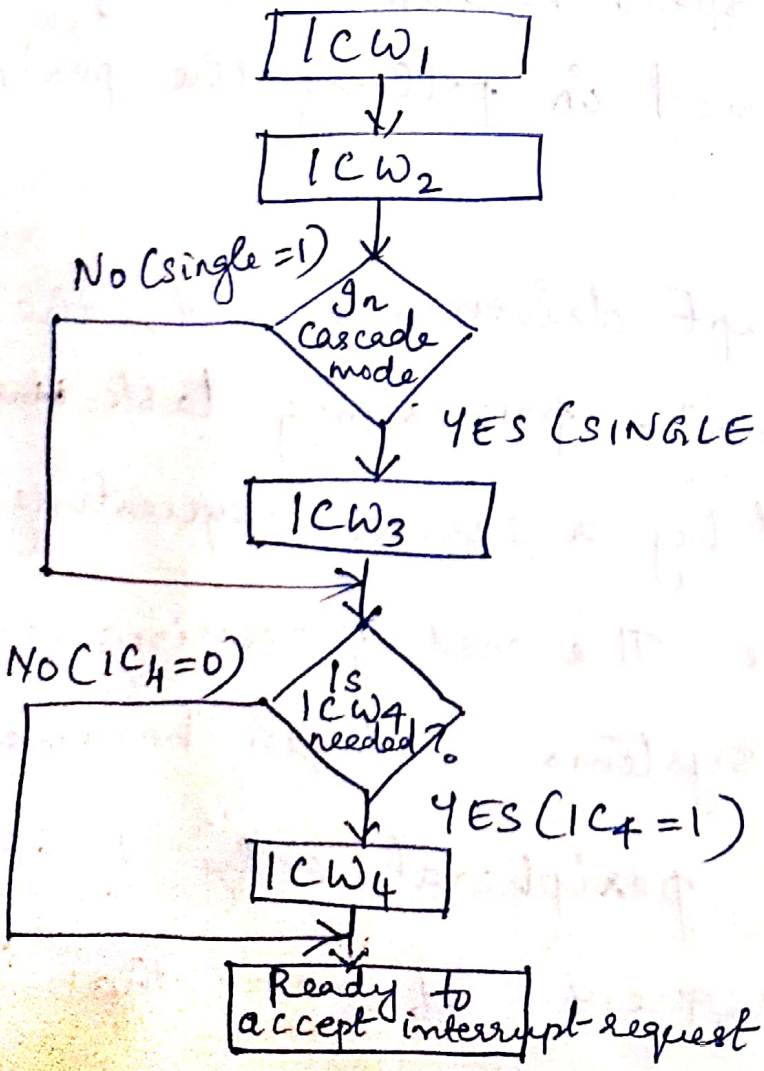


Fig: Initialisation Sequence of 8259A

Scanned with CamScanner

Interrupt sequence in an 8086-8259A systems is described as follows:-

1. One or more IR lines are raised high that set corresponding IRR bits.
2. 8259A resolves priority and sends an INT signal to CPU.
3. The CPU acknowledges with $\overline{INTA}$ pulse.
4. Upon receiving an $\overline{INTA}$ signal from the CPU, the highest priority ISR bit is set and the corresponding IRR bit is reset. The 8259A does not drive data bus during this period.
5. The 8086 will initiate a second $\overline{INTA}$ pulse. During this period 8259A releases an 8-bit pointer on to data bus from where it is read by the CPU.
6. This completes the interrupt cycle. The ISR bit is reset at the end of the second

Scanned with CamScanner

$\overline{INTA}$ pulse if automatic end of interrupt (AEOI) mode is programmed. Otherwise ISR bit remains set until an appropriate EOI command is issued at the end of interrupt subroutine.

Command words of 8259A

The command words of 8259A are classified into two groups, ie Initialization command words (ICW) and Operation Command Words (OCWs).

Initialization Command Words (ICWs)

Before it starts functioning, the 8259A must be initialized by writing two or four command words into the respective command word registers.

These are called as Initialization Command Words (ICWs).

If $A_0 = 0$ and $D_4 = 1$, the control word is recognized as ICW_1.

Scanned with CamScanner

It contains the control bits for edge / level triggered mode, single / cascade mode, call address interval and whether ICW_4 is required or not, etc.

If $A_0 = 1$, the control word is recognized as ICW_2.

The ICW_2 stores details regarding interrupt vector addresses.

Once ICW_1 is loaded, the following initialization procedure is carried out internally. (See fig)

a) The edge sense circuit is reset, ie by default 8259A interrupts are edge sensitive

b) IMR is cleared. (Interrupt mask register)

c) IR7 input is assigned the lowest priority

d) Slave mode address is set to 7.

e) Special mask mode is cleared and the status read is set to IRR.

f) If $IC_4 = 0$, all the functions of ICW_4 are set to zero.

Master/slave bit in ICW_4 is used in the buffered mode only.

Scanned with CamScanner

In 8086/88 based system, five most significant bits of the interrupt type byte are inserted in place of $T_7 - T_3$ respectively and the remaining 3 bits A_8, A_9 and A_{10} are inserted internally as 000.

Other 7 interrupt levels' vector addresses are internally generated automatically by 8259 using IR0 vector.

ICW_1 and ICW_2 are compulsory command words in initialization sequence of 8259A. ICW_3 & ICW_4 are optional.

ICW_3 is read only when there are more than one 8259As in the system, ie cascading is used (SNGL = 0).

The SNGL bit in ICW_1 indicates whether the 8259A is in the cascade mode or not.

The ICW_3 loads an 8 bit slave register.

Scanned with CamScanner

In the master mode ($S\overline{P} = 1$, or in buffer mode (M/S = 1 in ICW_4), the 8 bit slave register will be set bit wise to '1' for each slave in the system.

The requesting slave will then release the second byte of a CALL sequence.

In slave mode ($S\overline{P} = 0$, or if BUF = 1 & M/S = 0 in ICW_4) bits D_2 to D_0 identify the slave, ie 000 to 111 for slave 1 to slave 8. The slave compares the cascade inputs with these bits and if they are equal, the second byte of the CALL sequence is released by it on the data bus.

ICW_4 – The use of this command word depends on the IC_4 bit of ICW_1.

If $IC_4 = 1$, ICW_4 is used, otherwise it is neglected. Bit functions are as follows:-

SFNM – Special fully nested mode is selected if SFNM = 1.

BUF – If BUF = 1, the buffered mode is selected.

Scanned with CamScanner

ICW1

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	A_7	A_6	A_5	1	LTIM	ADI	SNGL	IC_4

- A_7-A_5 of interrupt vector address MCS 80/85 mode only. Don't care to 8086 mode
- LTIM: 1 - Level triggered, 0 - Edge triggered
- ADI: Call Address Interval. 1 - Interval of 4 bytes, 0 - Interval of 8 bytes. ADI = 1 for 8086 based system
- SNGL: 1 - Single, 0 - Cascade
- IC_4: 1 - ICW_4 needed, 0 - No ICW_4 needed

ICW2

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	T_7	T_6	T_5	T_4	T_3	A_{10}	A_9	A_8

T_7-T_3 - For 8086 system they are filled by the msb 5 bits of interrupt type and the lsb 3 bits are 0, pointing to Iro.

Fig: Initialization command words ICW1 and ICW2

Scanned with CamScanner

In the buffered mode, SP/EN acts as enable output and the master/slave is determined using the M/S bit of ICW_4.

M/S – If M/S = 1, 8259A is a master.

If M/S = 0, 8259A is a slave.

If BUF = 0, M/S is to be neglected.

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

$S_n = 1 - IR_n$ Input has a slave

$= 0 - IR_n$ Input does not have a slave

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

$D_2\ D_1\ D_0$ – 000 to 111 for IR_0 to IR_7 or slave 1 to slave 8

Fig: ICW_3 in Master & Slave mode

AEOI – If AEOI = 1, the automatic end of interrupt mode is selected.

MPM – If the MPM bit is 0, the MCS-85 system operation is selected and if MPM = 1, 8086/88 operation is selected.

Scanned with CamScanner

Fig. shows the ICW_4 bit positions

	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
ICW_4	1	0	0	0	SFNM	BUF	M/$\bar{S}$	AEOI	mPM

ICW_4 Bit functions

* Operation Command Words

Once 8259A is initialized using the Command words for initialisation; it is ready for its normal function, ie for accepting the interrupts but 8259A has its own ways of handling the received interrupts called as modes of operation. These modes of operations can be selected by programming ie writing three internal registers called as operation command word registers.

The data written into them (bit pattern) is called as operation command words.

In the operation command words OCW_1, OCW_2 and OCW_3, every bit corresponds to some operational feature of the mode selected except for a few bits those are

Scanned with CamScanner

either '1' or '0'.

The CW is used 'OCW1'

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	M_7	M_6	M_5	M_4	M_3	M_2	M_1	M_0

1 - Mask set
0 - Mask Reset

(a)

OCW2

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	R	SL	EOI	0	0	L_2	L_1	L_0

IR NO.	0	1	2	3	4	5	6	7
1	0	1	0	1	0	1	0	1
2	0	0	1	1	0	0	1	1
4	0	0	0	0	1	1	1	1

END OF INTERRUPT	0	0	1	NON SPECIFIC EOI COMMAND
	0	1	1	SPECIFIC EOI COMMAND
AUTOMATIC ROTATION	1	0	1	ROTATE ON NON SPECIFIC EOI COMMAND
	1	0	0	ROTATE IN AUTOMATIC EOI MODE(SET)
	0	0	0	ROTATE IN AUTOMATIC EOI (CLEAR)
SPECIFIC ROTATION	1	1	1	ROTATE ON SPECIFIC EOI COMMAND
	1	1	0	SET PRIORITY COMMAND*
	0	1	0	NO OPERATION

* In this mode L0 - L2 are used.

(b)

Scanned with CamScanner

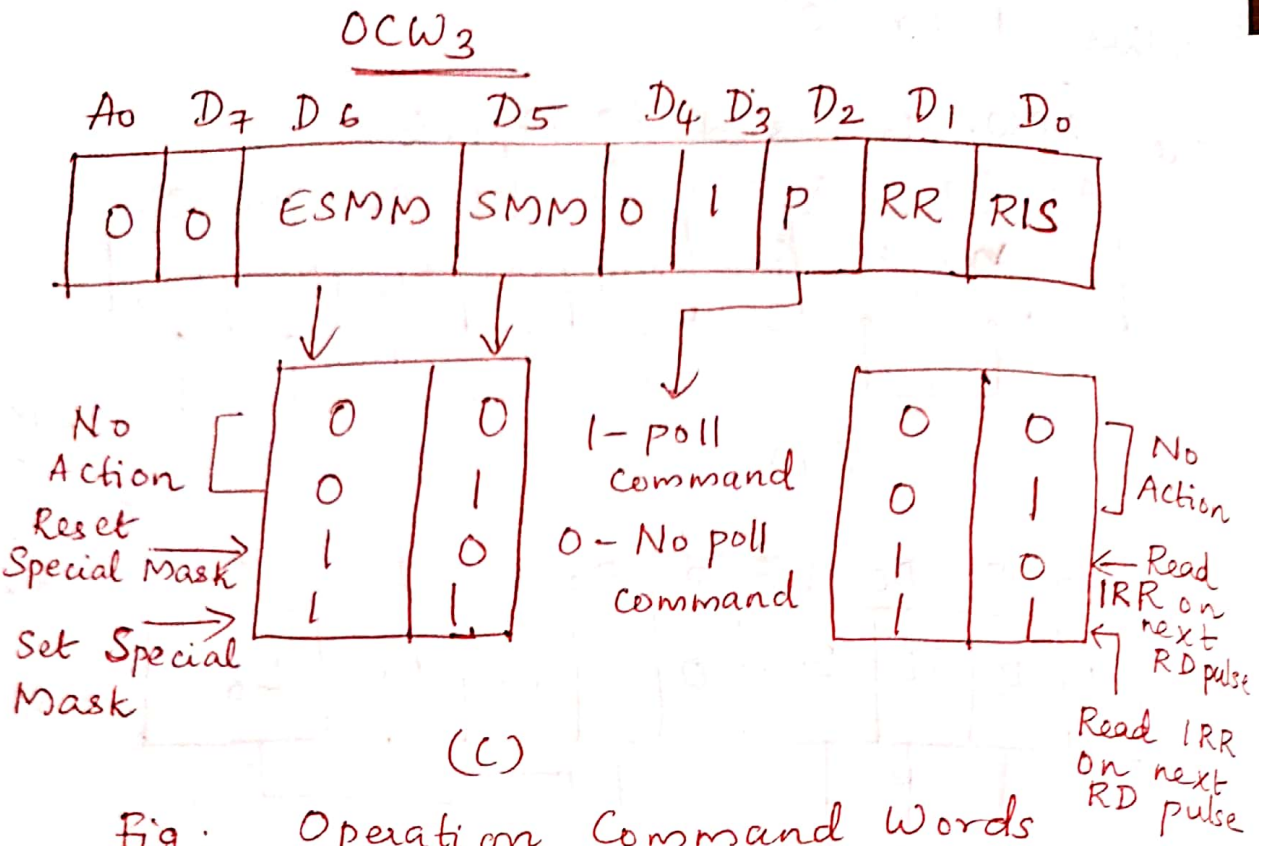


Fig: Operation Command Words

OCW_1 is used to mask the unwanted interrupt requests.

If the mask bit is 1, the corresponding interrupt request is masked, and if it is 0, the request is enabled.

In OCW_2, the three bits ie R, SL and EOI control the end of interrupt, the rotate mode and their combinations.

The three bits L_2, L_1, and L_0 in OCW_2 determine the interrupt level to be selected for operation, if the SL bit is active ie 1.

Scanned with CamScanner

In operation command word 3 (OCW_3), if the ESMM bit, ie Enable Special Mask mode bit is set to 1, the SMM bit is enabled to select or mask the Special Mask Mode. When ESMM bit is 0, the SMM bit is neglected.

If SMM bit, ie Special Mask Mode bit is 1, the 8259A will enter special mask mode provided ESMM = 1.

If ESMM = 1 and SMM = 0, the 8259A will return to the normal mask mode. (reset special mask)

* Operating modes of 8259

The different modes of operation of 8259A can be programmed by setting or resetting the appropriate bits of the ICWs or OCWs.

Fully Nested mode

This is the default mode of operation of 8259A.

IR_0 has the highest priority and IR_7 has the lowest one.

Scanned with CamScanner

When interrupt requests are noticed, the highest priority request amongst them is determined & the vector is placed on the data bus.

The corresponding bit of ISR is set & remains set till the microprocessor issues an EOI command just before returning from the service routine or the AEOI bit is set.

If the ISR (In Service) bit is set, all the same or lower priority interrupts are inhibited but higher levels will generate an interrupt, that will be acknowledged only if the microprocessor's Interrupt Enable flag (IF) is set.

The priorities can afterwards be changed by programming the rotating priority modes.

End of Interrupt (EOI)

The ISR bit can be reset either with

AEOI bit of ICW_1 or by EOI command, issued before returning from the interrupt service routine.

There are two types of EOI commands specific and non specific.

When 8259A is operated in the modes that preserve fully nested structure, it can determine which ISR bit is to be reset on EOI.

When non specific EOI command is issued to 8259A it will automatically reset the highest ISR bit out of those already set.

When a mode that may disturb the fully nested structure is used, the 8259A is no longer able to determine the last level acknowledged.

In this case, a specific EOI command is issued to reset a particular ISR bit.

An ISR bit that is masked by the corresponding IMR bit will not be cleared by a non specific EOI of 8259A, if it is in special mask mode.

Scanned with CamScanner

Automatic Rotation

This is used in applications where all the interrupting devices are of equal priority.

In this mode, an Interrupt Request (IR) level receives lowest priority after it is served while the next device to be served gets the highest priority in sequence.

Once all the devices are served like this the first device again receives highest priority

Automatic EOI mode

Till AEOI = 1 in ICW_4, the 8259A operates in AEOI mode.

In this mode, the 8259A performs a non specific EOI operation at the trailing edge of the last $\overline{INTA}$ pulse automatically.

This mode should be used only when a nested multilevel interrupt structure is not required with a single 8259A.

Scanned with CamScanner

Specific Rotation

In this mode, a bottom priority level can be selected, using L_2, L_1 and L_0 in OCW_2 and R = 1, SL = 1, EOI = 0.

The selected bottom priority fixes other priorities. If IR_5 is selected as a bottom priority, then IR_5 will have least priority and IR_4 will have a next higher priority.

Thus IR_6 will have the highest priority.

These priorities can be changed during an EOI command by programming the rotate on specific EOI command in OCW_2.

Special mask mode

In the special mask mode, when a mask bit is set in OCW_1, it inhibits further interrupts at that level and enables interrupt from other levels, which are not masked.

Edge and Level triggered mode

This mode decides whether the interrupt should be edge triggered or level triggered.

Scanned with CamScanner

If bit LTIM of $ICW_1 = 0$, they are edge triggered, otherwise the interrupts are level triggered.

Reading 8259 Status

The status of the internal registers of 8259A can be read using this mode.

The OCW_3 is used to read IRR and ISR while OCW_1 is used to read IMR.

Reading is possible only in no polled mode.

Poll command

In the polled mode of operation, the INT output of 8259A is neglected, though it functions normally, by not connecting INT output or by masking INT input of the microprocessor.

The poll mode is entered by setting P=1 in OCW_3.

The 8259A is polled by using software execution by microprocessor instead of the

Scanned with CamScanner

requests on INT input.

The 8259A treats the next $\overline{RD}$ pulse to the 8259A as an interrupt acknowledge.

An appropriate ISR bit is set, if there is a request.

The priority level is read and a data word is placed on to data bus, after $\overline{RD}$ is activated.

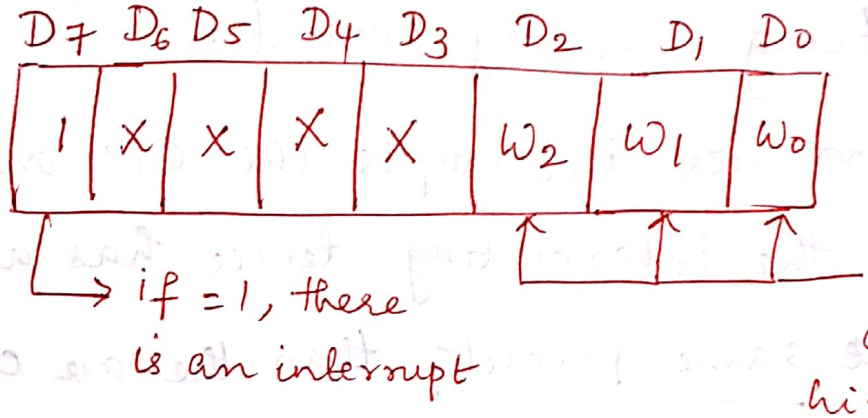


Fig: Data word of 8259

A poll command may give you more than 64 priority levels.

This has nothing to do with 8086 interrupt structure and interrupt priorities.

Special Fully Nested mode

This mode is used in more complicated systems, where cascading is used, and the priority has to be programmed in the master using ICW4.

Scanned with CamScanner

This is somewhat similar to the normal nested mode.

In this mode, when an interrupt request from a certain slave is in service, this slave can further send requests to the master, if the requesting device connected to the slave has higher priority than the one being currently served.

The master interrupts the CPU only when the interrupting device has a higher or the same priority than the one currently being served.

In normal mode, other requests than the one being served are masked out.

When entering the interrupt service routine the software has to check whether this is the only request from the slave.

This is done by sending a nonspecific EOI command to the slave and then reading its ISR and checking for zero.

Scanned with CamScanner

If its zero, a non specific EOI can be sent to the master, otherwise no EOI should be sent.

This mode is important since in the absence of this mode, the slave would interrupt the master only once and hence the priorities of the slave inputs would have been disturbed.

Buffered Mode

When the 8259A is used in the systems in which bus driving buffers are used on data buses (eg: cascade systems), the problem of enabling the buffers arises.

The 8259A sends a buffer enable signal on the $\overline{SP}/\overline{EN}$ pin, whenever data is placed on the bus.

Cascade mode

The 8259A can be connected in a system containing one master & 8 slaves (max.) to handle upto 64 priority levels.

The master controls the slaves using $CAS_0 - CAS_2$

Scanned with CamScanner

which act as chip select inputs (encoded) for slaves.

Here, the slave INT outputs are connected with master IR inputs.

When a slave request line is activated and acknowledged, the master will enable the slave to release the vector address during the second pulse of $\overline{INTA}$ sequence.

The cascade lines are normally low and contain slave address codes from the trailing edge of the first $\overline{INTA}$ pulse to the trailing edge of the second $\overline{INTA}$ pulse.

Each 8259A in the system must be separately initialized & programmed to work in different modes.

The ICW1 command must be issued twice, one for master and the other for the slave.

A separate address decoder is used to activate the chip select line of each 8259A.

Scanned with CamScanner

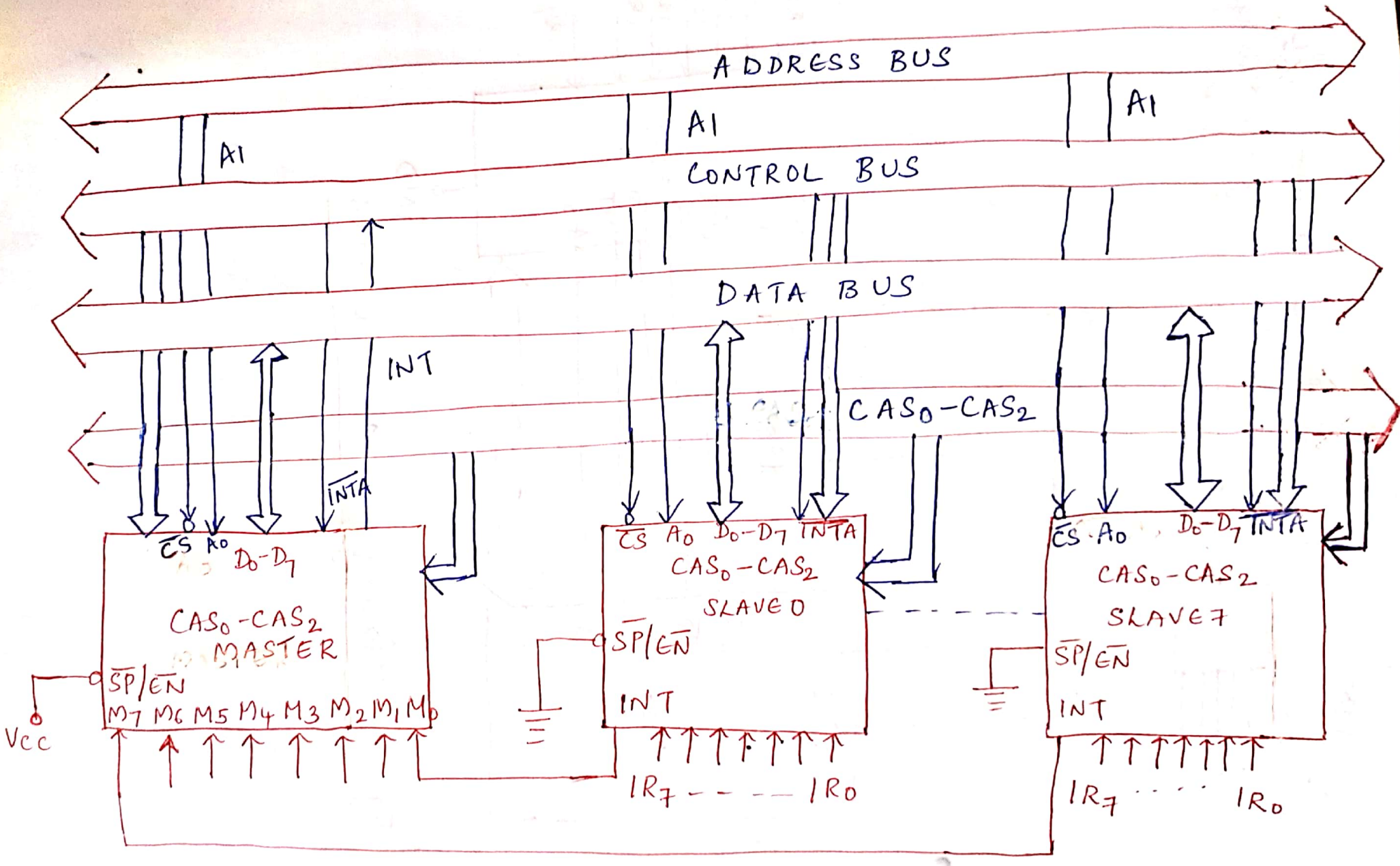


Fig: 8259A in Cascaded mode

Scanned with CamScanner

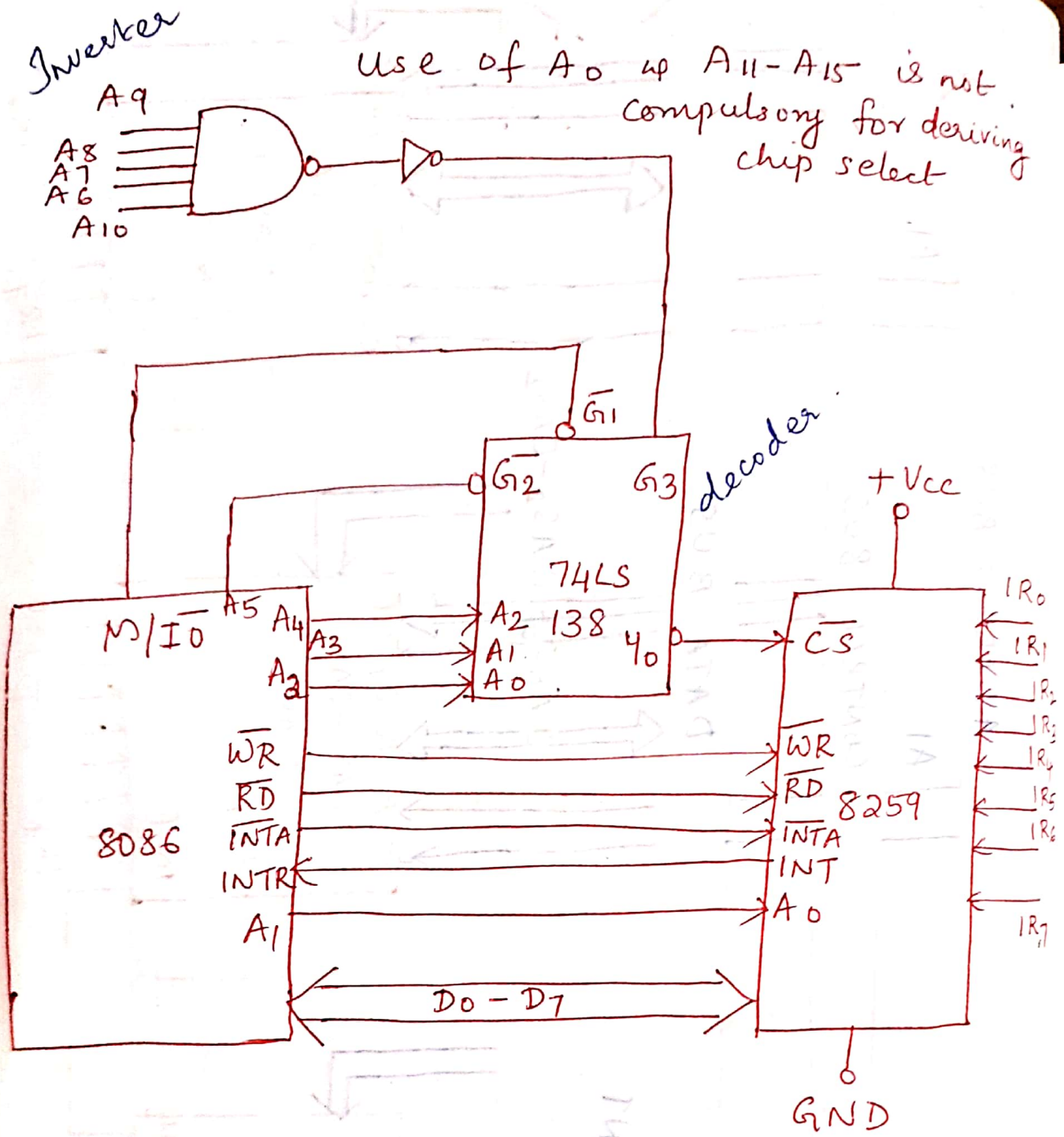


Fig: Interfacing 8259 with 8086

Scanned with CamScanner

Memory interfacing

(1) Semiconductor memory interfacing

Semiconductor memories are of two types viz RAM (Random Access Memory) and ROM (Read only memory).

Static RAM interfacing

The semiconductor RAMs are of broadly 2 types – static RAM and dynamic RAM. The semiconductor memories are organised as two dimensional arrays of memory locations. For eg. 4K×8 or 4k byte memory contains 4096 locations, where each location contains 8 bit data and only one of the 4096 locations can be selected at a time. Once a location is selected, all the bits in it are accessible using a group of conductors called 'data bus'. For addressing 4k bytes of memory, twelve address lines are required. To address a memory location out of N memory locations, we will require at least n bits of address, ie n address lines where $n = \log_2 N$. If the microprocessor has n address lines, then it is able to

address at the most N locations of memory where $2^n = N$. If out of N locations only P memory locations are to be interfaced, then the least significant p address lines out of the available n lines can be directly connected from the microprocessor to the memory chip while the remaining (n-p) higher order address lines may be used for address decoding (as inputs to chip selection logic). The memory address depends upon the hardware circuit used for decoding (i.e. the chip select ($\overline{CS}$). The output of the decoding circuit is connected with the $\overline{CS}$ pin of the memory chip. The general procedure of static memory interfacing with 8086 is as follows:-

1) Arrange the available memory chips so as to obtain 16 bit data bus width. The upper 8 bit bank is called 'odd address memory bank' and the lower 8 bit bank is called 'even address memory bank'.

2) Connect available memory address lines of

Scanned with CamScanner

memory chips with those of the micro-processor and also connect the memory $\overline{RD}$ and $\overline{WR}$ inputs to the corresponding processor control signals. Connect the 16 bit data bus of the memory bank with that of microprocessor 8086.

3) The remaining address lines of the micro-processor, $\overline{BHE}$ and A_0 are used for decoding the required chip select signals for odd and even memory banks. The $\overline{CS}$ of memory is derived from the o/p of the decoding circuit. As a good and efficient interfacing practise, the address map of the system should be continuous as far as possible ie there should be no windows in the map and no fold back space should be allowed. A memory location should have a single address corresponding to it, ie absolute decoding should be preferred and minimum hardware should be used for decoding. Linear decoding may be used to minimise the required hardware.

Scanned with CamScanner

Problem 1

Interface two 4K x 8 EPROMs and two 4K x 8 RAM chips with 8086. Select suitable maps.

Soln:

We know that after reset the IP and CS are initialised to form address FFFF0H. This address must lie in the EPROM. The address of RAM may be selected any where in the 1MB address space of 8086 but we will select the RAM address such that the address map of the system is continuous as in table.

Memory map for problem 1

Address	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	
	EPROM							8K x 8				
(last address)	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}			(2 x 4K = 8K)
	1	1	1	1	1	1	1	1	1			

8K x 8

	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	
FE000H	1	1	1	1	1	1	1	0	0	0	0	
	EPROM							8 K x 8				
	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}			
	0	0	0	0	0	0	0	0	0			

A-10
B-11
C-12
D-13
E-14
F-15
121

Scanned with CamScanner

Address	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_{09}
FDFFFH	1	1	1	1	1	1	0	1	1	1	1

RAM 8K x 8

	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
	1	1	1	1	1	1	1	1	1

FC000H	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_{09}
	1	1	1	1	1	1	0	0	0	0	0

	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
	0	0	0	0	0	0	0	0	0

RAM 8K x 8

Total 8K bytes of EPROM need 13 address lines $A_0 - A_{12}$ (since $2^{13} = 8K$). Address lines $A_{13} - A_{19}$ are used for decoding to generate the chip select. the $\overline{BHE}$ signal goes low when a transfer is at odd address or higher byte of data is to be accessed. Let us assume that the latched address, $\overline{BHE}$ and demultiplexed data lines are readily available for interfacing. figure shows the interfacing diagram for the memory system The memory system in this example contains in total four 4K x 8 memory chips. The two 4K x 8 chips of RAM and ROM are arranged in parallel to obtain 16 bit data bus width.

Scanned with CamScanner

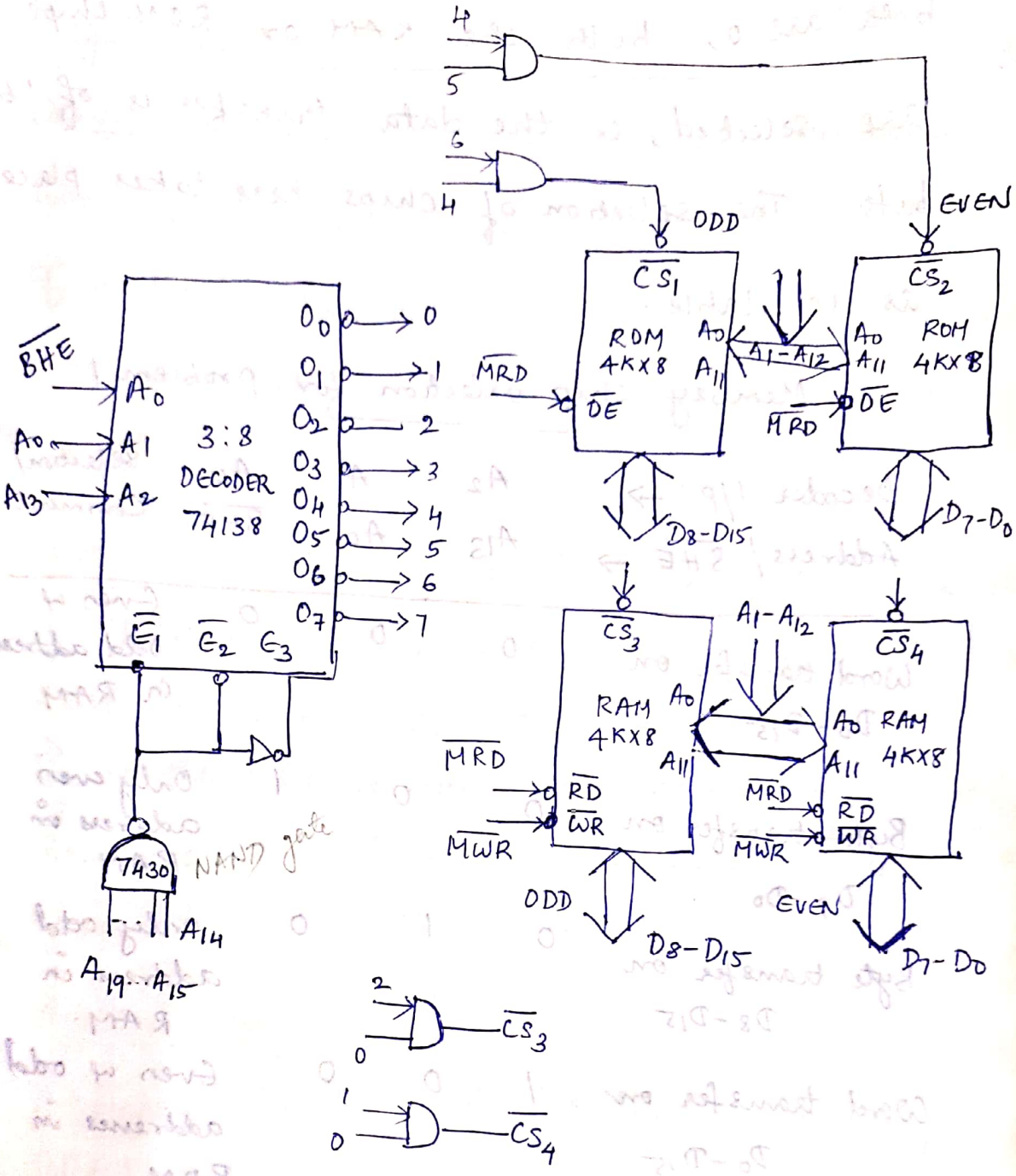


Fig: Interfacing problems

If A_0 is 0 ie the address is even and is in RAM, then the lower RAM chip is selected indicating 8 bit transfer is at an odd address. If the selected addresses are in ROM, the respective ROM chips are selected. If at a time A_0 and $\overline{BHE}$

both are 0, both the RAM or ROM chips are selected, ie the data transfer is of 16 bits. The selection of chips here takes place as in table.

Memory chip selection for problem 1

Decoder I/P → / Address / $\overline{BHE}$ →	A_2 / A_{13}	A_1 / A_0	A_0 / $\overline{BHE}$	Selection / Comment
Word transfer on $D_0 - D_{15}$	0	0	0	Even & odd address in RAM
Byte transfer on $D_7 - D_0$	0	0	1	Only even address in RAM.
Byte transfer on $D_8 - D_{15}$	0	1	0	Only odd address in RAM.
Word transfer on $D_0 - D_{15}$	1	0	0	Even & odd addresses in ROM
Byte transfer on $D_0 - D_7$	1	0	1	Only even address in ROM
Byte transfer on $D_8 - D_{15}$	1	1	0	Only odd address in ROM.

Scanned with CamScanner

Program 2.

Design an interface between 8086 CPU and two chips of 16K x 8 EPROM and two chips of 32K x 8 RAM. Select the starting address of EPROM suitably. The RAM address must start at 00000H.

Soln: The last address in the map of 8086 is $FFFFF_H$. After resetting, the processor starts from $FFFF0_H$. Hence this address must lie in the address range of EPROM. Fig shows the interfacing diagram and table shows complete map of the system.

Address map

Addresses	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}
FFFFFH	1	1	1	1	1	1	1	1	1	1
	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
	1	1	1	1	1	1	1	1	1	1
	32 KB EPROM									
F8000H	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}
	1	1	1	1	1	0	0	0	0	0
	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
	0	0	0	0	0	0	0	0	0	0
	32 KB EPROM									
0FFFFH	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}
	0	0	0	0	1	1	1	1	1	1
64 KB RAM	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
	1	1	1	1	1	1	1	1	1	1

Add banks of

Scanned with CamScanner

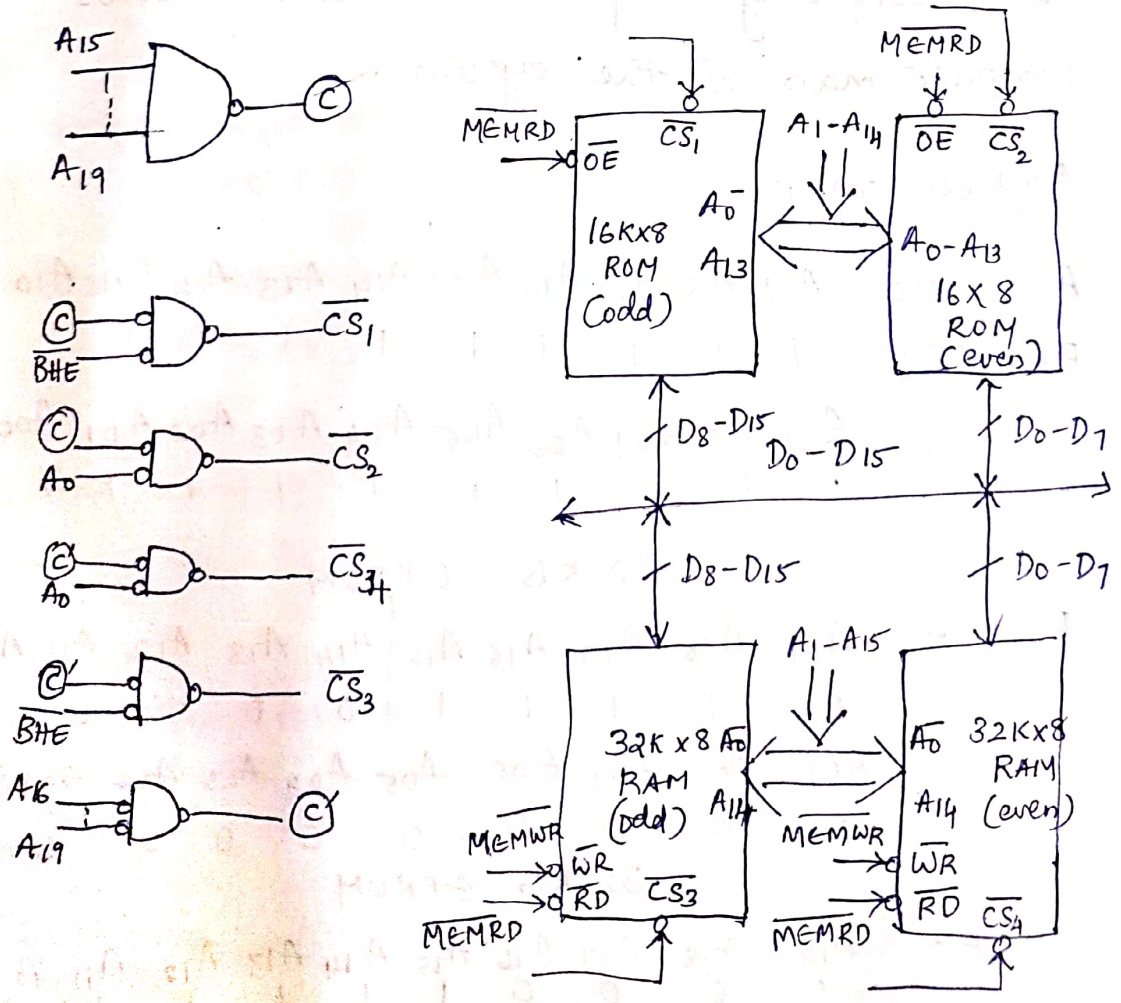
	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9
00000H	0	0	0	0	0	0	0	0	0	0	0

A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	0	0	0	0

64 KB RAM.

It is better not to use a decoder to implement the above map because it is not Continuous ie there is some unused address space between the last RAM address (0FFFFH) and the the first EPROM address (F8000H). Hence the logic is implemented using logic gates as in figure.

Interfacing problem 2



Scanned with CamScanner

Let us select a variable C for memory address pulse ie output of 6-input NAND gate. $\overline{BHE}$ is abbreviated as B. The chip selection logic can be designed as in table.

I/P		O/P	
A_0	B ($\overline{BHE}$)	C_1	C_2
0	0	0	0
0	1	1	0
1	0	0	1
1	1	1	1

$$C_2 = A_0$$
$$C_1 = \overline{BHE}$$

To find out $\overline{CS_1}$ and $\overline{CS_2}$ we will have to combine C_1 & C_2 with C.

I/P			O/P	
C_1	C_2	C	$\overline{CS_1}$	$\overline{CS_2}$
0	0	0	0	0
1	0	0	1	0
1	1	0	1	1
0	1	0	0	1

Table shows that $\overline{CS_1} = C + C_1 \neq C + \overline{BHE}$ and

$$\overline{CS_2} = C + C_2 = C + A_0$$

Similarly we can find CS_3 and CS_4.

Scanned with CamScanner

In the max. mode, the 8086 is prepared to operate in multiprocessor mode. All the control signals in this mode are derived by a chip called bus controller (8288). While interfacing memory or I/o devices in max. mode, the control signals derived by the bus controller (8288) are used. Instead of $\overline{RD}$ for memory operations $\overline{MRDC}$ (memory read command) output of 8288 is used and instead of $\overline{WR}$, $\overline{AMWC}$ (advanced memory write command) output of 8288 is used. The rest of the procedure in min. and max. mode is similar while interfacing the memories as well as peripherals Along with these $\overline{MRDC}$, $\overline{IOWC}$, $\overline{AIOWC}$, $\overline{AMWC}$ two more signals, namely $\overline{IOWC}$ and $\overline{MWTC}$ are provided by 8288. These signals are similar to $\overline{AIOWC}$ and $\overline{AMWC}$ in significance and operation but there activation is delayed by a clock cycle as compared to $\overline{AIOWC}$ and $\overline{MWTC}$.

Scanned with CamScanner

Dynamic RAM interfacing

Whenever a large memory is required in a microprocessor system, the memory subsystem is generally designed using dynamic RAM as it has various advantages eg: higher packaging density, lower cost and less power consumption. A typical static RAM cell may require six transistors while the dynamic RAM cell requires only a transistor along with a capacitor. Hence it is possible to obtain higher packaging density and hence low cost units are available. On the other side, there are some serious draw backs of dynamic RAMs. The basic dynamic RAM cell uses a capacitor to store the charge as a representation of data. This capacitor is manufactured as a diode that is reverse biased so that the storage capacitance comes into the picture. This storage capacitance is utilised for storing the charge representation of data but the reverse biased diode has a leakage current that tends to discharge the capacitor giving rise to the possibility of

Scanned with CamScanner

data loss. To avoid this possible data loss the data stored in dynamic RAM cell must be refreshed after a fixed time interval regularly. The process of refreshing the data in the RAM is known as refresh cycle. This activity is similar to reading the data from each cell of the memory, independent of the requirement of microprocessor, regularly. During this refresh period all other operations (accesses) related to the memory subsystem are suspended. Hence the refresh activity causes loss of time, resulting in reduced system performance. Keeping in view the advantages of dynamic RAM, like low power consumption, higher packaging density and low cost, most of the advanced computer systems are designed using dynamic RAMs of course at the cost of operating speed. The refresh mechanism and the additional hardware required makes the interfacing hardware in case of dynamic RAM, more complicated, as compared to static RAM interfacing circuit.

Scanned with CamScanner

A dedicated hardware chip called as dynamic RAM controller is the most important part of the interfacing circuit.

The refresh cycle is different from the memory read cycle in the foll: aspects.

1. The memory address is not provided by the CPU address bus, rather it is generated by a refresh mechanism counter known as refresh counter.
2. Unlike memory read cycle, more than one memory chip may be enabled at a time so as to reduce the number of total memory refresh cycles.
3. The data enable control of the selected memory chip is deactivated, and data is not allowed to appear on the system data bus during refresh, as more than one memory units are refreshed simultaneously. This is to avoid the data from the different chips to appear on the bus simultaneously.
4. Memory read is either a processor initiated or an external bus master initiated operation while memory refresh is an independent

Scanned with CamScanner

regular activity, initiated and carried out by the refresh mechanisms.

Generally, dynamic RAM is available in units of several kilobits to even megabits of memory (it is not in terms of bytes or nibbles as in static RAM). This memory is arranged internally in a two-D matrix array so that it will have n rows and m columns. The row address n and column address m are important for the refreshing operation. For eg. a typical 4K bit of dynamic RAM chip has an internally arranged bit array of dimension 64 x 64 ie 64 rows and 64 columns. Thus the row address and column address will require 6 bits each. These 6 bits for each row address and column address will be generated by the refresh counter, during refresh cycles. A complete row of 64 cells is refreshed at a time to minimize the refreshing time. Thus the refresh counter needs to generate only row addresses. The row addresses are multiplexed, over low order address lines.

Scanned with CamScanner

The refresh signals act to control the multiplexer ie when refresh cycle is in process the refresh counter puts the row address over the address bus for refreshing. Otherwise, the address bus of the processor is connected to the address bus of DRAM, during normal processor initiated activities. A timer, called refresh timer derives a pulse for refreshing action after each refresh interval, which can be qualitatively defined as the time for which a dynamic RAM cell can hold data charge level practically constant, ie no data loss takes place. Suppose the typical dynamic RAM chip has 64 rows, then each row should be refreshed after each refresh interval or in other words all the 64 rows are to be refreshed in a single refresh interval. This refresh interval depends upon the manufacturing technology of the dynamic RAM cell. It may range anywhere from 1 ms to 3 ms. Consider 2 ms as a typical refresh time interval. Hence, the frequency of the refresh

Scanned with CamScanner

Pulses will be calculated as shown.

Refresh Time (per row) $t_r = \frac{2 \times 10^{-3}}{64}$

Refresh frequency $f_r = \frac{64}{2 \times 10^{-3}} = 32 \times 10^{3}$ Hz

The block diagram explains the refreshing logic and 8086 interfacing with dynamic RAM.

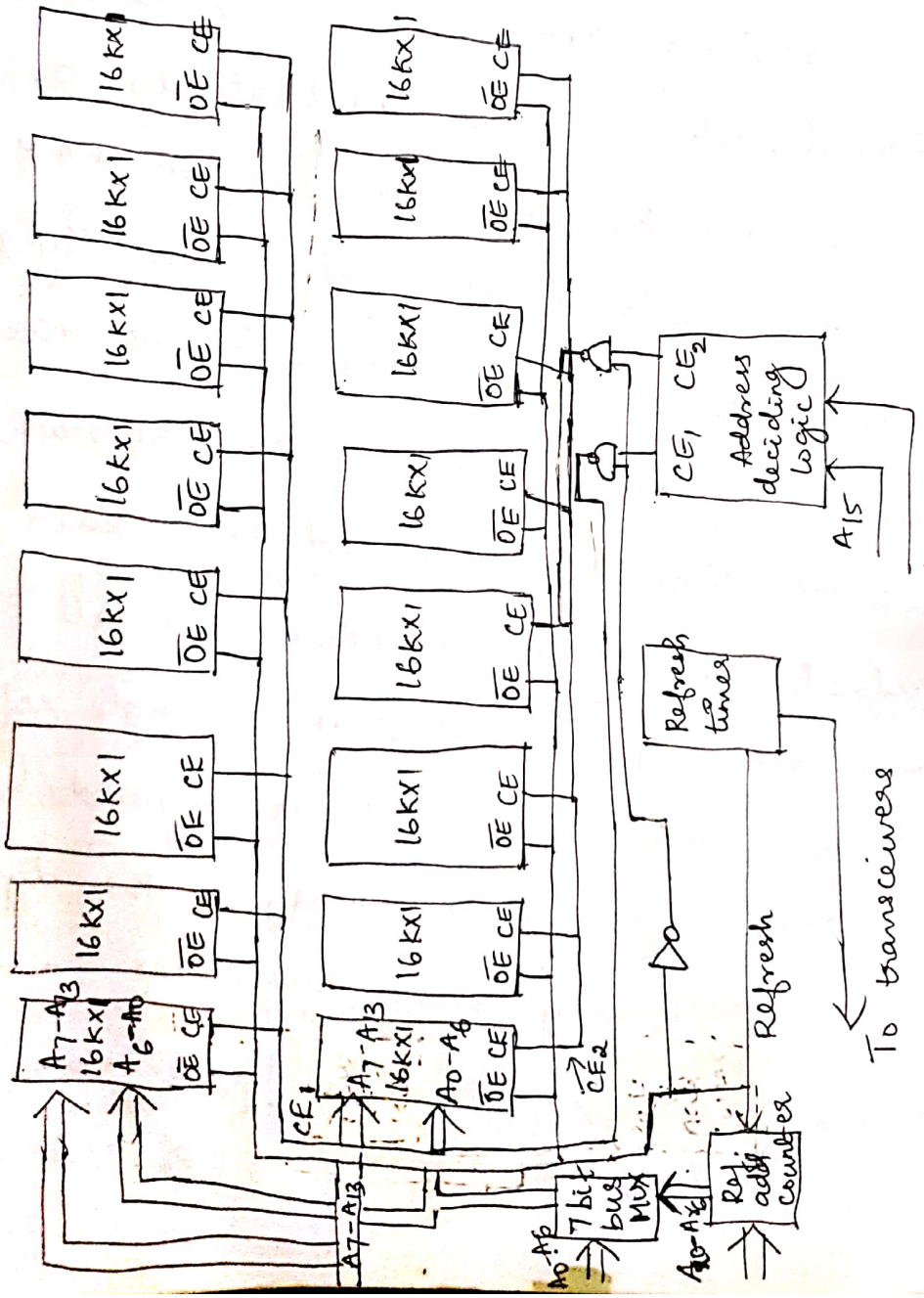


Fig: Dynamic RAM refreshing logic

Each of the used chips is a 16K x 1 bit dynamic RAM cell array. The system contains two 16K byte dynamic RAM cell array. All the address and data lines are assumed to be available from an 8086 microprocessor systems. The $\overline{OE}$ pin controls output data buffers of the memory chips. The CE pins are active high chip selects of memory chips. The refresh cycle starts if the refresh output of the refresh timer goes high, $\overline{OE}$ and CE also tend to go high. The high CE enables the memory chip for refreshing, while high $\overline{OE}$ prevents the data from appearing on the data bus, as said in the description of memory refresh cycle. The 16K x 1-bit dynamic RAM has an internal array of 128 x 128 (16x8, 16x8) cells, requiring 7 bits for row addresses. The lower order seven lines $A_0 - A_6$ are multiplexed with the refresh counter output $A_{R0} - A_{R6}$. The logic is shown in figure. The pin assignments for 2164 dynamic RAM is in figure. The $\overline{RAS}$ and $\overline{CAS}$ are row and column address strobes

Signal that validates other

Fig: Dynamic RAM refreshing logic

A 14

External devices, for
es that accept the data from a

I/O device with a CPU:

the data bus of the I/O por

ddress of the device and u

ry out device operations,
otherwise connect IOWR
ls are combined with the

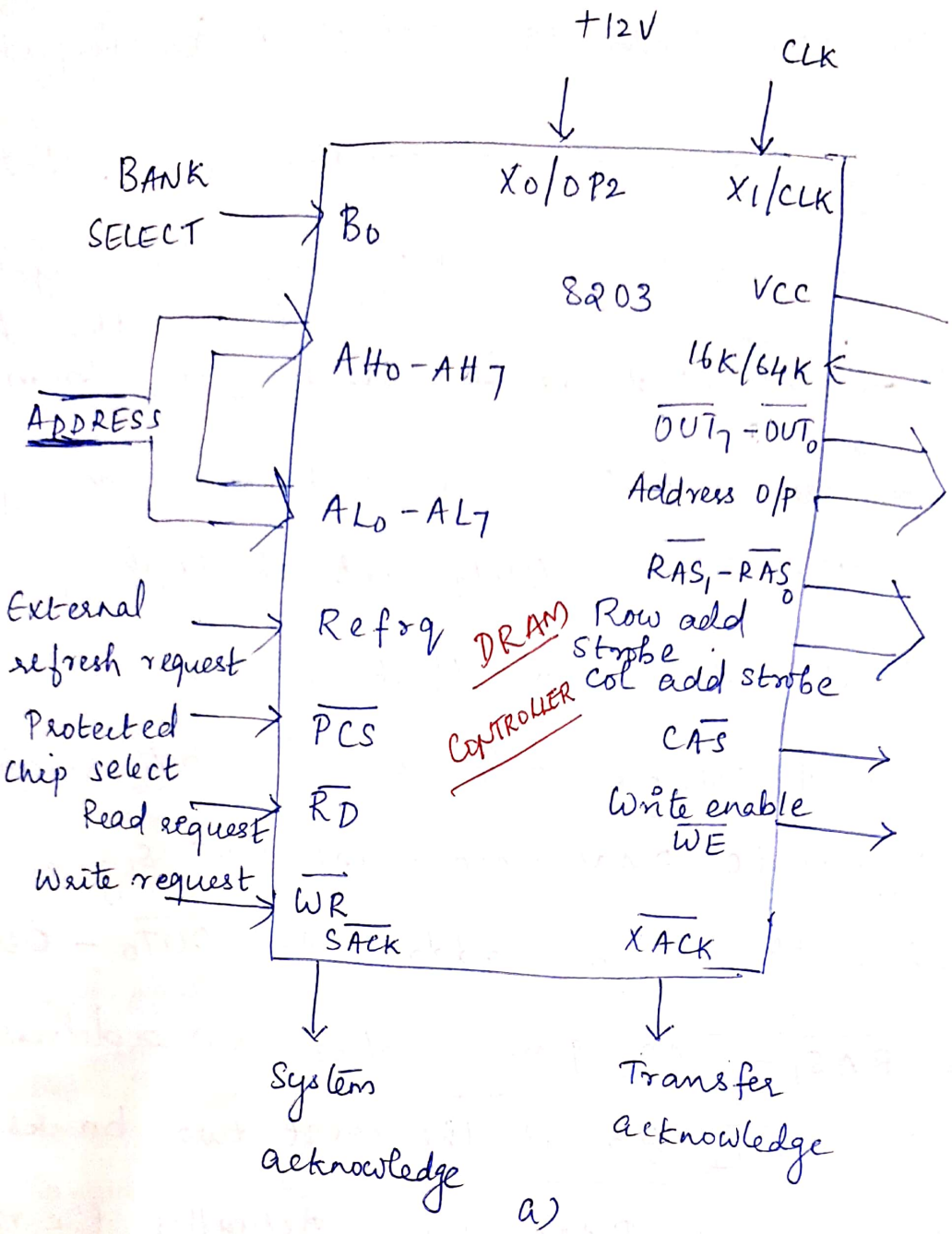
Scanned with CamScanner

And are driven by the dynamic RAM Controller outputs. $A_0 - A_7$ lines are the row or column address lines, driven by $OUT_0 - OUT_7$ outputs of the controller. The $\overline{WE}$ pin indicates memory write cycles. The D_{IN} and D_{OUT} pins are data pins for write and read operations, respectively. In practical circuits the refreshing logic is integrated inside dynamic RAM Controller chips like 8203, 8202 and 8207 etc. Intel's 8203 is a dynamic RAM Controller that supports 16K or 64K dynamic RAM chips. This selection is done using pin 16K/64K. If it is high, the 8203 is configured to control 16K dynamic RAM, else it controls 64K dynamic RAM. The address inputs of 8203 Controller accept address lines A_1 to A_{16} on lines $AL_0 - AL_7$ and $AH_0 - AH_7$. The B_0 line is used to select the even or odd bank. The $\overline{RD}$ and $\overline{WR}$ signals decode whether the cycle is a memory read or memory

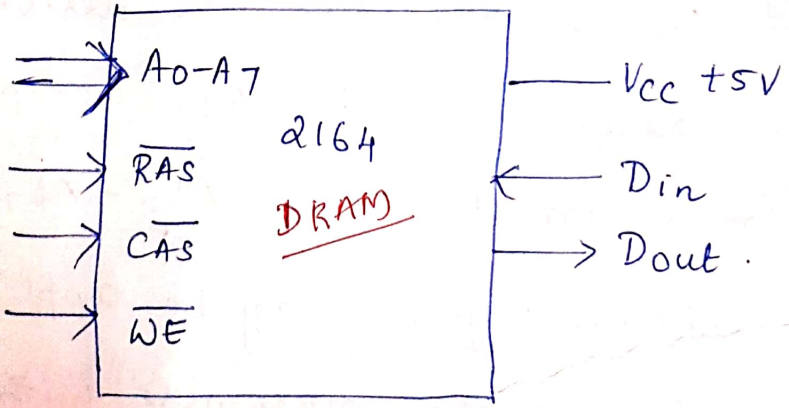
Scanned with CamScanner

write cycle and are accepted as inputs or to 8203 from the microprocessor. The WE signal specifies the memory write cycle and is an output from 8203 that drives the $\overline{WE}$ input of dynamic RAM memory chip. The $\overline{OUT_0}$ – $\overline{OUT_7}$ set of eight pins is an 8 bit output bus that carries multiplexed row and column address of a bit location in dynamic RAM chip. The $\overline{CAS}$ signal strobes the column address on $\overline{OUT_0}$ – $\overline{OUT_7}$. The $\overline{RAS_1}$ – $\overline{RAS_0}$ pins strobes row address on $\overline{OUT_0}$ – $\overline{OUT_7}$ for at the most two banks of 2164 dynamic RAM chips. Actually the row and column addresses are derived from the address lines A_1 – A_{16} accepted by the Controller on its inputs AL_0 – AL_7 and AH_0–AH_7. An external crystal may be applied between X_0 and X_1 pins, otherwise with the OP_2 pin at +12V, a clock signal may be applied at the pin CLK. The $\overline{PCS}$ pin accepts the chip select signal desired by an address decoder.

Scanned with CamScanner



a)



b)

Fig: a) Dynamic RAM controller
b) 1 bit Dynamic RAM.

Scanned with CamScanner

The REFREQ Pin is used whenever the memory refresh cycle is to be initiated by an external signal. The XACK signal indicates that data is available during a read cycle or it has been written if it is a write cycle. It can be used as a strobe for data latches (data flows) or as a ready signal to the processor. The SACK output signal marks the beginning of a memory access cycle. If a memory request is made during a memory refresh cycle, the SACK signal is delayed till the starting of memory read or write cycle. Fig. shows how the 8203 can be used to control a 256 K bytes memory subsystem for a maximum mode 8086 microprocessor system. This design assumes that data and address buses are inverted and latched, hence the inverting buffers and inverting latches are used (8283 - inverting buffer and 8287 - inverting latch). We can interface 512 K byte dynamic RAM with 8086 using an advanced dynamic RAM controller 8208. Most of the functions of 8208 and 8203 are similar but 8208 can be used to refresh the dynamic RAM using DMA approach. The memory system is divided into even and odd banks of 256 K byte each as

Mod-4

Fig: Interfacing 2164 (DRAM) using 8203.

(DRAM cont...)

Scanned with CamScanner

required for an 8086 system. The inverted $\overline{A}ACK$ output of 8208, latches the A0 and $\overline{BHE}$ signals required for selecting the banks. If the latched bank select signal and the $\overline{WE}$ / PCLK output of 8208 both go low it indicates a write operation to the respective bank.

Scanned with CamScanner

CS305- MPMC

Module 4

Interfacing memory, I/O,8255- detailed study, architecture, control word format and modes of operation, architecture and modes of operation of 8279 and 8257 (just mention the control word, no need to memorize the control word format)

Module 4- Memory interfacing

SEMICONDUCTOR MEMORY INTERFACING

Semiconductor memories are of two types, viz. RAM (Random Access Memory) and ROM (Read Only Memory).

Static RAM Interfacing

The semiconductor RAMs are of broadly two types–static RAM and dynamic RAM.
The semiconductor memories are
organised as two dimensional arrays of memory locations. For example, 4K × 8 or 4K byte memory contains 4096 locations, where each location contains 8-bit data and only one of the 4096 locations can be selected at a time. Once a location is selected all the bits in it are accessible using a group of conductors called 'data bus'. for addressing 4K bytes of memory, twelve address lines are required.
to address a memory location out of N memory locations , we will require at least n bits of address, i.e. n address lines where $n = \text{Log}_2 N$. if the microprocessor has n address lines, then it is able to address at the most N locations of memory, where $2^n = N$. if out of N locations only P memory locations are to be interfaced, then the least significant p address lines out of the available n lines can be directly connected from the microprocessor to the memory chip while the remaining $(n–p)$ higher order address lines may be used for address decoding (as inputs to the chip selection logic). The memory address depends upon the hardware circuit used for decoding the chip select ($\overline{\text{CS}}$). The output of the decoding circuit is connected with the $\overline{\text{CS}}$ pin of the memory chip.

The general procedure of static memory interfacing with 8086 is briefly described as follows:

1. Arrange the available memory chips so as to obtain 16-bit data bus width. The upper 8-bit bank is called 'odd address memory bank' and the lower 8-bit bank is called 'even address memory bank',
2. Connect available memory address lines of memory chips with those of the microprocessor and also connect the memory $\overline{\text{RD}}$ and $\overline{\text{WR}}$ inputs to the corresponding processor control signals. Connect the 16-bit data bus of the memory bank with that of the microprocessor 8086.
3. The remaining address lines of the microprocessor, $\overline{\text{BHE}}$ and A_0 are used for decoding the required chip select signals for the odd and even memory banks. The $\overline{\text{CS}}$ of memory is derived from the O/P of the decoding circuit.

As a good and efficient interfacing practice, the address map of the system should be continuous as far as possible, i.e. there should be no windows in the map and no fold back space should be allowed. A memory location should have a single address corresponding to it, i.e. absolute decoding should be preferred, and minimum hardware should be used for decoding. linear decoding may be used to minimise the required hardware.

Program

Interface two 4K × 8 EPROMS and two 4K × 8 RAM chips with 8086. Select suitable maps.

Solution We know that, after reset, the IP and CS are initialised to form address FFFF0H. Hence, this address must lie in the EPROM. The address of RAM may be selected any where in the 1MB address space of 8086, but we will select the RAM address such that the address map of the system is continuous, as shown in Table

Memory Map for Problem

Address	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
						EPROM						8K × 8								

Address	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
FE000H	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
FDFFFH	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
						RAM						8K × 8								
FC000H	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

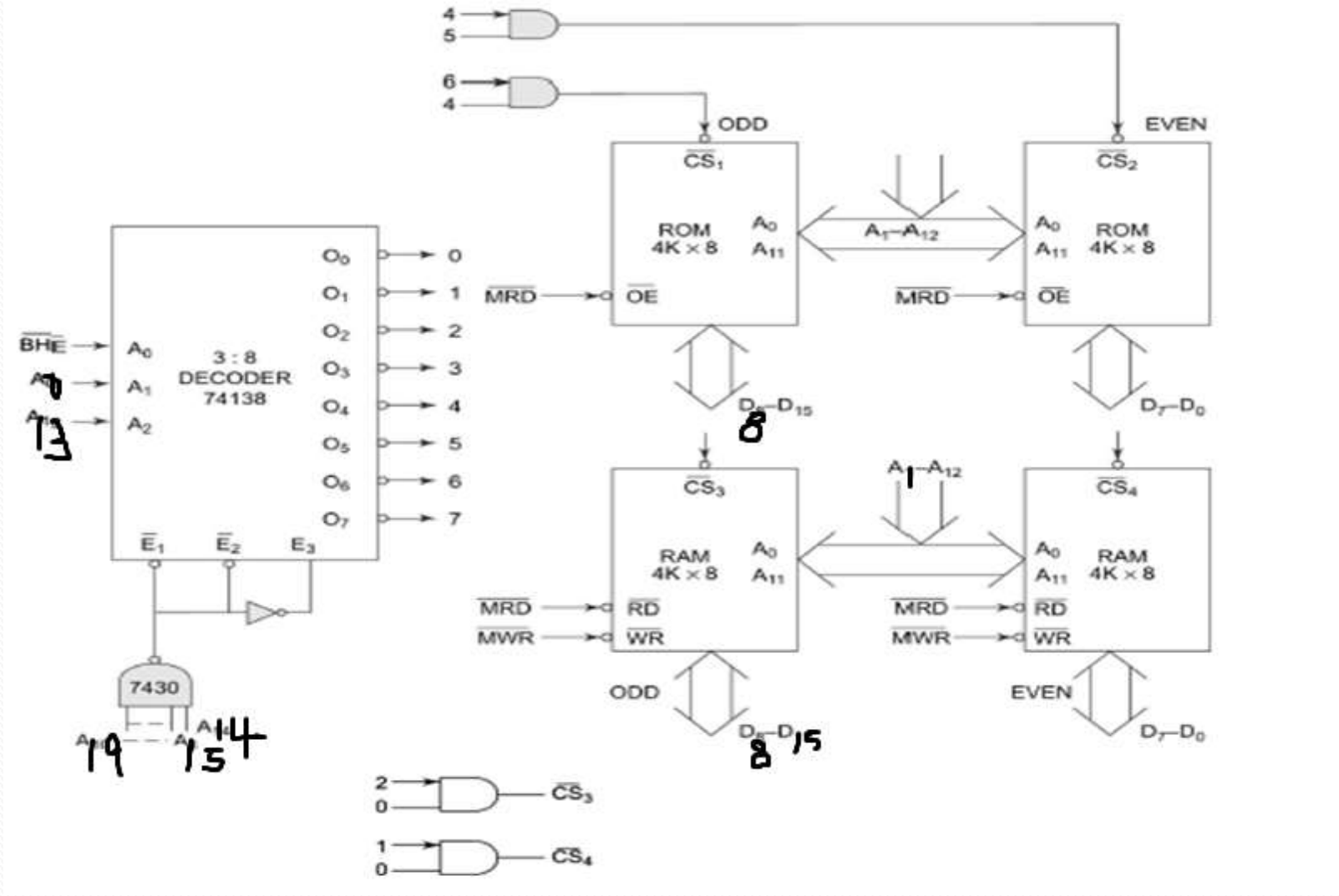
Total 8K bytes of EPROM need 13 address lines A_0–A_{12} (since $2^{13} = 8K$). Address lines A_{13}–A_{19} are used for decoding to generate the chip select. The $\overline{BHE}$ signal goes low when a transfer is at odd address or higher byte of data is to be accessed. Let us assume that the latched address, $\overline{BHE}$ and demultiplexed data lines are readily available for interfacing. Figure shows the interfacing diagram for the memory system.

The memory system in this example contains in total four 4K × 8 memory chips.

The two 4K × 8 chips of RAM and ROM are arranged in parallel to obtain 16-bit data bus width. If A_0 is 0, i.e. the address is even and is in RAM, then the lower RAM chip is selected indicating 8-bit transfer at an even address. If A_0 is 1, i.e. the address is odd and is in RAM, the $\overline{BHE}$ goes low, the upper RAM chip is selected, further indicating that the 8-bit transfer is at an odd address. If the selected addresses are in ROM, the respective ROM chips are selected. If at a time A_0 and $\overline{BHE}$ both are 0, both the RAM or ROM chips are selected, i.e. the data transfer is of 16 bits. The selection of chips here takes place as shown in Table .

Memory Chip Selection for Problem :

Decoder I/P → *Address/ $\overline{BHE}$ →*	A_2 A_{13}	A_1 A_0	A_0 $\overline{BHE}$	*Selection/* *Comment*
Word transfer on $D_0 - D_{15}$	0	0	0	Even and odd addresses in RAM
Byte transfer on $D_7 - D_0$	0	0	1	Only even address in RAM
Byte transfer on $D_8 - D_{15}$	0	1	0	Only odd address in RAM
Word transfer on $D_0 - D_{15}$	1	0	0	Even and odd addresses in ROM
Byte transfer on $D_0 - D_7$	1	0	1	Only even address in ROM
Byte transfer on $D_8 - D_{15}$	1	1	0	Only odd address in ROM


4
5
6
4
ODD
EVEN
$\overline{CS_1}$
$\overline{CS_2}$
ROM
4K × 8
A_0
A_{11}
A_1–A_{12}
$\overline{MRD}$
$\overline{OE}$
D_{15}–D_8
D_7–D_0
$\overline{BHE}$
A_0
A_1
A_2
3 : 8
DECODER
74138
O_0
O_1
O_2
O_3
O_4
O_5
O_6
O_7
0
1
2
3
4
5
6
7
$\overline{E_1}$
$\overline{E_2}$
E_3
$\overline{CS_3}$
$\overline{CS_4}$
RAM
4K × 8
$\overline{RD}$
$\overline{WR}$
$\overline{MWR}$
7430
2
0
1
0
$\overline{CS_3}$
$\overline{CS_4}$

In the maximum mode, the 8086 is prepared to operate in multiprocessor mode. All the control signals in this mode are derived by a chip called bus controller (8288). While interfacing memory or I/O devices in maximum mode, the control signals derived by the bus controller (8288) are used. Instead of $\overline{RD}$, for memory operations $\overline{MRDC}$ (Memory Read Command) output of 8288 is used and instead of $\overline{WR}$, $\overline{AMWC}$ (advanced memory write command) output of 8288 is used. The rest of the procedure in minimum and maximum mode is similar while interfacing the memories as well as peripherals.

Along with these $\overline{MRDC}$, $\overline{IOWC}$, $\overline{AIOWC}$, $\overline{AMWC}$ two more signals, namely $\overline{IOWC}$ and $\overline{MWTC}$, are provided by 8288. These signals are similar to $\overline{AIOWC}$ and $\overline{AMWC}$ in significance and operation but there activation is delayed by a clock cycle as compared to $\overline{AIOWC}$ and $\overline{MWTC}$.

DYNAMIC RAM INTERFACING

Whenever a large memory is required in a microcomputer system, the memory subsystem is generally designed using dynamic RAM as it has various advantages e.g. higher packaging density, lower cost and less power consumption.

A typical static RAM cell may require six transistors while the dynamic RAM cell requires only a transistor along with a capacitor. Hence it is possible to obtain higher packaging density and hence low cost units are available. On the other side, there are some serious drawbacks of dynamic RAMs. The basic dynamic RAM cell uses a capacitor to store the charge as a representation of data. This capacitor is manufactured as a diode that is reverse-biased so that the storage capacitance comes into the picture. This storage capacitance is utilized for storing the charge representation of data but the reverse-biased diode has a leakage current that tends to discharge the capacitor giving rise to the possibility of data loss. To avoid this possible data loss, the data stored in a dynamic RAM cell must be refreshed after a fixed time interval regularly. The process of refreshing the data in the RAM is known as *refresh cycle*. This activity is similar to reading the data from each cell of the memory, independent of the requirement of microprocessor, regularly. During this refresh period all other operations (accesses) related to the memory subsystem are suspended. Hence the refresh activity causes loss of time and results in reduced system performance.

keeping in view the advantages of
dynamic RAM, like low power consumption, higher packaging density and low cost, most of the advanced computer systems are designed using dynamic RAMs, of course, at the cost of operating speed. Also, the refresh mechanism and the additional hardware required makes the interfacing hardware, in case of dynamic RAM, more complicated, as compared to static RAM interfacing circuit. A dedicated hardware chip called as dynamic RAM controller is the most important part of the interfacing circuit.

The refresh cycle is different from the memory read cycle in the following aspects.

1. The memory address is not provided by the CPU address bus, rather, it is generated by a refresh mechanism counter known as refresh counter.
2. Unlike memory read cycle, more than one memory chip may be enabled at a time so as to reduce the number of total memory refresh cycles.
3. The data enable control of the selected memory chip is deactivated, and data is not allowed to appear on the system data bus during refresh, as more than one memory units are refreshed simultaneously. This is to avoid the data from the different chips to appear on the bus simultaneously.
4. Memory read is either a processor initiated or an external bus master initiated operation while memory refresh is an independent regular activity, initiated and carried out by the refresh mechanism.

Generally, dynamic RAM is available in units of several kilobits to even megabits of memory (it is not in terms of bytes or nibbles as in a static RAM). This memory is arranged internally in a two dimensional matrix array so that it will have n rows and m columns. The row address n and column address m are important for the refreshing operation. For example, a typical 4K bit dynamic RAM chip has an internally arranged bit array of dimension 64×64, i.e. 64 rows and 64 columns. Thus the row address and column address will require 6 bits each. These 6 bits for each row address and column address will be generated by the refresh counter, during the refresh cycles. A complete row of 64 cells is refreshed at a time to minimize the refreshing time. Thus the refresh counter needs to generate only row addresses. The row addresses are multiplexed, over lower order address lines. The refresh signals act to control the multiplexer, i.e. when refresh cycle is in process the refresh counter puts the row address over the address bus for refreshing.

Otherwise, the address bus of the processor is connected to the address bus of DRAM, during normal processor initiated activities. A timer, called *refresh timer*, derives a pulse for refreshing action after each refresh interval, which can be qualitatively defined as the time for which a dynamic RAM cell can hold data charge level practically constant, i.e. no data loss takes place. Suppose the typical dynamic RAM chip has 64 rows, then each row should be refreshed after each refresh interval, or in other words, all the 64 rows are to be refreshed in a single refresh interval. This refresh interval depends upon the manufacturing technology of the dynamic RAM cell. It may range anywhere from 1 ms to 3 ms. Let us consider 2 ms as a typical refresh time interval. Hence, the frequency of the refresh pulses will be calculated as shown:

$$\text{Refresh Time (per row) } t_r = \frac{2\times10^{-3}}{64}$$

$$\text{Refresh Frequency } f_r = \frac{64}{2\times10^{-3}} = 32\times10^{3} \text{ Hz}$$

The block diagram in Fig. explains the refreshing logic and 8086 interfacing with dynamic RAM. Each of the used chips is a16K × 1-bit dynamic RAM cell array. The system contains two 16K byte dynamic RAM units. All the address and data lines are assumed to be available from an 8086 microcomputer system. The $\overline{OE}$ pin controls output data buffers of the memory chips. The CE pins are active high chip selects of memory chips. The refresh cycle starts, if the refresh output of the refresh timer goes high, $\overline{OE}$ and CE also tend to go high. The high CE enables the memory chip for refreshing, while high $\overline{OE}$ prevents the data from appearing on the data bus, as said in the description of the memory refresh cycle. The 16K × 1-bit dynamic RAM has an internal array of 128 × 128 cells, requiring 7 bits for row addresses. The lower order seven lines A_0–A_6 are multiplexed with the refresh counter output $A_0 - A_6$. The logic is shown in Fig.

The pin assignments for 2164 dynamic RAM is shown in Fig. The $\overline{RAS}$ and $\overline{CAS}$ are row and column address strobes and are driven by the dynamic RAM controller outputs. $A_0 - A_7$ lines are the row or column address lines, driven by $OUT_0 - OUT_7$ outputs of the controller. The $\overline{WE}$ pin indicates memory write cycles. The D_{IN} and D_{OUT} pins are data pins for write and read operations, respectively. In practical circuits, the refreshing logic is integrated inside dynamic RAM controller chips like 8203, 8202 and 8207, etc.

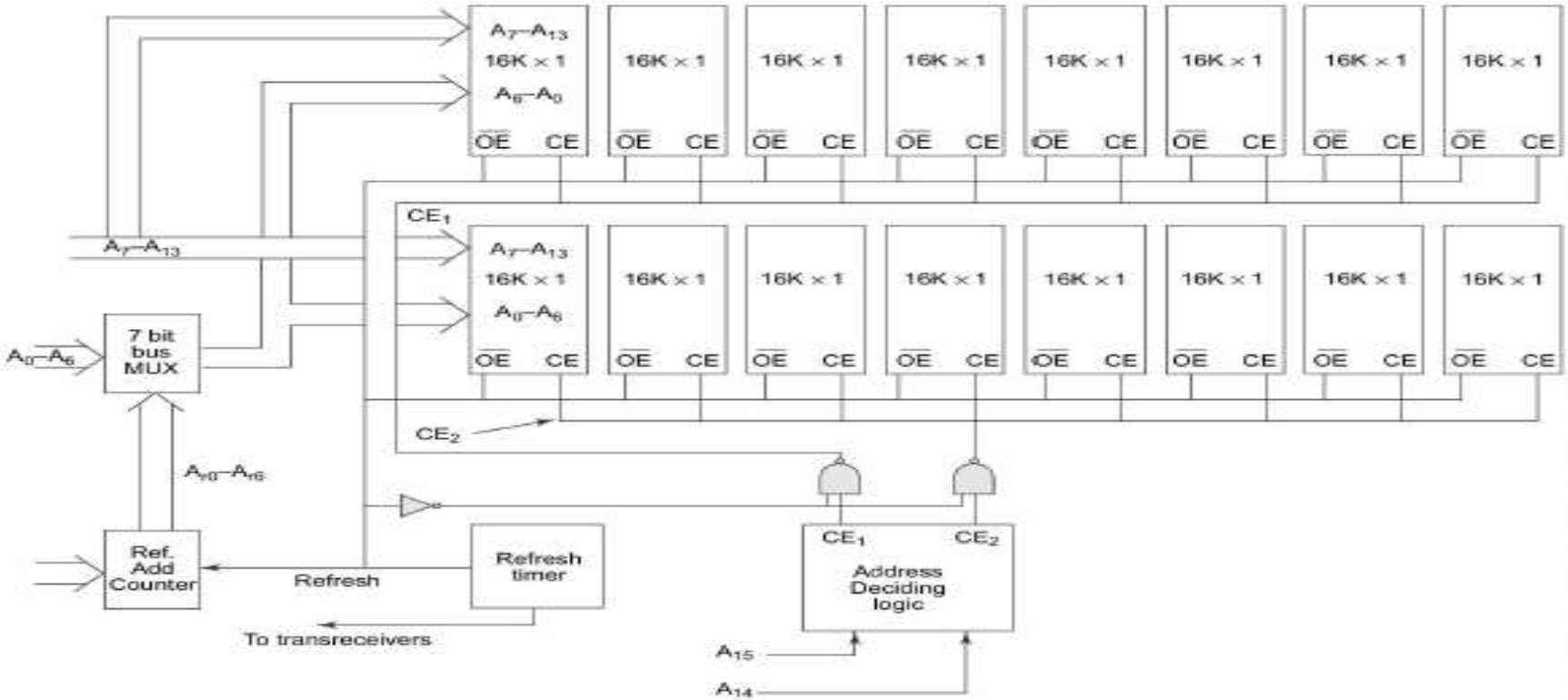
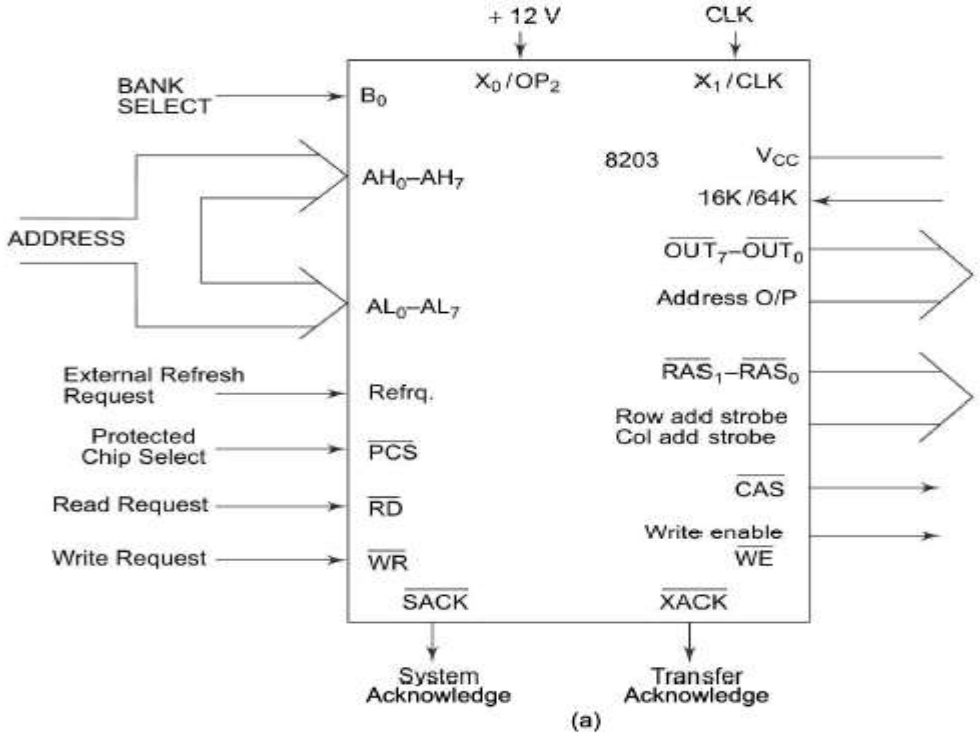


Fig. ***Dynamic RAM Refreshing Logic***

Intel's 8203 is a dynamic RAM controller that supports 16K or 64K dynamic RAM chips. This selection is done using pin 16K/64K. If it is high, the 8203 is configured to control 16K dynamic RAM, else it controls 64K dynamic RAM. The address inputs of 8203 controller accept address lines A_1 to A_{16} on lines AL_0–AL_7 and AH_0–AH_7. The B_0 line is used to select the even or odd bank. The $\overline{RD}$ and $\overline{WR}$ signals decode whether the cycle is a memory read or memory write cycle and are accepted as inputs to 8203 from the microprocessor. The $\overline{WE}$ signal specifies the memory write cycle and is an output from 8203 that drives the $\overline{WE}$ input of dynamic RAM memory chip. The $\overline{OUT_0}$ – $\overline{OUT_7}$ set of eight pins is an 8-bit output bus that carries multiplexed row and column addresses of a bit location in a dynamic RAM chip. The $\overline{CAS}$ signal strobes the column address on $\overline{OUT_0}$ – $\overline{OUT_7}$. The $\overline{RAS_1}$ – $\overline{RAS_0}$ pins strobes row address on $\overline{OUT_0}$ – $\overline{OUT_7}$ for at the most two banks of 2164 dynamic RAM chips. Actually the row and column addresses are derived from the address lines A_1 – A_{16} accepted by the controller on its inputs AL_0–AL_7 and AH_0–AH_7. An external crystal may be applied between X_0 and X_1 pins, otherwise, with the OP_2 pin at +12V, a clock signal may be applied at the pin CLK. The $\overline{PCS}$ pin accepts the chip select signal derived by an address decoder.



(a)

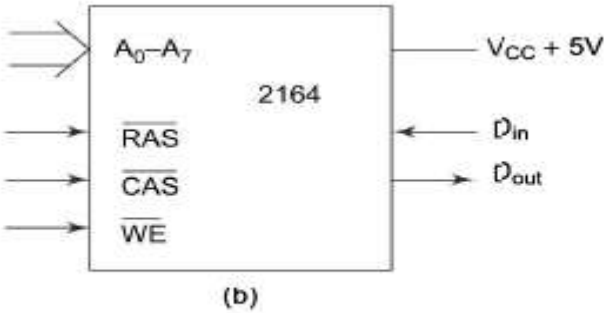


Fig. *(a) Dynamic RAM controller (b) 1-bit Dynamic RAM*

The REFREQ pin is used whenever the memory refresh cycle is to be initiated by an external signal. The $\overline{\text{XACK}}$ signal indicates that data is available during a read cycle or it has been written if it is a write cycle. It can be used as a strobe for data latches or as a ready signal to the processor. The $\overline{\text{SACK}}$ output signal marks the beginning of a memory access cycle. If a memory request is made during a memory refresh cycle, the $\overline{\text{SACK}}$ signal is delayed till the starting of memory read or write cycle. the 8203 can be used to control a 256K bytes memory subsystem for a maximum mode 8086 microprocessor system. This design assumes that data and address busses are inverted and latched, hence the inverting buffers and inverting latches are used (8283-inverting buffer and 8287-inverting latch).

We can interface a 512K byte dynamic RAM with 8086 using an advanced dynamic RAM controller 8208. Most of the functions of 8208 and 8203 are similar but 8208 can be used to refresh the dynamic RAM using DMA approach. The memory system is divided into even and odd banks of 256Kbyte each, as required for an 8086 system. The inverted $\overline{\text{AACK}}$ output of 8208 latches the A_0 and $\overline{\text{BHE}}$ signals required for selecting the banks. If the latched bank select signal and the $\overline{\text{WE}}$/PCLK output of 8208 both go low it indicates a write operation to the respective bank.

AACK –introduce wait states, PCLK-control signal

The Keyboard display controller-8279

We can use 8255 in interfacing keyboards and displays with 8086. The disadvantage of this method is that the processor has to refresh the display and check the status of the keyboard periodically using polling technique. Thus a considerable amount of CPU time is wasted, reducing the system operating speed and hence the throughput.

Intel's 8279 is a general purpose keyboard display controller that simultaneously drives the display of a system and interfaces a keyboard with the CPU, leaving it free for its routine task. The keyboard-display interface scans the keyboard to identify if any key has been pressed and sends the code of the pressed key to the CPU. It also transmits the data received from the CPU, to the display device. Both of these functions are performed by the controller in repetitive fashion without involving the CPU. The keyboard is interfaced either in the interrupt or the polled mode. In the interrupt mode, the processor is requested service only if any key is pressed, otherwise the CPU can proceed with its main task. In the polled mode, the CPU periodically reads an internal flag of 8279 to check for a key pressure. The keyboard section can interface an array of a maximum of 64 keys with the CPU. The keyboard entries (key codes) are debounced and stored in an 8-byte FIFO RAM, that is further accessed by the CPU to read the key codes. If more than eight characters are entered in the FIFO (i.e. more than eight keys are pressed), before any FIFO read operation, the overrun status is set. If a FIFO contains a valid key entry, the CPU is interrupted (in interrupt mode) or the CPU checks the status (in polling) to read the entry. Once the CPU reads a key entry, the FIFO is updated, i.e. the key entry is pushed out of the FIFO to generate space for new entries. The 8279 normally provides a maximum of sixteen 7-seg display interface with CPU. It contains a 16-byte display RAM that can be used either as an integrated block of 16 × 8-bits or two 16 × 4-bit blocks of RAM. The data entry to RAM block is controlled by CPU using the command words of the 8279.

Architecture and Signal Descriptions of 8279

The keyboard display controller chip 8279 provides (a) a set of four scan lines and eight return lines for interfacing keyboards (b) a set of eight output lines for interfacing display. Figure shows the functional block diagram of 8279 followed by its brief description.

I/O Control and Data Buffers The I/O control section controls the flow of data to/from the 8279. The data buffers interface the external bus of the system with internal bus of 8279. The I/O section is enabled only if $\overline{D}$ is low. The pins A_0, $\overline{RD}$ and $\overline{WR}$ select the command, status or data read/write operations carried out by the CPU with 8279.

Control and Timing Register and Timing Control These registers store the keyboard and display modes and other operating conditions programmed by CPU. The registers are written with $A_0 = 1$ and $\overline{WR} = 0$. The timing and control unit controls the basic timings for the operation of the circuit. Scan counter divide down the operating frequency of 8279 to derive scan keyboard and scan display frequencies.

Scan Counter The scan counter has two modes to scan the key matrix and refresh the display. In the encoded mode, the counter provides a binary count that is to be externally decoded to provide the scan lines for keyboard and display (four externally decoded scan lines may drive up to 16 displays). In the decoded scan mode, the counter internally decodes the least significant 2 bits and provides a decoded 1 out of 4 scan on SL_0–SL_3 (four internally decoded scan lines may drive up to 4 displays). The keyboard and display both are in the same mode at a time.

Return Buffers and Keyboard Debounce and Control This section scans for a key closure row-wise. If it is detected, the keyboard debounce unit debounces the key entry (i.e. wait for 10 ms). After the debounce period, if the key continues to be detected. The code of the key is directly transferred to the sensor RAM along with SHIFT and CONTROL key status.

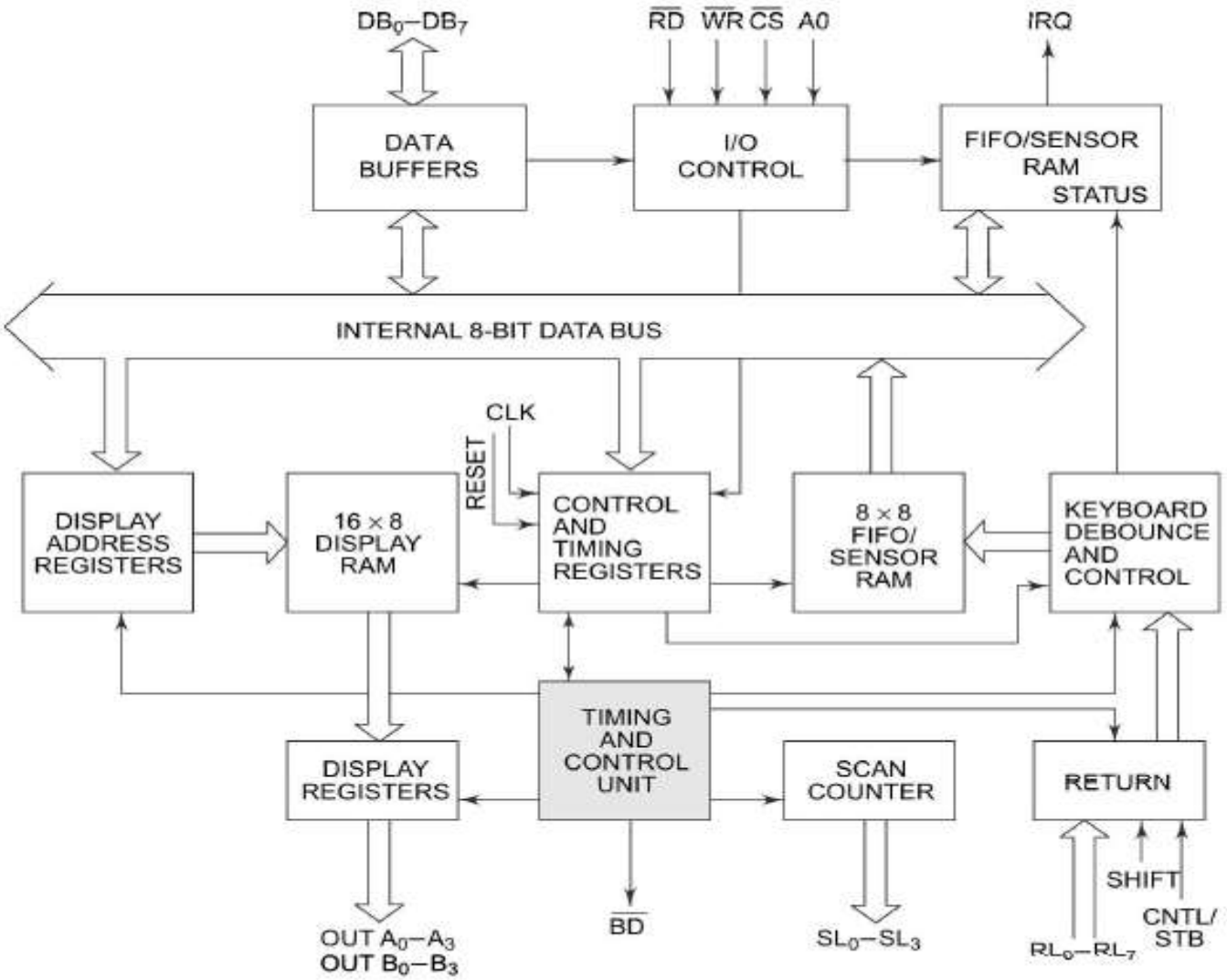
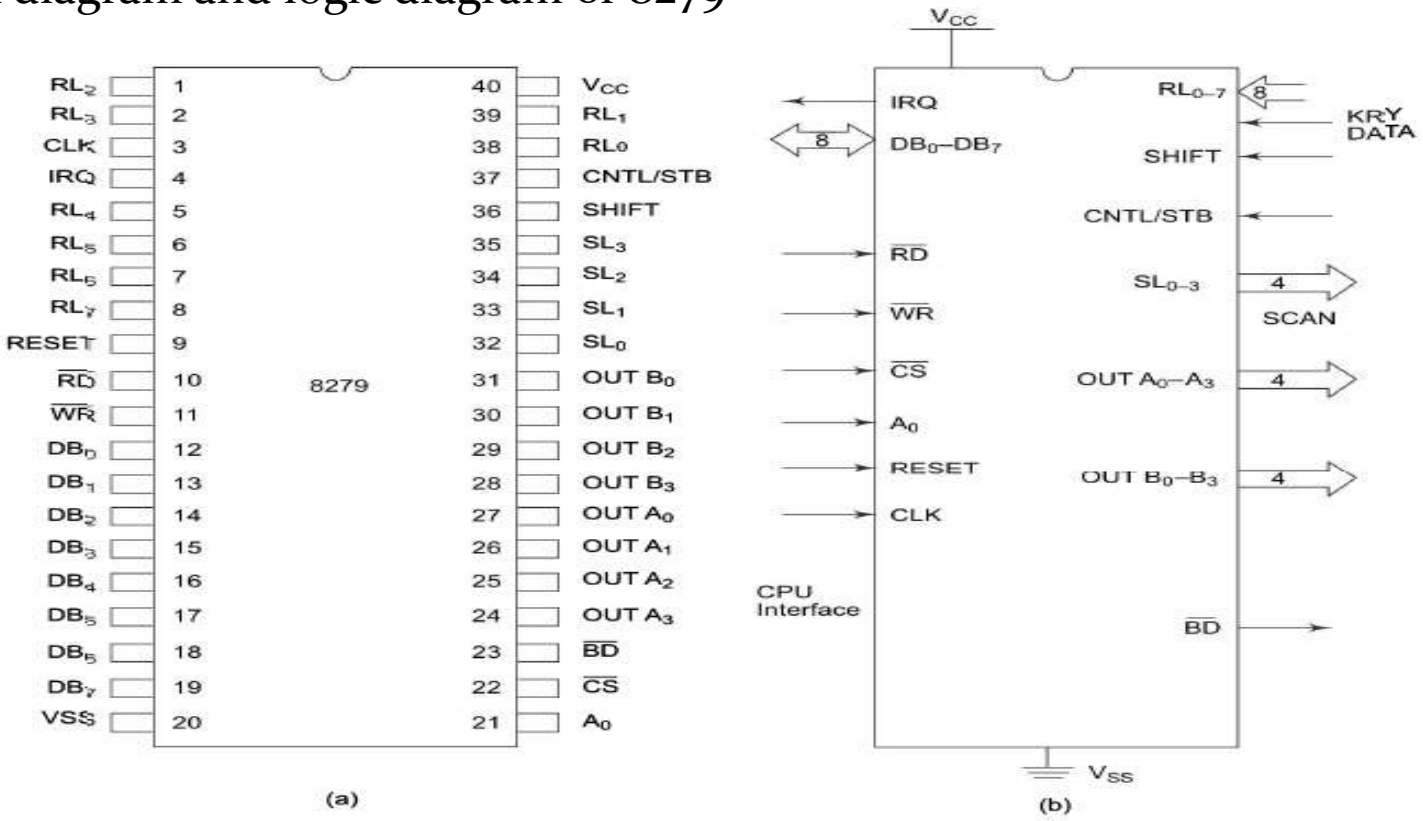


Fig. *8279 Internal Architecture*

FIFO/Sensor RAM and Status Logic In keyboard or strobed input mode, this block acts as 8-byte first-in-first-out (FIFO) RAM. Each key code of the pressed key is entered in the order of the entry, and in the meantime, read by the CPU, till the RAM becomes empty. The status logic generates an interrupt request after each FIFO read operation till the FIFO is empty. In scanned sensor matrix mode, this unit acts as sensor RAM. Each row of the sensor RAM is loaded with the status of the corresponding row of sensors in the matrix. If a sensor changes its state, the IRQ line goes high to interrupt the CPU.

Display Address Registers and Display RAM The display address registers hold the address of the word currently being written or read by the CPU to or from the display RAM. The contents of the registers are automatically updated by 8279 to accept the next data entry by CPU. The 16-byte display RAM contains the 16-bytes of data to be displayed on the sixteen 7-seg displays in the encoded scan mode.

Pin diagram and logic diagram of 8279



The signal discription of each of the pins of 8279 is presented below in brief:

DB_0–DB_7 These are bidirectional data bus lines. The data and command words to and from the CPU are transferred on these lines.

CLK This is a clock input used to generate internal timings required by 8279.

RESET This pin is used to reset 8279. A high on this line resets 8279. After resetting 8279, its in sixteen 8-bit display, left entry encoded scan, 2-key lock out mode. The clock prescaler is set to 31.

$\overline{CS}$ Chip Select: A low on this line enables 8279 for normal read or write operations. Otherwise, this pin should remain high.

A_0 A high on the A_0 line indicates the transfer of a command or status information. A low on this line indicates the transfer of data. This is used to select one of the internal registers of 8279.

$\overline{RD}$, $\overline{WR}$ (Input/Output) READ/WRITE input pins enable the data buffers to receive or send data over the data bus.

IRQ This interrupt output line goes high when there is data in the FIFO/sensor RAM. The interrupt line goes low with each FIFO RAM read operation. However, if the FIFO RAM further contains any key-code entry to be read by the CPU, this pin again goes high to generate an interrupt to the CPU.

V_{ss}, V_{cc} These are the ground and power supply lines for the circuit.

SL_0–SL_3-Scan Lines These lines are used to scan the keyboard matrix and display digits. These lines can be programmed as encoded or decoded, using the mode control register.

RL_0–RL_7-Return Lines These are the input lines which are connected to one terminal of keys, while the other terminal of the keys are connected to the decoded scan lines. These are normally high, but pulled low when a key is pressed.

SHIFT The status of the shift input line is stored along with each key code in FIFO in the scanned keyboard mode. Till it is pulled low with a key closure it is pulled up internally to keep it high.

CNTL/STB-CONTROL/STROBED I/P Mode In the keyboard mode, this line is used as a control input and stored in FIFO on a key closure. The line is a strobe line that enters the data into FIFO RAM, in the strobed input mode. It has an internal pull up. The line is pulled down with a key closure.

$\overline{BD}$ -Blank Display This output pin is used to blank the display during digit switching or by a blanking command.

$OUTA_0$–$OUTA_3$ and $OUTB_0$–$OUTB_3$ These are the output ports for two 16×4 (or one 16×8) internal display refresh registers. The data from these lines is synchronized with the scan lines to scan the display and keyboard. The two 4-bit ports may also be used as one 8-bit port.

Modes of Operation of 8279

The modes of operation of 8279

(i) Input (Keyboard) modes
(ii) Output (Display) modes

Input (Keyboard) Modes 8279 provides three input modes

1. Scanned Keyboard Mode This mode allows a key matrix to be interfaced using either encoded or decoded scans. In the encoded scan, an 8 × 8 keyboard or in decoded scan, a 4 × 8 keyboard can be interfaced. The code of key pressed with SHIFT and CONTROL status is stored into the FIFO RAM.

2. Scanned Sensor Matrix In this mode, a sensor array can be interfaced with 8279 using either encoded or decoded scans. With encoded scan 8 × 8 sensor matrix or with decoded scan 4 × 8 sensor matrix can be interfaced. The sensor codes are stored in the CPU addressable sensor RAM.

3. Strobed input In this mode, if the control line goes low, the data on return lines, is stored in the FIFO byte by byte.

Output (Display) Modes 8279 provides two output modes for selecting the display options.

1. Display Scan In this mode, 8279 provides 8 or 16 character multiplexed displays those can be organized as dual 4-bit or single 8-bit display units.

2. Display Entry (right entry or left entry mode) 8279 allows options for data entry on the displays. The display data is entered for display either from the right side or from the left side.

All these modes may be selected by programming the 8279 suitably.

Details of Modes of Operation

Keyboard Modes

(i) Scanned Keyboard Mode with 2 Key Lockout In this mode of operation, when a key is pressed, a debounce logic comes into operation. During the next two scans, other keys are checked for closure and if no other key is pressed the first pressed key is identified. The key code of the identified key is entered into the FIFO with SHIFT and CNTL status, provided the FIFO is not full, i.e. it has at least one byte free. If the FIFO does not have any free byte, naturally the key data will not be entered and the error flag is set. If FIFO has at least one byte free, the above code is entered into it and the 8279 generates an interrupt (on IRQ line) to the CPU to inform about the previous key closures. If another key is found closed during the subsequent two scans, no entry to FIFO is made. If all the other keys are released before the first key, the keycode is entered into FIFO. If the first pressed key is released before the others, the first will be ignored. A keycode is entered to FIFO only once for each valid depression, independent of other keys pressed along with it, or released before it. If two keys are pressed within a debounce cycle (simultaneously), no key is recognized till one of them remains closed, and the other is released. The last key, that remains depressed is considered as single valid key depression.

(ii) Scanned Keyboard with N-Key Rollover In this mode, each key depression is treated independently. When a key is pressed, the debounce circuit waits for 2 keyboard scans and then checks whether the key is still depressed. If it is still depressed, the code is entered in FIFO RAM. Any number of keys can be pressed simultaneously and recognized in the order, the keyboard scan recorded them. All the codes of such keys are entered into FIFO. Note that, in this mode, the first pressed key need not be released before the second is pressed. All the keys are sensed in the order of their depression, rather in the order the keyboard scan senses them, and independent of the order of their release.

(iii) Scanned Keyboard Special Error Mode This mode is valid only under the N-Key rollover mode. This mode is programmed using *end interrupt/error mode set* command. If during a single debounce period (two keyboard scans) two keys are found pressed, this is considered a simultaneous depression and an error flag is set. This flag, if set, prevents further writing in FIFO but allows generation of further interrupts to the CPU for FIFO read. The error flag can be read by reading the FIFO status word. The error flag is set by sending normal clear command with CF = 1.

(iv) Sensor Matrix Mode In the Sensor Matrix mode, the debounce logic is inhibited. The 8-byte FIFO RAM now acts as 8 × 8-bit memory matrix. The status of the sensor switch matrix is fed directly to sensor RAM matrix. Thus the sensor RAM bits contains the row-wise and column-wise status of the sensors in the sensor matrix. The IRQ line goes high, if any change in sensor value is detected at the end of a sensor matrix scan or the sensor RAM has a previous entry to be read by the CPU. The IRQ line is reset by the first data read operation, if AI = 0, otherwise, by issuing the end interrupt command. AI is a bit in read sensor RAM word.

Display Modes There are various options of data display. For example, the command number of characters can be 8 or 16, with each character organised as single 8-bit or dual 4-bit codes. Similarly there are two display formats. The first one is known as left entry mode or type writer mode, since in a type writer the first character typed appears at the left-most position, while the subsequent characters appear successively to the right of the first one. The other display format is known as right entry mode, or calculator mode, since in a calculator the first character entered appears at the rightmost position and this character is shifted one position left when the next character is entered. Thus all the previously entered characters are shifted left by one position when a new character is entered.

(i) Left Entry Mode In the left entry mode, the data is entered from the left side of the display unit. Address 0 of the display RAM contains the leftmost display character and address 15 of the RAM contains the right most display character. It is just like writing in our note books, i.e. from left to write. If the 8279 is in autoincrement mode, the display RAM address is automatically updated with successive reads or writes. The first entry is displayed on the leftmost display and the sixteenth entry on the rightmost display. The seventeenth entry is again displayed at the leftmost display position.

(ii) Right Entry Mode In the right entry mode, the first entry to be displayed is entered on the rightmost display. The next entry is also placed in the right most display but after the previous display is shifted left by one display position. The leftmost character is shifted out of that display at the seventeenth entry and is lost, i.e. it is pushed out of the display RAM.

Command Words of 8279

All the command words or status words are written or read with $A_0 = 1$ and $\overline{CS} = 0$ to or from 8279.

(a) Keyboard Display Mode Set The format of the command word to select different modes of operation of 8279 is given below with its bit definitions.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
0	0	0	D	D	K	K	K	1

D	*D*	*Display modes*
0	0	Eight 8-bit character Left entry
0	1	Sixteen 8-bit character Left entry (Default after reset)
1	0	Eight 8-bit character Right entry
1	1	Sixteen 8-bit character Right entry

K	*K*	*K*	*Keyboard modes*
0	0	0	Encoded scan, 2 key lockout (Default after reset)
0	0	1	Decoded scan, 2 key lockout
0	1	0	Encoded scan N-Key roll over
0	1	1	Decoded scan N-Key roll over
1	0	0	Encoded scan sensor matrix
1	0	1	Decoded scan sensor matrix
1	1	0	Strobed Input Encoded scan
1	1	1	Strobed Input Decoded scan

(b) Programmable Clock The clock for operation of 8279 is obtained by dividing the external clock input signal by a programmable constant called prescaler.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
0	0	1	P	P	P	P	P	1

PPPPP is a 5-bit binary constant. The input frequency is divided by a decimal constant ranging from 2 to 31, decided by the bits of an internal prescaler, PPPPP.

(c) Read FIFO/Sensor RAM The format of this command is given as shown below:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
0	1	0	AI	X	A	A	A	1

X — don't care
AI — Auto Increment flag
AAA — Address pointer to 8 bit FIFO RAM

This word is written to set up 8279 for reading FIFO/sensor RAM. In scanned keyboard mode, AI and AAA bits are of no use. The 8279 will automatically drive data bus for each subsequent read, in the same sequence, in which the data was entered. In sensor matrix mode, the bits AAA select one of the 8 rows of RAM. If AI flag is set, each successive read will be from the subsequent RAM location.

(d) Read Display RAM This command enables a programmer to read the display RAM data.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
0	1	1	AI	A	A	A	A	1

The CPU writes this command word to 8279 to prepare it for display RAM read operation. AI is auto increment flag and AAAA, the 4-bit address, points to the 16-byte display RAM that is to be read. If AI = 1, the address will be automatically, incremented after each read or write to the display RAM. The same address counter is used for reading and writing.

(e) Write Display RAM

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
1	0	0	AI	A	A	A	A	1

AI — Auto Increment flag
AAAA — 4-bit address for 16-bit display RAM to be written
Other details of this command are similar to the 'Read Display RAM Command'.

(f) Display Write Inhibit/Blanking

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
1	0	1	X	IW	IW	BL	BL	1
Output nibbles		→		A	B	A	B	

The IW (Inhibit Write flag) bits are used to mask the individual nibble as shown in the above command. The output lines are divided into two nibbles ($OUTA_0$–$OUTA_3$ and $OUTB_0$–$OUTB_3$), those can be masked by setting the corresponding IW bit to 1. Once a nibble is masked by setting the corresponding IW bit to 1, the entry to display RAM does not affect the nibble even though it may change the unmasked nibble. The blank display bit flags (BL) are used for blanking A and B nibbles. Here D_0 and D_2 corresponds to $OUTB_0$–$OUTB_3$ while D_1 and D_3 corresponds to $OUTA_0$–$OUTA_3$ for blanking and masking respectively.

If the user wants to clear the display, blank (BL) bits are available for each nibble as shown in the format. Both BL bits will have to be cleared for blanking both the nibbles.

(g) Clear Display RAM

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
1	1	0	CD_2	CD_1	CD_0	CF	CA	1

The CD_2, CD_1, CD_0 is a selectable blanking code to clear all the rows of the display RAM as given below. The characters A and B represent the output nibbles.

CD_2	CD_1	CD_0	
1	0	x	All zeros (x don't care) AB = 00
1	1	0	A_3–A_0 = 2 (0010) and B_3–B_0 = 00 (0000)
1	1	1	All ones (AB = FF), i.e. clear RAM

CD_2 must be 1 for enabling the clear display command. If $CD_2 = 0$, the clear display command is invoked by setting CA = 1 and maintaining CD_1, CD_0 bits exactly same as above. If CF = 1, FIFO status is cleared and IRQ line is pulled down. Also the sensor RAM pointer is set to row 0. If CA = 1, this combines the effect of CD and CF bits. Here, CA represents Clear All and CF represents Clear FIFO RAM.

End Interrupt/Error Mode Set

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
1	1	1	E	x	x	x	x	1

x—do not care

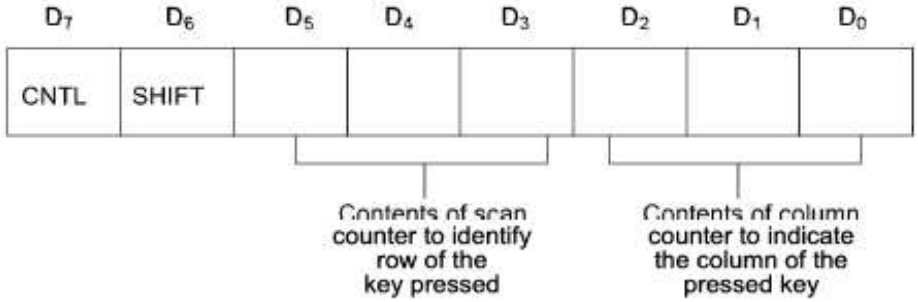
For the sensor matrix mode, this command lowers the IRQ line and enables further writing into the RAM. Otherwise, if a change in sensor value is detected, IRQ goes high that inhibits writing in the sensor RAM.

For N-key roll over mode, if the E bit is programmed to be '1', the 8279 operates in Special Error mode.

Key-code and Status Data Formats

This briefly describes the formats of the key-code/sensor data in their respective modes of operation and the FIFO Status Word formats of 8279.

Key-code Data Formats After a valid key closure, the key code is entered as a byte code into the FIFO RAM, in the following format, in scanned keyboard mode. The data format of the keycode in scanned



keyboard mode is given below. The keycode format contains 3-bit contents of the internal row counter, 3-bit contents of the column counter and status of the SHIFT and CNTL keys.

In the sensor matrix mode, the data from the return lines is directly entered into an appropriate row of sensor RAM, that identifies the row of the sensor that changed its status. The SHIFT and CNTL keys are ignored in this mode. RL bits represent the return lines. Rn represents the sensor RAM row number that is equal to the row number of the sensor array in which the status change was detected.

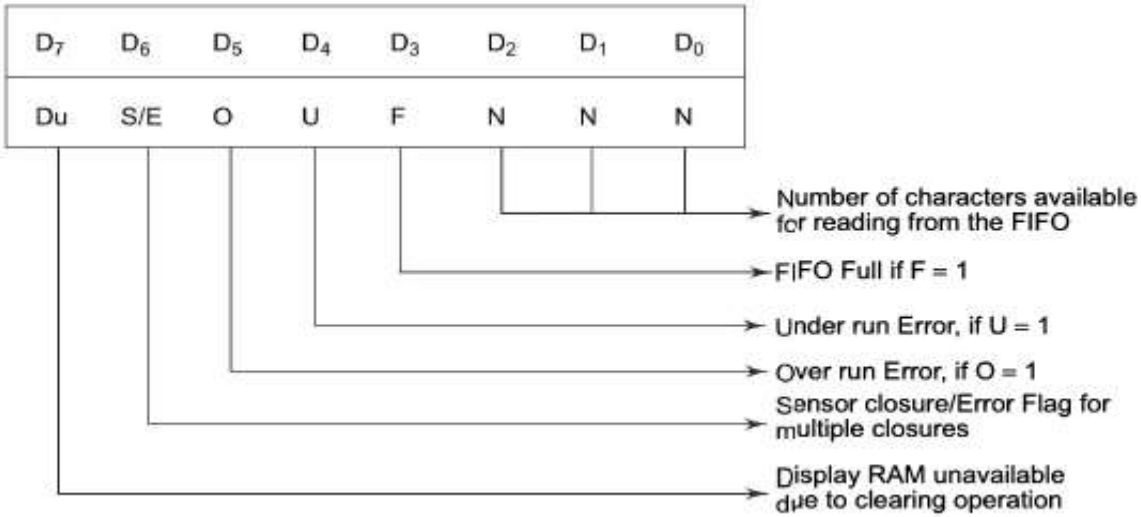
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Rn	RL_7	RL_6	RL_5	RL_4	RL_3	RL_2	RL_1	RL_0

Data Format of the Sensor Code in sensor matrix mode is given above.

In strobed input mode, data is entered to the FIFO RAM at the rising edges of CNTL/STB line, in the same format as in the sensor matrix mode.

FIFO Status Word The FIFO status word is used in keyboard and strobed input mode to indicate the error. Overrun error occurs, when an already full FIFO is attempted an entry. Underrun error occurs when an empty FIFO read is attempted. FIFO status word also has a bit to show the unavailability of FIFO RAM because of the ongoing clearing operation. In sensor matrix mode, a bit is reserved to show that at least one sensor closure indication is stored in the RAM. The S/E bit shows the simultaneous multiple closure error in special error mode.

The status word contains FIFO status, error and display unavailable signals. This is read, when $A_0 = 1$, $\overline{RD} = 0$ and $\overline{CS} = 0$. Data is read when A_0, $\overline{CS}$, $\overline{RD}$ are low. The source of data is specified by the read FIFO or read display command. The address of RAM being read is automatically incremented, if AI = 1. FIFO read always increments the address independent of AI. Data is written to the display RAM, with A_0, $\overline{WR}$ and $\overline{CS}$ tied low. The address is specified by the previous read display or write display command. The address is auto-incremented for subsequent write operations, if AI is set to. The FIFO status word format is as shown below:



INTERFACING I/O PORTS

I/O ports or Input/Output ports are the devices through which the microprocessor communicates with other devices or external data sources/destinations. Input activity, as one may expect, is the activity that enables the microprocessor to read data from external devices , for example keyboards, joysticks, mouse, etc. These devices are known as input devices as they feed data into a microprocessor system.

Output activity transfers data from the microprocessor to the external devices, for example CRT display, 7-segment displays, printers, etc. The devices which accept the data from a microprocessor system are called output devices. Thus for a microprocessor the input activity is similar to read operation, while the output activity is similar to write operation. Note that an input device can only be read and an output device can only be written.

Hence $\overline{\text{IORD}}$ operation is related with reading data from an input device and not an output device and $\overline{\text{IOWR}}$ operation means writing data to an output device and not an input device. The control word and status word may be written and read respectively in both, input or output, devices, in case of programmable devices.

After executing an OUT operation, the data appears on the data bus and simultaneously a device select signal is generated from the address and control signals. Now, if the data is to be there, at the output of the

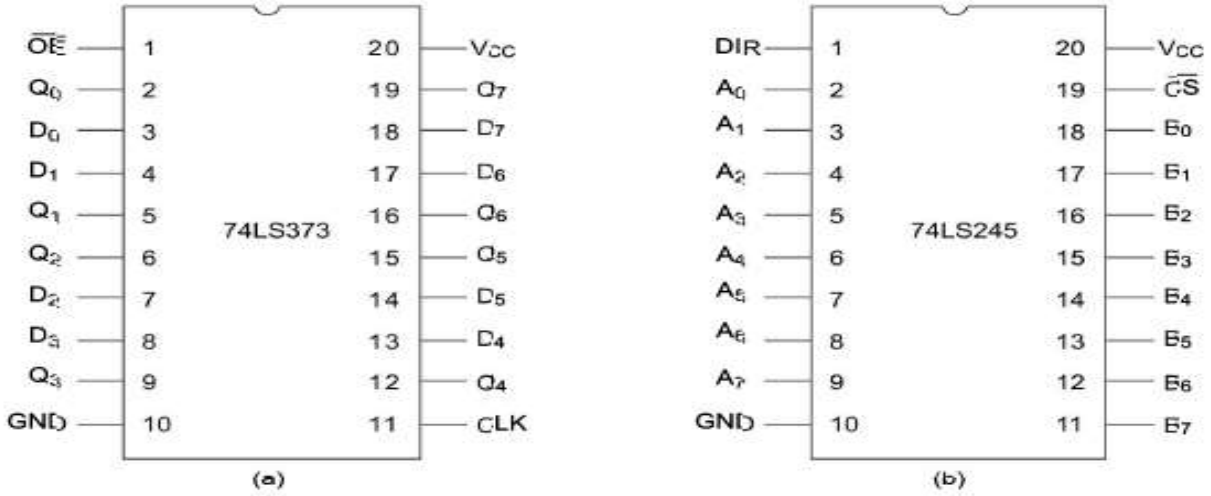


Fig. *(a) Latch (O/P port) (b) Buffer (I/P port)*

device till the next change, it must be latched. Also if the output port is to source large currents the port lines must be bufferred. Hence the latch acts as a good output port. The chip 74LS373, contains eight bufferred latches and can be used as an 8-bit output port. While reading, an input device one must take care that much current should not be sourced or sunk from the data lines to avoid loading. To overcome this problem, one may use a tristate buffer as an input device. An input port may not be a latch as it reads the status of a signal at a particular instant. The chip 74LS245 contains eight buffers and may be used as an 8-bit input port. Actually, 74LS245 is a bidirectional buffer, but while using it as an input device, only one direction is useful. This direction of data transfer in 74LS245 is selected using its DIR pin. The pin diagrams of 74LS373 and 74LS 245 are shown in Figs : (a) and (b). The $\overline{OE}$ and $\overline{CS}$ are the chip selects of 74LS373 and 74LS245 respectively. Ds and Qs are corresponding latch inputs and outputs respectively.

The CLK pin is clock input for D flip-flops. If DIR is 1, then the direction is from A(I/Ps) to B(O/Ps), otherwise the data direction is from B(I/Ps) to A(O/Ps).

Steps in Interfacing an I/O Device The following steps are performed to interface a general I/O device with a CPU:

(i) Connect the data bus of the microprocessor system with the data bus of the I/O port.
(ii) Derive a device address pulse by decoding the required address of the device and use it as the chip select of the device.
(iii) Use a suitable control signal, i.e. $\overline{IORD}$ and/or $\overline{IOWR}$ to carry out device operations, i.e. connect $\overline{IORD}$ to $\overline{RD}$ input of the device if it is an input device, otherwise connect $\overline{IOWR}$ to $\overline{WR}$ input of the device. In some cases the $\overline{RD}$ or $\overline{WR}$ control signals are combined with the device address pulse to generate the device select pulse.

Methods of Interfacing I/O Devices There are two methods of interfacing general I/O devices.

(i) I/O mapped
(ii) Memory-mapped

The principal distinction in the two approaches is that in I/O mapped interfacings the devices are viewed as distinct I/O devices and are addressed accordingly. While *in memory-mapped scheme,* the devices are viewed as memory locations and are addressed likewise. In I/O mapped interfacing, all the available address lines of a microprocessor may not be used for interfacing the devices. The processor 8086 has 20 address lines. The I/O mapped scheme may use at the most 16 address lines $A_0 - A_{15}$ or even 8 address lines for address decoding. The unused higher order address lines are logic zero, while addressing the device. An I/O mapped device requires the use of IN and OUT instructions for accessing them. The I/O mapped method requires less hardware for decoding, as less number of address lines are used. In case of 8086, a maximum of 64K input and 64K byte output devices or 32K input and 32K word output devices can be interfaced. In addition to address and data busses, to address an input device, we require the $\overline{\text{IORD}}$ signal and to address an output device, we use $\overline{\text{IOWR}}$ signal for the respective operations. The $\overline{\text{IOWR}}$ and $\overline{\text{IORD}}$ signals are used for I/O mapped interfacing.

In memory-mapped interfacing, all the available address lines are used for address decoding. Thus each memory-mapped I/O device with 8086 has a 20-bit address, i.e. 8086 can have as many as 1M memory-mapped input and as many byte output devices. Practically this is impossible, as memory-mapped I/O devices consume the addresses in the memory map of the CPU. 1M byte devices will require the complete 1Mbyte of the memory map and nothing will be left as program memory. Also the memory locations and the memory-mapped devices cannot have common addresses. The $\overline{\text{MRDC}}$ and $\overline{\text{MRTC}}$ signals are used for interfacing in memory-mapped I/O scheme. All the applicable data transfer instructions (e.g. MOV, LEA) can be used to communicate with memory-mapped I/O devices. However, I/O operations are much more sluggish

compared to the memory operations which are faster. Moreover, complex decoding hardware is required in this case since all the address lines are used for decoding.

In case of the 8086 systems, the memory-mapped method is seldom used. Hence all the peripheral devices in most of the practical systems are essentialy I/O mapped devices.

8086 has a 16-bit data bus, hence interfacing of 8-bit devices with 8086 need special consideration. Usually, 8-bit I/O devices are interfaced with lower order data bus of 8086, i.e. D_0–D_7. The 16-bit devices are interfaced directly with the 16-bit data bus, using A_0 and $\overline{BHE}$ pins of 8086. The problems explain the actual method of interfacing I/O devices with 8086. The interfacing hardware always need supporting application programs to carry out the desired operations. each interfacing example is accompanied with a supporting 8086 ALP.

Problem:

Interface an input port 74LS245 to read the status of switches SW_1 to SW_8. The switches, when shorted, input a '1' else input a '0' to the microprocessor system. Store the status in register BL. The address of the port is 0740H.

Solution The hardware interface circuit is shown in Fig. . The address, control and data lines are assumed to be readily available at the microprocessor system.

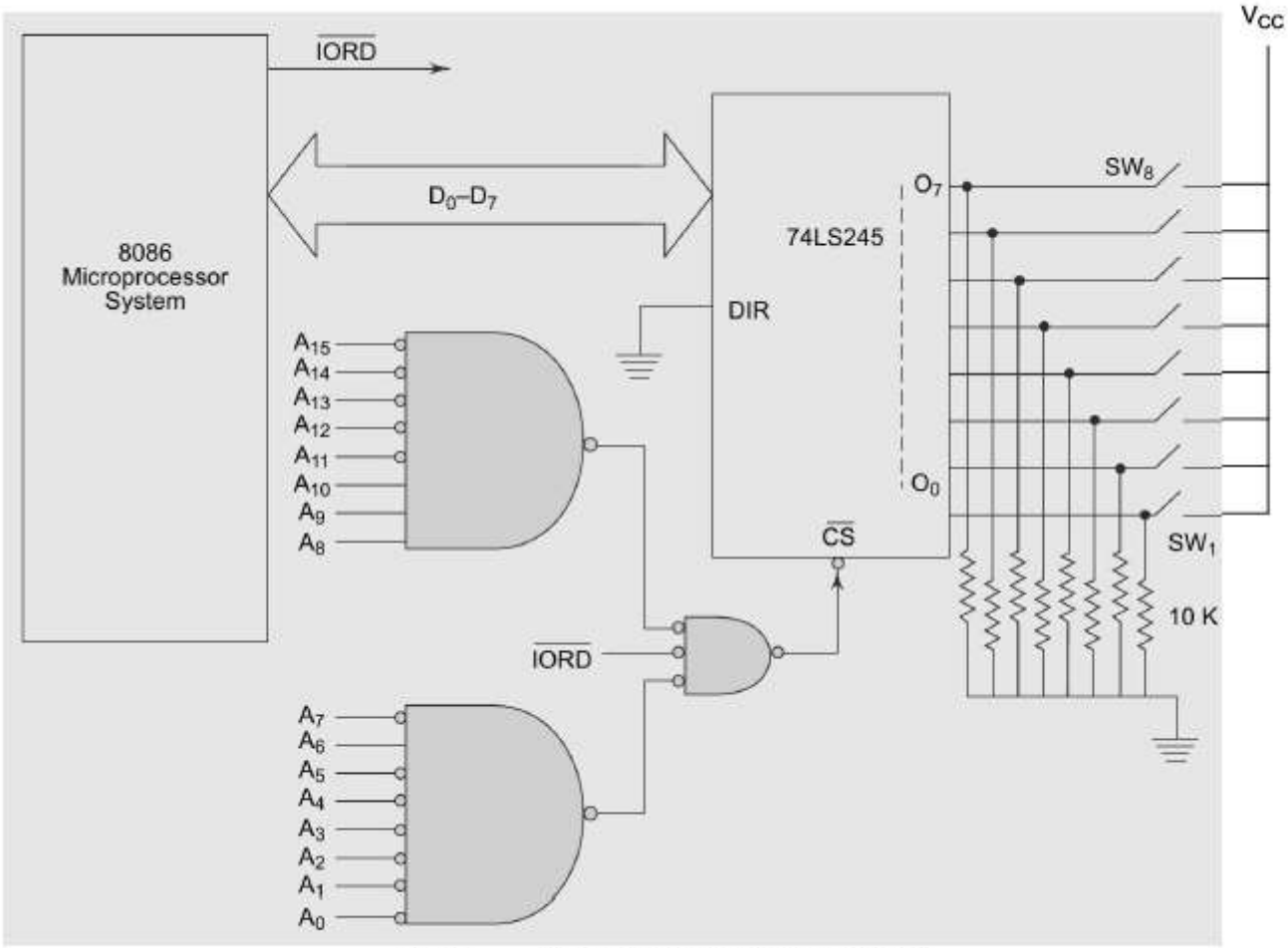


Fig. *Interfacing Input Port 74LS245*

```
THE ALP IS GIVEN AS FOLLOWS:
MOV BL, 00H       ; Clear BL for status
MOV DX, 0740H     ; 16-bit port addrenss in DX
IN AL, DX         ; Read port 0740H for switch positions
MOV BL, AL        ; Store status of switches from AL into BL
HLT               ; Stop
```

Program — *ALP for Problem*

Here LSB bit of BL corresponds to the status of SW_1 and likewise the MSB of BL corresponds to the status of SW_8. The '1' indicates 'on' or shorted switch and the '0' indicates an 'off' or opened switch. The pull-up registers in Fig. are necessary because the open switches should input a '0' to the system but the TTL port 74LS245 will read the free input as '1'. (Free TTL inputs are always read as logic '1'.)

Multiplexed 7-seg Displays To display a single digit, at least one output port is required. Suppose one wants a 5-digit display for some practical application, Five such ports will be required, i.e. the hardware will be very complex and costly. To minimize the complexity and cost of hardware one may implement an almost visually identical display using only two ports. Suppose there are four 7-seg displays, numbered 1, 2, 3 and 4. One of the two ports, say port1 selects one of these displays, say display 1 at a time, while port 2 sends the data to be displayed (i.e. a,b,c,d,e,f,g and dp) to the first display for a fixed short duration. Port 1 next, selects the display 2 and port 2 sends the appropriate display data to it. Each of the display unit remains active for a short duration and the process continues in a loop. This is repeated at a high frequency so that the complete display containing more than one7-seg display appears to be stationary due to the persistence of vision. Instead of BCD to 7-seg decoder circuit, look up table technique may be used for converting BCD numbers to equivalent 7-seg codes.

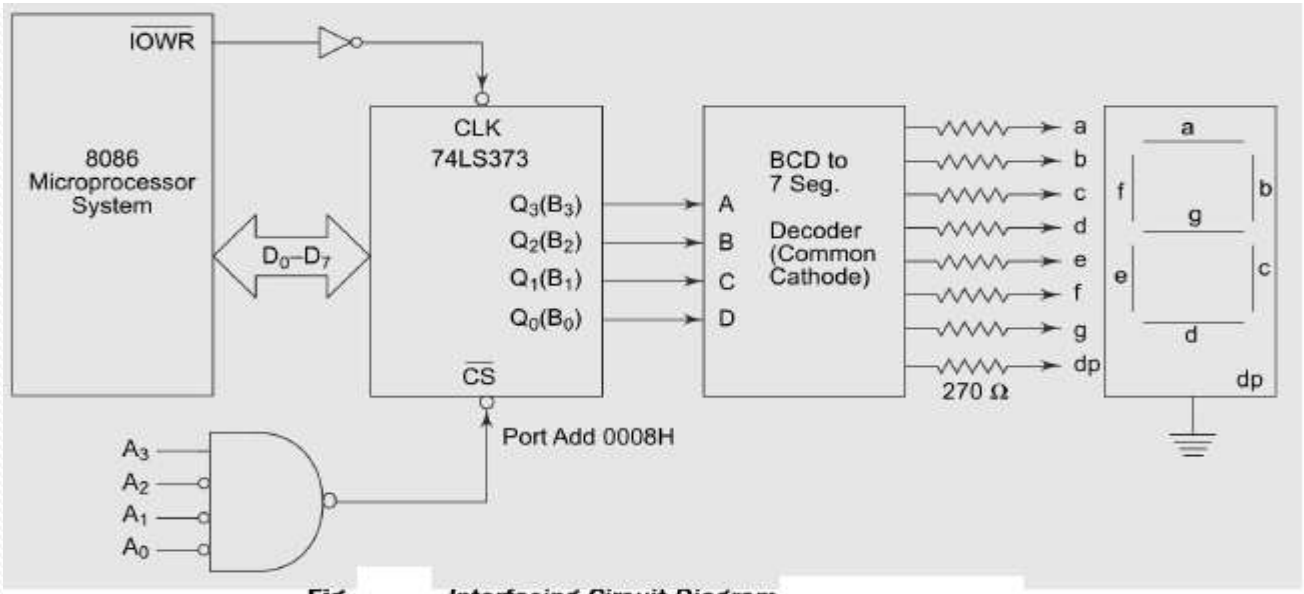


Fig. *Interfacing Circuit Diagram*

For interfacing a 16-bit output port with 8086, we may use two 8-bit output ports to form a 16-bit port, with a single address, as shown in Fig. . Both the 8-bit ports are in this case addressed as a single 16-bit port with a single address.

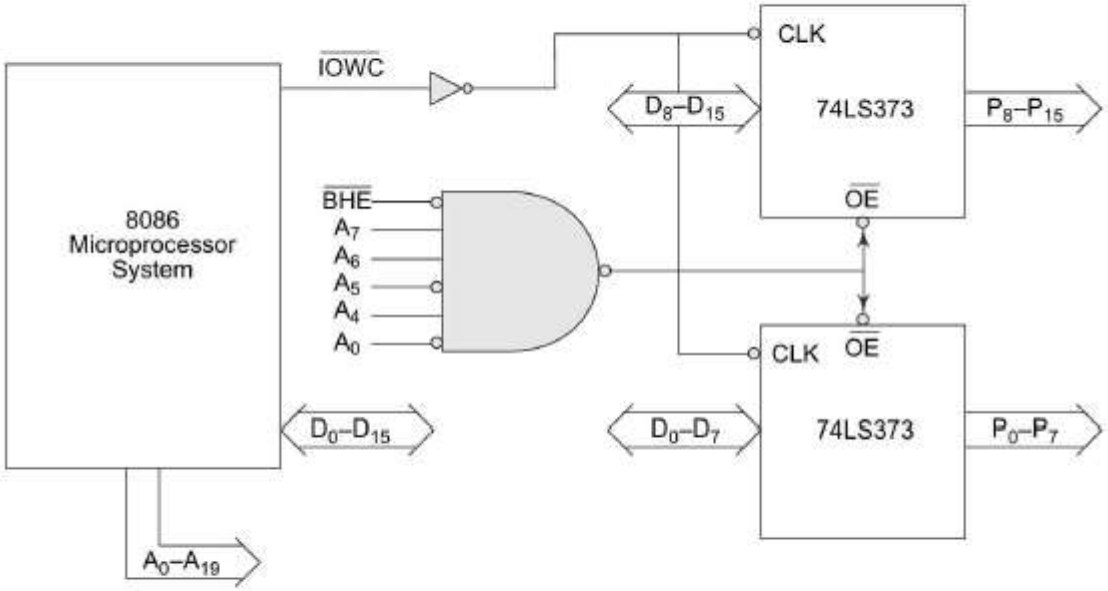


Fig. ***Interfacing a Circuit of 16 bit output port***

The OUT instruction of 8086 is able to output 16-bit data directly in a single cycle, and the programming technique is identical to that of the 8-bit port. The instruction uses 16-bit register AX as source operand as shown:

```
OUT Port_Add, AX
OUT [DX], AX
```

A 16-bit input port may also be interfaced similarly. Note the use of A0 and BHE signals in interfacing the 16-bit ports. One may find out address of the output port in Fig. :

The ports were either input or output, but in 8255 the same port may be programmed either to work as input port or as output port. Hence the chip is called Programmable Input-Output (PIO) device.

PIO 8255 [PROGRAMMABLE INPUT-OUTPUT PORT]

The parallel input-output port chip 8255 is also known as programmable peripheral input-output port. The Intel's 8255 is designed for use with Intel's 8-bit, 16-bit and higher capability microprocessors. It has 24 input/output lines which may be individually programmed in two groups of twelve lines each, or three groups of eight lines. The two groups of I/O pins are named as Group A and Group B. Each of these two groups contain a subgroup of eight I/O lines called as 8-bit port and another subgroup of four I/O lines or a 4-bit port. Thus Group A contains an 8-bit port A along with a 4-bit port, C upper. The port A lines are identified by symbols PA_0–PA_7 while the port C lines are identified as PC_4–PC_7. Similarly, Group B contains an 8-bit port B, containing lines PB_0 – PB_7 and a 4-bit port C with lower bits PC_0–PC_3. The port C upper and port C lower can be used in combination as an 8-bit port C. Both the port Cs are assigned the same address. Thus one may have either three 8-bit I/O ports or two 8-bit and two 4-bit I/O ports

from 8255. All of these ports can function independently either as input or as output ports. This can be achieved by programming the bits of an internal register of 8255 called as Control Word Register (CWR). The internal block diagram and the pin configuration of 8255 are shown in Figs (a) and (b).

The 8-bit data bus buffer is controlled by the read/write control logic. The read/write control logic manages all of the internal and external transfers of both data and control words. $\overline{RD}$, $\overline{WR}$, A_1, A_0 and RESET are the inputs, provided by the microprocessor to the READ/WRITE control logic of 8255. The 8-bit, 3-state bidirectional buffer is used to interface the 8255 internal data bus with the external system data bus. This buffer receives or transmits data upon the execution of input or output instructions by the microprocessor. The control words or status information is also transferred through the buffer.

The signal descriptions of 8255 are briefly presented as follows:

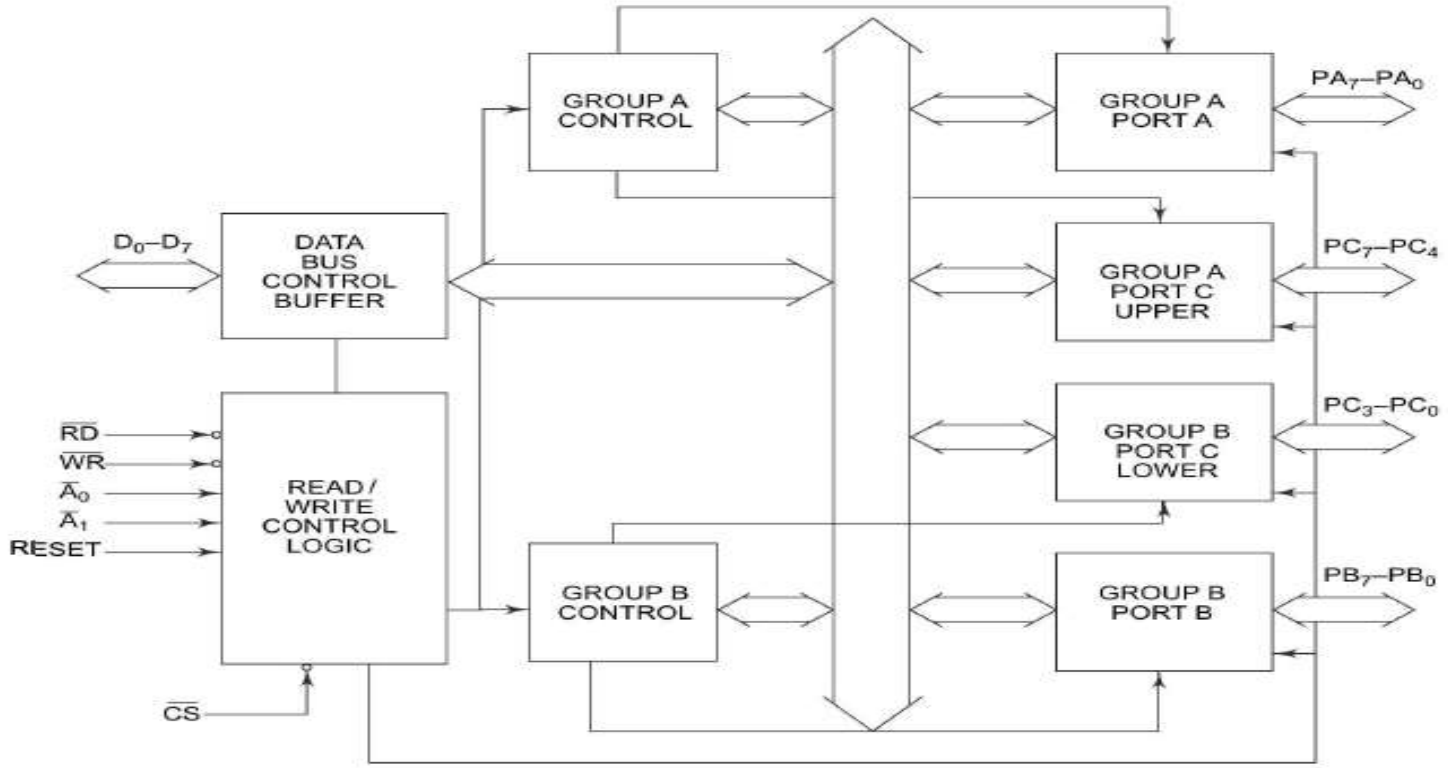


Fig. (a) ***8255 Internal Architecture***

PA_7–PA_0 These are eight port A lines that act as either latched output or buffered input lines depending upon the control word loaded into the control word register.

PC_7–PC_4 Upper nibble of port C lines. They may act as either output latches or input buffers lines. This port also can be used for generation of handshake lines in mode1 or mode 2.

PC_3–PC_0 These are the lower port C lines, other details are the same as PC_7–PC_4 lines.

PB_0–PB_7 These are the eight port B lines which are used as latched output lines or buffered input lines in the same way as port A.

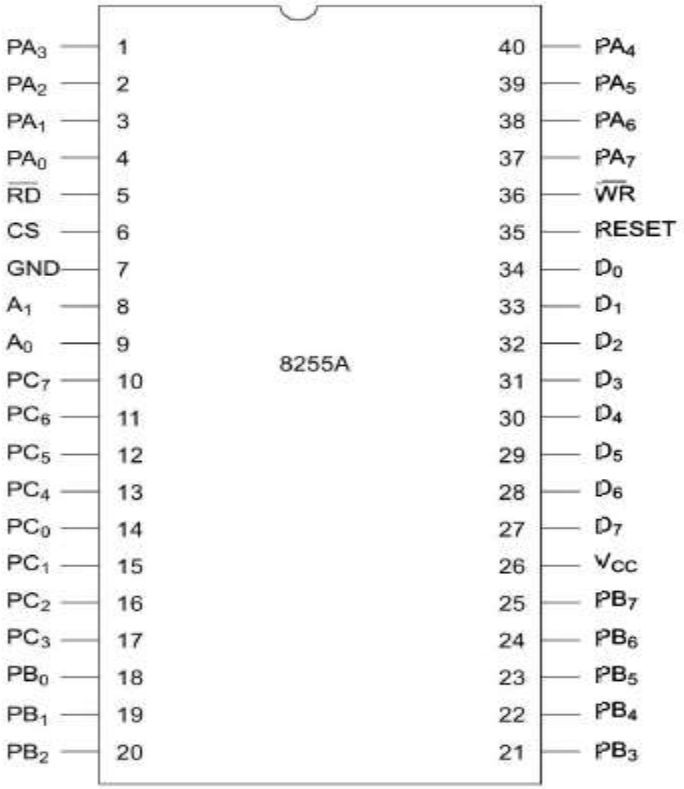


Fig. (b) ***8255A Pin Configuration***

$\overline{RD}$ This is the input line driven by the microprocessor and should be low to indicate read operation to 8255.

$\overline{WR}$ This is an input line driven by the microprocessor. A low on this line indicates write operation.

$\overline{CS}$ This is a chip select line. If this line goes low, it enables the 8255 to respond to $\overline{RD}$ and $\overline{WR}$ signals, otherwise $\overline{RD}$ and $\overline{WR}$ signals are neglected.

A_1–A_0 These are the address input lines and are driven by the microprocessor. These lines ($A_1 - A_0$) with $\overline{RD}$, $\overline{WR}$ and $\overline{CS}$ form the following operations for 8255. These address lines are used for addressing any one of the four registers, i.e. three ports and a control word register as given in Tables (a), (b) and (c).

In case of 8086 systems, if the 8255 is to be interfaced with lower order data bus, the A_0 and A_1 pins of 8255 are connected with A_1 and A_2 respectively.

Table (a)

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	*Input (Read) cycle*
0	1	0	0	0	Port A to data bus
0	1	0	0	1	Port B to data bus
0	1	0	1	0	Port C to data bus
0	1	0	1	1	CWR to data bus

Table ! (b)

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	*Output (Write) cycle*
1	0	0	0	0	Data bus to Port A
1	0	0	0	1	Data bus to Port B
1	0	0	1	0	Data bus to Port C
1	0	0	1	1	Data bus to CWR

Table (c)

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	*Function*
X	X	1	X	X	Data bus tristated
1	1	0	X	X	Data bus tristated

D_0–D_7 These are the data bus lines those carry data or control word to/from the micro-processor.
RESET A logic high on this line clears the control word register of 8255. All ports are set as input ports by default after reset.

MODES OF OPERATION OF 8255

There are two basic modes of operation of 8255–I/O mode and Bit Set-Reset mode (BSR). In the I/O mode, the 8255 ports work as programmable I/O ports, while in BSR mode only port C(PC_0–PC_7) can be used to set or reset its individual port bits. Under the IO mode of operation, further there are three modes of operation of 8255, so as to support different types of applications, viz. *mode 0* , *mode 1* and *mode 2*. These modes of operation are discussed below

BSR Mode

In this mode, any of the 8-bits of port C can be set or reset depending on B_0 of the control word. The bit to be set or reset is selected by bit select flags B_3, B_2 and B_1 of the CWR as given in Table The CWR format is shown in Fig. (a).

Table

B_3	B_2	B_1	*Selected Bits of port C*
0	0	0	B_0
0	0	1	B_1
0	1	0	B_2
0	1	1	B_3
1	0	0	B_4
1	0	1	B_5
1	1	0	B_6
1	1	1	B_7

I/O MODES

MODE 0 (Basic I/O mode) This mode is also known as *basic input/output mode*. This mode provides simple input and output capability using each of the three ports. Data can be simply read from and written to the input and output ports respectively, after appropriate initialisation.

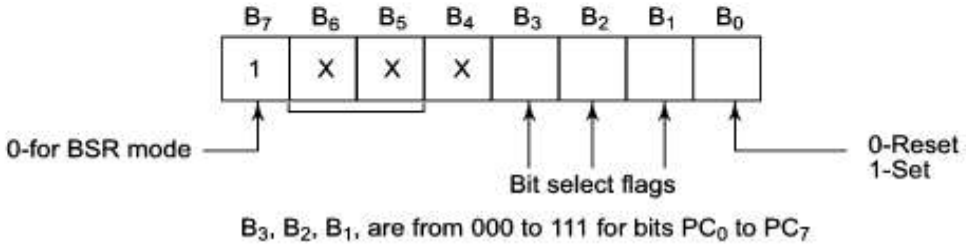
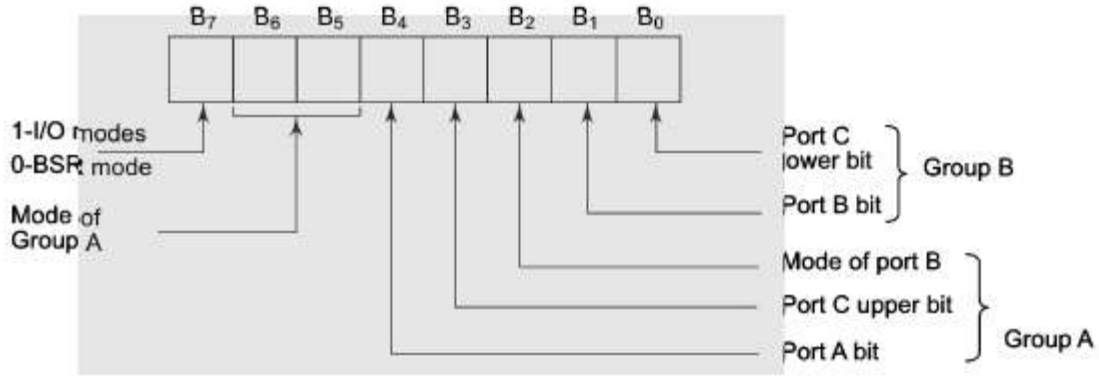


Fig. (a) ***BSR Mode Control Word Register Format***

The salient features of this mode are as listed below:

(i) Two 8-bit ports (port A and port B) and two 4-bit ports (port C upper and lower) are available. The two 4-bit ports can be combinedly used as a third 8-bit port.
(ii) Any port can be used as an input or output port.
(iii) Output ports are latched. Input ports are not latched.
(iv) A maximum of four ports are available so that overall 16 I/O configurations are possible.

All these modes can be selected by programming a register internal to 8255, known as Control Word Register (CWR) which has two formats. The first format is valid for I/O modes of operation, i.e. modes 0, mode 1 and mode 2 while the second format is valid for bit set/reset (BSR) mode of operation. This format is shown in Fig. (b).



Group A modes

B_6	B_5	Mode
0	0	mode 0
0	1	mode 1
1	0	mode 2
1	1	x

(i) Port B mode is either 0 or 1 depending upon B2 bit.

(ii) A port is an output port if the port bit is 0 else it is input port

Fig. (b) *I/O Mode Control Word Register Format*

MODE 1 (Strobed I/O mode) This mode is also called as strobed input/output mode. In this mode the handshaking signals control the input or output action of the specified port. Port C lines PC_0–PC_2, provide strobe or handshake lines for port B. This group which includes port B and PC_0–PC_2 is called as group B for strobed data input/output. Port C lines PC_3–PC_5 provide strobe lines for port A. This group including port A and PC_3–PC_5 forms group A. Thus port C is utilized for generating handshake signals. The salient features of mode 1 are listed as follows:

(i) Two groups—group A and group B are available for strobed data transfer.
(ii) Each group contains one 8-bit data I/O port and one 4-bit control/data port.
(iii) The 8-bit data port can be either used as input or an output port. Both the inputs and outputs are latched.
(iv) Out of 8-bit port C, P C_0–PC_2 are used to generate control signals for port B and PC_3–PC_5 are used to generate control signals for port A. The lines PC_6, PC_7 may be used as independent data lines.

The control signals for both the groups in input and output modes are explained as follows:

Input control signal definitions (mode 1)

$\overline{\textbf{STB}}$ **(Strobe input)**—If this line falls to logic low level, the data available at 8-bit input port is loaded into input latches.

IBF(Input buffer full)—If this signal rises to logic 1, it indicates that data has been loaded into the latches, i.e. it works as an acknowledgement. IBF is set by a low on $\overline{\textbf{STB}}$ and is reset by the rising edge of $\overline{\textbf{RD}}$ input.

INTR (Interrupt request) This active high output signal can be used to interrupt the CPU whenever an input device requests the service. INTR is set by a high at $\overline{\text{STB}}$ pin and a high at IBF pin. INTE is an internal flag that can be controlled by the bit set/reset mode of either PC_4 ($INTE_A$) or PC_2 ($INTE_B$) as shown in Figs (a) and (b). INTR is reset by a falling edge on $\overline{\text{RD}}$ input. Thus an external input device can request the service of the processor by putting the data on the bus and sending the strobe signal. Figure explains the signal definitions clearly.

The strobed data input cycle waveforms are shown in Fig. (a).

Output control signal definitions (mode 1)

OBF (Output buffer full)—This status signal, whenever falls to logic low, indicates that the CPU has written data to the specified output port. The OBF flip-flop will be set by a rising edge of $\overline{\text{WR}}$ signal and reset by a low going edge at the $\overline{\text{ACK}}$ input.

$\overline{\text{ACK}}$ (Acknowledge input)— $\overline{\text{ACK}}$ signal acts as an acknowledgement to be given by an output device. $\overline{\text{ACK}}$ signal, whenever low, informs the CPU that the data transferred by the CPU to the output device through the port is received by the output device.

INTR (Interrupt request)—Thus an output signal that can be used to interrupt the CPU when an output device acknowledges the data received from the CPU. INTR is set when $\overline{\text{ACK}}$, $\overline{\text{OBF}}$ and INTE are '1'. It is reset by a falling edge on $\overline{\text{WR}}$ input. The INTEA and INTEB flags are controlled by the bit set-reset mode of PC_6 and PC_2, respectively.

The waveforms in Fig. may help in understanding the handshake data transfers. The following Figs (a) and (b) explains the signal definitions of mode1 in output mode clearly.

Input control signal definitions in Mode 1

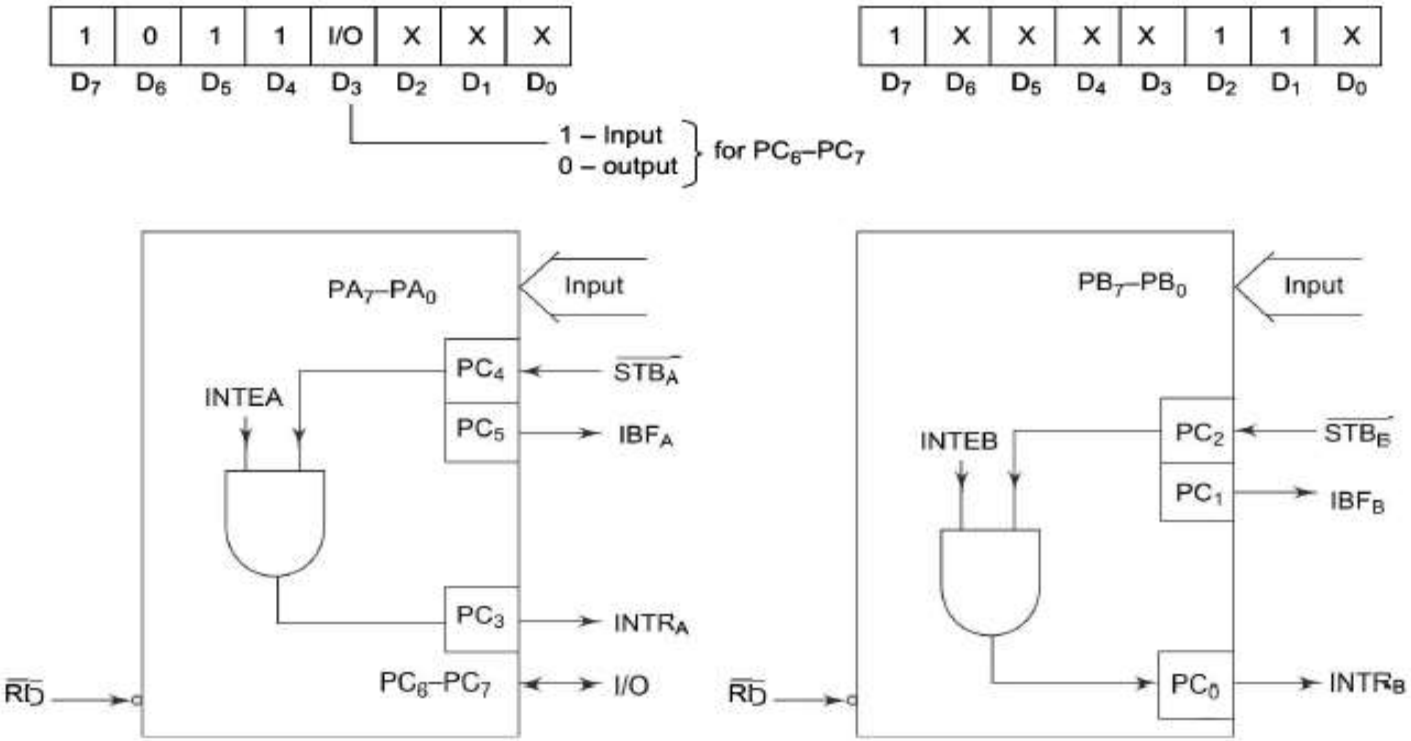


Fig. *(a) Mode1 Control Word Group A I/P (b) Mode 1 Control Word Group B I/P*

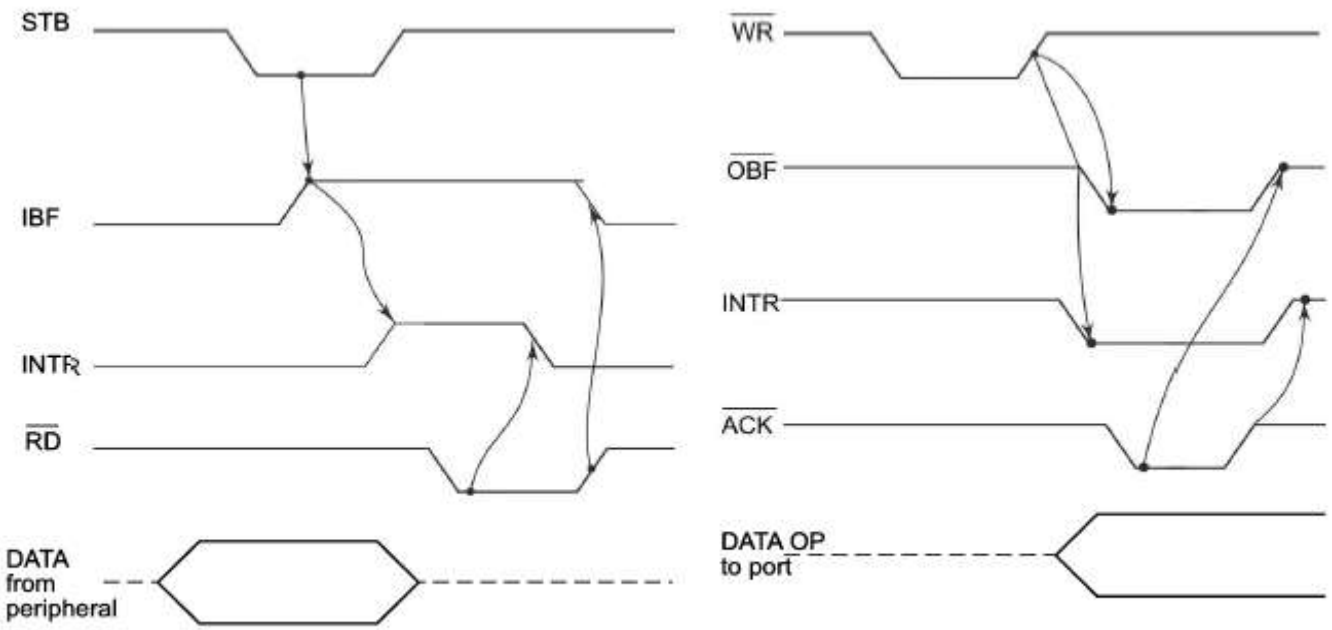


Fig. *(a) Mode1 Strobed Input Data Transfer (b) Mode1 Strobed Data Output*

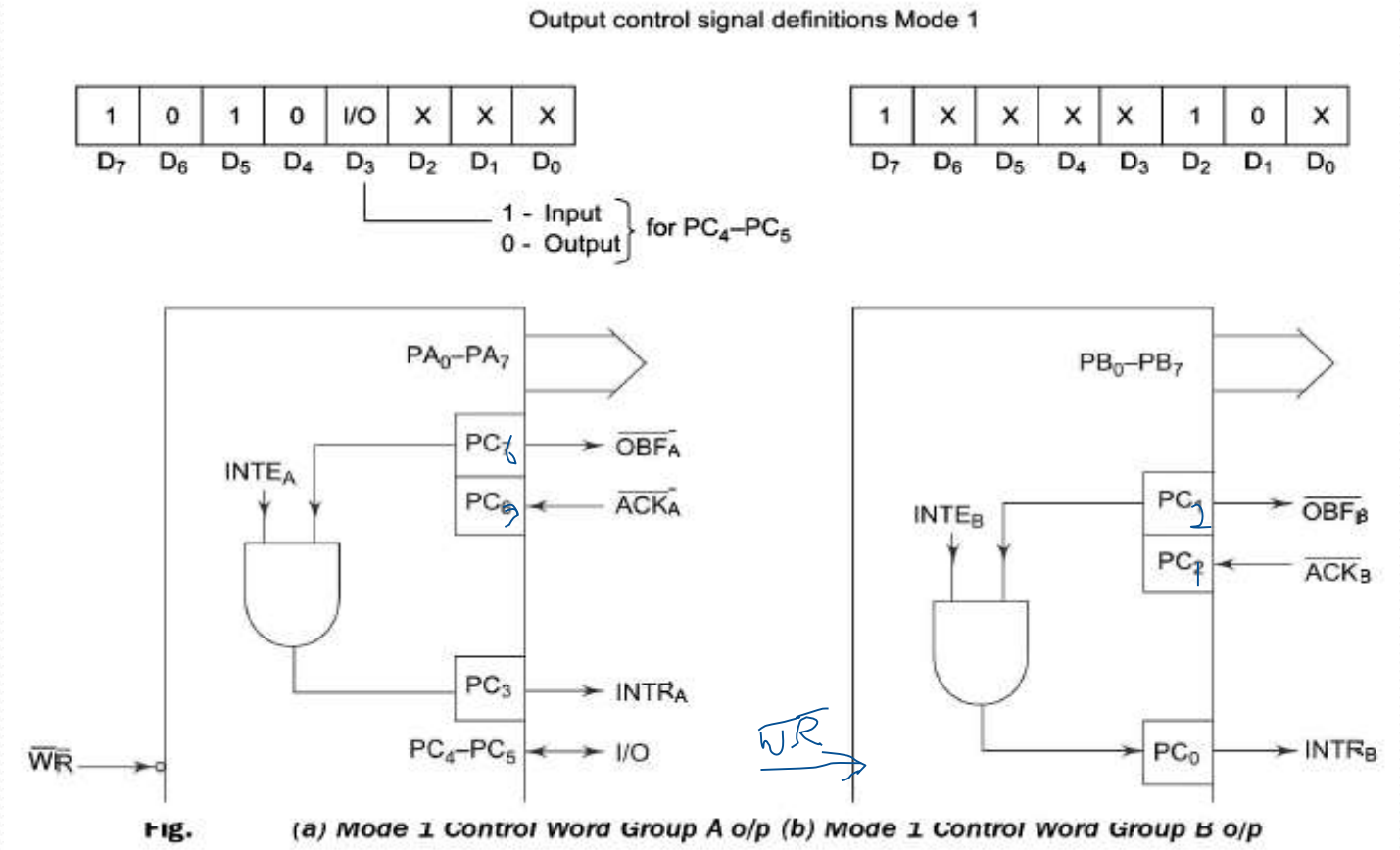


Fig. (a) Mode 1 Control Word Group A o/p (b) Mode 1 Control Word Group B o/p

The group A is used in mode 1 for handshake data transfer so that port A is used for data transfer and port C lines PC_3–PC_5 are used as handshake lines. Port B lines are used for checking the printer status, hence port B is used as input port in mode 0. Port C lower is used as output port for enabling the printer.

MODE 2 (Strobed bidirectional I/O) This mode of operation of 8255 is also known as *strobed bidirectional I/O* . This mode of operation provides 8255 with an additional feature for communicating with a peripheral device on an 8-bit data bus. Handshaking signals are provided to maintain proper data flow and synchronization between the data transmitter and receiver. The interrupt generation and other functions are similar to mode 1. Thus in this mode, 8255 is a bidirectional 8-bit port with handshake signals. The $\overline{RD}$ and $\overline{WR}$ signals decide whether the 8255 is going to operate as an input port or output port.

The salient features of mode 2 of 8255 are listed as follows:

1. The single 8-bit port in group A is available.
2. The 8-bit port is bidirectional and additionally a 5-bit control port is available.
3. Three I/O lines are available at port C,viz. PC_2–PC_0.
4. Inputs and outputs are both latched.
5. The 5-bit control port C (PC_3–PC_7) is used for generating/accepting handshake signals for the 8-bit data transfer on port A.

Control signal definitions in mode 2

INTR (Interrupt request) As in mode 1, this control signal is active high and is used to interrupt the microprocessor to ask for transfer of the next data byte to/from it. This signal is used for input (read) as well as output (write) operations.

Control signals for output operations

$\overline{OBF}$ (Output buffer full) This signal, when falls to logic low level, indicates that the CPU has written data to port A.

$\overline{ACK}$ (Acknowledge) This control input, when falls to logic low level, acknowledges that the previous data byte is received by the destination and the next byte may be sent by the processor. This signal enables the internal tristate buffers to send out the next data byte on port A.

INTE1 (A flag associated with $\overline{OBF}$) This can be controlled by bit set/reset mode with PC_6.

Control signals for input operations

$\overline{STB}$ (Strobe input) A low on this line is used to strobe in the data into the input latches of 8255.

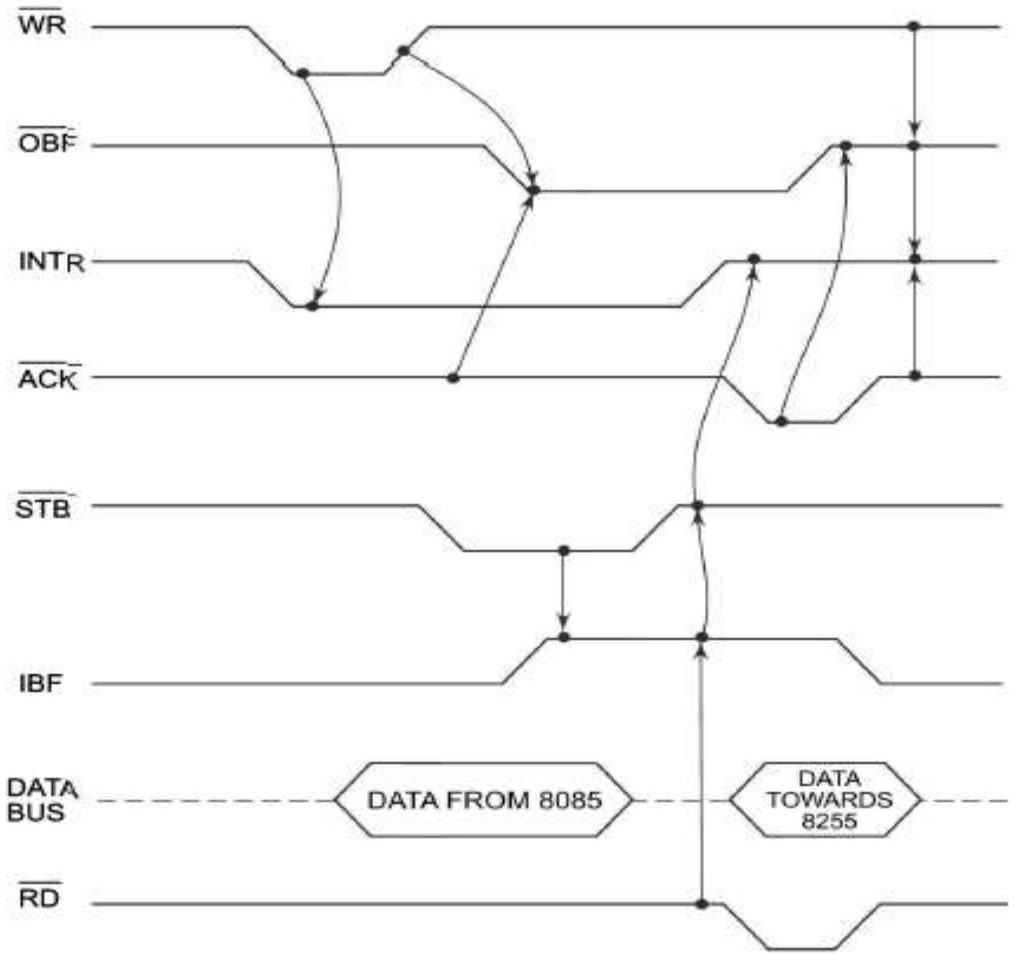


Fig. ***Mode 2 Bidirectional Data Transfer***

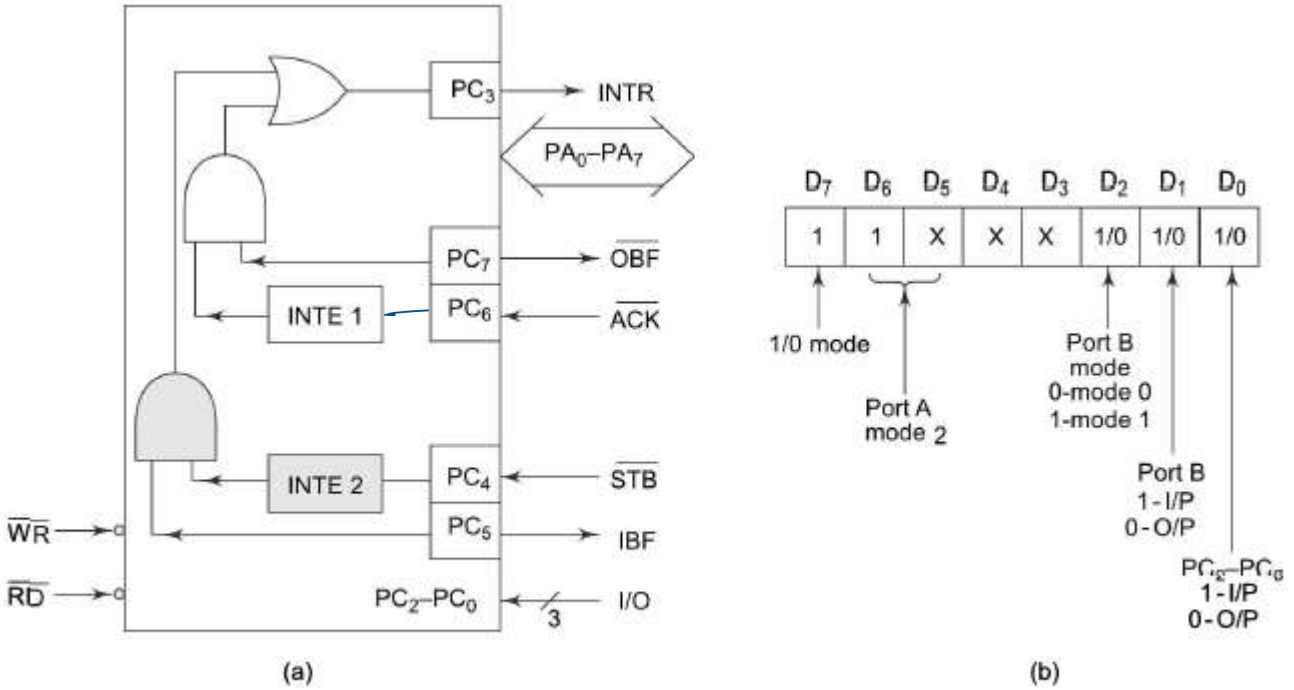


Fig. *(a) Mode 2 pins (b) Mode 2 control word*

IBF (Input buffer full) When the data is loaded into the input buffer, this signal rises to logic '1'. This can be used as an acknowledgement that the data has been received by the receiver.

The waveforms in Fig. show the operation in mode 2 for output as well as input port.

Note: $\overline{WR}$ must occur before $\overline{ACK}$ and $\overline{STB}$ must be activated before $\overline{RD}$.

Figure (a) shows a schematic diagram containing an 8-bit bidirectional port, 5-bit control port and the relation of INTR with the control pins. Port B can either be set to mode 0 or mode 1 while port A (Group A) is in mode 2. Mode 2 is not available for port B. Figure (b) shows the necessary control word.

The INTR goes high only if either IBF, INTE2, $\overline{STB}$ and $\overline{RD}$ go high or $\overline{OBF}$, INTE1, $\overline{ACK}$ and $\overline{WR}$ go high. The port C can be read to know the status of the peripheral device, in terms of the control signals, using the normal I/O instructions.

Problem

Interface 16-bit 8255 ports with 8086. The address of port A is F0H.

Solution To implement a 16-bit port two 8255s are required. One will act as the lower 8-bit port, i.e. D_0–D_7, while the other will act as the upper 8-bit port D_8–D_{15}. The overall scheme is as shown in Fig. . While initialising AL and AH (AX) both should be loaded with a suitable (common) control word. In this system, port A, port B and port C all may work as 16-bit ports.

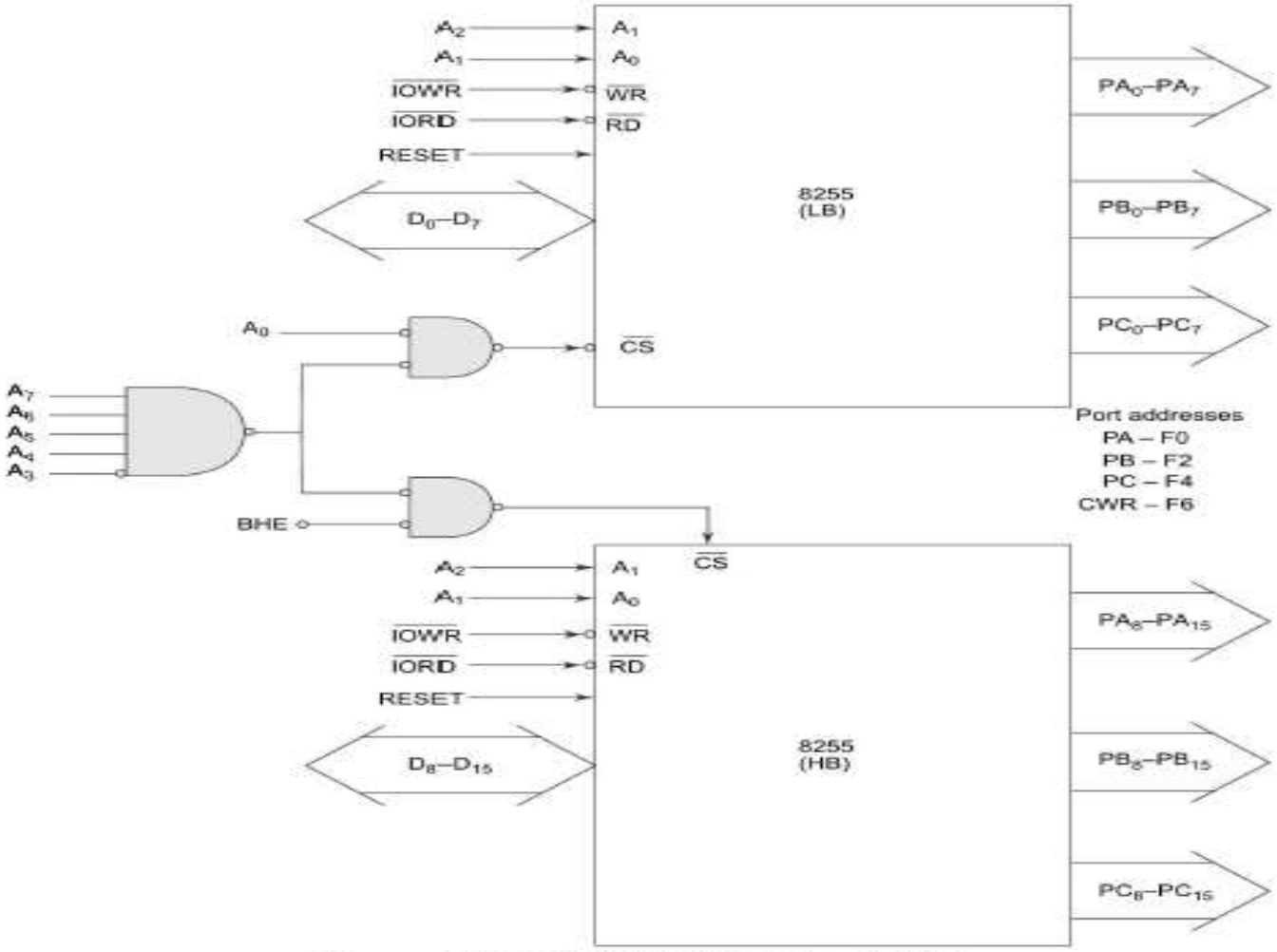


Fig. *Interfacing 16-bit 8255 ports with 8086*

DMA CONTROLLER 8257

The *Direct Memory Access* or DMA mode of data transfer is the fastest amongst all the modes of data transfer. In this mode, the device may transfer data directly to/from memory without any interference from the CPU. The device requests the CPU (through a DMA controller) to hold its data, address and control bus, so that the device may transfer data directly to/from memory. The DMA data transfer is initiated only after receiving HLDA signal from the CPU. For facilitating DMA type of data transfer between several devices, a DMA controller may be used.

Intel's 8257 is a four channel DMA controller designed to be interfaced with their family of microprocessors. The 8257, on behalf of the devices, requests the CPU for bus access using local bus request input i.e. HOLD in minimum mode. In maximum mode of the microprocessor $\overline{RQ}/\overline{GT}$ pin is used as bus request input. On receiving the HLDA signal (in minimum mode) or $\overline{RQ}/\overline{GT}$ signal (in maximum mode) from the CPU, the requesting device gets the access of the bus, and it completes the required number of DMA cycles for the data transfer and then hands over the control of the bus back to the CPU.

Internal Architecture of 8257

The internal architecture of 8257 is depicted in Fig. ' . The chip supports four DMA channels, i.e. four peripheral devices can independently request for DMA data transfer through these channels at a time. The DMA controller has 8-bit internal data buffer, a read/write unit, a control unit, a priority resolving unit along with a set of registers. '

Register Organisation of 8257 The 8257 performs the DMA operation over four independent DMA channels. Each of the four channels of 8257 has a pair of two 16-bit registers, viz. DMA *address register* and *terminal count register*. Also, there are two common registers for all the channels, namely, *mode set register* and *status register*. Thus there are a total of ten registers. The CPU selects one of these ten registers using address lines A_0–A_3. Table ' shows how the A_0–A_3 bits may be used for selecting one of these registers.

DMA Address Registers Each DMA channel has one DMA address register. The function of this register is to store the address of the starting memory location, which will be accessed by the DMA channel. Thus the starting address of the memory block which will be accessed by the device is first loaded in the DMA address register of the channel. Naturally, the device that wants to transfer data over a DMA channel, will access the block of memory with the starting address stored in the DMA Address Register.

Terminal Count Register As in the previous case, each of the four DMA channels of 8257 has one terminal count register (TC). This 16-bit register is used for ascertaining that the data transfer through a DMA channel ceases or stops after the required number of DMA cycles. Thus this register should be appropriately written before the actual DMA operation starts. The low order 14-bits of the terminal count register are initialised with the binary equivalent of the number of required DMA cycles minus one. After each DMA cycle, the terminal count register content will be decremented by one and finally it becomes zero after the required number of DMA cycles are over.

The bits 14 and 15 of this register indicate the type of the DMA operation (transfer). If the device wants to write data into the memory, the DMA operation is called DMA write operation. Bit 14 of the register in this case will be set to one and bit 15 will be set to zero. Table ' gives details of DMA operation selection and the corresponding bit configuration of the bits 14 and 15 of the TC register.

Mode Set Register The mode set register is used for programming the 8257 as per the requirements of the system. The function of the mode set register is to enable the DMA channels individually and also to set the various modes of operation. A DMA channel should not be enabled till the DMA address register and the terminal count register contain valid information, otherwise, an unwanted DMA request may initiate a DMA cycle, probably destroying the valid memory data.

Table *8257 Register Selection*

Register	*Byte*	*Address Inputs*			*F/L*		*BI-Directional Data Bus*							
		A_3	A_2	A_1	A_0		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	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0

Table

Register	*Byte*	*Address Inputs*			*F/L*		*BI-Directional Data Bus*							
		A_3	A_2	A_1	A_0		D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
	MSB	0	0	0	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
CH-1 DMA Address	LSB	0	0	1	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	MSB	0	0	1	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-1 Terminal Count	LSB	0	0	1	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	MSB	0	0	1	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
CH-2 DMA Address	LSB	0	1	0	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	MSB	0	1	0	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-2 Terminal Count	LSB	0	1	0	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	MSB	0	1	0	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
CH-3 DMA Address	LSB	0	1	1	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	MSB	0	1	1	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-3 Terminal Count	LSB	0	1	1	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	MSB	0	1	1	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
MODE SET (Programme only)	—	1	0	0	0	0	AL	TCS	EW	RP	EN3	EN2	EN1	EN0
STATUS (Read only)	—	1	0	0	0	0	0	0	0	UP	TC3	TC2	TC1	TC0

A_0–A_{15} DMA Starting Address, C_0–C_{13} Terminal Count Value (N-1), Rd & Wr-DMA verify (00), Write (01) or Read (10) cycle selection. AL Auto Load, TCS-TC STOP, EW-Extended Write, RP-Rotating Priority, EN3-EN0-Channel Mask Enable, UP-Update Flag, TC3-TC0-Terminal Count Status Bits.

The mode set register format is shown in Fig. . It is thus extremely important that the mode set register should be programmed by the CPU for enabling the DMA channels only after initializing the DMA address register and terminal count register appropriately.

Table *DMA Operation Selection Using A_{15}/RD and A_{14}/WR*

Bit 15	*Bit 14*	*Type of DMA Operation*
0	0	Verify DMA Cycle
0	1	Write DMA Cycle
1	0	Read DMA Cycle
1	1	(Illegal)

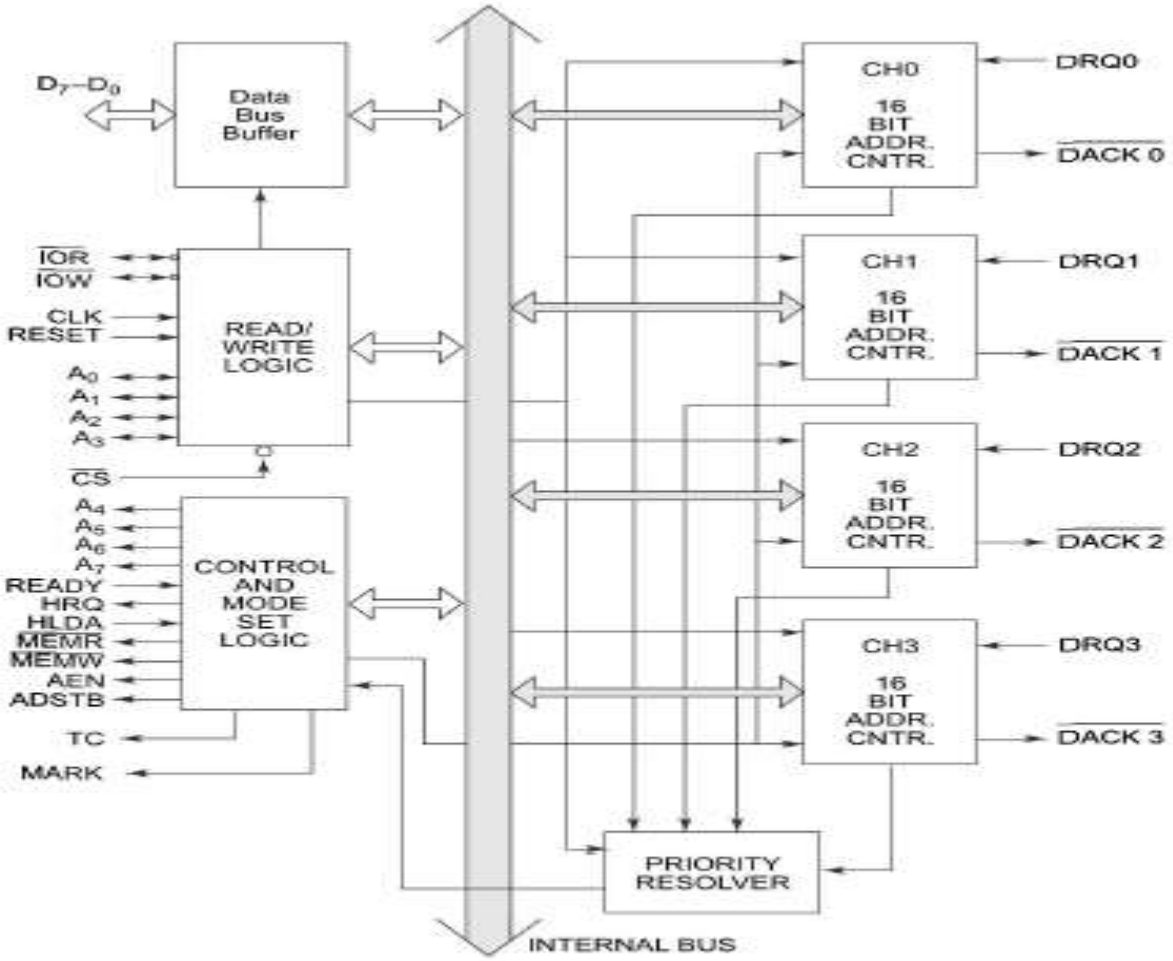


Fig. *Internal Architecture of 8257*

The bits D_0–D_3 enable one of the four DMA channels of 8257. For example, if D_0 is '1', channel 0 is enabled. If bit D_4 is set, rotating priority is enabled, otherwise, the normal, i.e. fixed priority is enabled.

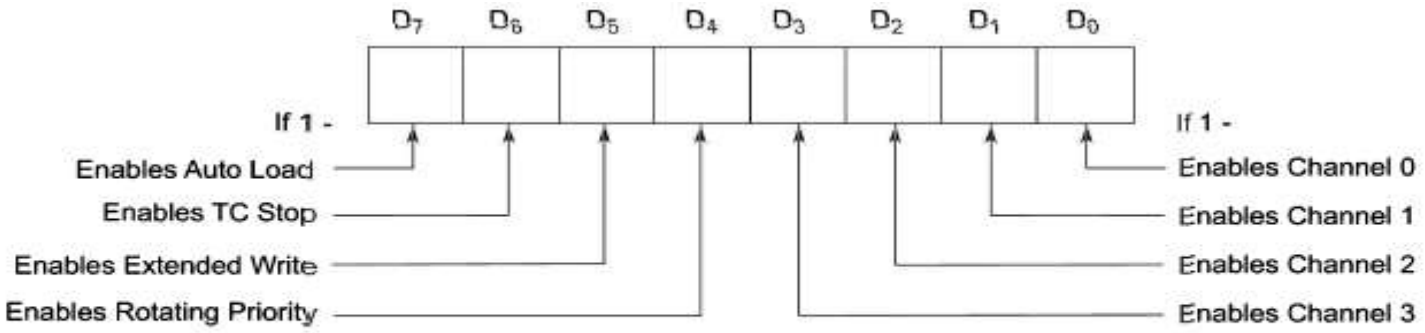


Fig. ***Bit Definitions of the Mode Set Register***

If the TC STOP bit is set, the selected channel is disabled after the *terminal count* condition is reached, and it further prevents any DMA cycle on the channel. To enable the channel again, this bit must be reprogrammed. If the TC STOP bit is programmed to be zero, the channel is not disabled, even after the count reaches zero and further requests are allowed on the same channel.

The auto load bit, if set, enables channel 2 for the repeat block chaining operations, without immediate sofware intervention between the two successive blocks. The channel 2 registers are used as usual, while the channel 3 registers are used to store the block reinitialisation parameters, i.e. the DMA starting address and the terminal count. After the first block is transferred using DMA, the channel 2 registers are reloaded with the corresponding channel 3 registers for the next block transfer, if the *Update* flag is set.

The extended write bit, if set to '1', extends the duration of $\overline{\text{MEMW}}$ and $\overline{\text{IOW}}$ signals by activating them earlier. This is useful in interfacing the peripherals with different access times. If the peripheral is not accessed within the stipulated time, it is expected to give the 'NOT READY' indication to 8257, to request it to add one or more wait states in the DMA cycle. The mode set register can only be written into.

Status Register The status register of 8257 is shown in Fig. . The lower order 4-bits of this register contain the terminal count status for the four individual channels. If any of these bits is set, it indicates that the specific channel has reached the terminal count condition. These bits remain set till either the status is read by the CPU or the 8257 is reset. The update flag is not affected by the read operation. This flag can only be cleared by resetting 8257 or by resetting the auto load bit of the mode set register. If the update flag is set, the contents of the channel 3 registers are reloaded to the corresponding registers of

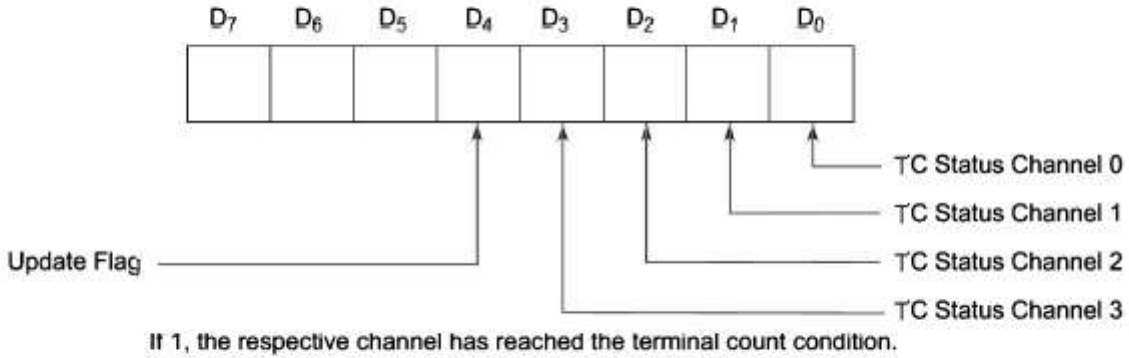


Fig. ***Bit Definitions of Status Register of 8257***

channel 2, whenever the channel 2 reaches a terminal count condition, after transferring one block and the next block is to be transferred using the autoload feature of 8257. The update flag is set every time, the channel 2 registers are loaded with contents of the channel 3 registers. It is cleared by the completion of the first DMA cycle of the new block. This register can only be read.

Data Bus Buffer, Read/Write Logic, Control Unit and Priority Resolver

The 8-bit, tristate, bidirectional buffer interfaces the internal bus of 8257 with the external system bus under the control of various control signals. In the slave mode, the read/write logic accepts the I/O Read or I/O Write signals, decodes the A_0–A_3 lines and either writes the contents of the data bus to the addressed internal register or reads the contents of the selected register depending upon whether $\overline{IOW}$ or $\overline{IOR}$ signal is activated. In master mode, the read/write logic generates the $\overline{IOR}$ and $\overline{IOW}$ signals to control the data flow to or from the selected peripheral. The control logic controls the sequences of operations and generates the required control signals like AEN, ADSTB, $\overline{MEMR}$, $\overline{MEMW}$, TC and MARK along with the address lines A_4–A_7, in master mode. The priority resolver resolves the priority of the four DMA channels depending upon whether normal priority or rotating priority is programmed.

Signal Descriptions of 8257

Figure shows pin configuration of 8257, followed by the functional description of each signal.

DRQ_0–DRQ_3 These are the four individual channel DMA request inputs, used by the peripheral devices for requesting DMA services. The DRQ_0 has the highest priority while DRQ_3 has the lowest one, if the fixed priority mode is selected.

$\overline{DACK_0}$ – $\overline{DACK_3}$ These are the active-low DMA acknowledge output lines which inform the requesting peripheral that the request has been honoured and the bus is relinquished by the CPU. These lines may act as strobe lines for the requesting devices.

D_0–D_7 These are bidirectional, data lines used to interface the system bus with the internal data bus of 8257. These lines carry command words to 8257 and status word from 8257, in slave mode, i.e. under the control of CPU. The data over these lines may be transferred in both the directions. When the 8257 is the bus master (master mode, i.e. not under CPU control), it uses D_0–D_7 lines to send higher byte of the generated address to the latch. This address is further latched using ADSTB signal. The address is transferred over D_0–D_7 during the first clock cycle of the DMA cycle. During the rest of the period, data is available on the data bus.

$\overline{\text{IOR}}$ This is an active-low bidirectional tristate input line that acts as an input in the slave mode. In slave mode, this input signal is used by the CPU to read internal registers of 8257. This line acts as output in master mode. In master mode, this signal is used to read data from a peripheral during a memory write cycle.

$\overline{\text{IOW}}$ This is an active-low, bidirectional tristate line that acts as input in slave mode to load the contents of the data bus to the 8-bit mode register or upper/lower byte of a 16-bit DMA address register or terminal count register. In the master mode, it is a control output that loads the data to a peripheral during DMA memory read cycle (write to peripheral).

CLK This is a clock frequency input required to derive basic system timings for the internal operation of 8257.

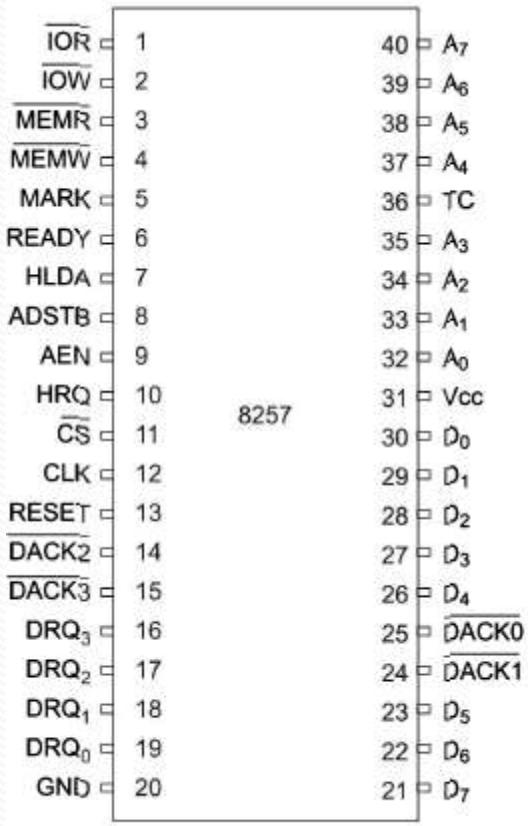


Fig. ***Pin Diagram of 8257***

RESET This active-high asynchronous input disables all the DMA channels by clearing the mode register and tristates all the control lines.

A_0–A_3 These are the four least significant address lines. In slave mode, they act as input which select one of the registers to be read or written. In the master mode, they are the four least significant memory address output lines generated by 8257.

$\overline{CS}$ This is an active-low chip select line that enables the read/write operations from/to 8257, in slave mode. In the master mode, it is automatically disabled to prevent the chip from getting selected (by CPU) while performing the DMA operation.

A_4–A_7 This is the higher nibble of the lower byte address generated by 8257 during the master mode of DMA operation.

READY This is an active-high asynchronous input used to stretch memory read and write cycles of 8257 by inserting wait states. This is used while interfacing slower peripherals.

HRQ The hold request output requests the access of the system bus. In the non-cascaded 8257 systems, this is connected with HOLD pin of CPU. In the cascade mode, this pin of a slave is connected with a DRQ input line of the master 8257, while that of the master is connected with HOLD input of the CPU.

HLDA The CPU drives this input to the DMA controller high, while granting the bus to the device. This pin is connected to the HLDA output of the CPU. This input, if high, indicates to the DMA controller that the bus has been granted to the requesting peripheral by the CPU.

$\overline{MEMR}$ This active-low memory read output is used to read data from the addressed memory locations during DMA read cycles.

$\overline{MEMW}$ This active-low three state output is used to write data to the addressed memory location during DMA write operation.

ADSTB This output from 8257 strobes the higher byte of the memory address generated by the DMA controller into the latches.

AEN This output is used to disable the system data bus and the control the bus driven by the CPU. This may be used to disable the system address and data bus by using the enable input of the bus drivers to inhibit the non-DMA devices from responding during DMA operations. This also may be used to transfer the higher byte of the generated address over the data bus. If the 8257 is I/O mapped, this should be used to disable the other I/O devices, when the DMA controller address is on the address bus.

TC Terminal count output indicates to the currently selected peripheral that the present DMA cycle is the last for the previously programmed data block. If the TC STOP bit in the mode set register is set, the selected channel will be disabled at the end of the DMA cycle. The TC pin is activated when the 14-bit content of the terminal count register of the selected channel becomes equal to zero. The lower order 14 bits of the terminal count register are to be programmed with a 14-bit equivalent of (n-1), if n is the desired number of DMA cycles.

MARK The modulo 128 mark output indicates to the selected peripheral that the current DMA cycle is the 128th cycle since the previous MARK output. The mark will be activated after each 128 cycles or integral multiples of it from the beginning of the data block (the first DMA cycle), if the total number of the required DMA cycles (n) is completely divisible by 128.

V_{cc} This is a +5V supply pin required for operation of the circuit.

GND This is a return line for the supply (ground pin of the IC).

DMA TRANSFERS AND OPERATIONS

The 8257 is able to accomplish three types of operations, viz. verify DMA operation, write operation and read operation. The complete operational sequence of 8257 is described using a state diagram in Fig. for a single channel.

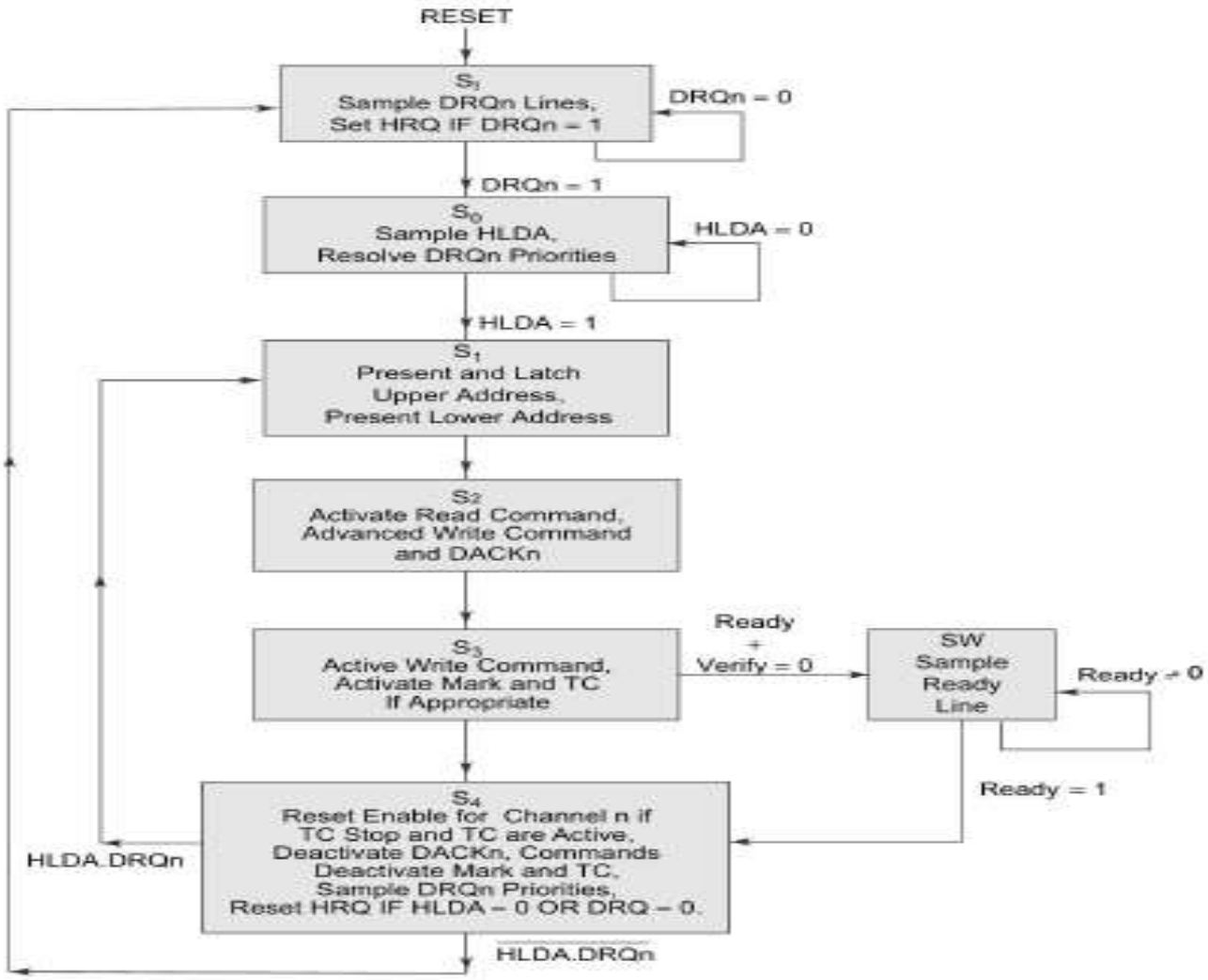
A single byte transfer using 8257 may be requested by an I/O device using any one of the 8257 DRQ inputs. In response, the 8257 sends HRQ signal to the CPU at its HLD input and waits for acknowledgement at the HLDA input. If the HLDA signal is received by the DMA controller, it indicates that the bus is available for the transfer. The $\overline{\text{DACK}}$ line of the used channel is pulled down by the DMA controller to indicate the I/O device that its request for the DMA transfer has been honoured by the CPU. The DMA controller generates the read and write commands to transfer the byte from/to the I/O device. The $\overline{\text{DACK}}$ line is pulled transfer is over, to indicate the DMA controller that the transfer, as requested by the device, is over. The HRQ line is lowered by the DMA controller to indicate the CPU that it may regain the control of the bus. The DRQ must be high until acknowledged and must go low before S_4 state of the DMA operation state diagram to avoid another unwanted transfer.

If more than one channel requests service simultaneously, the transfer will occur as a *burst transfer*. This will be discussed further in case of 8237. No overhead is required in switching from one channel to another. In each S_4, the DRQ lines are sampled and the highest priority request is recognized during the next transfer. Once the higher priority transfer is over; the lower priority transfer requests may be served, provided their DRQ lines are still active. The HRQ line is maintained active till all the DRQ lines go low.

The burst or continuous transfer, described above may be interrupted by an external device by pulling down the HLDA line. After each transfer, the 8257 checks the HLDA line. If it is found active, it completes the current transfer and releases the HRQ line (i.e. sends it low) and returns to its idle state. If the DRQ line is still active, the 8257 will again activate HRQ and proceed as already described. The 8257 uses four clock cycles to complete a transfer. The 8257 has a READY input to interface it with low speed devices. The READY pin status is checked in S_3 of the state diagram. If it is low, the 8257 enters a wait state. The status is sampled in every state till it goes high. Once the READY pin goes high, the 8257 continues from state S_4 to complete the transfer. The 8257 can be interfaced as a memory mapped device or an I/O mapped device. If it is connected as memory mapped device, proper care must be taken while programming Rd/A_{15} and Wr/A_{14} bits in the terminal count register.

Priorities of the DMA Requests

The 8257 can be programmed to select any of the two priority schemes using the command register. The first is the *fixed priority scheme*, while the second is the *rotating priority scheme*. In the fixed priority scheme, each device connected to a channel is assigned a fixed priority. In this scheme, the $DREQ_3$ has the lowest priority followed by DRQ_2 and DRQ_1. DRQ_0 has the highest priority. In the rotating priority mode, the priorities assigned to the channels are not fixed. At any point of time, suppose DRQ_0 has highest priority and DRQ_3 the lowest, then after the device at channel 0 gets the service, its priority goes down and the channel 0 becomes the lowest priority channel. Channel1 now becomes the highest priority channel, and remains the highest priority channel till it gets the service. Once channel 1 is served, it becomes the lowest priority channel and the channel 2 now becomes the highest priority channel. If you select the rotating priority, in a single chip DMA system any device requesting the service is guaranteed to be recognized after no more than three higher priority requests, thus avoiding dominance of any one channel. The priority allotment in the rotating priority mode is as shown in Fig.



DRQn refers to anv DRQ line of an enabled DMA channel

Fig. ***DMA Operation State Diagram***

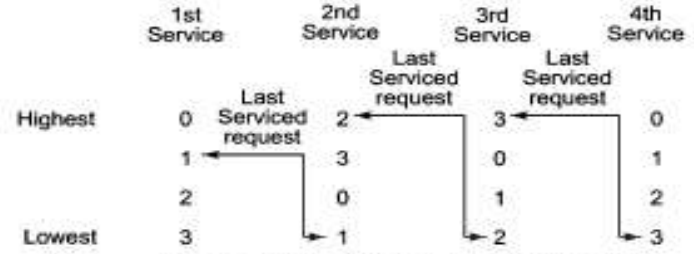


Fig. *Priority Allotment in Rotating Priority Mode*

Programming and Reading the 8257 Registers

The selected register may be read or written depending upon the instruction executed by the CPU but the mode set register can only be written in, while the status register can only be read. The 16-bit register pair of each channel is read or written in two successive read or write operations. The Least Significant Byte (LSB) and the Most Significant Byte (MSB) of each register for a specific channel has the same address, but they are differentiated by an internal First/Last (F/L) flip-flop. If the F/L flip-flop is 0, it indicates the first operation, i.e. the LSB is to be read or written, otherwise, it is the last operation, i.e. the MSB is to be read or written. The F/L flip-flop can be cleared by resetting 8257. Thus the first operation after RESET will always be a LSB operation and the successive one for the same register address will be a MSB operation. The least significant three address bits A_0–A_2 indicate the specific register for a specific channel. The A_3 address line is used to differentiate between all the channel registers and the common registers, i.e. mode set and status registers. The higher order address lines A_4–A_{15} may be used to derive the chip select signal $\overline{CS}$ of 8257. All the accesses to any of the terminal count registers and DMA address registers must be in pairs, i.e. the LSB accesses must be followed by the MSB accesses. In verify transfer mode, no actual data transfer takes place. In this mode, the 8257 acts in the same way as read or write transfer to generate addresses, but no control lines are activated.

Interfacing 8257 with 8086

Once a DMA controller is initialised by a CPU properly, it is ready to take control of the system bus on a DMA request, either from a peripheral or itself (in case of memory-to-memory transfers). The DMA controller sends a HOLD request to the CPU and waits for the CPU to assert the HLDA signal. The CPU relinquishes the control of the bus before asserting the HLDA signal. Once the HLDA signal goes high, the DMA controller activates the $\overline{DACK}$ signal to the requesting peripheral and gains the control of the system bus. The DMA controller is the sole master of the bus, till the DMA operation is over. The CPU remains in the HOLD status (all of its signals are tristated except HOLD and HLDA), till the DMA controller is the master of the bus. In other words, the DMA controller interfacing circuit implements a switching arrangement for the address, data and control busses of the memory and peripheral subsystem from/to the CPU to/from the DMA controller. A conceptual implementation of the system is shown in Fig.

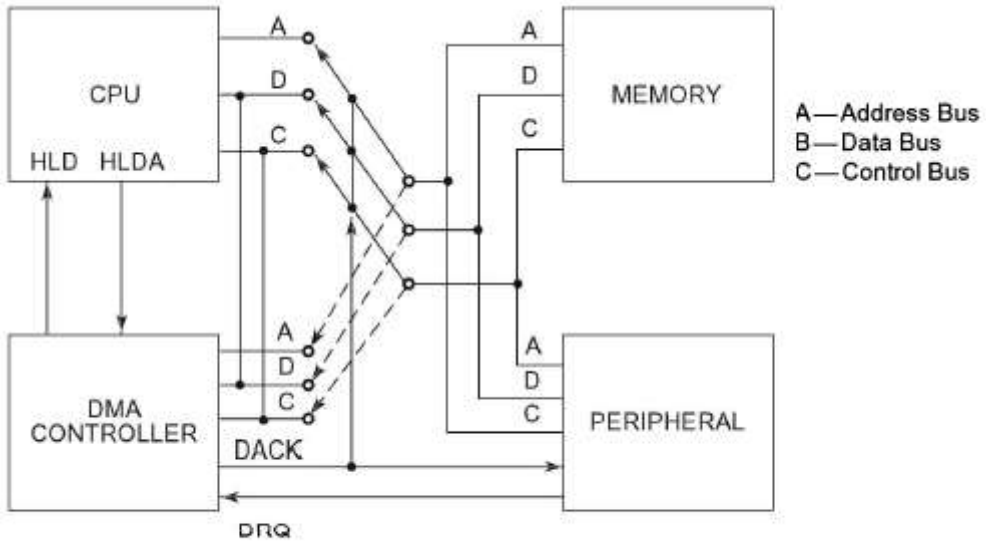


Fig. *Interfacing a Typical DMA Controller with a System*

Interfacing problem-2: Connecting two, 16k*8 ROM and two, 32k*8RAM with 8086

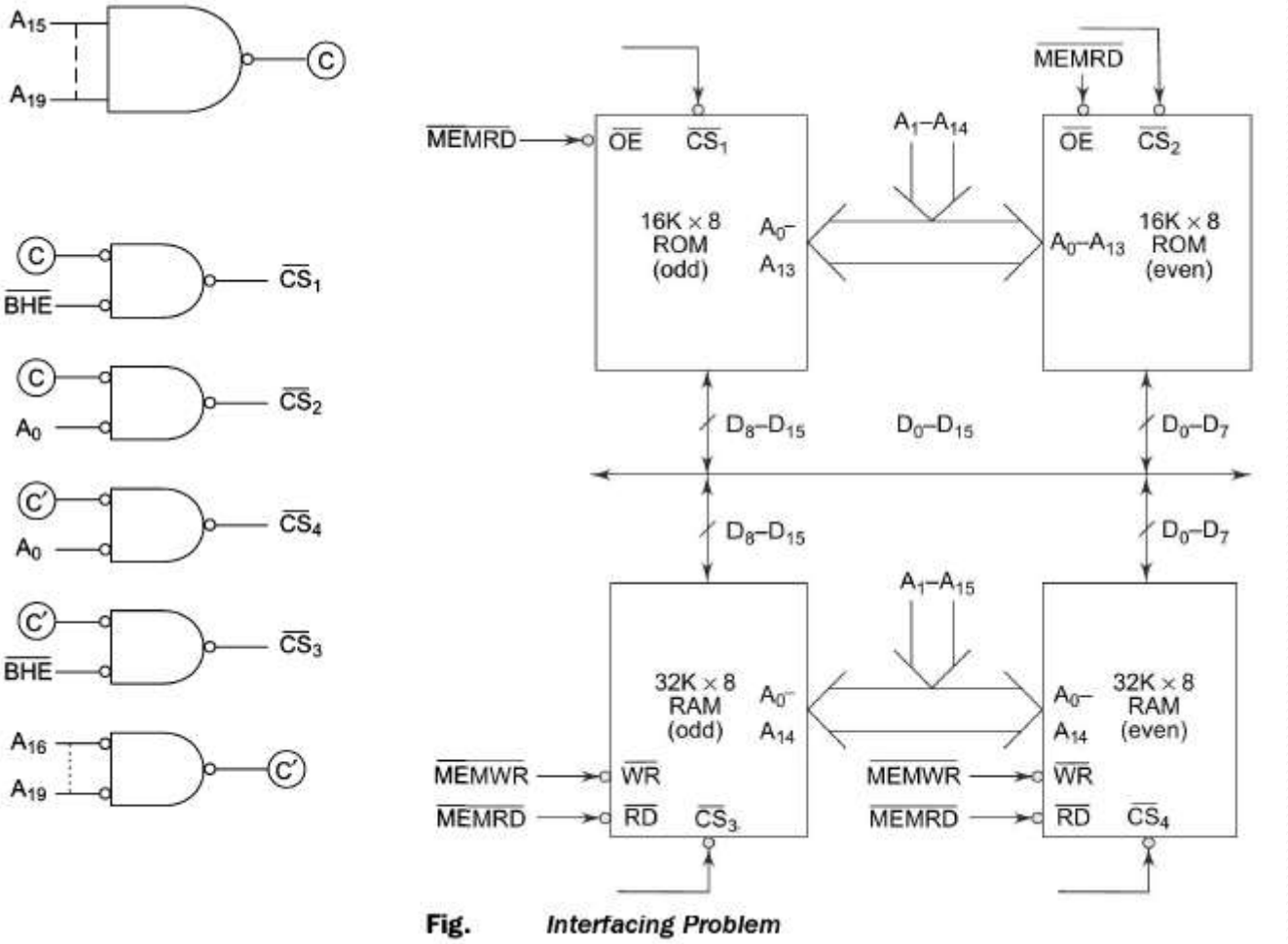


Fig. *Interfacing Problem*

Draw 4 memory chips, we can select odd and even memory address banks using chip select. Derive CS using address lines.

Table

I/P		O/P	
A_0	$B(\overline{BHE})$	C_1	C_2
0	0	0	0
0	1	1	0
1	0	0	1
1	1	1	1

$C_2 = A_0$

$C_1 = \overline{BHE}$

To find out $\overline{CS_1}$ and $\overline{CS_2}$ we will have to combine C_1 and C_2 with C.

Table

I/P			O/P	
C_1	C_2	C	$\overline{CS_1}$	$\overline{CS_2}$
0	0	0	0	0
1	0	0	1	0
1	1	0	1	1
0	1	0	0	1

Table shows that $\overline{CS_1} = C + C_1 = C + \overline{BHE}$ and $\overline{CS_2} = C + C_2 = C + A_0$

Similarly we can find out CS_3 and CS_4.

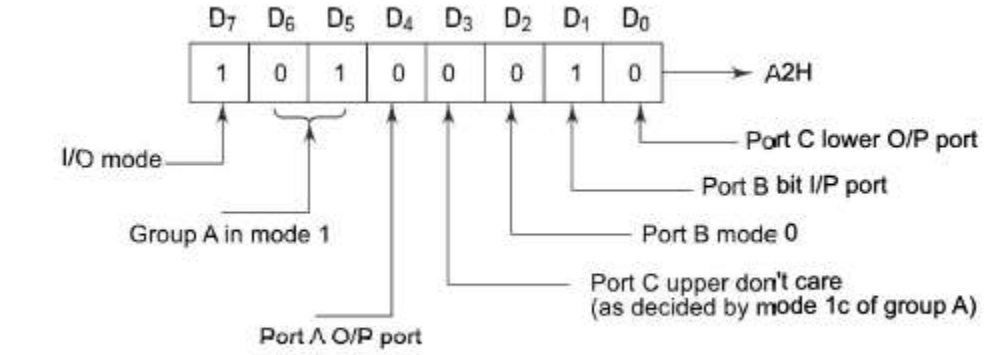
Solution The last address in the map of 8086 is FFFFFH. After resetting, the processor starts from FFFF0H. Hence this address must lie in the address range of EPROM. Figure ! : shows the interfacing diagram, and Table shows complete map of the system.

Table *Address Map for Problem !*

Addresses	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
								32KB			EPROM									
F8000H	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0FFFFH	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
								64KB RAM												
00000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

It is better not to use a decoder to implement the above map because it is not continuous, i.e. there is some unused address space between the last RAM address (0FFFFH) and the first EPROM address (F8000H). Hence the logic is implemented using logic gates, as shown in Fig.

8255- Control word example for I/O



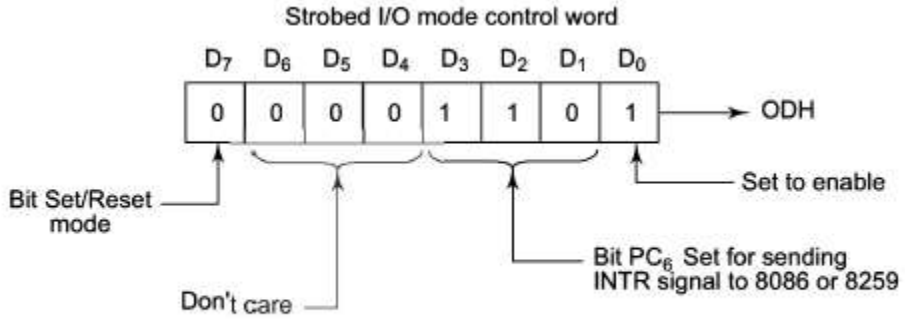


Fig. ***Bit Set/Reset Control Word***

Module 5

8051 Microcontroller

Microprocessor is a single chip CPU, **microcontroller** contains, a CPU and much of the remaining circuitry of a complete microcomputer system in a single chip. **Microcontroller** includes RAM, ROM, serial and parallel interface, timer, interrupt schedule circuitry (in addition to CPU) in a single chip.

1) Computer System Vs Embedded System:

Microprocessor widely used in the computer system. And microcontroller is used in embedded system.

If the microprocessor is the heart of computer system then microcontroller is the heart of the embedded system.

2) Architecture:

The microprocessor uses **Von Neumann architecture** where data and program present in the same memory module .The microcontroller uses **Harvard architecture**. In this module, data and program get stored in separate memory. The microcontroller can access data and program at the same time as it is in a separate memory. This is one of the reasons microcontrollers is faster than the

microprocessor.

KTUNOTES.IN

3) Memory and I/O Components:

The microprocessor can not operate without peripheral components. It has only processing unit and have to attach all the required components externally to operate. Whereas micro control has small processing unit along with internal memory to store and I/O components to give input. So it can work independently.

4) Circuit Size and its Complexity:

As we have to connect components externally, microprocessor circuit becomes large and complex. In microcontroller all the components are internally connected, its circuit becomes too small.

5) Efficient Techniques to use in Compact System:

The microcontroller can be used in the compact system as it has a small size. So microcontroller is better and efficient technique in the compact system than the microprocessor.

Downloaded from Ktunotes.in

6) Cost:

Microprocessor requires external components to operate. So the Cost of the microprocessor is higher than the microcontroller.

7) Power consumption:

Microprocessor requires external components and its circuits also complex. It requires more power consumption. So it is difficult to operate microprocessor using battery power. The microcontroller has very low external components. it manages all its operation inside the single chip. So it consumes very low power supply as compared to the microprocessor. We can operate microcontroller on the externally connected stored power such as a battery.

8) Power saving feature:

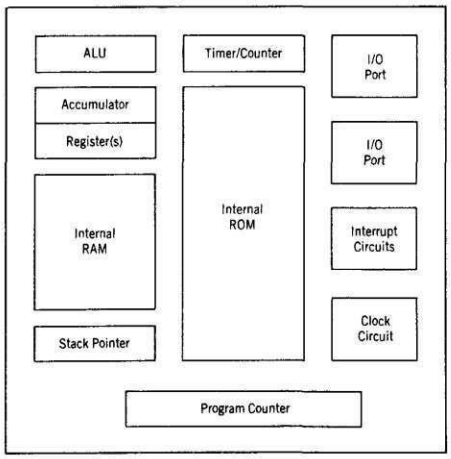
The microcontroller can have multiple modes of operation such as higher performance, balance, idle or power saving mode. So if we operate microcontroller in power saving mode, the power consumption re reduce even more. Most of the Microprocessor does not have this power saving feature.

9) ProcessngSpeed:

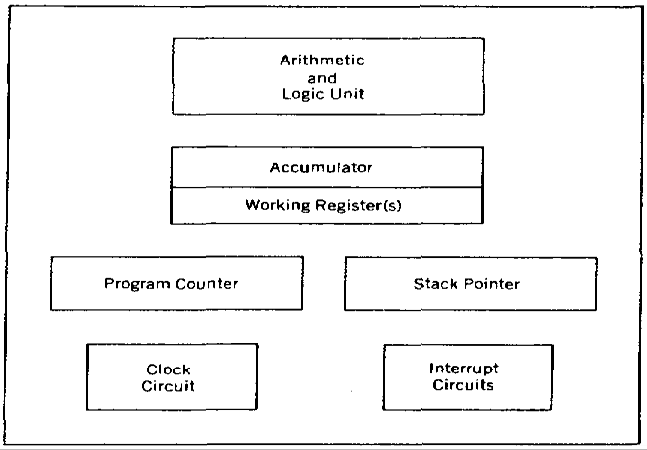
The microprocessor has very less internal registers. It has to rely on external storage. So all the memory operations are carried out using memory based external commands. Results in high processing time.The microcontroller have many registers for instruction execution. Fetching data and storing data require internal commands. So its execution and processing time are lower than the microprocessor.

KTUNOTES.IN

A Block Diagram of a Microcontroller



A Block Diagram of a Microprocessor



Downloaded from Ktunotes.in

1 Overview

- The Intel 8051 is a very popular general purpose microcontroller widely used for small scale embedded systems.
- The 8051 is an 8-bit microcontroller with 8 bit data bus and 16-bit address bus.
- The 16 bit address bus can address a 64K(216) byte code memory space and a separate 64K byte of data memory space.
- The 8051 has 4K on-chip read only code memory and 128 bytes of internal Random Access Memory (RAM)
- Besides internal RAM, the 8051 has various
  - Special Function Registers (SFR) such as the Accumulator, the B register, and many other control registers.
  - 34 8-bit general purpose registers in total. The ALU performs one 8-bit operation at a time. They are register A,B and 32 working registers
- Two 16 bit /Counter timers
- Three internal interrupts (one serial), 2 external interrupts.
- Four 8-bit I/O ports (3 of them are dual purposed). One of them used for serial port,Some 8051 chips come with UART for serial communication and ADC for analog to digital conversion

KTUNOTES.IN

8051 Chip Pins

- 40 pins on the 8051 chip. Most of these pins are used to connect to I/O devices or external data and code memory.
- 4 I/O port take 32 pins(4 x 8 bits) plus a pair of XTALS pins for crystal clock
- A pair of Vcc and GND pins for power supply (the 8051 chip needs +5V 500mA to function properly)
- A pair of timer pins for timing controls, a group of pins (EA, ALE, PSEN) for internal and external data and code memory access controls
- A pair of Crystal clock pins(XTAL1,2)
- One Reset pin for reboot purpose

Downloaded from Ktunotes.in

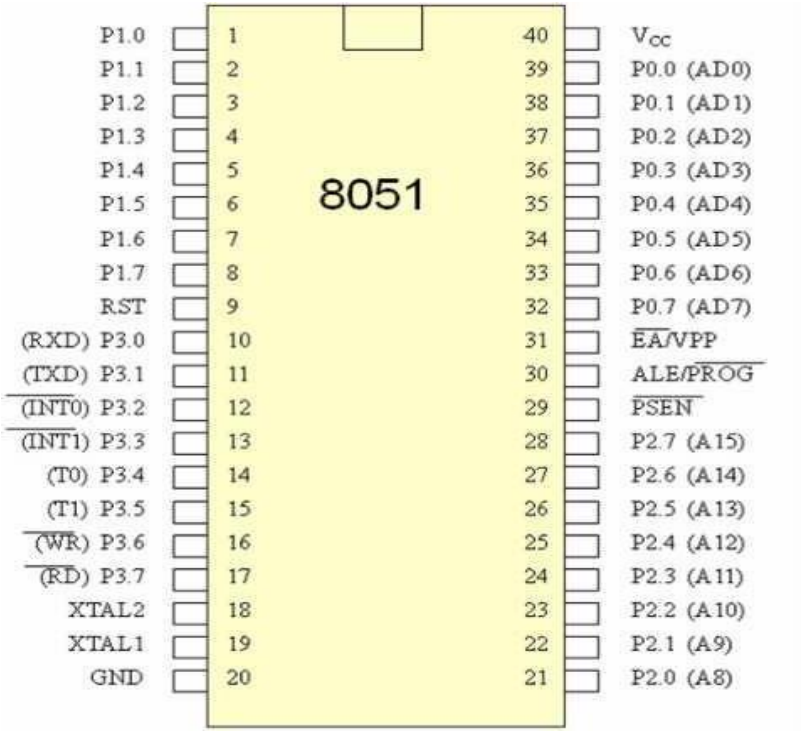


Fig Pin out Diagram of the 8051 Microcontroller

The Pin Connection for External Code and Data Memory

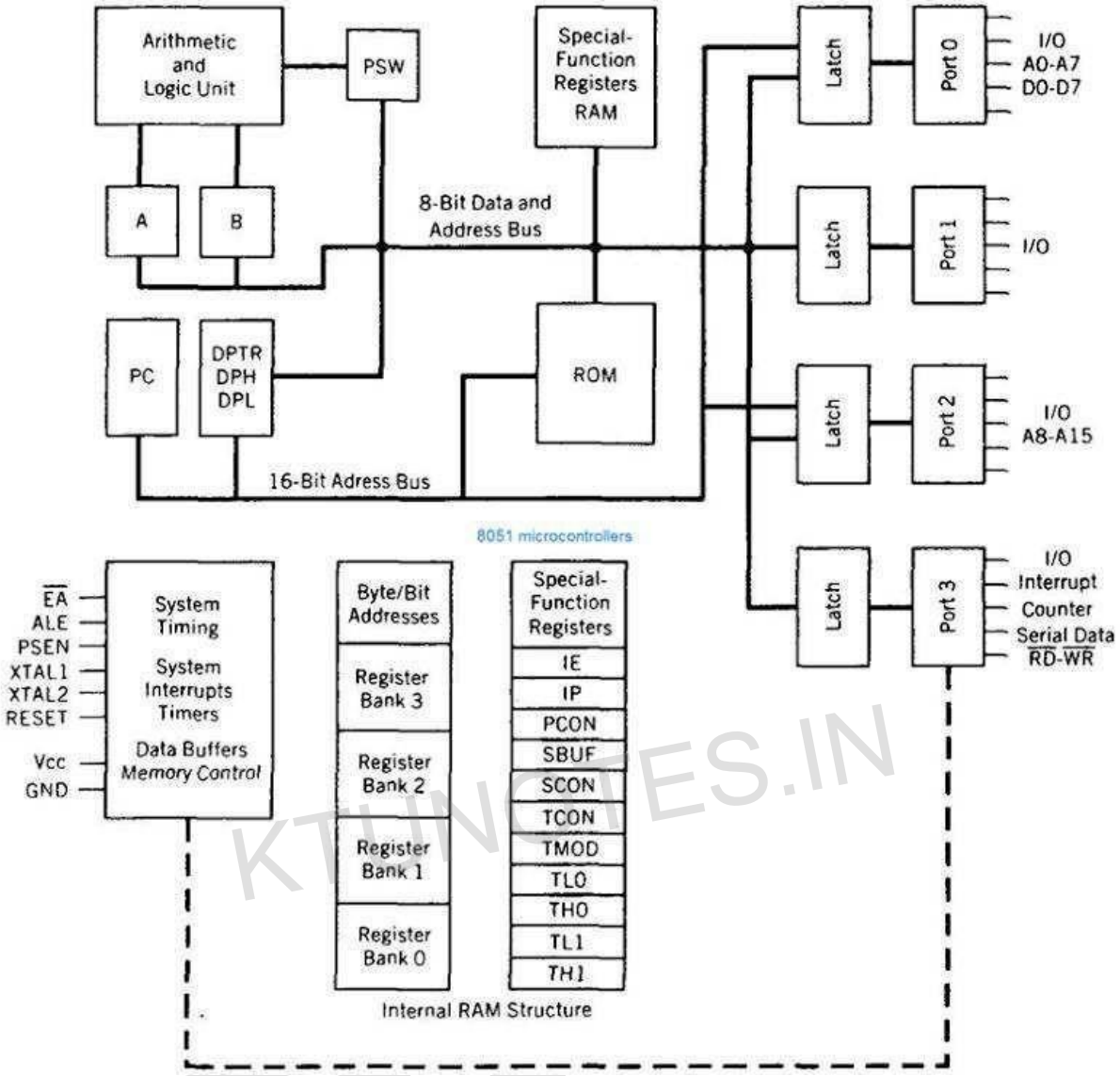
- The EA' (External Access) pin is used to control the internal or external memory access.
- The signal 0 is for external memory access and signal 1 for internal memory access.
- The PSEN' (Program Store Enable) is for reading external code memory when it is low (0) and EA is also 0.
- The ALE (Address Latch Enable) activates the port 0 joined with port 2 to provide 16 bit external address bus to access the external memory.
- The ALE multiplexes the P0: 1 for latching address on P0 as A0-A7 in the 16 bit address
- buss, 0 for latching P0 as data I/O.
- P0.x is named ADx because P0 is multiplexed for Address bus and Data bus at different clock time. WR' provides the signal to write external data memory RD' provides the signal to read external data and code memory.

8051 Internal Architecture

Fig 8051 Internal Architecture

KTUNOTES.IN

Downloaded from Ktunotes.in



System Clock and Oscillator Circuits

- The 8051 requires an external oscillator circuit. The oscillator circuit usually runs around 12MHz. The pulse is used to synchronize the system operation in a controlled pace.
- A machine cycle is minimum amount time a simplest machine instruction must take An 8051 machine cycle consists of 12 crystal pulses (ticks). The first 6 crystal pulses (clock cycle) is used to fetch the opcode and the second 6 pulses are used to perform the operation on the operands in the ALU.
- Pins used are
    - XTAL1
    - XTAL2

Downloaded from Ktunotes.in

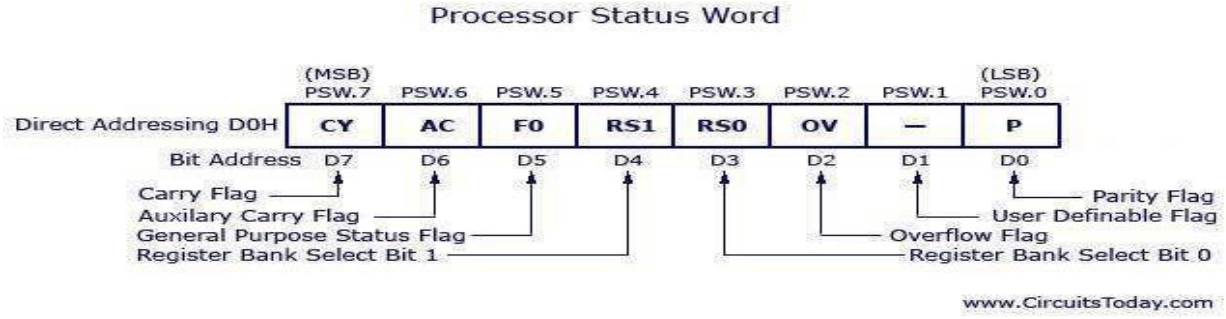
- The CPU has many important registers. The Program Count (PC) always holds the code memory location of next instruction. The CPU is the heart of any computer which is in charge of computer operations.
- It fetches instructions from the code memory into the instruction Register (IR), analyzes the opcode of the instruction, updates the PC to the location of next instruction, fetches the operand from the data memory if necessary, and finally performs the operation in the Arithmetic-Logic Unit (ALU) within the CPU.
- The B register is a register just for multiplication and division operation which requires more register spaces for the product of multiplication and the quotient and the remainder for the division. The immediate result is stored in the accumulator register (Acc) for next operation and the Program Status Word (PSW) is updated depending on the status of the operation result

.

Flags and PSW (Program Status Word) SFR for CPU status

Flags are one bit registers provided to store result of certain program instructions. Other instructions can test the conditions of the flags and make decisions based on the flag states. The flags are grouped into PSW and power control register (PCON).8051 has four math flags that respond automatically to output of math operations and three general purpose user flags can be set to 1 or 0 by the programmer. The math flags include Auxiliary carry(AC),Carry(C),Overflow(OV) and Parity(P).User flags are F0,GF0 and GF0F0 stored in PSW and GF0 & GF1 are stored in PCON

KTUNOTES.IN



- P: parity check flag
- OV: ALU overflow flag
- RS0/RS1: Register bank specification mode
- 00: bank 0 (00H-07H); 01: bank1; 10: bank 2; 11: bank 3(18H- 1FH)
- F0: User defined lag
- CY: ALU carry out

Downloaded from Ktunotes.in

- AC: ALU auxiliary carry out

Ports :

There are four 8-bit ports: P0, P1, P2 and P3. All of them are dual purpose ports except P1 which is only used for I/O.

- PORT P1 (Pins 1 to 8): The port P1 is a port dedicated for general I/O purpose. The other ports P0, P2 and P3 have dual roles in addition to their basic I/O function. PORT P0 (pins 32 to 39): When the external memory access is required then Port P0 is multiplexed for address bus and data bus that can be used to access external memory in conjunction with port P2. P0 acts as A0-A7 in address bus and D0-D7 for port data. It can be used for general purpose I/O if no external memory presents.
- PORT P2 (pins 21 to 28): Similar to P0, the port P2 can also play a role (A8-A15) in the address bus in conjunction with PORT P0 to access external memory.
- PORT P3 (Pins 10 to 17):
  - In addition to acting as a normal I/O port, • P3.0 can be used for serial receive input pin(RXD)
  - P3.1 can be used for serial transmit output pin(TXD) in a serial port
  - P3.2 and P3.3 can be used as external interrupt pins(INT0' and INT1')
  - P3.4 and P3.5 are used for external counter input pins(T0 and T1)
  - P3.6 and P3.7 can be used as external data memory write and read control signal pins(WR' and RD')read and write pins for memory access.

Internal Memory

The 8051 microcontroller contains internal RAM and internal ROM.RAM memory for variable data rhat can be altered as the program runs.

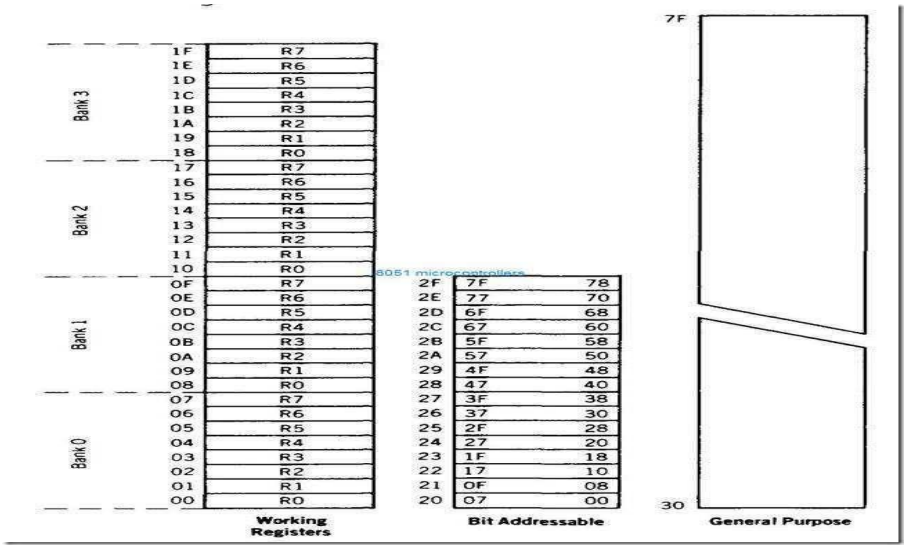
Internal RAM

The 128 byte internal RAM shown in below fig: is organized into three distinct areas

- The first 128 bytes of RAM (from 0000 to 0x7F) are called the direct memory, and can be used to store data.
- The lowest 32 bytes of RAM (00h to 1Fh) are reserved for 4 general register banks. The register banks numbered 0 to 3Each bank consist of 8 registers namely R0 to R7 with 8bit width and register can be addressed by its name or by its RAM address. The default bank is bank 0.

KTUNOTES.IN

Downloaded from Ktunotes.in



- A bit addressable area of 16 byte occupies RAM byte address 20h to 2Fh, forming total 128 byte addressable bit. An addressable bit may be specified by its bit address of 00h to 7Fh
- General purpose RAM area above the bit area from 30h to 7Fh.

Stack and Stack pointer

The stack rfers to an area in internal RAM to store and retrieve data quickly.The 8bit stack pointer (SP) register is used by the 8051 to hold internal RAM addressthat is called top of the stack.

Special Function Registers (SFRs)

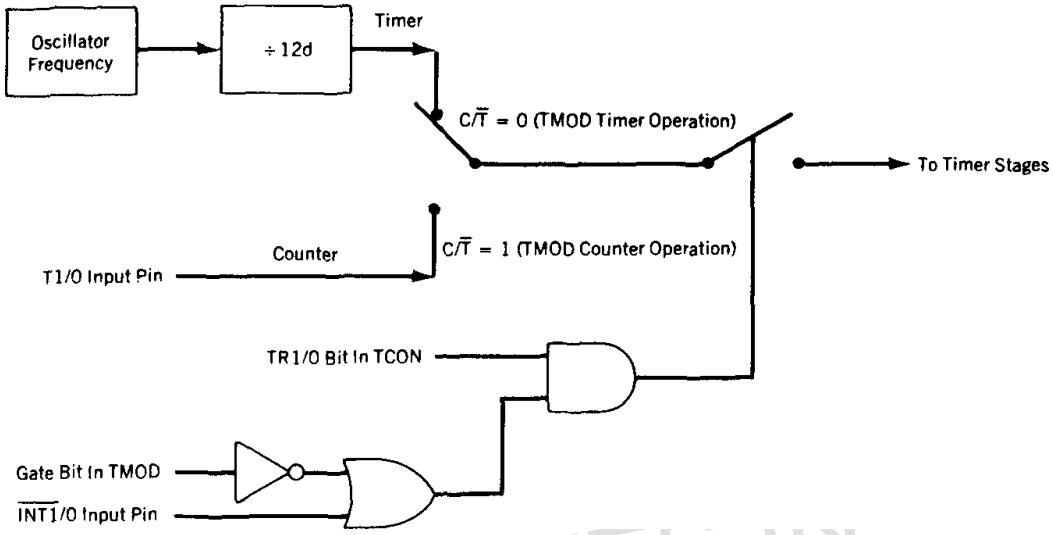
- The second 128 bytes are used to store Special Function Registers (SFR) that 8051 program can configure and control the ports, timer, interrupts, serial communication, and other tasks.
- The SFR is the upper area of addressable memory, from address 0x80 to 0xFF. This area consists of a series of memory-mapped ports and registers.
- All port input and output can therefore be performed by get and set operations on SFR port name such as P3.
- Also, different status registers are mapped into the SFR for checking the status of the 8051, and changing some operational parameters of the 8051.
- All 8051 CPU registers, I/O ports, timers and other architecture components are accessible in 8051 C through SFRs
- They are 21 SFRs and accessed in normal internal RAM (080H – 0FFH)

Downloaded from Ktunotes.in

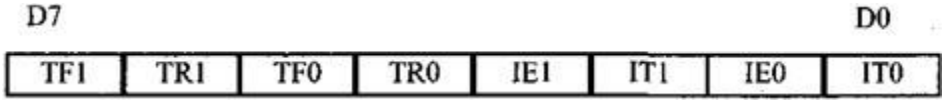
In addition to I/O ports, the most frequently used SFRs to control and configure 8051 operations are:TCON (Timer CONtrol),TMOD (Timer MODe),TH0/TH1 and TL0/TL1 (Timer's high and low bytes),SCON (Serial port CONtrol),IP (Interrupt Priority),IE (Interrupt Enable)

Counters and Timers

FIGURE 2.11 Timer/Counter Control Logic



Many microcontroller applications require the counting of external events such as frequency of a pulse train or generation of precise internal time delay between computer actions. For this purpose ,the 8051 contains two 16bit up counters T0 and T1.Each counter may be programmed to count internal clock pulses i.e. acting as timer or programmed to count external pulses as counter. The two counters are divided into two 8bit registers called timer low(TL0,TL1) and high(TH0,TH1) bytes. All counter actions are controlled by bits state in timer mode control register TMOD and timer control register TCON. Working as a "Timer", the timer is incremented by one every machine cycle. A machine cycle consists of 12 oscillator periods, so the count rate is 1/12 of the oscillator frequency. Working as a "Counter", the counter is incremented in response to a falling edge transition in the external input pins.



TCON (Timer/Counter) Register (Bit-addressable)

Downloaded from Ktunotes.in

TF1 TCON.7 Timer 1 overflow flag. Set by hardware when timer/counter 1 overflows. Cleared by hardware as the processor vectors to the interrupt service routine.

TR1 TCON.6 Timer 1 run control bit. Set/cleared by software to turn timer/counter 1 on/off.

TF0 TCON.5 Timer 0 overflow flag. Set by hardware when timer/counter 0 overflows. Cleared by hardware as the processor vectors to the service routine.

TR0 TCON.4 Timer 0 run control bit. Set/cleared by software to turn timer/counter 0 on/off.

IE1 TCON.3 External interrupt 1 edge flag. Set by CPU when the external interrupt edge (H-to-L transition) is detected. Cleared by CPU when the interrupt is processed. *Note:* This flag does not latch low-level triggered interrupts.

IT1 TCON.2 Interrupt 1 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupt.

IE0 TCON.1 External interrupt 0 edge flag. Set by CPU when external interrupt (H-to-L transition) edge is detected. Cleared by CPU when interrupt is processed.

IT0 TCON.0 Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupt.

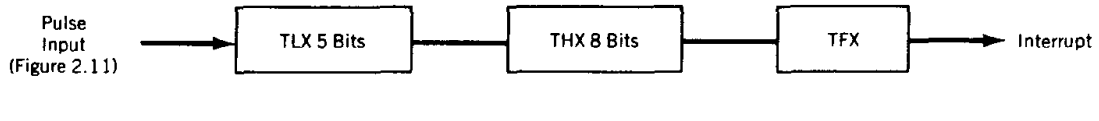
- Timer 0 and Timer 1 have four operating modes.

M1	M0	Mode Control
0	0	(Mode 0) 13 bit count mode
0	1	(Mode 1) 16 bit count mode
1	0	(Mode 2) Auto reload mode
1	1	(Mode 3) Multiple mode

KTUNOTES.IN

Downloaded from Ktunotes.in

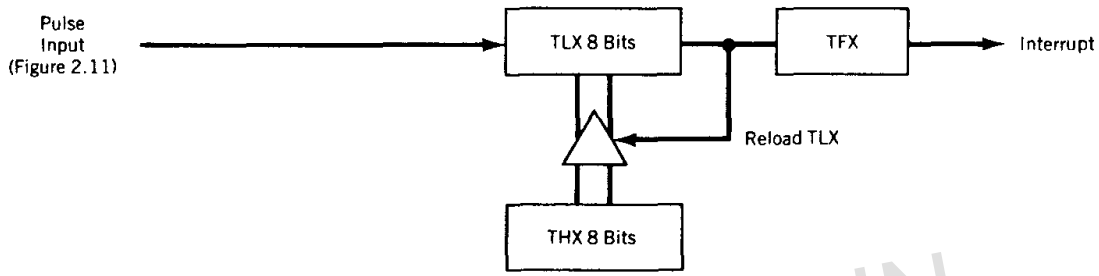
FIGURE 2.12 Timer 1 and Timer 0 Operation Modes



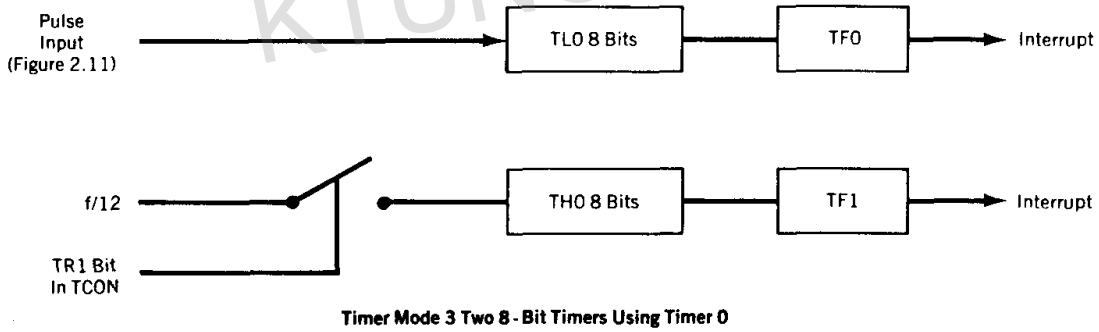
Timer Mode 0 13-Bit Timer/Counter



Timer Mode 1 16-Bit Timer/Counter



Timer Mode 2 Auto-Reload of TL from TH



Timer Mode 3 Two 8-Bit Timers Using Timer 0

KTUNOTES.IN

Interrupts

- Interrupt is an internal or external event that suspends a program and transfers the control to an event handler or ISR to handle the event.
- After the service is over, the control is back to the suspended program to resume the execution.

IE enables interrupts

- EX0/EX1: Enable external interrupt INT0/INT1

Downloaded from Ktunotes.in

- ET0/ET1: Enable Timer 0/Timer1 interrupt
- ES: Enable serial interrupt
- EA: Enable global interrupt
- Set 1 in a bit to enable its interrupt, e.g. EA =1;
- reset 0 to masks that interrupt, e.g., EA = 0;

Interrupt Flags

- The interrupt flags are set to 1 when the interrupts occur, the flag 0 indicates no interrupt.
- IE0/IE1 in TCON - For External Interrupts
- TF0/TF1 in TCON - For Timer Interrupts
- TI/RI in SCON - For Serial Interrupts

Interrupt Priority

- There are two types of interrupt priority
- User Defined Priority and Automatic Priority
- User Defined Priority
  - The IP register is used to define priority levels by users. The high priority interrupt can preempt the low priority interrupt. There are only two levels of interrupt priority.
- Automatic Priority
  - In each priority level, a priority is given in order of INT0, TF0, INT1, TF1, SI.
  - For example, if two external interrupts are set at same priority level, then INT0 has precedence over INT1.

KTUNOTES.IN

1) External Interrupts

• An external interrupt is triggered by a low level or negative edge on INT0 and INT1 which depends on the external interrupt type setting. Set up an external interrupt type by IT0 and IT1 of TCON SFR.

2) Timer/Counter Interrupts

- This unit can be used as a counter to count external pulses on P3.4 and P3.5 pins or it can be used to count the pulses produced by the crystal oscillator of the microcontroller.
- Timer Interrupt is caused by Timer 0/ Timer1 overflow.

3) Serial Interrupts

Downloaded from Ktunotes.in

- Serial communication with Universal Asynchronous Receive Transmit (UART) protocol transmits or receives the bits of a byte one after the other in a timed sequence on a single wire.
- It is used to communicate any serial port of devices and computers.
- The serial interrupt is caused by completion of a serial byte transmitting or receiving.
- The transmit data pin (TxD) is at P3.1 and the receive data pin (RxD) is at P3.0.
- All communication modes are controlled through SCON, a non bit addressable SFR. The SCON bits are defined as SM0, SM1, SM2, REN, TB8, RB8, TI, RI.

IE (Interrupt Enable Register) SFR used for interrupt control

- EX0/EX1 : Enables(1)/disables(0) the external interrupt 0
- and the external interrupt 1 on port P3.2 / P3.3
- ET0/ET1 : Enables(1)/disables(0) the Timer0 and Timer1
- interrupt via TF0/1
- ES : Enables(1)/disables(0) the serial port interrupt for sending and receiving data
- EA : Enables(1)/disables(0) all interrupts

IE : Interrupt Enable Register (Bit Addressable)

If the bit is 0, the corresponding interrupt is disabled. If the bit is 1, the corresponding interrupt is enabled.

EA	–	–	ES	ET1	EX1	ET0	EX0

EA	IE.7	Disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, interrupt source is individually enable or disabled by setting or clearing its enable bit.
-	IE.6	Not implemented, reserved for future use*.
-	IE.5	Not implemented, reserved for future use*.
ES	IE.4	Enable or disable the Serial port interrupt.
ET1	IE.3	Enable or disable the Timer 1 overflow interrupt.
EX1	IE.2	Enable or disable External interrupt 1.
ET0	IE.1	Enable or disable the Timer 0 overflow interrupt.
EX0	IE.0	Enable or disable External Interrupt 0.

IP (Interrupt Priority Register) SFR used for IP setting

- PX0/1: External interrupt 0/1 priority level

Downloaded from Ktunotes.in

- PT0/1/2: Timer0, Timer1, Timer2(8052) interrupt priority level
- PS: Serial port interrupt priority level

Depending on operation they perform, all instructions are divided in several groups:

- Arithmetic Instructions
- Branch Instructions
- Data Transfer Instructions
- Logic Instructions
- Bit-oriented Instructions

The first part of each instruction, called MNEMONIC refers to the operation an instruction performs (copy, addition, logic operation etc.). Mnemonics are abbreviations of the name of operation being executed. For example:

- INC R1 – Means: Increment register R1 (increment register R1);
- LJMP LAB5 – Means: Long Jump LAB5 (long jump to the address marked as LAB5);
- JNZ LOOP – Means: Jump if Not Zero LOOP (if the number in the accumulator is not 0, jump to the address marked as LOOP);

The other part of instruction, called OPERAND is separated from mnemonic by at least one whitespace and defines data being processed by instructions. Some of the instructions have no operand, while some of them have one, two or three. If there is more than one operand in an instruction, they are separated by a comma. For example:

- RET – return from a subroutine;
- JZ TEMP – if the number in the accumulator is not 0, jump to the address marked as TEMP;
- ADD A,R3 – add R3 and accumulator;
- CJNE A,#20,LOOP – compare accumulator with 20. If they are not equal, jump to the address marked as LOOP;

Arithmetic instructions

KTUNOTES.IN

Downloaded from Ktunotes.in

Arithmetic instructions perform several basic operations such as addition, subtraction, division, multiplication etc. After execution, the result is stored in the first operand. For example:

ADD A,R1 – The result of addition (A+R1) will be stored in the accumulator.

ARITHMETIC INSTRUCTIONS

Mnemonic	Description	Byte	Cycle
ADD A,Rn	Adds the register to the accumulator	1	1
ADD A,direct	Adds the direct byte to the accumulator	2	2
ADD A,@Ri	Adds the indirect RAM to the accumulator	1	2
ADD A,#data	Adds the immediate data to the accumulator	2	2
ADDC A,Rn	Adds the register to the accumulator with a carry flag	1	1
ADDC A,direct	Adds the direct byte to the accumulator with a carry flag	2	2
ADDC A,@Ri	Adds the indirect RAM to the accumulator with a carry flag	1	2
ADDC A,#data	Adds the immediate data to the accumulator with a carry flag	2	2
SUBB A,Rn	Subtracts the register from the accumulator with a borrow	1	1
SUBB A,direct	Subtracts the direct byte from the accumulator with a borrow	2	2
SUBB A,@Ri	Subtracts the indirect RAM from the accumulator with a borrow	1	2
SUBB A,#data	Subtracts the immediate data from the accumulator with a borrow	2	2
INC A	Increments the accumulator by 1	1	1
INC Rn	Increments the register by 1	1	2
INC Rx	Increments the direct byte by 1	2	3
INC @Ri	Increments the indirect RAM by 1	1	3
DEC A	Decrements the accumulator by 1	1	1
DEC Rn	Decrements the register by 1	1	1
DEC Rx	Decrements the direct byte by 1	1	2
DEC @Ri	Decrements the indirect RAM by 1	2	3
INC DPTR	Increments the Data Pointer by 1	1	3
MUL AB	Multiplies A and B	1	5

Downloaded from Ktunotes.in

DIV AB	Divides A by B	1	5
DA A	Decimal adjustment of the accumulator according to BCD code	1	1

Branch Instructions

There are two kinds of branch instructions:

- Unconditional jump instructions: upon their execution a jump to a new location from where the program continues execution is executed.
- Conditional jump instructions: a jump to a new program location is executed only if a specified condition is met. Otherwise, the program normally proceeds with the next instruction.

BRANCH INSTRUCTIONS

Mnemonic	Description	Byte	Cycle
ACALL addr11	Absolute subroutine call	2	6
LCALL addr16	Long subroutine call	3	6
RET	Returns from subroutine	1	4
RETI	Returns from interrupt subroutine	1	4
AJMP addr11	Absolute jump	2	3
LJMP addr16	Long jump	3	4
SJMP rel	Short jump (from –128 to +127 locations relative to the following instruction)	2	3
JC rel	Jump if carry flag is set. Short jump.	2	3
JNC rel	Jump if carry flag is not set. Short jump.	2	3
JB bit,rel	Jump if direct bit is set. Short jump.	3	4
JBC bit,rel	Jump if direct bit is set and clears bit. Short jump.	3	4
JMP @A+DPTR	Jump indirect relative to the DPTR	1	2
JZ rel	Jump if the accumulator is zero. Short jump.	2	3
JNZ rel	Jump if the accumulator is not zero. Short jump.	2	3

KTUNOTES.IN

Downloaded from Ktunotes.in

CJNE A,direct,rel	Compares direct byte to the accumulator and jumps if not equal. Short jump.	3	4
CJNE A,#data,rel	Compares immediate data to the accumulator and jumps if not equal. Short jump.	3	4
CJNE Rn,#data,rel	Compares immediate data to the register and jumps if not equal. Short jump.	3	4
CJNE @Ri,#data,rel	Compares immediate data to indirect register and jumps if not equal. Short jump.	3	4
DJNZ Rn,rel	Decrements register and jumps if not 0. Short jump.	2	3
DJNZ Rx,rel	Decrements direct byte and jump if not 0. Short jump.	3	4
NOP	No operation	1	1

Data Transfer Instructions

Data transfer instructions move the content of one register to another. The register the content of which is moved remains unchanged. If they have the suffix "X" (MOVX), the data is exchanged with external memory.

KTUNOTES.IN

DATA TRANSFER INSTRUCTIONS

Mnemonic	Description	Byte	Cycle
MOV A,Rn	Moves the register to the accumulator	1	1
MOV A,direct	Moves the direct byte to the accumulator	2	2
MOV A,@Ri	Moves the indirect RAM to the accumulator	1	2
MOV A,#data	Moves the immediate data to the accumulator	2	2
MOV Rn,A	Moves the accumulator to the register	1	2
MOV Rn,direct	Moves the direct byte to the register	2	4
MOV Rn,#data	Moves the immediate data to the register	2	2
MOV direct,A	Moves the accumulator to the direct byte	2	3
MOV direct,Rn	Moves the register to the direct byte	2	3
MOV direct,direct	Moves the direct byte to the direct byte	3	4
MOV direct,@Ri	Moves the indirect RAM to the direct byte	2	4

Downloaded from Ktunotes.in

MOV direct,#data	Moves the immediate data to the direct byte	3	3
MOV @Ri,A	Moves the accumulator to the indirect RAM	1	3
MOV @Ri,direct	Moves the direct byte to the indirect RAM	2	5
MOV @Ri,#data	Moves the immediate data to the indirect RAM	2	3
MOV DPTR,#data	Moves a 16-bit data to the data pointer	3	3
MOVC A,@A+DPTR	Moves the code byte relative to the DPTR to the accumulator (address=A+DPTR)	1	3
MOVC A,@A+PC	Moves the code byte relative to the PC to the accumulator (address=A+PC)	1	3
MOVX A,@Ri	Moves the external RAM (8-bit address) to the accumulator	1	3-10
MOVX A,@DPTR	Moves the external RAM (16-bit address) to the accumulator	1	3-10
MOVX @Ri,A	Moves the accumulator to the external RAM (8-bit address)	1	4-11
MOVX @DPTR,A	Moves the accumulator to the external RAM (16-bit address)	1	4-11
PUSH direct	Pushes the direct byte onto the stack	2	4
POP direct	Pops the direct byte from the stack/td>	2	3
XCH A,Rn	Exchanges the register with the accumulator	1	2
XCH A,direct	Exchanges the direct byte with the accumulator	2	3
XCH A,@Ri	Exchanges the indirect RAM with the accumulator	1	3
XCHD A,@Ri	Exchanges the low-order nibble indirect RAM with the accumulator	1	3

KTUNOTES.IN

Logic Instructions

Logic instructions perform logic operations upon corresponding bits of two registers. After execution, the result is stored in the first operand.

Mnemonic	Description	Byte	Cycle
ANL A,Rn	AND register to accumulator	1	1
ANL A,direct	AND direct byte to accumulator	2	2
ANL A,@Ri	AND indirect RAM to accumulator	1	2

Downloaded from Ktunotes.in

ANL A,#data	AND immediate data to accumulator	2	2
ANL direct,A	AND accumulator to direct byte	2	3
ANL direct,#data	AND immediae data to direct register	3	4
ORL A,Rn	OR register to accumulator	1	1
ORL A,direct	OR direct byte to accumulator	2	2
ORL A,@Ri	OR indirect RAM to accumulator	1	2
ORL direct,A	OR accumulator to direct byte	2	3
ORL direct,#data	OR immediate data to direct byte	3	4
XRL A,Rn	Exclusive OR register to accumulator	1	1
XRL A,direct	Exclusive OR direct byte to accumulator	2	2
XRL A,@Ri	Exclusive OR indirect RAM to accumulator	1	2
XRL A,#data	Exclusive OR immediate data to accumulator	2	2
XRL direct,A	Exclusive OR accumulator to direct byte	2	3
XORL direct,#data	Exclusive OR immediate data to direct byte	3	4
CLR A	Clears the accumulator	1	1
CPL A	Complements the accumulator (1=0, 0=1)	1	1
SWAP A	Swaps nibbles within the accumulator	1	1
RL A	Rotates bits in the accumulator left	1	1
RLC A	Rotates bits in the accumulator left through carry	1	1
RR A	Rotates bits in the accumulator right	1	1
RRC A	Rotates bits in the accumulator right through carry	1	1

Bit-oriented Instructions

Similar to logic instructions, bit-oriented instructions perform logic operations. The difference is that these are performed upon single bits.

Mnemonic	Description	Byte	Cycle

Downloaded from Ktunotes.in

CLR C	Clears the carry flag	1	1
CLR bit	Clears the direct bit	2	3
SETB C	Sets the carry flag	1	1
SETB bit	Sets the direct bit	2	3
CPL C	Complements the carry flag	1	1
CPL bit	Complements the direct bit	2	3
ANL C,bit	AND direct bit to the carry flag	2	2
ANL C,/bit	AND complements of direct bit to the carry flag	2	2
ORL C,bit	OR direct bit to the carry flag	2	2
ORL C,/bit	OR complements of direct bit to the carry flag	2	2
MOV C,bit	Moves the direct bit to the carry flag	2	2
MOV bit,C	Moves the carry flag to the direct bit	2	3

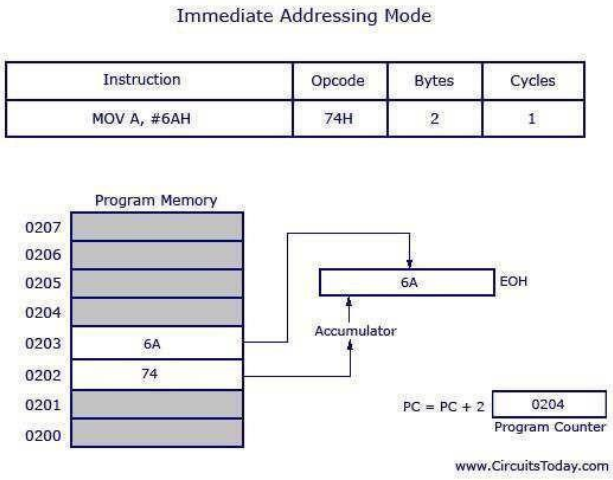
ADDRESSING MODE IN 8051.

Immediate Addressing Mode

MOV A, #6AH

In general we can write MOV A, #data

This addressing mode is named as "immediate" because it transfers an 8-bit data immediately to the accumulator (destination operand).



The picture above describes the above instruction and its execution. The opcode for MOV A, # data is 74H. The opcode is saved in program memory at 0202 address. The data 6AH is saved in program memory 0203..

Downloaded from Ktunotes.in

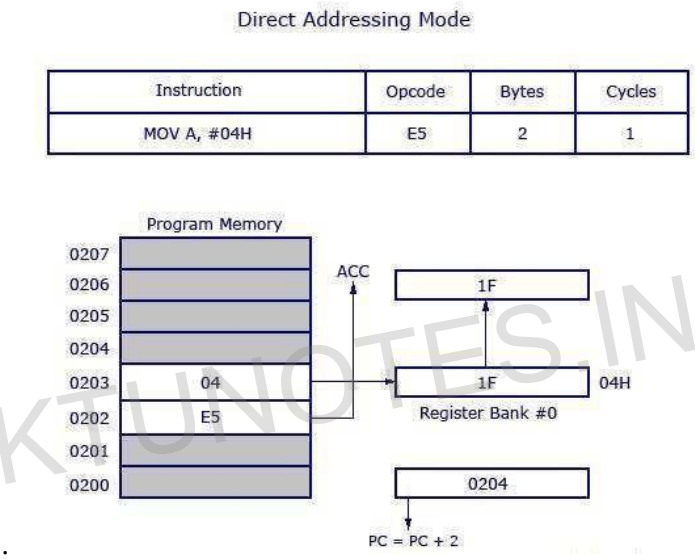
- The *'#'* symbol before 6AH indicates that operand is a data (8 bit). If *'#'* is not present then the hexadecimal number would be taken as address.

Direct Addressing Mode

This is another way of addressing an operand. Here the address of the data (source data) is given as operand. Lets take an example.

MOV A, 04H

Here 04H is the address of register 4 of register bank#0. When this instruction is executed, what ever data is stored in register 04H is moved to accumulator. In the picture below we can see, register 04H holds the data 1FH. So the data 1FH is moved to accumulator.



KTUNOTES.IN

As shown in picture above this is a 2 byte instruction which requires 1 cycle to complete. Program counter will increment by 2 and stand in 0204. The opcode for instruction ***MOV A, address*** is E5H. When the instruction at 0202 is executed (E5H), accumulator is made active and ready to receive data. Then program control goes to next address that is 0203 and look up the address of the location (04H) where the source data (to be transferred to accumulator) is located. At 04H the control finds the data 1F and transfers it to accumulator and hence the execution is completed.

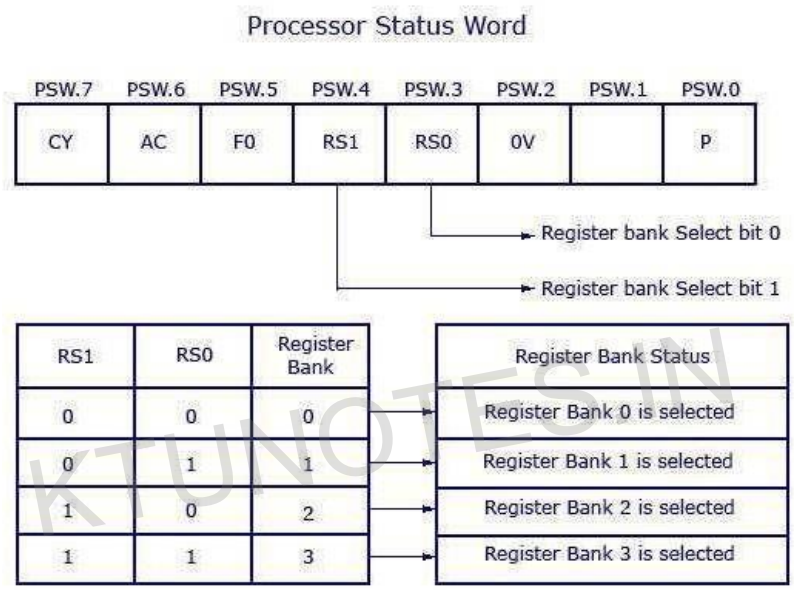
Register Direct Addressing Mode

In this addressing mode the register name directly (as source operand). An example is shown below.

MOV A, R4

Downloaded from Ktunotes.in

At a time registers can take value from R0,R1…to R7. There are 4 register banks named 0,1,2 and 3. Each bank has 8 registers named from R0 to R7. At a time only one register bank can be selected. Selection of register bank is made possible through a Special Function Register (SFR) named Processor Status Word (PSW). PSW is an 8 bit SFR where each bit can be programmed. Bits are designated from PSW.0 to PSW.7 Register banks are selected using PSW.3 and PSW.4 These two bits are known as register bank select bits as they are used to select register banks. A picture below shows the PSW register and the Register Bank Select bits with status.



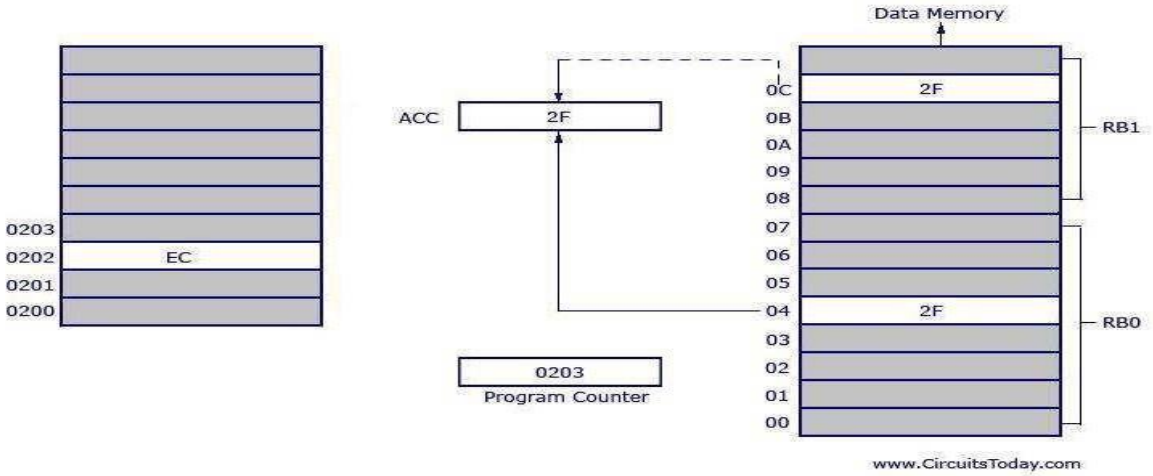
KTUNOTES.IN

So in register direct addressing mode, data is transferred to accumulator from the register (based on which register bank is selected).

Downloaded from Ktunotes.in

Register Direct Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, R4	ECH	1	1



.

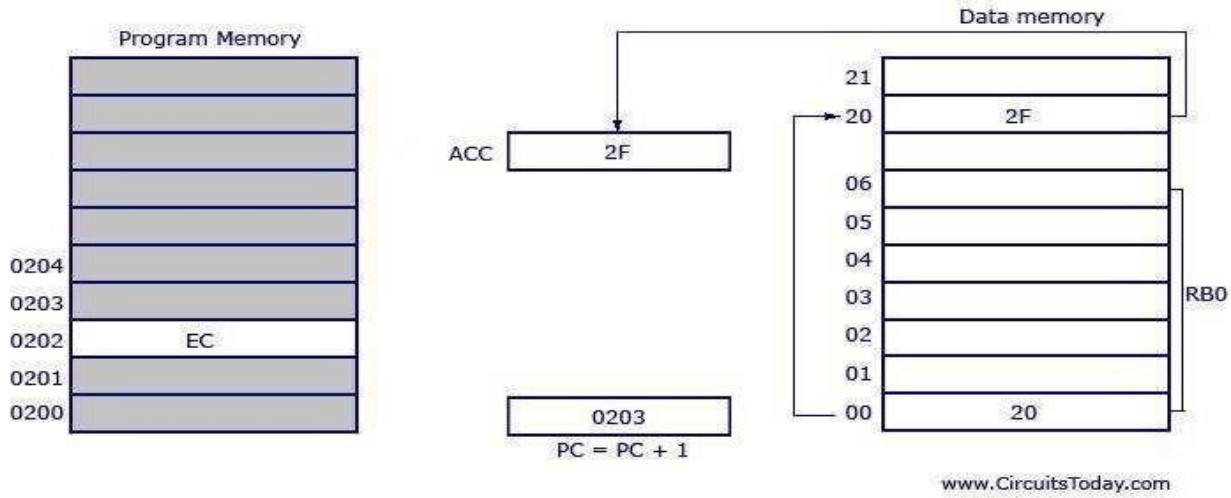
Register Indirect Addressing Mode

MOV A, @R0

KTUNOTES.IN

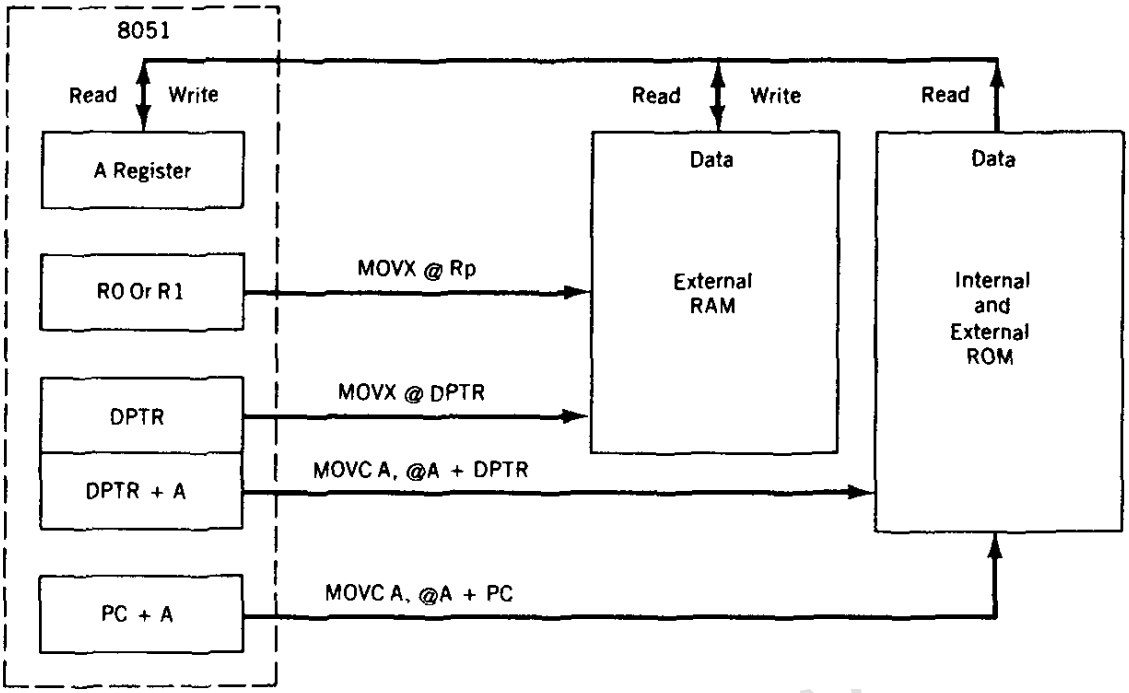
Register Indirect Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOV A, @ R0	E6H	1	1



Downloaded from Ktunotes.in

FIGURE 3.2 External Addressing using MOVX and MOVC



MOVC A, @A+DPTR and MOVC A, @A+PC where DPTR is data pointer and PC is program counter (both are 16 bit registers).

KTUNOTES.IN

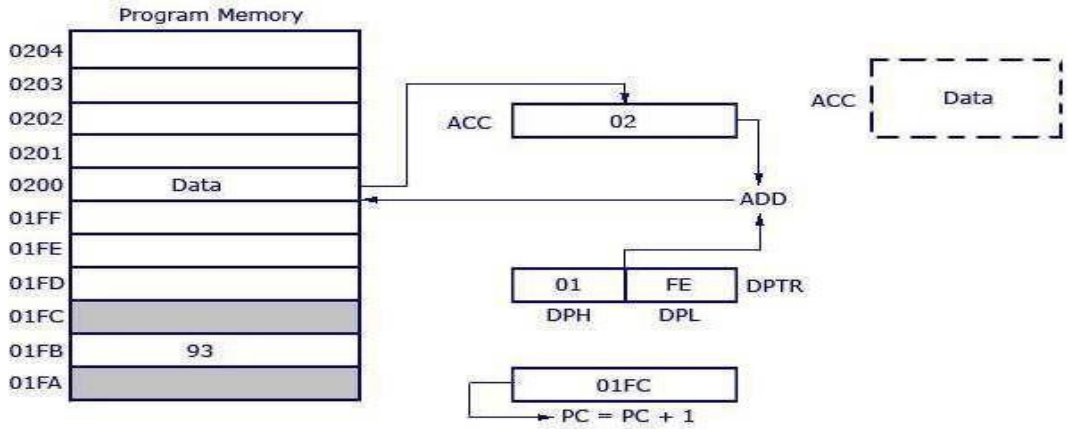
MOVC A, @A+DPTR

The opcode for the instruction is 93H. DPTR holds the value 01FE, where 01 is located in DPH (higher 8 bits) and FE is located in DPL (lower 8 bits). Accumulator now has the value 02H. A 16 bit addition is performed and now 01FE H+02 H results in 0200 H. whatever data is in 0200 H will get transferred to accumulator. The previous value inside accumulator (02H) will get replaced with new data from 0200H. It is a 1 byte instruction. The other example MOVC A, @A+PC works the same way as above example. The only difference is, instead of adding DPTR with accumulator, here data inside program counter (PC) is added with accumulator to obtain the target address.

Downloaded from Ktunotes.in

Indexed Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOVC A,@A +DPTR	93H	1	2



8051 TIMERS AND PROGRAMMING

The 8051 has two timers: Timer 0 and Timer 1. They can be used either as timers or as event counters. In this section we first discuss the timers' registers and then show how to program the timers to generate time delays.

KTUNOTES.IN

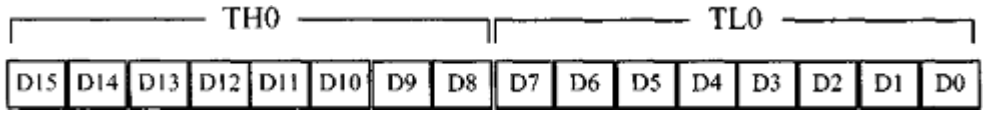
Basic registers of the timer

Both Timer 0 and Timer 1 are 16 bits wide. Since the 8051 has an 8-bit architecture, each 16-bit timer is accessed as two separate registers of low byte and high byte. Each timer is discussed separately.

Timer 0 registers

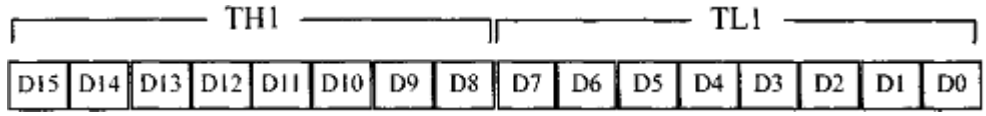
The 16-bit register of Timer 0 is accessed as low byte and high byte. The low byte register is called TLO (Timer 0 low byte) and the high byte register is referred to as THO (Timer 0 high byte). These registers can be accessed like any other register, such as A, B, RO, Rl, R2, etc. For example, the instruction "MOV TLO , #4FH" moves the value 4FH into TLO, the low byte of Timer 0. These registers can also be read like any other register. For example, "MOV R5 , THO" saves THO (high byte of Timer 0) in R5.

Downloaded from Ktunotes.in



Timer 0 Registers

Timer 1 registers

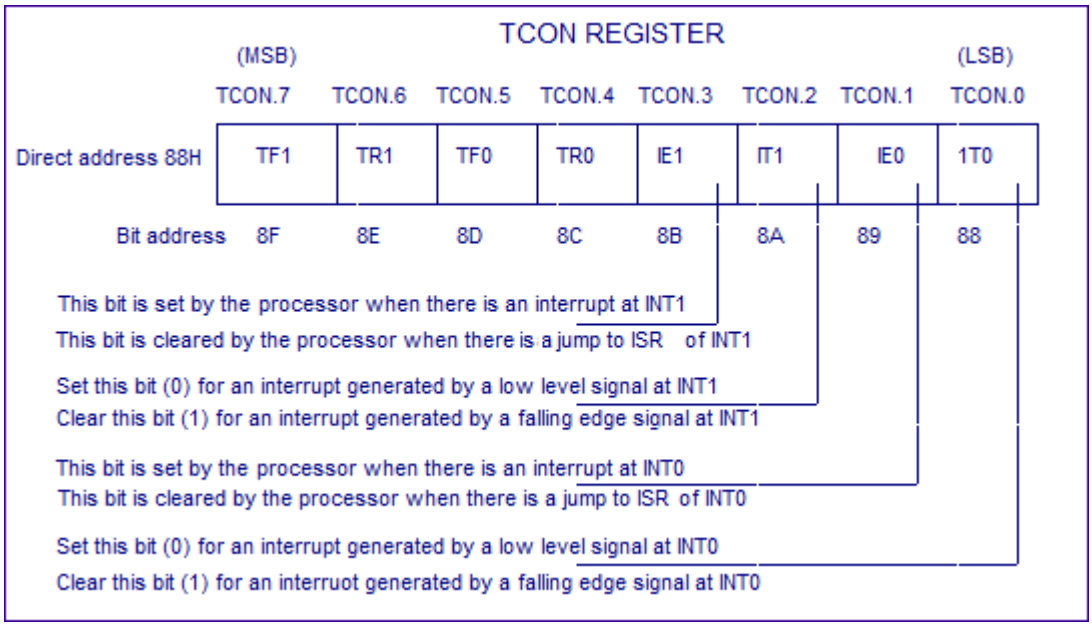


Timer 1 Registers

Timer I is also 16 bits, and its 16-bit register is split into two bytes, referred to as TLl (Timer I low byte) and TH1 (Timer 1 high byte). These registers are accessible in the same way as the registers of timer 0

TCON (Timer/Counter Control Register) control

- TF0/TF1: Timer0/1 overflow flag is set when the timer/counter overflows, reset by program
- TR0/TR1: Timer0/1 run control bit is set to start, reset to stop the timer0/1
- IE0/IE1: **E**xternal **I**nterrupt 0/1 edge detected flag: 1 is set when a falling edge interrupt on the external port 0/1



Downloaded from Ktunotes.in

TMOD (timer mode) register

Both timers 0 and 1 use the same register called TMOD, to set the various timer operation modes. TMOD is an 8-bit register in which the lower 4 bits are set aside for Timer 0 and the upper 4 bits for Timer 1. In each case, the lower 2 bits are used to set the timer mode and the upper 2 bits to specify the operation.

TMOD Register

(MSB) (LSB)

GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer 1				Timer 0			

GATE Gating control when set. The timer/counter is enabled only while the INTx pin is high and the TRx control pin is set. When cleared, the timer is enabled whenever the TRx control bit is set.

C/T Timer or counter selected cleared for timer operation (input from internal system clock). Set for counter operation (input from Tx input pin).

M1 Mode bit 1

M0 Mode bit 0

M1	M0	Mode	Operating Mode
0	0	0	13-bit timer mode 8-bit timer/counter THx with TLx as 5-bit prescaler
0	1	1	16-bit timer mode 16-bit timer/counters THx and TLx are cascaded; there is no prescaler
1	0	2	8-bit auto reload 8-bit auto reload timer/counter; THx holds a value that is to be reloaded into TLx each time it overflows.
1	1	3	Split timer mode

KTUNOTES.IN

M1, MO

MO and Ml select the timer mode. As shown in Figure 9-3, there are three modes: 0, 1, and 2. Mode 0 is a 13-bit timer, mode 1 is a 16-bit timer, and mode 2 is an 8-bit timer. We will concentrate on modes 1 and 2 since they are the ones used most widely. We will soon describe the characteristics of these modes, after describing the rest of the TMOD register.

C/T (clock/timer)

This bit in the TMOD register is used to decide whether the timer is used as a delay generator or an event counter. If C/T = 0, it is used as a timer for time delay generation. The clock source for the

Downloaded from Ktunotes.in

time delay is the crystal frequency of the 8051. This section is concerned with this choice. The timer's use as an event counter is discussed in the next section.

GATE

Every timer has a means of starting and stopping. Some timers do this by software, some by hardware, and some have both software and hardware controls. The timers in the 8051 have both. The start and stop of the timer are controlled by way of software by the TR (timer start) bits TRO and TR1. This is achieved by the instructions "SETB TR1″ and "CLR TR1″ for Timer 1, and "SETB TRO" and "CLR TRO" for Timer 0. The SETB instruction starts it, and it is stopped by the CLR instruction. These instructions start and stop the timers as long as GATE = 0 in the TMOD register. The hardware way of starting and stopping the timer by an external source is achieved by making GATE = 1 in the TMOD register GATE = 0, meaning that no external hardware is needed to start and stop the timers. In using software to start and stop the timer where GATE = 0. all we need are the instructions "SETB TRx" and "CLR TRx".

KTUNOTES.IN

Downloaded from Ktunotes.in