Microprocessors and Assembly Language Programming (3-1-3)

Evaluation:

	Theory	Practical	Total
Sessional	30	20	50
Final	50	-	50
Total	80	20	100

Course Objectives:
The purpose of the course is to provide the basics fundamentals and operations of microprocessor. It provides knowledge to program microprocessor using assembly language and design microprocessor based systems and interfaces.

Course Contents:

1. Introduction to microprocessor **(3 hrs)**
- 1.1 Brief description: Microprocessor, Microcontroller, Microcomputer
- 1.2 Application of microprocessor
- 1.3 Evolution of microprocessor: INTEL series

2. Architectural Details and Instruction set of 8085 and 8086 microprocessor (10 hrs)
- 2.1 Internal architecture and description
- 2.2 Instruction set
- 2.3 Addressing modes
- 2.4 Instruction cycle, Machine cycle, t-states
- 2.5 Timing Diagram

3. Assembly Language Programming **(12 hrs)**
- 3.1 Introduction
- 3.2 Format of an assembly language instruction
- 3.3 Basic assembly language programs of 8085
- 3.4 ALP development tools:Editor,Assembler,Linker,Debuger,Locator.Emulator
- 3.5 Macro Assembler and Assembler Directives
- 3.6 8086 Assembly Language Programs in MASM/TASM
- 3.7 Modular Programming
  - 3.7.1 Linking and Relocation
  - 3.7.2 Stacks Procedures
  - 3.7.3 Macros Program Design
  - 3.7.4 String Manipulation

4. Bus Structure and Memory Devices **(4 hrs)**
- 4.1 Introduction: Data/Address/Control bus
- 4.2 Synchronous and Asynchronous bus
- 4.3 Memory Classification
- 4.4 Memory Interfacing and Addressing Decoding

5. Interrupt (6 hrs)

5.1 Introduction
5.2 Interrupt Sources: Hardware, Software, Processor
5.3 Interrupt Types: Maskable, Non-Maskable Interrupt
5.4 8086 Interrupts
5.5 Interrupt Vector Table
5.6 Vector Chain and Polled Interrupt
5.7 Interrupt Processing

6. Input / Output Interfaces (10 hrs)

6.1 Serial I/O standards: 8251A USART
6.2 8259A Programmable Interrupt Controller(PIC)
6.3 8255A Programmable Peripheral Interface(PPI)
6.4 8254 Programmable Interrupt Timer(PIT) and its application
6.5 DMA and DMA controller

Laboratory:

1. A minimum of 10 laboratory exercises shall be done with the use of SDK-85/SDK-86 or equivalent microprocessor trainer kit and Simulators.
2. Numerous assembly language programming exercises are to be done both with the help of microprocessor trainer kit and Macro-Assemblers in PC.

Text Books:

1. Liu. Yu-cheng and Gibson Glenn A., Microprocessor Systems: The 8080 8088 family Architecture. Programming and Design.PHI, 1998. ISBN: 81-203-0409-8
2. Brey. Barry B..Intel Microprocessors.PHI. 1998. ISBN:

References:

1. Antonakos. J. L. An Introduction to the Intel family of microprocessors, 3rd ed. Pearson Education Asia. ISBN: 81-7808-312-4
2. Triebel, Walter A. and Singh Avvbtar, The 8088 and 8086 microprocessors: Programming Interfacing, Software, Hardware, and Applications PHI. 1998. ISBN
3. L.A Leventhal, Introduction to Microprocessor software, Hardware & Programming Prentice Hall of India. Pvt. Ltd., 1995.
4. A.P. Malvino, An Introduction to Microcomputers. Prentice Hall of India. Pvt. Ltd 1995
5. P.K. Ghosh, P.R. Sridhar, 0000 to 8085;Introduction to Microprocessor for Engineers and Scientists. Prentice Hall of India Pvt. Ltd 1997
6. Rajaraman, V. and Radhakrishnan T., Essentials of Assembly Language Programming for the IBM PC, PHI, 1998. ISBN: 81-203-1425-5

Microprocessor and Assembly Language Programming

* Microprocessor → CPU of a computer built into a single IC chip.

* Microcomputer → A computer having microprocessor as its CPU.

* Microcontroller → entire computer built into a single IC chip.

Generation of Computers

1st Generation → Vacuum tubes
2nd Generation → Transistors
3rd Generation → IC chip / LSI (Integrated Circuit)
4th Generation → µP / VLSI / ULSI
5th Generation → AI / Bio-chips

IC Fabrication Technology

SSI → Small Scale Integration → Upto 10 gates/chip
MSI → Medium Scale Integration → 10 - 100 gates/chi
LSI → Large Scale " → 1000 - ~~100~~ 1000 "
VLSI → Very Large Scale " → ~~100,000~~ 1000 - 1000,000 "
ULSI → Ultra Large Scale "→ > 100000 gates/chip

History of Microprocessor

→ INTEL Series

1971	4004 → First microprocessor (4-bit)
1972	8008 → 8-bit
1974	8080 → 8-bit
1976	8085 → 8-bit
1978	8086 → 16-bit 8088

80286

80386

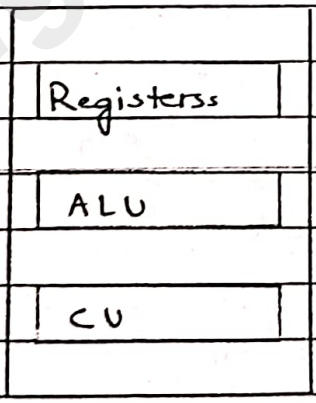
Pentium I/II/III/IV

I/O Interfacing Devices

Introduction to microprocessor & microcomputer

→ A microprocessor is a multipurpose, programmable, clock driven, register based electronic device that reads binary instructions from a storage device called memory, accepts binary data as input and processes data according to those instructions & provides result as output.

In short, CPU of a digital computer built into a single IC chip is called microprocessor.



The clock speed of MP ranges from MHz to GHz.

A microcomputer on the other hand is the digital computer having microprocessor as its CPU.

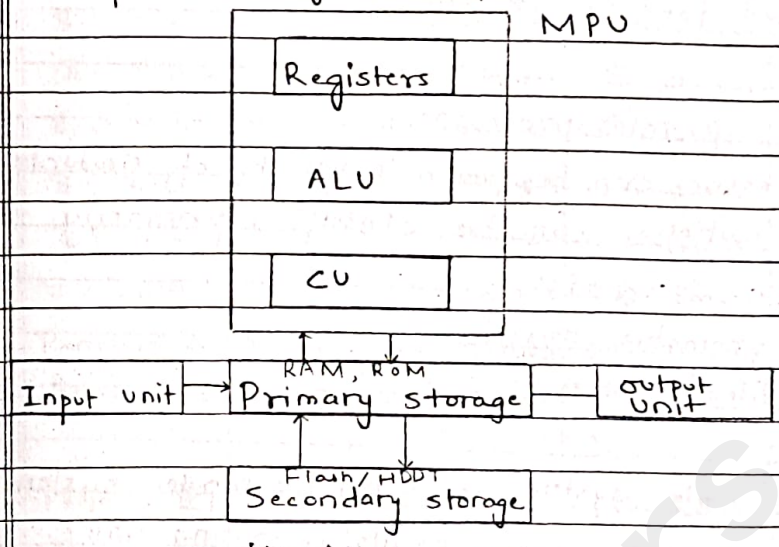


fig: Microcomputer

The advancement in the development of semiconductor devices (IC's) led to the invention of microcontroller.

A microcontroller is a semiconductor device which is fabricated to include mpu, memory, I/O and other peripherals (USB pendrive) within the same IC-package.

In short, entire digital computer built into a single IC-chip is known as microcontroller.

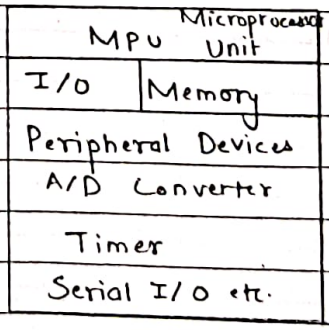


fig: Microcontroller

→ clock speed is in the range of MHz.
→ It is used for specific purpose.

Application of microprocessor
→ Microprocessor can be found in variety of products
→ The applications can be classified primarily into two categories.
① Re-programable system
② Embedded system

Reprogrammable system is a microcomputer system that uses microprocessor as its computing unit.

In embedded system, the MP is a part of final product and is not available for reprogramming to end-~~order~~ user. eg: Traffic light control system

Summary of Applications
1. Microcontroller
2. Industrial Controls
3. Robotics
4. Medical Equipment (CT scanner, etc)
5. Washing Machine
6. Traffic light control, etc.

Evolution of microprocessor (Summary Table)

	Processor	Year	Clock Speed	Databus	Address bus
I	4004	1971	108 KHz	4-bit	10-bit
N	8008	1972	200 KHz	8-bit	14-bit
T	8080	1974	2 MHz	8-bit	16-bit
E	8085	1976	5 MHz	8-bit	16-bit
L	8086	1978	5 MHz	16-bit	20-bit
	8088	1979	5 MHz	8-bit	20-bit
S	80386	1985	16 MHz	32-bit	32-bit
E	Pentium I	1993	60 MHz	32/64-bit	32-bit
R	Pentium II	1997	233 MHz	64 bit	36-bit
I	:				
E	Pentium IV	2000	1.4GHz	64 bit	64-bit
S	Xenon	2001	1.7GHz		
	Pentium M	2003	1.7GHz		
	Dual Core	2005	2.8 GHz	32bit	
	Core2 Duo	2006	2.66 GHz	32bit	
	~~Atom~~	~~2008~~	~~1.86 GHz~~		
	2nd gen core	2010	3.8 GHz		
	3rd gen core	2012	2.9 GHz		
	I_7	2008			
	I_5	2009			
	I_3	2010			
	I_9				

Bus Organisation / structure of a microprocessor

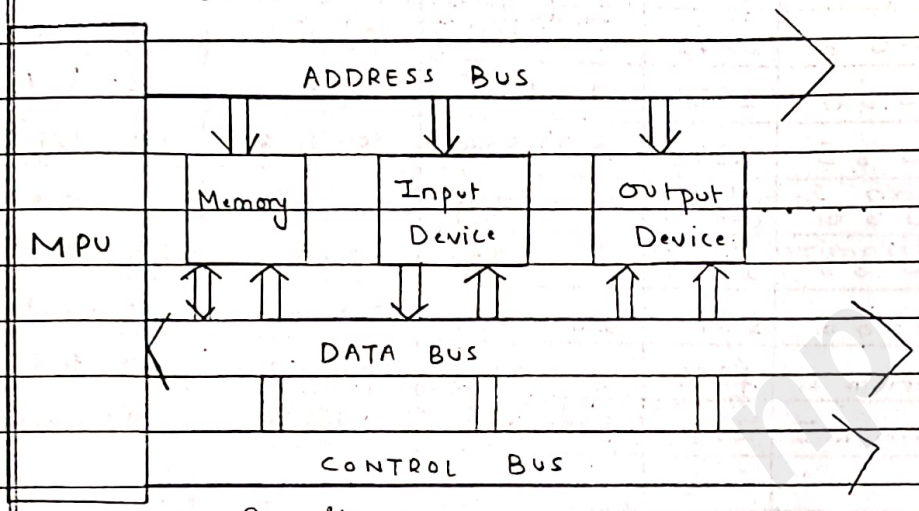


fig- Bus Structure of a Microprocessor

→ A bus is a path or group of wires through which data & information (bits) travels.
→ It is an electronic path through which binary bits of data flow takes place.
→ It consists of:

1. Address Bus
→ Address bit travels (that carries address bits)
→ Multiple of 4
→ Total addressable memory for 'n' bits is 2^n.
For eg: 8085 µP has 16-bit address bus.
Addressable Memory $= 2^{16}$
$= 65536$ bytes
$= 64$ KB
→ unidirectional

2. Data Bus
→ that carries data.
→ indicates the data bit capacity of µP.
→ Bidirectional
→ It maybe 4-bit, 8-bit, 16-bit, 32-bit & 64-bit

3 Control Bus

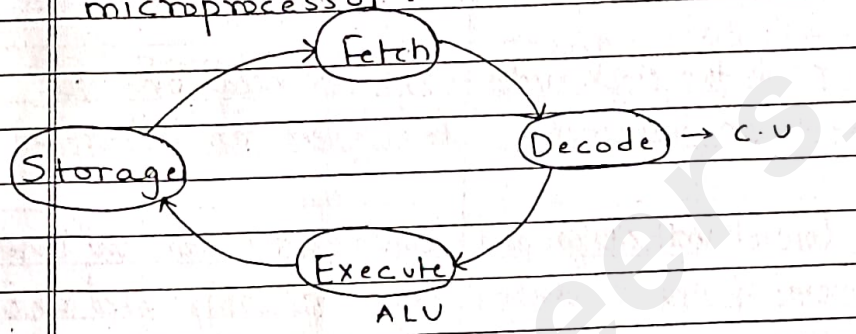
→ That carries control signals.

→ The width of a bus depend upon the types of control signals used.

→ Control signals maybe memory read, memory write, I/O read, I/O write.

Concept of Fetch, Decode & Execute :

→ It constitutes a basic instruction cycle of a microprocessor.



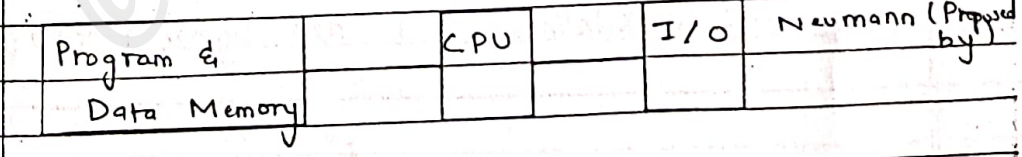
Computer Architecture

Von-Neumann Architecture

Hardvard Architecture

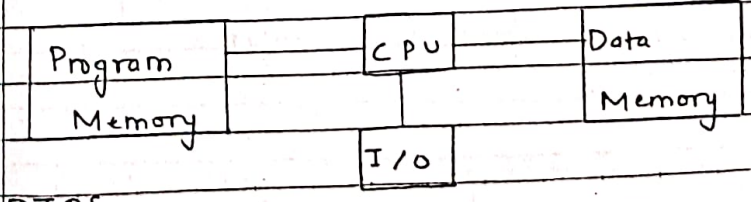
* Von-Neumann Architecture

John Von-Neumann (Proposed by)



* Hardvard Architecture

~~Hardvard~~ Howard Hathaway Aiken



BIOS
→ Bootstrap loading

Differentiate between Von-Nieuman & Hardvard Architecture

Von-Nieuman Architecture	Hardvard Architecture.
→ It is a theoretical design based on the stored-program computer concept.	→ It is a modern computer architecture based on the Harvard Mark I relay-based comp. model
→ It uses same physical memory address for instructions and data.	→ It uses separate memory address for instructions and data.
→ Processor needs two clock cycles to execute an instruction.	→ Processor needs one cycle to complete an instruction.
→ Simpler control unit design and development of one is cheaper and faster.	→ Control unit for two buses is more complicated which adds to the development cost.
→ Data transfers and instruction fetches cannot be performed simultaneously.	→ Data transfers and instruction fetches can be performed at the same time.
→ Used in personal computers, laptops and workstations	→ Used in micro controllers and signal processing.

Qn: Explain the function of these signals

CHAPTER 2 : Introduction to 8085 µP.

Features :

1. Introduced in 1976.
2. 40-pin DIP (Dual Inline Package)
3. 8-bit µP
   - Data bus width = 8-bit
   - Address bus " = 16-bit
   - Total Addressable memory = $2^{16} = 65536$ bytes = 64 KB
4. It uses single +5V power supply.
5. Operating frequency (max^m) = 5 MHz.
6. Upward compatible with 8080A
7. Supports large no. of instructions.

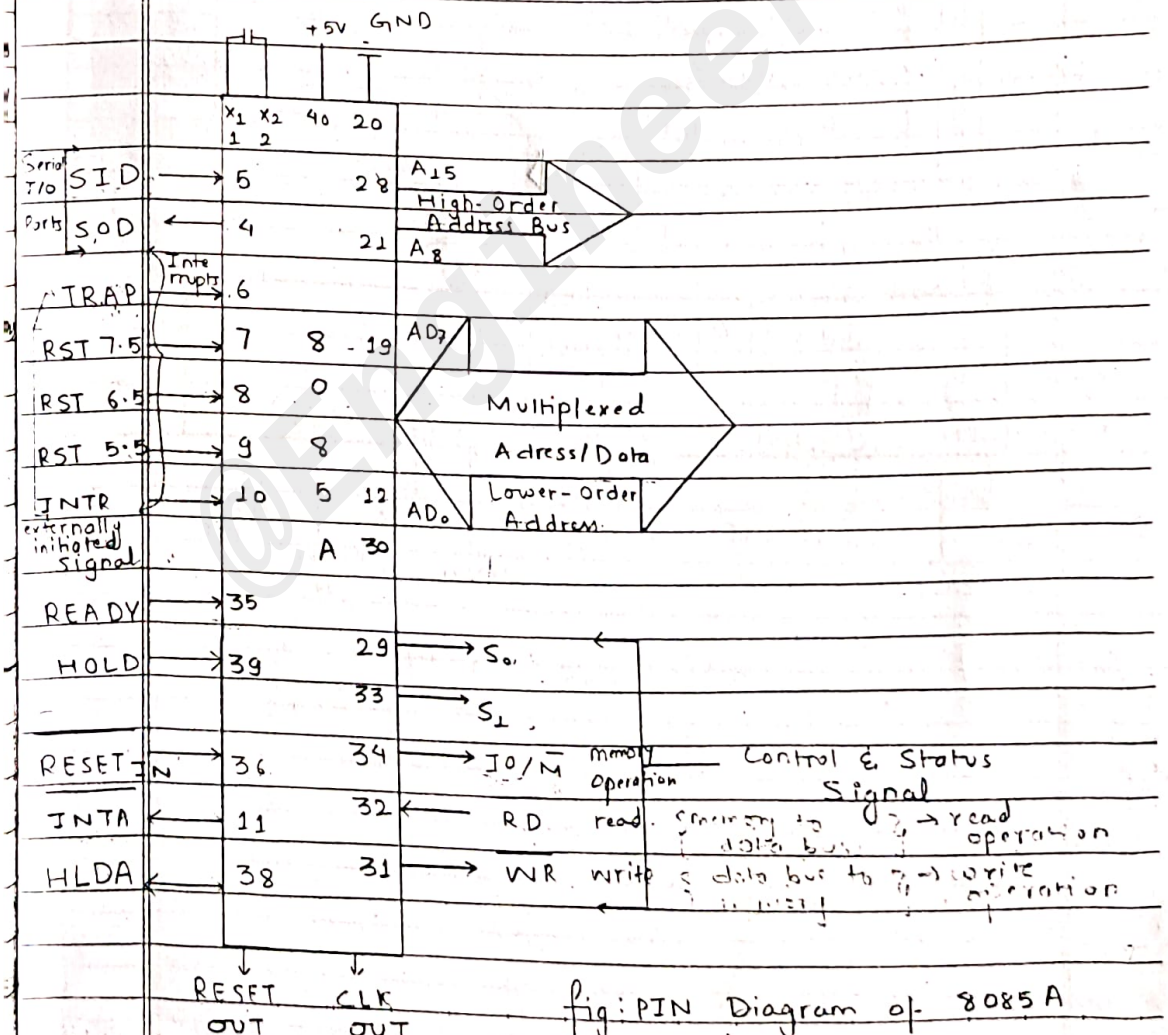
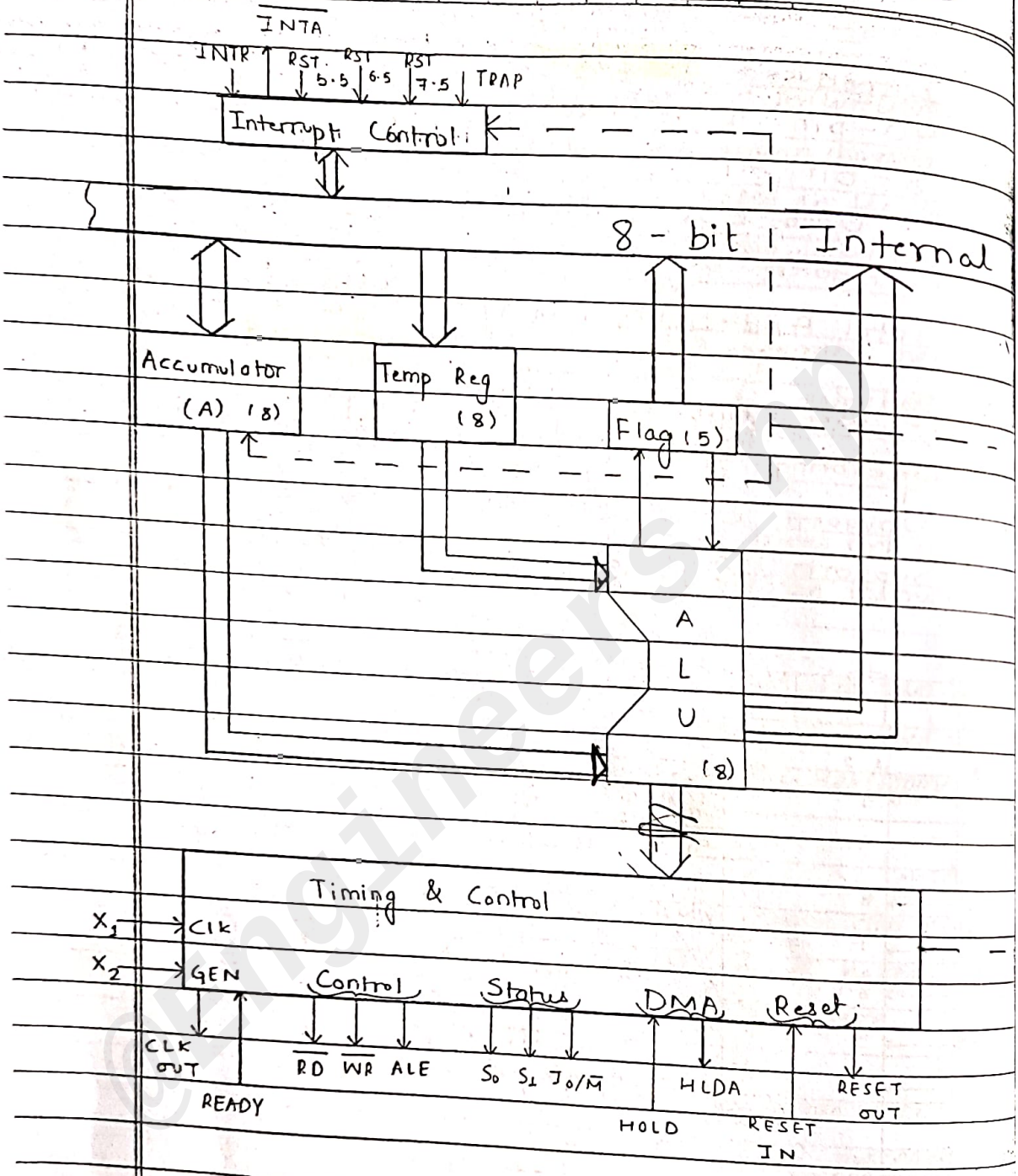


fig: PIN Diagram of 8085A


INTA
INTR
RST 5.5
RST 6.5
RST 7.5
TRAP
Interrupt Control
8 - bit Internal
Accumulator (A) (8)
Temp Reg (8)
Flag (5)
A
L
U
(8)
Timing & Control
X1
X2
CLK
GEN
Control
Status
DMA
Reset
CLK OUT
READY
RD
WR
ALE
S0
S1
IO/M
HOLD
HLDA
RESET IN
RESET OUT

Fetch → Decode → Execute

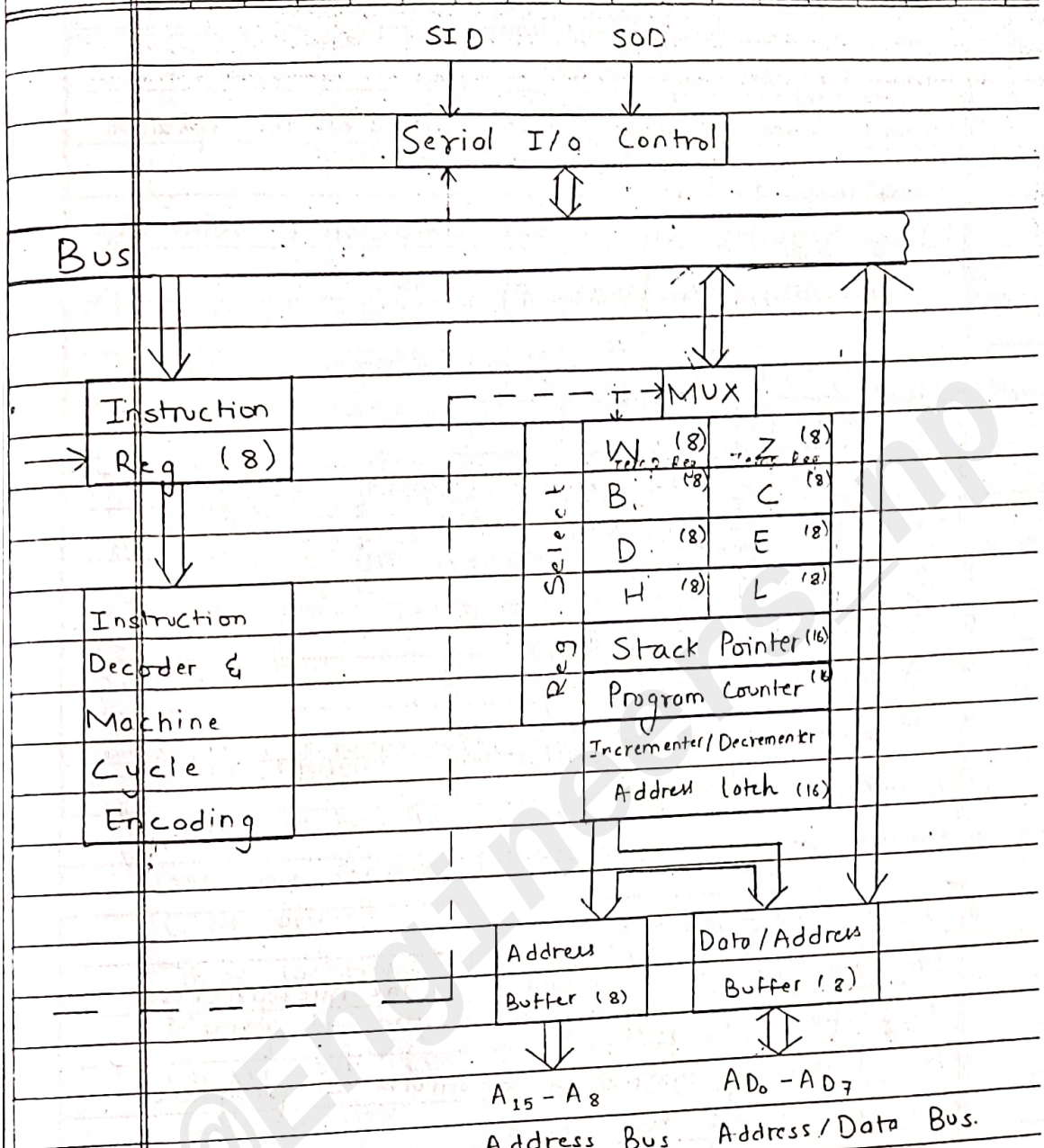


Fig: Block diagram of 8085A

Flag Register of 8085

→ Flags are indicators that indicates the operation ongoing inside ALU.

→ Flag register are always associated with ALU.

→ There are five flags of 8085.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z	X	AC	X	P	X	CY

Fig: Flag register of 8085

$D_7 = 1$ → set ; -ve → 1 → is set

1. Sign Flag → 0 → +ve
2. Zero Flag → 0 → 0 aayo bhane 1
3. Auxillary Carry → 1 Flag
4. Parity Flag → 0
5. Carry Flag → 1

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	1	0	1	0	1	0
1	0	0	0	1	0	1	0
1	0	0	1	1	0	0	0

→ Even number of 1 cha bhane set = 1

8085 Instructions

→ An instruction is a command to the microprocessor to perform a given task.

→ A collection of instructions constitutes a program.

→ It consists of two parts:

1. Opcode (Operation code)

It is the first part of the instruction that represents the what kind of operation or task to be performed.

2. Operand:

Operands are the 2nd part of the instruction. This indicates the data to be operated in the instruction.

eg: MVI A, 25H

MVI A → opcode
25H → Operand

Important

8085 Sets Instruction

Types of Instruction (~~According to byte size~~)

→ ~~Three categories of in~~

→ 8085 instructions can be grouped into five basic categories

1. Data Transfer Instructions
2. Arithmetic Instructions
3. Logic & bit manipulation Instructions
4. Branching Instruction
5. Machine control Instruction

1 Data Transfer Instructions:

- This instruction transfer or copies the instruction from one location (source) to another location (destination)
- These locations may be registers or memory location or I/O.
- Memory to memory transfer is not supported.
- Register pair (BC, DE, HL) is used for 16-bit data transfer.

Address: 0000H to FFFFH

Data (8-bit)

* Note: The I/O port or memory is address

V.V.Imp

	Instruction	Function	Example
1.	MVI Rd, data	This instruction immediately copies the 8-bit data into the register.	MVI A, 25H MVI B, 3A
2.	MOV Rd, Rs	This instruction copies the content of same Register into destination	MOV A, B MOV B, H
3.	LXI Rp, 16-bit data	This instruction immediately loads the 16-bit data into register pair (Bc, DE and HL)	LXI, B, 2050H LXI H, 2076H
4.	IN 8-bit (port-addr)	This instruction copies the data byte from the port address specified in the instruction into Accumulator (A).	IN 03H 8-bit data
5.	OUT 8-bit (port_addr)	This instruction copies the data byte from the Accumulator into the port address specified in the instruction.	OUT 01H.
6.	STA 16-bit address (store Accumulator direct).	This instruction copies the data byte from the Accumulator into the memory location specified by 16-bit address in the Accumulator.	STA 2050H A → 2050 8-bit data
7.	LDA 16-bit address (load Accumulator direct).	This copies the data type from the memory location specified by 16-bit address in the instruction into the Accumulator.	LDA, 2050 H 2050 LDA 8-bit data (A)

Instruction	Function	Example
STAX Rp (Store Accumulator Indirect)	This instruction copies the data byte from Accumulator into the memory locations specified by the register pair in the instruction.	STAX B STAX D A → BC pair memory 8-bit data
LDAX Rp (Load Accumulator Indirect)	This instruction copies the data byte from the memory location specified by register pair in the instruction into the Accumulator.	LDAX B LDAX D data A ← BC pair 2050H
Mov M, R	Copies the content of specified register into the memory location specified by HL pair.	MOV M, B
SHLD address	Store H & L Register Direct	SHLD 2050H
LHLD address	Load H&L Register Direct	LHLD 2050H
XCHG	Exchange the content of H & L with D & E.	

2. Arithmetic Instruction:

These instructions are capable of performing arithmetic operations including adding, subtraction, increment, decrement etc. The basic instructions are as follows.

	Instruction	Function	Example
1.	ADD R/M	• The content of Register / Memory is added to A and the result is stored in A.	ADD B A ← A+B
		• Memory (M) is specified by HL pair.	ADD M A ← A+M
2.	ADC R/M	• The content of R/M is added to A along with carry flag (CF) and result is stored in A.	ADC B A ← A+B+CF→1 ADC M A ← A+M+CF
3.	ADI 8-bit data	• The 8-bit data is immediately added to A and the result is stored in A.	ADI 32H A ← A+32H
4.	ACI 8-bit data	• The 8-bit data is added to A along CF.	ACI 32H A ← A+32H+CF
5.	SUB R/M	• The content of R/M is subtracted from A and the result is stored in A.	SUB C A ← A-C
6.	SBB R/M	• The content of R/M is subtracted from A along with borrow & the result is stored in A.	SBB C A ← A-C-BF
7.	SUI 8-bit data	• The 8-bit data is subtracted from A and the result is stored in A	SUI 25H A ← A-25H
8.	INR R/M	• It increments the content of Register or Memory by 1.	INR C C ← C+1.

	Instruction	Function	Example
9	DCR R/M	• It decrements the content of R/M by 1	DCR D D ← D-1
10	INX Rp	• It increments the content of Register Pair by 1.	INX B BC ← BC+1
11	DCX Rp	• It decrements the content of Register Pair by 1.	DCX D DE ← DE-1

3. Logic / Bit Manipulation Instruction

These instructions are capable of performing logical operations including logical multiplication (AND), logical addition (OR), XOR, complement (not), rotate, compare, etc.

	Instruction	Function	Example
1.	ANA R/M	• The content of A are logically ANDed with the content of R/M and the result is stored in A.	ANA B A ← A·B ANA M
2	ANI 8-bit data	• The content of A are logically ANDed with the immediate data specified in the instruction and the result is stored in A.	ANI 25h
3.	ORA R/M	• OR	
4.	ORI 8-bit data	• OR	
5.	XRA R/M	• XOR	
6.	XRI 8-bit data	• XOR	

Qn. If A = 25H. What will be the result of execution of ANI 32H. 8 4 2 1

Ans;

A = 25 = 0 0 1 0 0 1 0 1

32 = 0 0 1 1 0 0 1 0

AND 0 0 1 0 0 0 0 0

→ 20H 2 0

Qn. What are the status of flag in this execution.

SF : Sign Flag → 0

CF : Carry Flag → 0

AF : Auxilliary Flag → 0

ZF : Zero Flag → 0

PF : Parity Flag → 0

7. Rotate Instruction

ⓐ RLC (Rotate Accumulator Left)

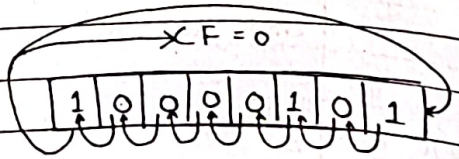
ⓑ RRC (Rotate " Right)

ⓒ RAL (" " Left with Carry)

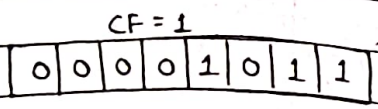
ⓓ RAR (" " Right " ")

ⓐ RLC

For A = 85H



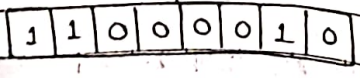
A = 0BH



ⓑ RRC

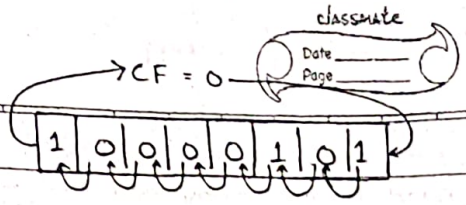
For A = 85H

∴ A = C2H

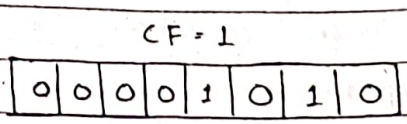


ⓒ RAL

For A = 85H

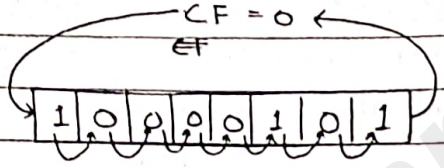


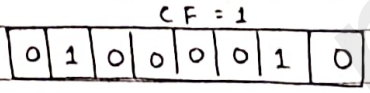
A = 0AH



ⓓ RAR

For A = 85H





8. Compare Instruction

ⓐ CMP R/M

This instruction compares the content of R/M with A. The result of comparison is:

If A < R/M : CF is set, ZF is reset.

If A = R/M : ZF is set, CF is reset.

If A > R/M : CF & ZF both are reset. Eg: CMP B

ⓑ CPI 8-bit data:

This instruction compares the immediate 8-bit data with A. The result of the comparison is;

If A < 8-bit data : CF is set, ZF is reset.

If A = 8-bit data : ZF is set, CF is reset.

If A > 8-bit data : CF and ZF both are reset. Eg: CPI 32H.

* stack = temporary memory

4. Branching Instructions

These instructions are capable of transferring the program sequence from one location into another. It may be: I. Unconditional branch:

	Instruction	Function	Example.
1	JMP 16-bit address	The program sequence is transferred to the memory location specified by 16-bit address	JMP C050H
2	CALL 16-bit address	The program sequence is transferred to the subroutine program at the memory location specified by 16-bit address in the instruction.	CALL D050H
3	RET	The program sequence is transferred from the subroutine program to the CALLing program	RET

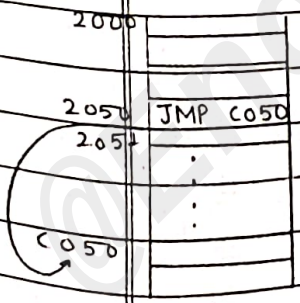


Illustration of JMP (Jump)

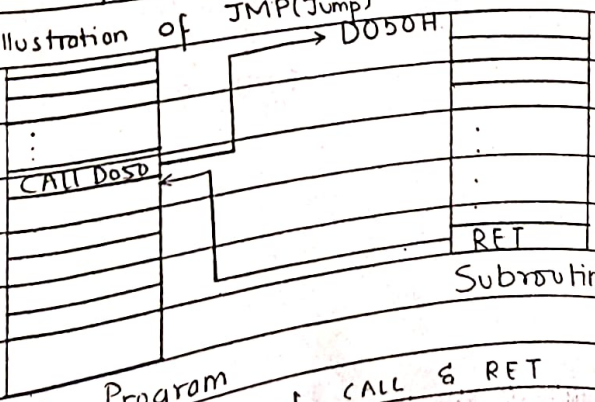


Illustration of CALL & RET

II. Conditional Branch

Instruction Illustration	Function	Example.
JC, 16-bit add	Jump if Carry (CF→1)	JC 2030H
JNC "	Jump if no-Carry (CF→0)	JNC 2020H
JP "	Jump if positive (SF→0)	
JM "	Jump if minus/negative (SF→1)	
JZ "	" " zero (ZF→1)	
JNZ "	" " not zero (ZF→0)	
JPE "	" " parity even (PF→1)	
JPO "	" " " odd (PF→0)	

5. Machine Control Instructions

NOP → No operation is perfored. eg: NOP

HLT → The CPU finishes executing the current instruction & stops any further execution. eg: HLT

WAP to add two 8-bit numbers stored in memory location 2050H and 2060H respectively. Store the result in memory location 2070H.

→

Program		2050	Data1
LDA 2050H	A ← Data1	2060	Data2
MOV B, A	B ← Data1	2070	Sum
LDA 2060H	A ← Data1		
ADD B	A ← A+B		
STA 2070H			
HLT			

WAP to add two 8-bit numbers stored in memory location 2050H and 2060H respectively. Store the sum in memory location 2070H & carry (if occurs) in 2071H.

2050	Data 1
2060	Data 2
2070	Sum
2071	Carry

→ Program

```
LDA  2050H      A ← Data 1
Mov  B, A       B ← Data 1
MVI  C, 00H
LDA  2050H
Mov  B, A
LDA  2060H
ADD  B          A ← A + B
JNC  loop
INR  C
loop: STA  2070H
Mov  A, C
STA  2071H
HLT.
```

WAP to multiply two immediate numbers. And store the result in C050H.

→ Program

```
MVI  A, 00H
MVI  B, 05H
MVI  C, 03H
loop: ADD  B,   A ← 5+0+5+5
DCR  C          C = 2, 1, 0
JNZ  loop
STA  C050H
HLT
```

Types of Instruction (on the basis of size)

① One byte - Instruction:

These instruction occupy a total ~~sum~~ memory of one byte. It includes the opcode & operand in the same byte. eg: ADD B, MOV A, B, RAL, ANA B, etc.

② Two - byte Instruction:

These instructions occupy a total memory of two bytes. The first byte specifies the opcode and the second byte specifies the operand.
eg: MVI A, 32H, ADI 25H, IN 30H, etc.

③ Three byte Instructions:

These instruction occupy a total memory of three bytes. The first byte specifies the opcode and ~~of~~ the remaining two bytes specifies the operand.
eg: STA 2050H, JNC 3050H, LXI H, 2070H.

Imp

Addressing Modes of 8085:

- An instruction consits of opcode and operand. The way in which operands are expressed within an instruction is called ~~insts~~ addressing modes.
- Addressing modes explain how operands are addressed within an instruction.
- There are five addressing mode of 8085.

① Immediate Addressing Mode:

If the 8-bit or 16-bit operands/data are immediately expressed within an instruction then it is called immediate addressing mode.

Eg: MVI A, 29H, LXI B, 2050H, ADI 32, ANI FFH

② Register Addressing Mode:

If the operands are stored in register and the operation is in betn register that are expressed in the instruction then it is called Register Addressing Mode. Eg: MOV A, B, ADD B, ANA D.

③ Direct Addressing Mode:

If the operand's effective address (address where the operand is stored) is expressed with in an instruction then it is called direct addressing mode.

eg: STA 2050, LDA 3050H, SHLD 2050, IN 01H, OUT 03H.

④ Indirect Addressing Mode:

If the register pair which contain the address of the data is specified within an instruction, then it is called indirect addressing mode. Here, register pair holds the ~~operation~~ memory location. Eg: STAX D, LDAX B, MOV A, M.

⑤ Implied Addressing Mode:

If the opcode in an instruction tells about the operand & the operand is not visible within an instruction then it is called implied addressing mode.

Eg: HLT, RET, NOP, RAL, RRC, etc.

Timing Diagram of 8085 Instructions:

Related Terms:

1. Instruction Cycle: The time taken to complete the execution of an instruction is called instruction cycle. It is the combination of machine cycles

2. Machine Cycle: The time taken by the processor to access memory location, I/O ports, or to acknowledge an interrupt once is called machine cycle. It is combⁿ of T-states.

3. T-state: It is the subdivision of operation performed in one clock cycle of the processor's clock.

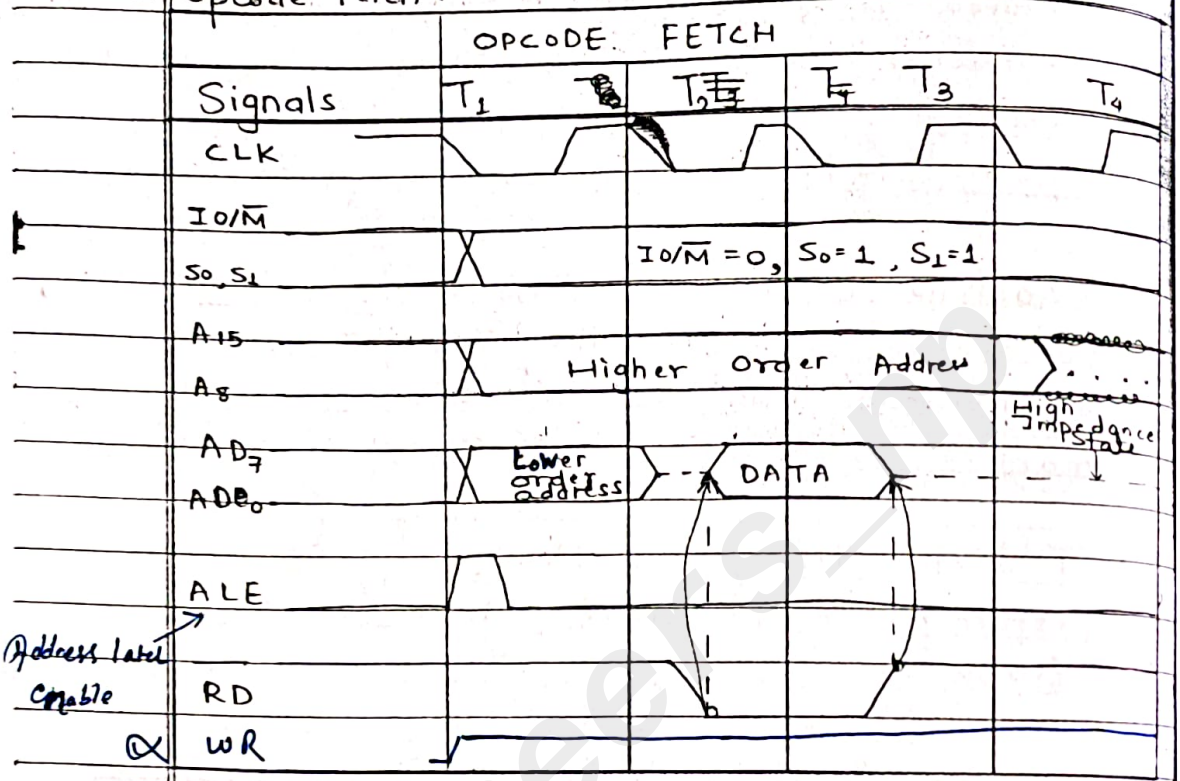
Machine Cycle & the Status Single

Machine cycle	Status Signal			T-states
	$IO/\bar{M}$	S_0	S_1	
Opcode Fetch	0	1	1	4T/6T
Memory Read	0	0	1	3T
Memory Writ	0	1	0	3T
I/O Read	1	0	1	3T
I/O Write	1	1	0	3T

Other machine cycles may be Interrupt Acknowledge & Bus Idle.

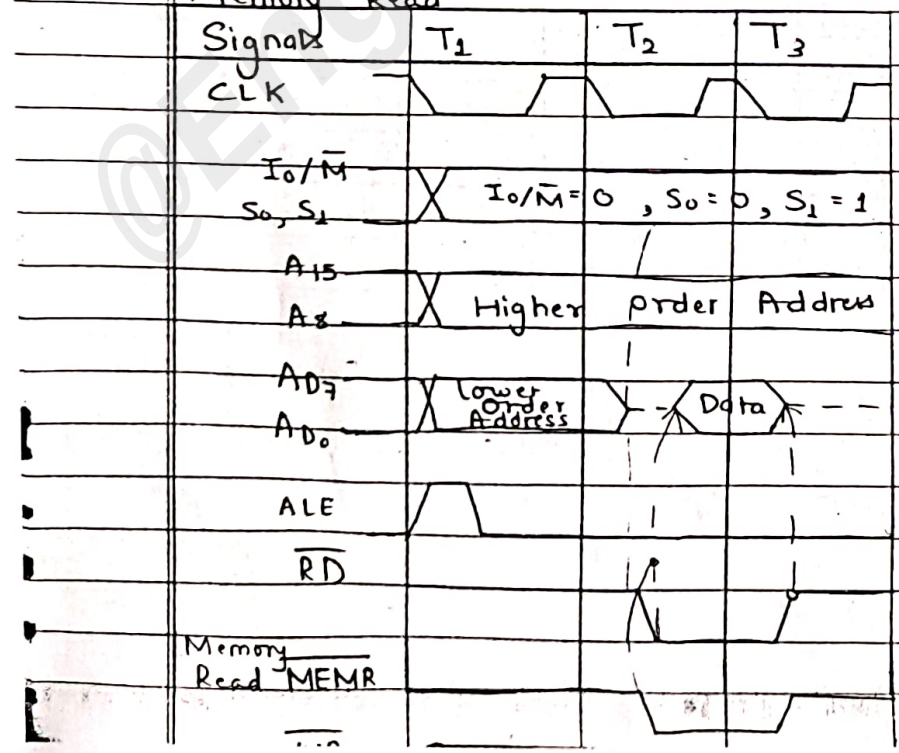
Note: Timing diagram is a graphical representation of an instruction cycle that consists of fetch, decode and execute cycle.

1. Opcode Fetch:



State change @ falling edge of clock cycle.

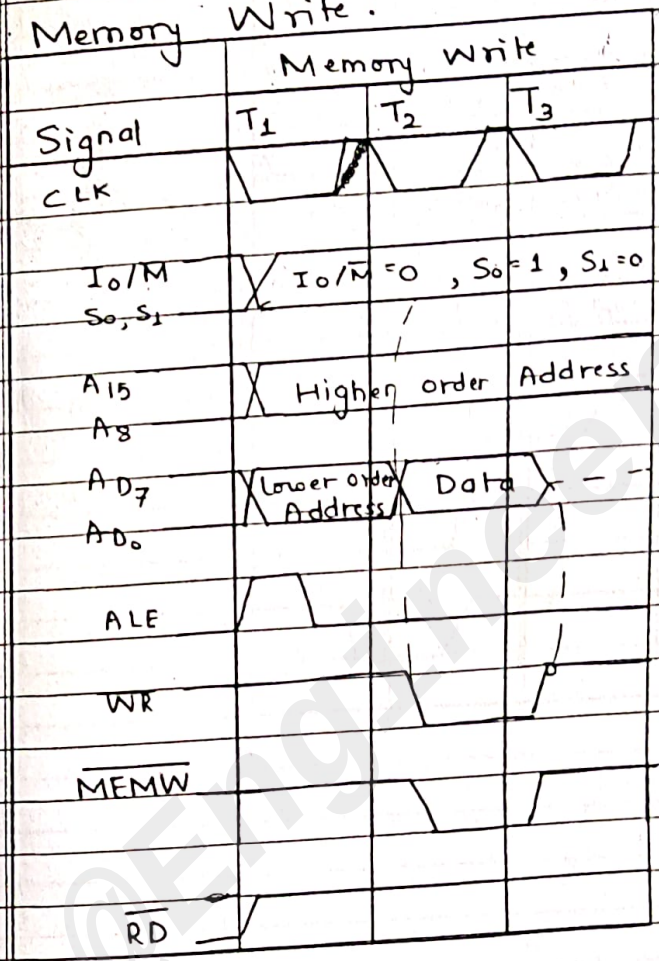
2. Memory Read



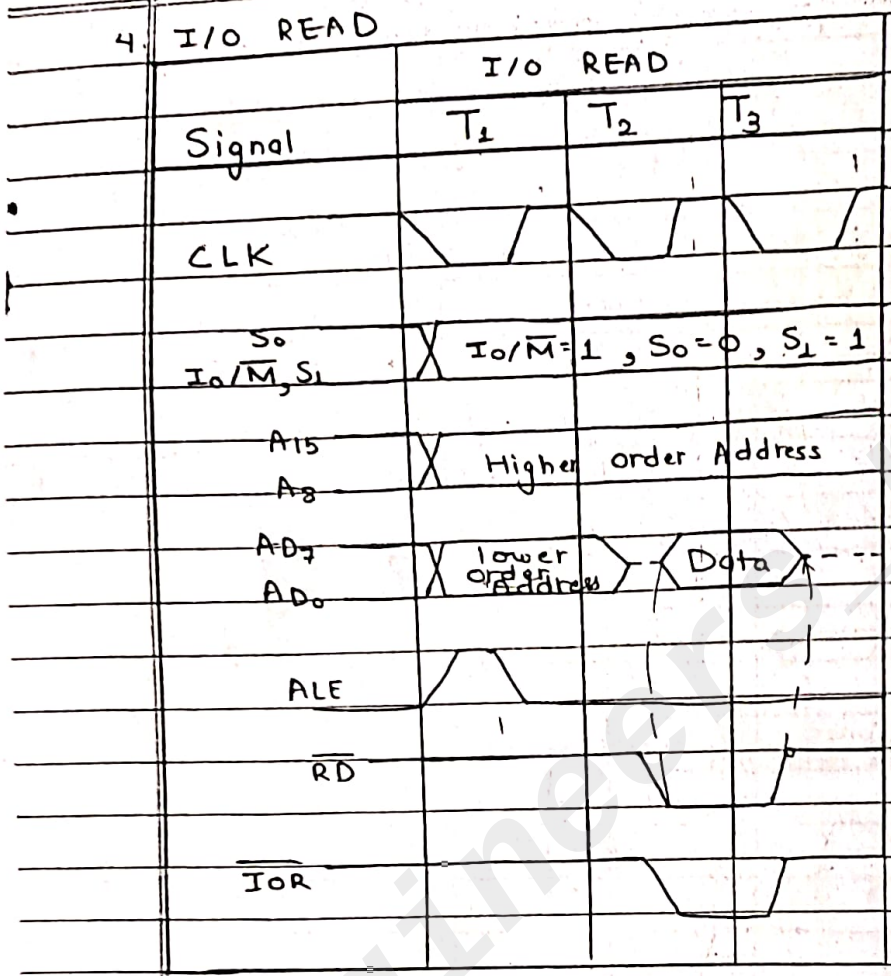
Signals	T_1	T_2	T_3
$\overline{WR}$			

→ Program counter places higher order bus at higher bus and lower order bus at lower bus at time

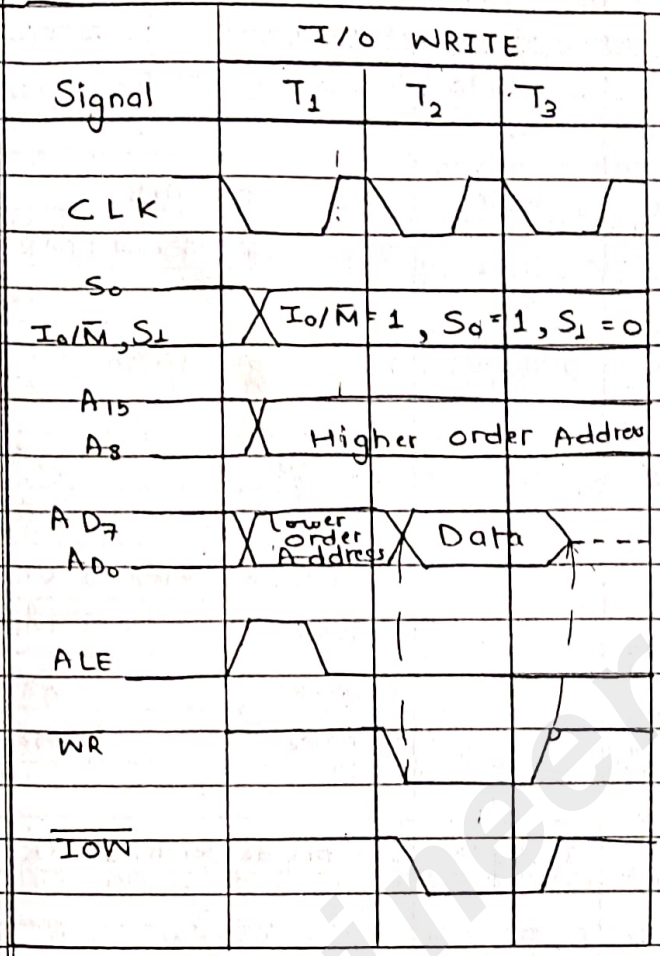
3. Memory Write:



4. I/O READ



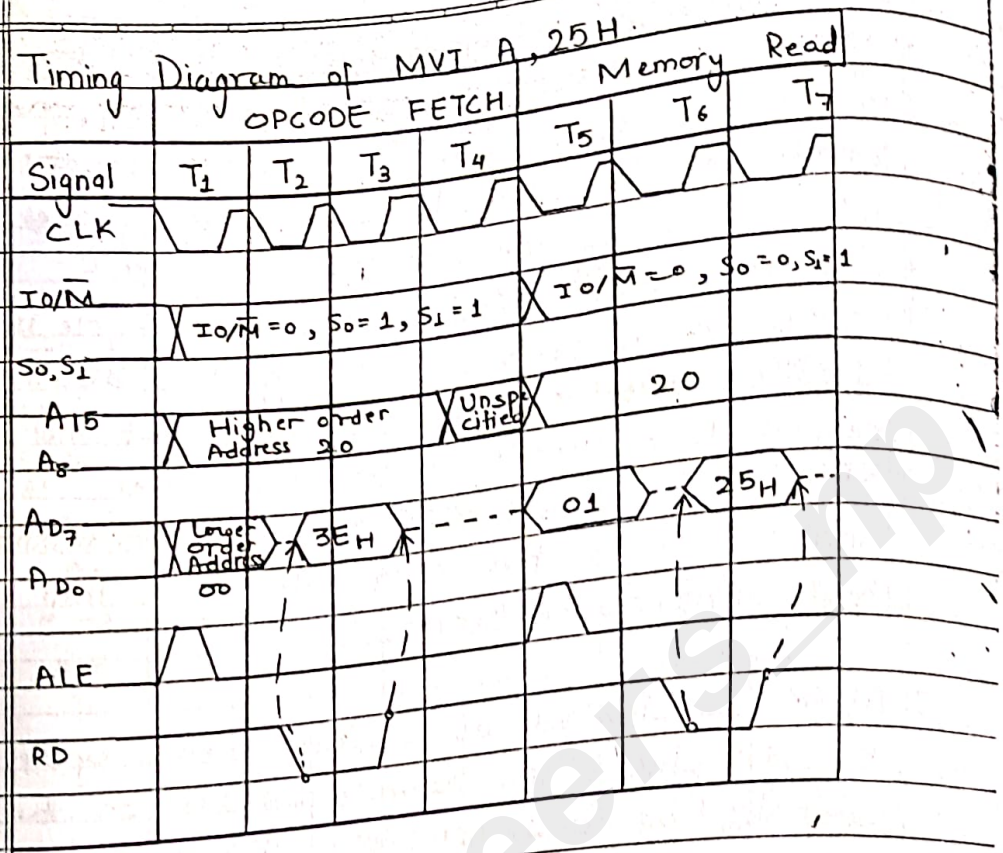
5. I/O WRITE



List of Instructions (along with machine cycles)

	Instruction	Size	Machine Cycle.
1	MOV A, B	1 byte	1. Opcode Fetch
2.	MVI A, 32H	2 byte	1. Opcode Fetch 2. Memory Read (MR)
3.	MVI M, 32H	2 byte	1. Opcode Fetch 2. MR 3. MW
4.	STA 2065H LDA 2050H	3 byte	1. Opcode Fetch 1 OF 2. MR 2 MR 3. MR 3 MR 4. MW 4 MR
5.	MOV A, M	1 byte	1. Opcode Fetch 2. MR
6.	MOV M, A	1 byte	1. OF 2. MW
7.	IN 84H	2 byte	1. OF 2. MR 3. I/O Read
8.	OUT 01H	2 byte	1. OF 2. MR 3. I/O Writ
9.	ADI 12H	2 byte	1. OF 2. MR
10.	JN2 C050H	3 byte	1. OF 2. MR 3. MR (If condition is true)
11.	STAX B	1 byte	1. OF 2. MW

Timing Diagram of MVI A, 25H.



The above timing diagram can be explained as:

1. This instruction consists of two bytes: The first is opcode & the second is data byte. The μP need these bytes to be read from the memory and thus requires two machine cycles,
   I. Opcode Fetch
   II. Memory Read.

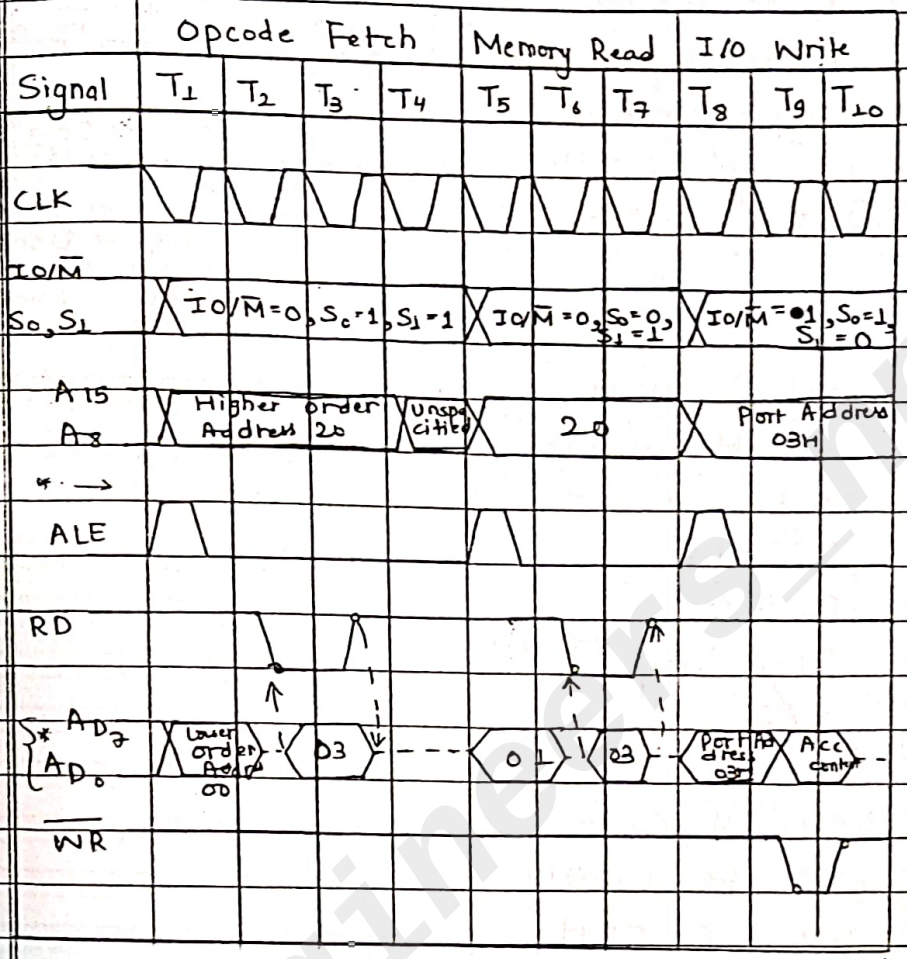
2. At T_1, the μP identifies that it is a opcode fetch cycle. It places $IO/\overline{M} = 0$, $S_0 = 1$, $S_1 = 1$. It places the memory address (say 2000H) from the program counter on the address bus A_{15}-A_8 and AD_7-AD_0 and increments the program counter to next memory address 2001H to print to the next machine code. The ALE goes high during T_1. The 8085 asserts the $\overline{RD}$

control signal which enables the memory and memory places the byte 3EH from location 2000H on the data bus. The 8085 places the opcode in the instruction register & disables the $\overline{RD}$ signal. The fetch cycle is completed in state T_3. During T_4, 8085 decodes the opcode and finds out that a second byte needs to be read. After T_3 state, the content of $A_{15} - A_8$ are unknown and the data bus $AD_7 - AD_0$ goes into high impedance state.

3. After completion of opcode fetch cycle, the MP places the address 2001H on the address bus and increments the program counter at T_5. The second machine cycle is identified as memory Read cycle according to the control signal generated i.e $IO/\overline{M} = 0$, $S_0 = 0$, $S_1 = 1$. The ALE goes high during T_5 state. At T_6, the $\overline{RD}$ signal becomes active and enable the memory chip.

4. Then the memory places the data byte 25H on the data bus, and the 8085 reads and store the byte in Accumulator during T_7.

Qn: Draw and explain the timing diagram for OUT 03H.



OUT 03H

2000 → D3

2001 → 03

Assembly Language Program

Qn. WAP to find the sum of 5 data bytes stored from memory 2000H and store the sum in memory location 2005H.

→
```
LXI H, 2000H
MVI C, 05H
```

2000 → Data 1
⋮
2004 → Data 5
2005 → Sum.

```
    MVI A, 00H
    LXI H, 2000H
    MVI C, 05H
loop: ADD M
    DCR C
    JNZ loop
    STA
```

↳ INX H

```
MVI A, 00H
LXI H, 2000H
MVI C, 05H
loop: ADD M
    INX H
    DCR C
    JNZ loop
    STA 2005
    HLT
```

Qn. To transfer a block of 10 bytes of data stored from memory 2000H into another block of memory starting from 3000H.

→
```
LXI H, 2000H
LXI D, 3000
MVI C, 0AH
loop: MOV A, M
```

```
STAX D
INX H
INX D
DCR C
JNZ loop
HLT
```

Reverse,

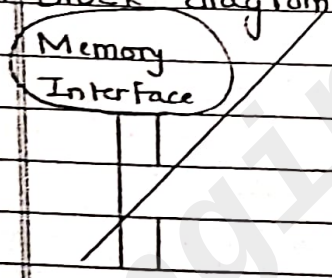
To comare two 8 bit data and store larger no in. your desire memory location.

CHAPTER 3: 8086 MP

Basic Features

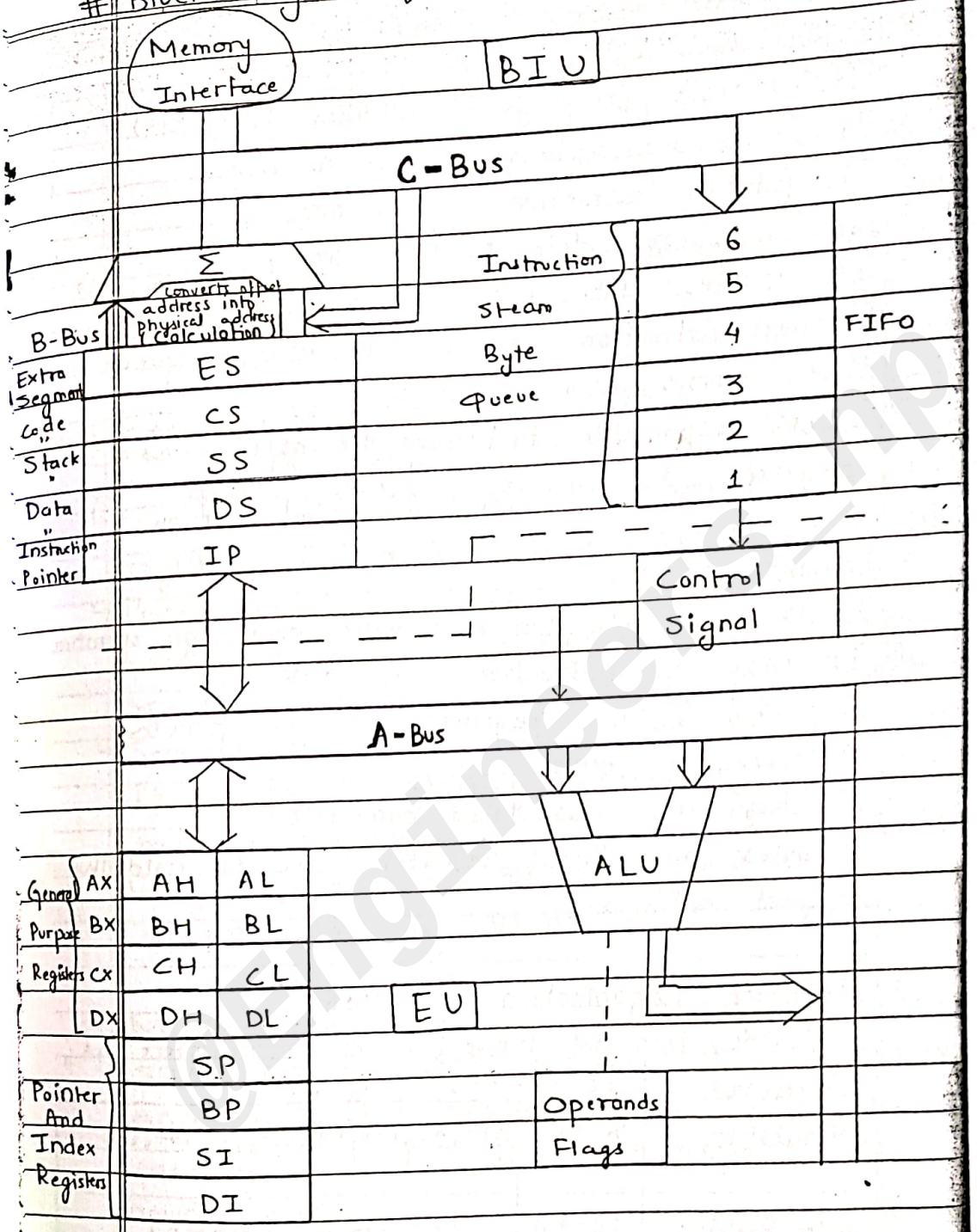
- A 16-bit MP introduced in 1978.
- 40-pin DIP.
- +5V power supply
- Data bus width → 16-bit & address bus width → 20
  Total Addressable Memory = $2^{20} \approx 1$ MB.
- 8086 block is divided into two parts: BIU & EU
- Memory Segmentation
- Pipelining of Instruction
- Operating modes: 1. Minimum mode (Single Processor)
  2. Maximum mode (Multi Processor)
- 256 vectored interrupts

Block diagram of 8086.



- Physical Address → 20-bit
- Effective / Offset Address → 16-bit

Block diagram of 8086



BIU → Bus Interface Unit

EU → Execution Unit

Features of BIU

1. It generates the physical address (20-bit)
2. It sends out addresses to the memory.
3. It fetches instruction from memory.
4. It sends out data to the memory.
5. It writes data in the output port.
6. Fetched instruction are stored in 6-byte queue that works in FIFO basis.
7. It is responsible to decode the instructions & generates the necessary control signals.

Features of EU

1. It is responsible for arithmetic and logic operation
2. It receives instruction from BIU.
3. General Purpose registers are used for the storage of data.
4. 9-Flags are associated with ALU.
5. Pointer and Index register used to hold the offset address.

Register Organisation of 8086

8086 has got powerful set of registers as;

1	General Purpose Registers	16-bit	AX, BX, CX, DX
		8-bit	AH, AL, BH, BL, CH, CL, DH, DL
2	Pointer Registers	16-bit	SP (Stack Pointer) BP (Base Pointer)
3	Index Registers	16-bit	SI (Source Index) DI (Destination Index)

4.	Segment Register	16-bit	CS (Code Segment)
			DS (Data ")
			ES (Extra ")
			SS (Stack ")
5.	Instruction Register	16-bit	IP (Instruction Pointer)
6.	Flag Register	16-bit	9 Flags

↳ General Purpose Registers are 16-bit or 8-bit registers as shown used for the storage of data.
↳ Pointer and index registers are 16-bit registers used to hold the 16-bit offset address. These registers are associated with memory segment.
↳ Segment registers are used to hold the upper 16-bit of the starting address of the memory segment.
↳ Flag register are used to indicate the operation ongoing inside ALU.

Memory Segmentation

↳ 8086 has got 1MB memory that ranges from 00000H ~~and~~ to FFFFFH. These portion of memory is divided into four segment each having 64KB.
↳ The ease of memory segmentation is that we can address the memory of 8086 16-bit address although the physical address is 20-bit
↳ 16-bit segment registers (DS, CS, SS & ES) are associated with these memory segment that holds the upper 16-bit of the starting address of the memory segment.

↳ Data Segment → Data are operated in this memory
code " → Instructions / opcode " " "
Stack " → Stack operation
Extra " → Data / Strings Operations.

↳ Pointer & Index registers within the memory segment gives the offset address:
i.e CS → IP
DS → SI / DI
SS → SP or BP
ES → DI

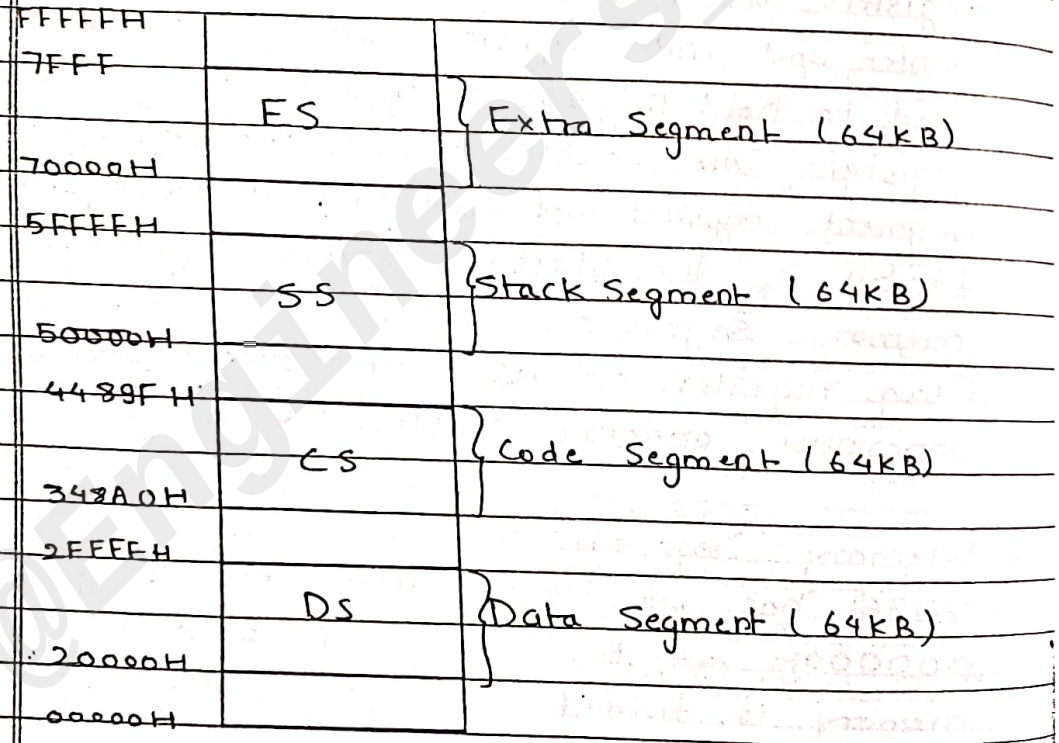
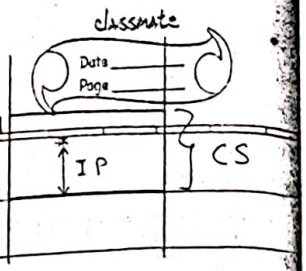


Fig: Memory Segmentation.



Physical Address Calculation:

CS = 348AH

IP = 1234H

PA = (Segment Address) X 10 + Offset Address

= (348A) X 10 + 1234H

= 348A0 + 1234

= 35AD4H

Flag Register of 8086:

D_{15}	D_{14}	D_{13}	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

↳ Flags are associated with ALU that indicates the status/operation.

Flags of 8086

Conditional Flag
- → Sign Flag (SF)
- → Zero Flag (ZF)
- → Auxillary Carry Flag (AF)
- → Parity Flag (PF)
- → Carry Flag (CF)
- → Overflow Flag (OF)

Control Flag
- → Interrupt Flag (IF)
- → Trap Flag (TF)
- → Direction Flag (DF)

Control Flags

1. Interrupt Flags:

It is interrupt enable/disable flag. If it is set to 1, the maskable INTR of 8086 is enabled and if it is 0, the INTR is disabled. It can be set/Reset by using instruction

STI → Set Interrupt

CLI → Clear Interrupt

2. Direction Flag:

It is used for string operations. If it is set to '1', string bytes are accessed from high memory address to low memory address. When it is '0', the string bytes are accessed from low memory to high memory address.

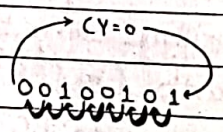
3. Trap Flag:

It allows users to execute one instruction of a program at a time of debugging. Also known as single step control flag. When it is set to '1', program can be run in single step mode.

WAP to find the number of 1's present in a byte of data.

→ Suppose the data is 25 = 00100101

```
        MVI  A, 25H
        MVI  C, 08H
        MVI  B, 00H
loop:   RAL
        JNC  here     ; JC for no. of 0's
        INR  B
here:   DCR  C
        JNZ  loop
        MOV  A, B
        STA  2050H
        HLT
```



Instruction Sets of 8086 μP:

8086 μP has got following sets of instructions:

1. Data Transfer Instruction
2. Arithmetic Instruction
3. Logic / Bit Manipulation Instruction
4. Program Execution Transfer Instruction
5. String Instruction
6. Process Control Instruction

1. Data Transfer Instruction

↳ Used to transfer data from source to destination

↳ Operand can be constant, memory location register or I/O port address.

It consists of the following:

@ MOV Des, Src:

eg: MOV CX, 0025H
MOV AL, BL
MOV BX, [0205H]

ⓑ PUSH Operand:

It pushes the operands onto the top of stack.

Eg: PUSH BX.

ⓒ POP Des:

It pops the operand from the top of the ~~state~~ stack to Des.

↳ Des can be general purpose registers, segment registers (except CS). or memory location.

Eg: POP AX.

(d) XCHG Des, Src:
→ This instruction exchanges src with Des.
→ It can't exchange, has memory locations directly
Eg: XCHG, DX, AX.

(e) IN Accumulator, Port Address:
→ It transfers the operand from specified port to the Accumulator.
Eg: IN AX, 0028H.

(f) OUT port address, Accumulator:
→ It transfers operand from Accumulator to the specified port.
Eg: OUT 0028H, AX.

(g) LEA Register, Src:
→ It loads the 16-bit register with the offset address of the data specified by the src.
Eg: LEA BX, [DI].

(h) LDS Des, Src:
→ It loads the 32-bit pointer from memory source to destination register & DS (Data Segment)
→ The offset is placed in the destination register and the segment is placed in DS.
Eg: LDS BS, [0301H].

(i) LES Des, Src:
→ It loads the 32-bit pointer from memory source to destination register and ES.
→ The offset is placed to des register and the segment is placed in ES. Eg: LES BX, [0301H]

(j) LAHF

→ It copies the lower byte of flag register to AH.

(k) SAHF

→ It copies the content of AH to lower byte of flag.

(l) PUSHF

→ It pushes flag register onto the top of stack.

(m) POPF

→ It pops the top of stack into the flag register.

2. Arithmetic Instructions

@ ADD Des, Src

→ It adds a byte to byte or word to word.

→ It affects AF, CF, OF, PF, SF, ZF flags.

Eg:

```
ADD AL, 72H          AL ← AL+72H
ADD DX, AX
ADD AX, [BX]         AX ← AX + (BX)
                       ↳ Memory bhitrako data
```

(b) ADC Des, Src

→ It adds two operands with CF.

→ It affects AF, CF, OF, PF, SF, ZF flags.

Eg:

```
ADC AL, 72H
ADC DX, AX
ADD AX, [BX]
```

(c) SUB Des, Src

→ It subtructs a byte from byte. or word from word
→ It affects AF, CF, OF, PF, SF, ZF flags.
→ CF acts as a borrow flag.

Eg:

```
SUB AL, 72H
SUB DX, AX
SUB AX, [BX]
```

(d) SBB Des, Srz

→ It subtructs the two operand and also borrow from the result.
→ It affects AF, CF, OF, PF, SF, ZF Flag.

Eg:

```
SBB AL, 72H
SBB DX, AX
SBB AX, [BX]
```

(e) INC Src

→ It increments the byte or word by 1.
→ The operand can be register or memory.
→ It affects AF, OF, PF, SF, ZF flags.
→ CF is not affected.

Eg: INC bl

(f) Dec Src

→ It decrements the byte / word by 1.

Eg: Dec bX

(g) AAA (ASCII Adjust After Addition).

→ The data entered from the terminal is in ASCII format.

→ In ASCII 0-9 are represented by 30H - 39H.

→ This instruction allows us to add the ASCII codes

(h) AAS

(i) AAM

(j) AAD

(k) NEG Src

→ It creates 2's complement of the given number / Src.

(l) MVL Src

→ It is an unsigned multiplication instruction.

→ It multiplies two bytes to produce a word or two word to produce a double word.

AX = AL * Src.

AX = AX * Src

→ This instruction assumes one of the operands in AL or AX.

→ Src can be register or memory.

(m) DIV Src

→ It is unsigned division.

→ The operand is stored in AX, Divisor is Src and the result is stored as:

AH = remainder,

AL = quotient

3) Logic / BIT Manipulation Instruction

a) NOT Src
→ It complements each bit of src to produce 1's complement of specified operand.
→ The operand can be register or memory.

b) AND Des, Src
→ AND operator of Src and Des
→ Src can be immediate number, register or memory.
→ Des can be register or memory.
→ Both operands can't be memory simultaneously.
Eg: AND AL, BL.

c) OR Des, Src

d) XOR Des, Src

e) SHL Des, Count
→ It shifts bits of byte/word left by count.
→ It puts zeros in LSB.
→ MSB is shifted into Carry Flag.

f) SHR Des, Count
→ It shifts bits of byte/word right by count
→ It puts zeros in MSB.
→ LSB is shifted into CF.

g) ROL Des, Count
→ It rotates bits of byte/word left by count.
→ MSB is transferred to LSB and CF.

(h) ROR Des, Count
→ Rotate right by count
→ LSB is transferred to MSB and CF.

(i) CMP Des, Src
→ It compares two specified bytes/words.
→ Src, Des can be constant, register or memory.
→ Flags are modified according to the result.

⇒ See 8085 (for Flags condn)

4. Program Execution Transfer Instruction.
(a) CALL Des:
→ This instruction is used to call a subroutine or procedure.
→ The address of next instruction after CALL is saved onto stack.

(b) RET
→ It returns the control from procedure to CALLing program.

(c) JMP Des
→ Unconditional jump from one place to another.

(d) JXX Des
→ Conditional Jump Instructions (that effects the Flags)
→ The following table shows the list of instruction:

Conditional Jump Instructions	Condition
JA → Jump if above	CF → 0, ZF → 0
JAE → Jump if above or equal	CF → 0
JB → Jump if below	CF → 1
JBE → Jump if below or equal	CF → 1, ZF → 1
JC → Jump if carry	CF → 1
JNC → Jump if no carry	CF → 0
JNZ → Jump if zero	ZF → 1
JPE / JNZ → Jump if no zero	ZF → 0
JPE → Jump if parity even	PF → 1
JPO → Jump if parity odd.	PF → 0

6. LOOP Des

→ This is looping instruction and the number of loop/iteration is placed in cx register

→ With each iteration, the content of cx are decremented.

5. String Instructions

→ String in assembly language is just a sequentially stored bytes/words.

→ By using the instructions, the size of the program is considerably reduced.

→ It consists of the following instructions:

@ CMP Des, Src:

It ~~consis~~ compares the string bytes or words.

(b) SCAS String:

It scons a string. It compares the string with byte in AL or with word in AX.

© MOVS/MOVB/MOVSW :

→ It causes moving of bytes or word from one string to another. In this instruction the source is in data segment and the destination is in extra segment.

→ SI and DI store the offset values for the source & destination string.

ⓓ Rep (Repeat) :

→ This is an instruction prefix.

→ It causes the repetition of instruction until cx becomes zero.

ⓔ Processor Control Instruction :

These instructions controls the processor itself. Some of these instructions are:

I. STC : It sets the CF to 1.

II. CLC : It clears the CF to 0.

III. CMC : It complements the CF.

IV. STD : It sets the direction Flag (DF) to 1.

V. CLD : It clears the DF to 0.

Sure Top

Assembly Language Programming of 8086

* Sample Program (To display a string "Hello World")

Assembler Directives

```
.Model small
.stack 100h          → stack segment ko size 100
.Data
    String db "Hello World!$"
.Code
main proc
.STARTUP  mov ax, @DATA ; @DATA is predefined symbol for
                        initial address of DS
          mov ds, ax    ; initialize data segment.
          mov dx, offset string ; load the offset
          OR
          LEA dx, string        address into dx
          mov ah, 09h ; ax = 09h for string display
                                until $.
          Int 21h ; DOS interrupt function.
.EXIT     mov ax, 4c00h ; End request with AX = 4c00h
          Int 21h       ; return control to OS
main endp               ; end procedure
end main                ; end program
```

ALP Development Tools:

1. Editor → .asm
2. Assembler → One pass Assembler
   → Two pass Assembler (.obj)
3. Linker (.obj)
4. Debugger → .exe
5. Emulator (comb^n of ~~hdw one~~ hardware and software)

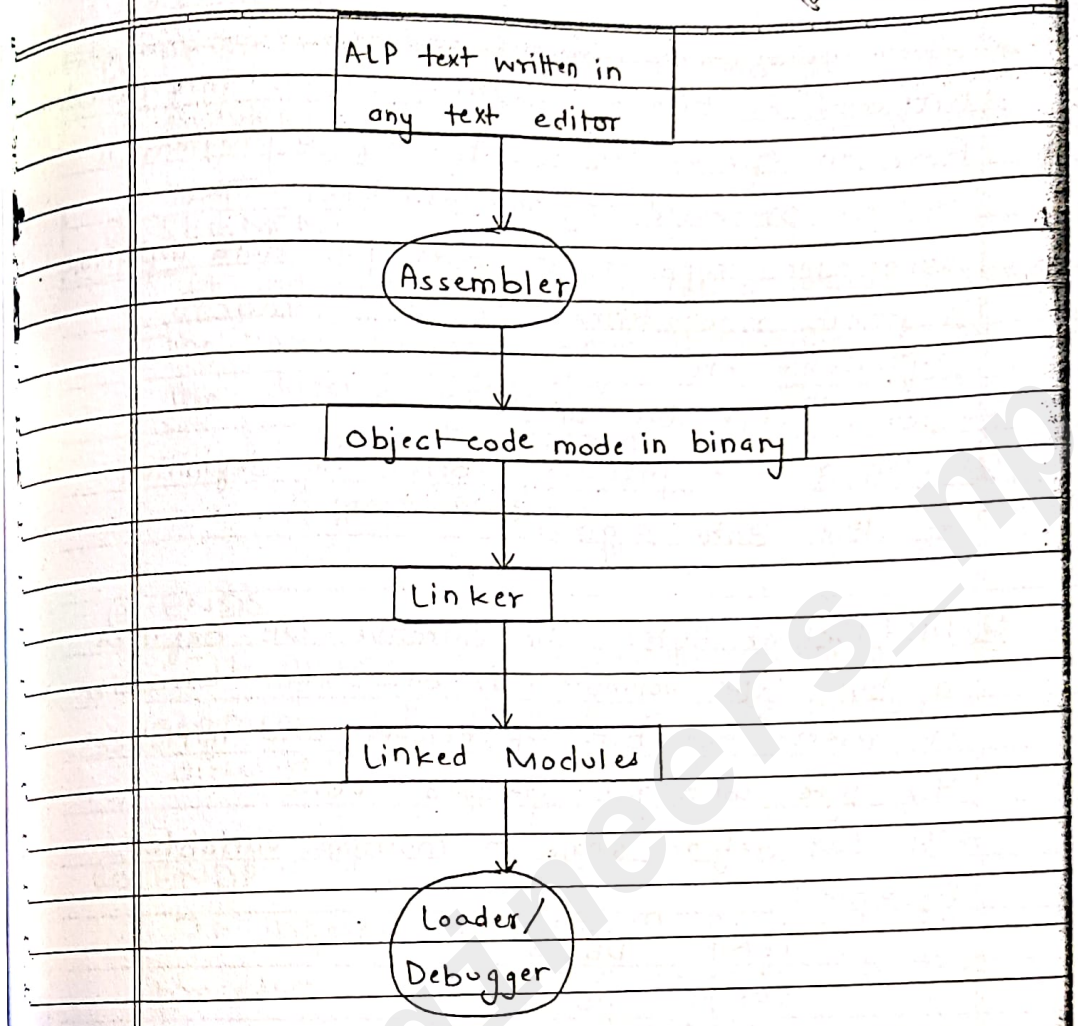


Fig: Flowchart for the development of the ALP tools.

Qn: Differentiate one pass assembler and two pass assembler.

Imp # Assembler Directives

An ALP contains two types of statements: instruction and assembler directives. The instructions are translated into machine codes whereas directives are not converted into machine codes. The directives are statements to give directions to the assembler to

perform the task of assembly process. They indicate how an operand or a section of a program is to be processed by the assembler. The assembler supports directives for data definition, segment organizations, procedure, macro definitions etc.

↳ Some imp directives are;

① .DATA : It provides shortcut in definition of the data segment.

② DB (Define Byte) : The directive DB defines a byte type variable. It directs the assembler to reserve one byte of memory and initialize the byte with the specified value.

→ It can define single or multiple variables.

Example :

```
TEMP    DB  5
TEMP    DB  ?
TEMP    DB  11,22,33,44.
```

③ DW (Define Word) → 2 bytes

④ DD (Define double word) → 4 bytes

⑤ DQ (Define Quad word) → 8 bytes

⑥ .CODE : This provide shortcut in definition of the code segment. It is basically specified to distinguish different code segment when there are multiple CS in the program.

Eg: .CODE [name]
optional

(7) PROC : It tells the start of the procedure or subroutine.

(8) SEGMENT :

It indicates the beginning of a logical segment. The name of the segment is written before the directive SEGMENT.

Eg: Program SEGMENT
Segment name

(9) ENDS :

It informs the assembler; the end of the segment. It is used with the segment directive.

(10) ENDP :

It informs the assembler the end of the procedure. This directive together with PROC is used to enclose a procedure.

(11) END

This directive is used after the last statement of the program to inform the assembler that this is the end of a program module.

(12) ASSUME

It tells the name of a logical segment which is to be used for specified segment

(13) STACK

It indicates the start of the stack segment. The default size of stack is 1024 bytes.

(14) • Model:

It provides the info of the segment size that is to be used in a program module. The size of the model may be tiny, small, large, etc.

(15) • Startup:

It indicates the start of the program when using program.

(16) • EXIT:

It indicates the end of the program when using program.

DOS Interrupt Function of 8086:

→ The list of interrupts are:

* INT 10H function:

① Int 10h/AH = 00H → Set video mode input
② Int 10h/ah = 01H → Set text mode cursor shape
③ Int 10h/ah = 02H → Set cursor position
④ Int 10h/ah = 06H → Scroll window up
⑤ Int 10h/ah = 08H → read character and attribute at cursor position.
⑥ Int 10h/ah = 0AH → write character only at cursor position.

* INT 21h functions:

① Int 21h/ah = 01h → read character from standard input, with echo, result is stored in AL.
② Int 21h/ah = 02h → write character to standard output. eg: mov ah, 02h (the character must be in dl)
mov dl, 'a'
Int 21h.

③ Int 21h/ah=05h → o/p character to printer.

④ Int 21h/ah=09h → output of a string at DS:DX
String must be terminated by $.

⑤ Int 21h/ah=0Ah → Input of a string to DS:DX
first byte is buffer size, second byte is number of characters actually used.
→ $ not required to terminate the string.

⑥ Int 21h/ah = 4Ch → Return control to operating system (Stop program).

WAP in 8086 to add 8-bit number

```
.model small
.stack 100h
.data                                   dB → Define Byte
    val1 dB 25h
    val2 dB 35h
.code
  main proc
.startup { mov ax,@DATA
           mov ds,ax
           mov bl,val1
           add bl,val2
           mov result,bl
.exit    { mov ax,4c00h  or  mov ah,4ch
           Int 21h
  main endp
  end main
```

emu8086 Assembler

Date
Page

WAP to display alphabets from a to z.

→
```
.model small
.stack 100h
.data
    alpha dB 'a'
.code
    main proc
        .startup
            mov cx, 26
            mov bl, alpha
            mov dl, bl
again :     mov ah, 02h   } a display....
            Int 21h
            Inc dl
            loop again
        .exit
    main endp
    end main
```

WAP to display numbers from 0 to 9.

→
```
.model small
space macro
.stack 100h
    mov dl, ' '
.data   mov ah, 02h
        int 21h
    endm
.stack 100h
.data
    digit dB '0'
.code
    display proc
        .startup
            mov cx, 10
            mov bl, digit
again :     mov dl, bl
            mov ah, 02h
            int 21h
            space
            inc bl
            loop again
        .exit
    display endp
    end display
```

Imp

MACRO :

It is a group of instructions. The macro assembler generates the code in the program each time where the macro is called. Macro's can be defined by MACRO and ENDM directives.

Example :

```
space  macro ; macro definition
(name)
mov  dl , c
mov  ah , 02h  } ; body
Int 21h
endm ; end macro
```

Macro Vs Procedure

	Procedure	Macro
1.	Accessed by CALL and RET instruction during program execution.	Accessed during Assembly with name given to macro.
2.	Machine code is generated only once in memory.	Machine code is generated each time when it is called.
3.	Less memory (memory efficient)	More memory is required.
4.	Parameters can be passed in register, memory location, stack.	Parameters passed as a part of the statement which calls macro.
5.	long codes	Short codes
6.	eg: main proc { main endp end main	eg: space macro { endm

WAP to display alphabets 'a to z' with space. (Use macro)

→
```
.model small
space macro
mov dl, ' '
mov ah, 02h
Int 21h
endm
.stack 100h
.data
    alpha dB 'a'
.code  display proc
    .startup
    mov cx, 26
    mov bl, alpha
again: mov dl, bl
    mov ah, 02h
    Int 21h
    space
    Inc bl
    loop again
    .exit
    display endp
    end display
```

WAP to display a string 'microprocessor' characterwise in console.

→
```
.model small
.stack 100h
.data
    string dB 'microprocessor"
.code
main proc
  .startup
    mov cx, 14
    mov si, offset string
again: mov dl, [si]
    mov dl, bl
    mov ah, 02h
    Int 21h
    Inc Si
    loop again
  .exit
main endp
end main
```

To display a string 'microprocessor' in reverse order characterwise.

→
```
.model small
.stack 100h
.data
    string dB 'microprocessor"
.code
main proc
  .startup
    mov cx, 14
    mov si, offset string
```

```
add si,13
again: mov bl,[si]
       mov dl,bl
       mov ah,02h
       int 21h
       Dec si
       loop again
       .exit
       main endp
       end main
```

WAP to transfer a block of data from memory location list1 to list2. Consider the block of six bytes.

→ Title copy of block of data.

```
Dosseg
.model small
.stack 100h
.data
    list1 dB 10,20,30,40,50,60
    list2 dB 6dup(?)
.code
main proc
    .startup
    mov si,offset list1
    mov di,offset list2
    mov cx,06
back: mov al,[si]
    mov [di],al
    inc si
    inc di
    loop back
    .exit
    main endp
    end main
```

You are given two strings "assembly program" and "language". Now write an assembly language program to display the string 'assembly language program'.

→

```
.model small
  display macro
    mov dx, offset string2
    mov ah, 09h
    int 21h
    end m
.stack 100h
.data
    String1 dB "Assembly Program"
    String2 dB "Language"
.code
  main proc
    .startup
        mov  cx, 09
        mov Si, offset String1
back : mov dl, [Si]
        mov ah, 02h
        Int 21h
        Inc Si
        loop back
        display
        mov di, offset string1
        add dl, 10
again : mov dl, [di]
        mov ah, 02h
        Int 21h
        Inc di
        loop again
    .exit
main endp
```

8086 modes of operation

1. Minimum mode (Single Processor Mode) | MN / $\overline{MX}$ = 1 |

2. Maximum mode (Multi Processor Mode) | MN / $\overline{MX}$ = 0 |

GND		40	V_{CC}
AD_0-AD_{15}	2-16 and 39	35-38	A_{16}/S_3 – A_{19}/S_6
NMI	17	34	$\overline{BHE}$ / S_7
INTR	18	33	MN/$\overline{MX}$
CLK	19	32	$\overline{RD}$
GND	20	31	HOLD
RESET	21	30	HLDA
READY	22	29	$\overline{WR}$
$\overline{TEST}$	23	28	M / $\overline{IO}$
$\overline{INTA}$	24	27	DT/$\overline{R}$
ALE	25	26	$\overline{DEN}$

Starting from 31: For maximum pin

$\overline{RQ}$ / GT_0

$\overline{RQ}$ / GT_1

$\overline{LOCK}$

$\overline{S_2}$

$\overline{S_1}$

$\overline{S_0}$

Figure: Pin configurations of minimum and maximum modes.

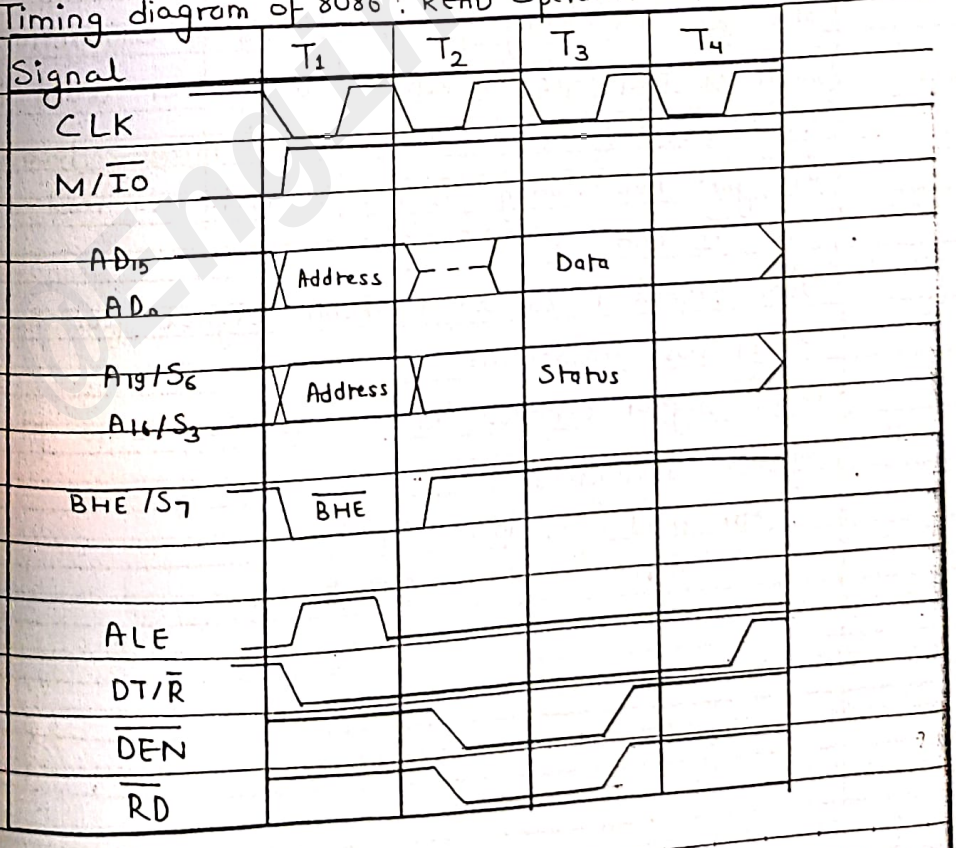
Table :

S_4	S_3	Segment Register
0	0	ES
0	1	SS
1	0	CS
1	1	DS

$\overline{BHE}$	A_0	Word / Byte Access
0	0	Whole word
0	1	Upper byte
1	0	Lower byte
1	1	none

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Characteristic
0	0	0	Interrupt Acknowledge
0	0	1	I/O Read
0	1	0	I/O write
0	1	1	Halt
1	0	0	Opcode Fetch
1	0	1	Memory Read
1	1	0	Memory write
1	1	1	Passive

Timing diagram of 8086 : READ Operation (Minimum Mode)



→ Figure shows the timing diagram for 8086 memory read bus cycle for minimum modes of operation. During T_1 clock of bus cycle, the 8086 sends out the 20-bit memory address on the address bus. As the address lines operates in time multiplexed mode, the address remain on these lines only for clock. T_1. For the minimum modes of operation, pin MN/$\overline{MX}$ is kept high. The 8086 sends a high signal M/$\overline{IO}$ line to indicate that the data transfer will take place with memory. During T_1, an ALE (Address Latch Enable) is sent out so that the address is latched in the external devices.

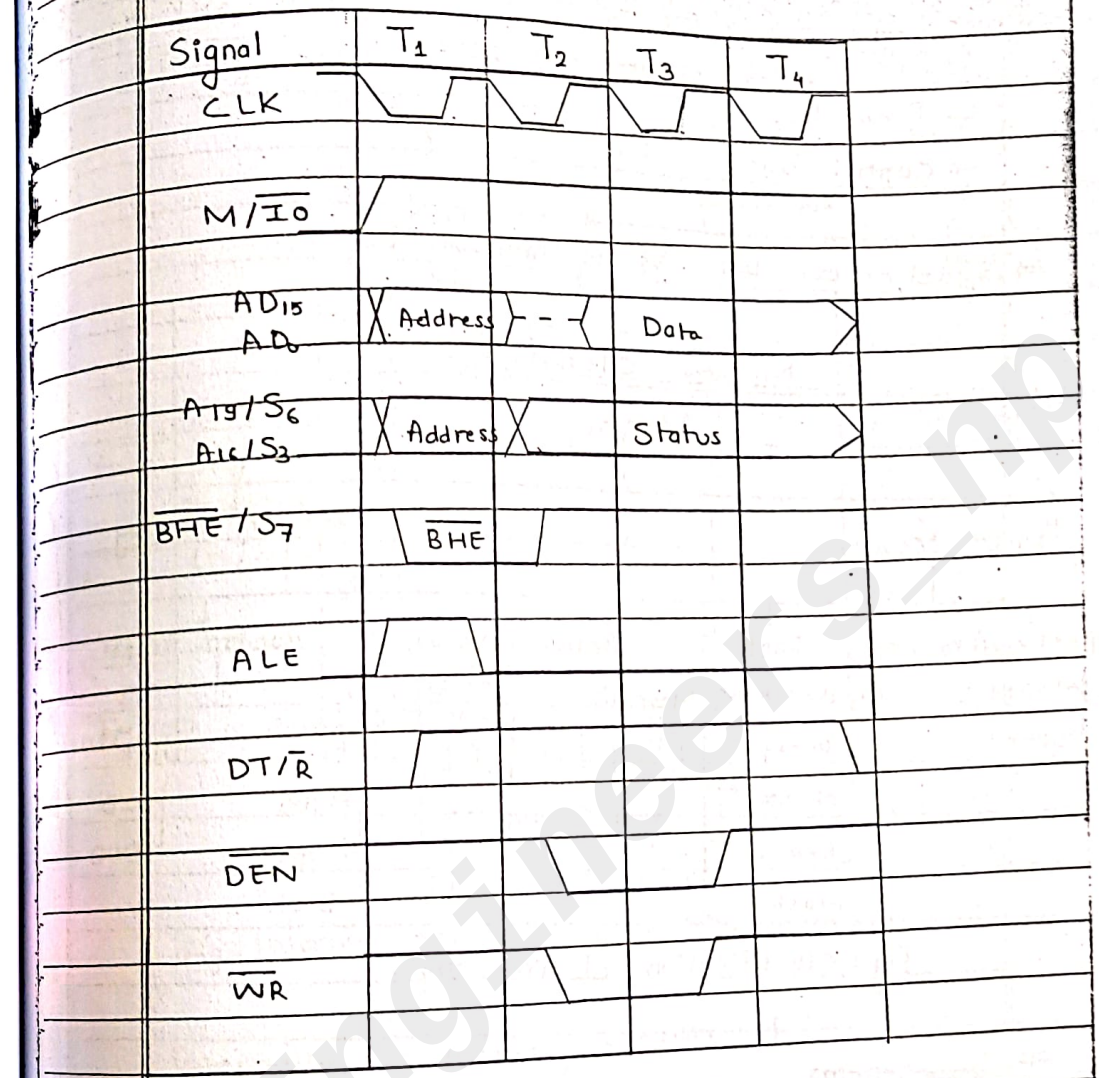
- During T_2, the AD_0-AD_{15} gives into high impedance state.
- During T_3 and T_4 data are transmitted on this bus.
- During T_2, T_3 & T_4, the address lines A_{16}/S_3 - A_{19}/S_6 carry ~~signals~~ status signals S_3, S_4, S_5 & S_6.
- $\overline{BHE}$ goes low during T_1. It works in conjunction with A_0 that decides whether a byte or word is to be transmitted from/to the addressed memory location
- $\overline{RD}$ goes low during T_2 and the data is read out at T_2 and T_3.
- $\overline{DEN}$ signal is an output enable signal for the octal bus transceivers.
- DT/$\overline{R}$ signal controls the direction of data flow throw transceivers.

  For read operation, DT/$\overline{R}$ → 0

  write " , DT/R → 1

Maximum Mode
(Read & Write) → yourself

For Write Operation (Minimum Mode)



Chapter 4: Bus Structure & Memory Device

Bus configuration of a microprocessor (Refer to ch1)
- ↳ Address bus
- ↳ Data bus
- ↳ control bus

Synchronous bus Vs Asychronous bus

Memory Classification

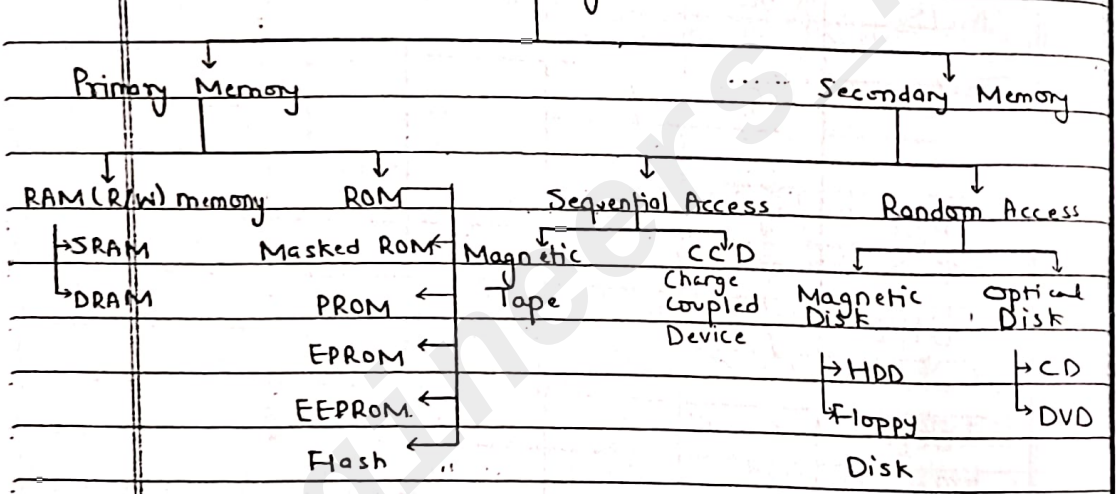


Fig: Classification of memory.

~~Synchronous~~
- → A memory is a collection of storage cells that is used to store data and information.
- → A memory is a storage device.
- → Flipflop is the memory unit that stores 1 bit of information.

Structure of a memory

→ The internal structure of a 8X8 memory is shown:

→ Every memory unit has similar type of structure.

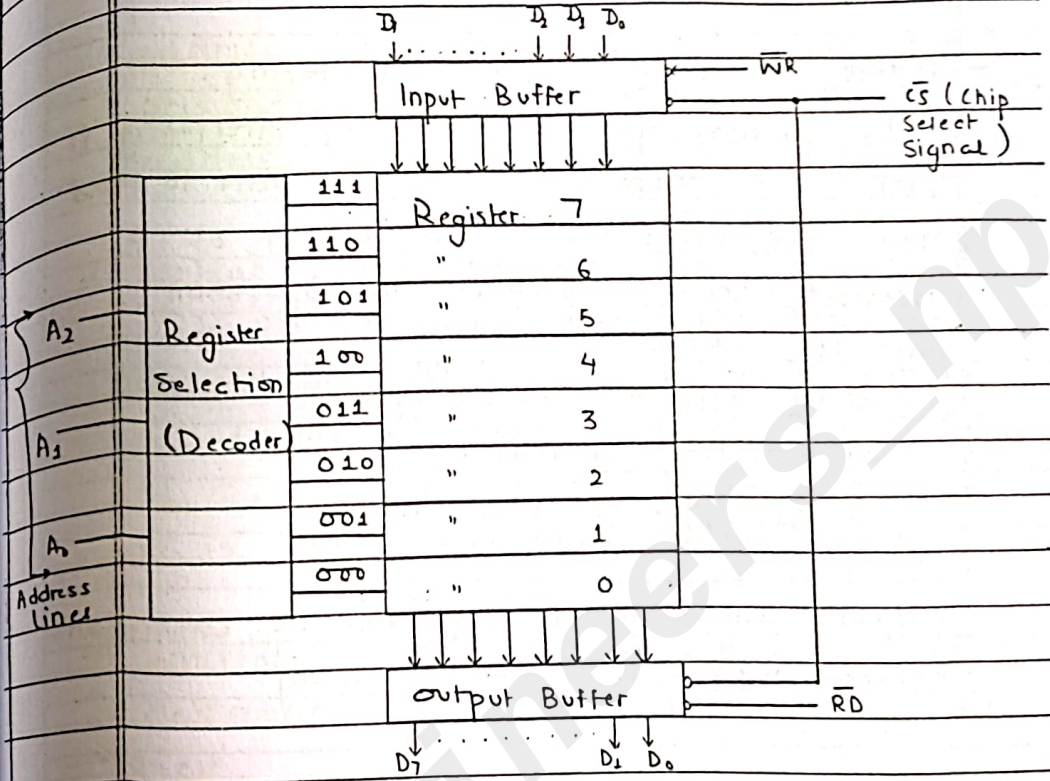


Fig: Internal structure of a 8X8 memory.

Internally, a memory consists of:

↳ address decoder

↳ Input buffer

↳ output buffer

↳ registers

along with data lines, addres lines & control signals.

The memory capacity of ony memory can be written as; $C = M \times N$

$= 2^K \times N$

Where, M → no. of memory locations

K → address bus width

N → Data bus width

For eg: 8×8 memory consists of
address bits = 3
Data bits = 8

Basic Concept of Memory Interfacing:

→ The primary function of memory interfacing is that the microprocessor should be able to read from and write into a give register of a memory chip. To perform the operation, the μP should

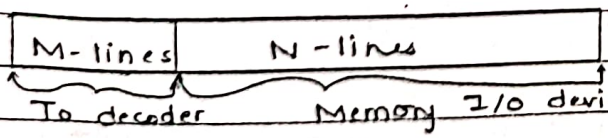
① be able to select the chip.
② Identify the register
③ Enable the appropriate buffer.

→ Memory Interfacing is the process of communication between microprocessor and memory.

Address Decoding

→ The process of generating chip select ($\overline{CS}$) signal using the address lines of a μP and a decoder or logic to interface memory or I/O devices with microprocessor is called address decoding.

→ The process of determining the range of address allocated by the μP during memory interfacing is called address decoding.



Address decoding is of two types:

1. Full address decoding:

If all the address lines of a μP system is used to process a memory or I/O devices, it is a full address decoding. Also known as unique or absolute address decoding since the address is unique.

2 Partial address decoding:

If all the address lines of the μP system is not used to address a memory or I/O, it is called partial address decoding. The address of the memory or I/O is not unique so it is called non-unique address decoding.

Interface a (2048 X 8) RAM with 8085 μP. Also determine the range of address.

→ Here,

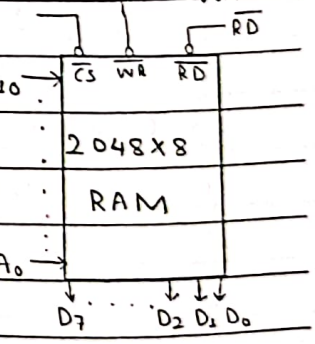
Memory capacity of RAM is:

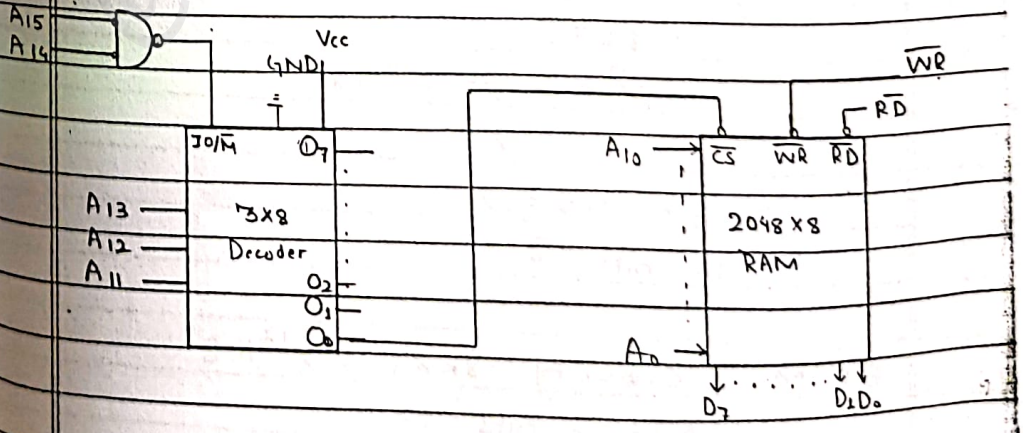
$C = 2048 \times 8$ Incomplete →

$= 2^{11} \times 8$

∴ No. of address lines = 11

No. of data lines = 8





Address Decoding

	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
Initial	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Address		0					0				0			0		
Final	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
Address		0					7				F				F	

The range of address is 0000H to 07FFH.

Interface 2KB RAM, 4KB EEPROM and 8KB EPROM with 8085 µP. Also determine the range of address. (with starting address PU 20 4000H).

→ Here,

Memory capacity of RAM is,

$= 2k \times 8$

$= 2^1 \cdot 2^{10} \times 8$

$= 2^{11} \times 8$

∴ Address bits = 11

Data bits = 8

Memory capacity of EEPROM is,

$= 4k \times 8$

$= 2^2 \cdot 2^{10} \times 8$

$= 2^{12} \times 8$

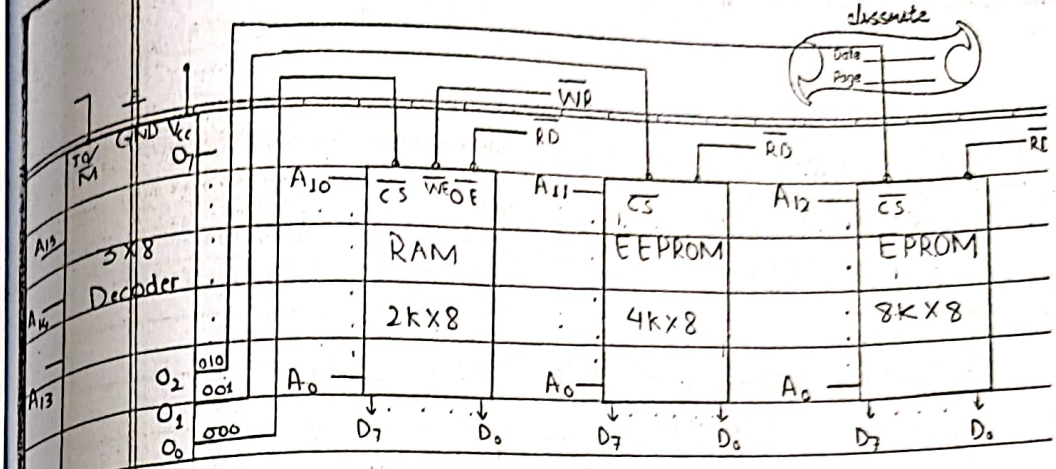
∴ Address bits = 12

Data bits = 8

EPOM : capacity = $8k \times 8 = 2^{13} \times 8$

∴ Address bits = 13

Data bits = 8



Address Decoding

	Address	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Range	
RAM	Initial	0	0	0	x	x	0	0	0	0	0	0	0	0	0	0	0	xx=00 0000H 07FFH	xx=01 0800H 0FFFH
RAM	Final	0	0	0	x	x	1	1	1	1	1	1	1	1	1	1	1	xx=10 1000H 17FFH	xx=11 1800H 1FFFH
EEPROM	Initial	0	0	1	x	0	0	0	0	0	0	0	0	0	0	0	0	For x=0 2000H	For x=1 3000H
EEPROM	Final	0	0	1	x	1	1	1	1	1	1	1	1	1	1	1	1	2FFFH	3FFFH
EPROM	Initial	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000H to	
EPROM	Final	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	5FFFH	

$n \times 2^n$

Design an address decoding circuit to interface 4K x 8 RAM with starting address 8000H.

→

Memory capacity of RAM is,

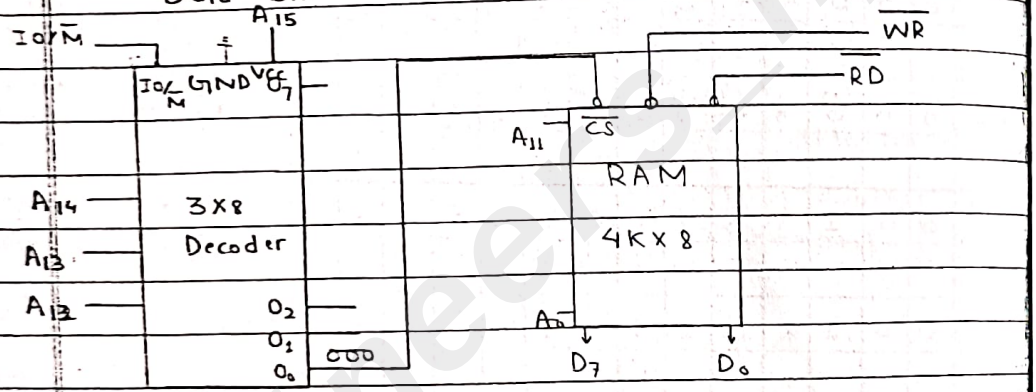
$= 4K \times 8$

$= 2^2 \cdot 2^{10} \times 8$

$= 2^{12} \times 8$

∴ Address bits = 12

Data bits = 8



	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A0	Range
Initial	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000H
Final	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	4FFFH

Interface 8KB RAM with 8086 MP. Also determine the range of address.

→ Memory capacity of RAM,

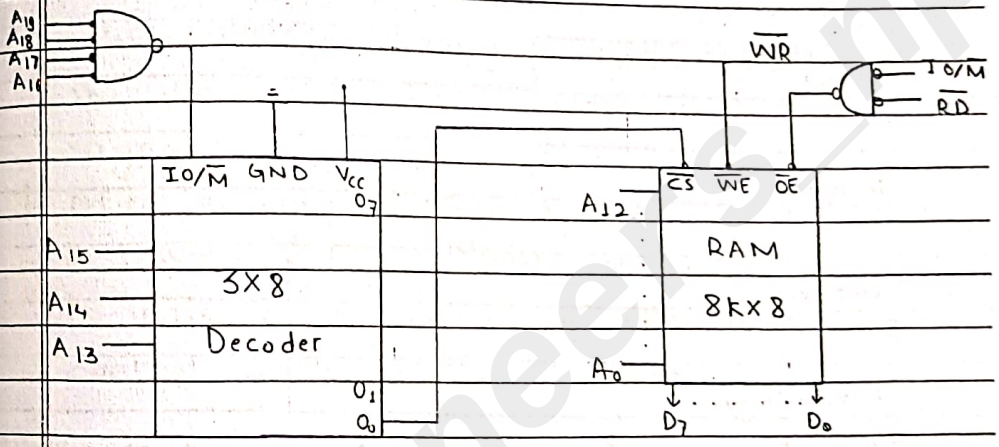
$= 8k \times 8$

$= 2^3 \cdot 2^{10} \times 8$

$= 2^{13} \times 8$

∴ Address bits = 13

Data bits = 8



Synchronous & Asynchronous Bus

↳ In a synchronous bus, the occurence of the events on the bus is determined by a clock.

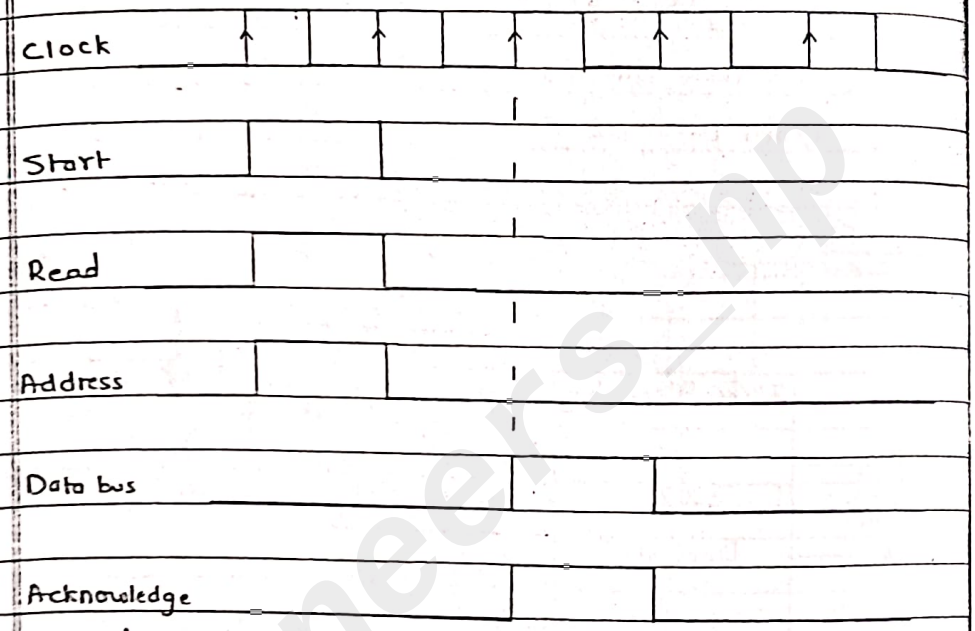


Fig: Synchronous memory read operation

Here, The CPU issues a START signal to indicate the presence of address and control information on the bus. Then it issues the memory read signal and places the memory address on the address bus. The addressed memory module recognizes the address and after a delay ~~on~~ of one clock cycle, it ~~pass~~ places data and acknowledgement signal on the buses. In this way the synchronous memory read operation is completed.

↳ The transition takes place at the leading edge of clock cycle.

↳ In sychronous bus, all devices are tied to a fixed rate & hence the system can't take

the advantage of device performance.

Asynchronous Bus:

In this bus, the timing is maintained in such a way that the occurence of one events on the bus follows and depends on the occurence of the previous events.

↳ Here, the CPU places the memory read (control) and address signals on the bus.

↳ The addressed memory module responds with the data and SSYNC (Slave synchronous signal)

↳ After allowing for these signals to stabilize it, it issues MSYNC (Master synchronous signal) to indicate the presence of valid address and control signal on the bus.

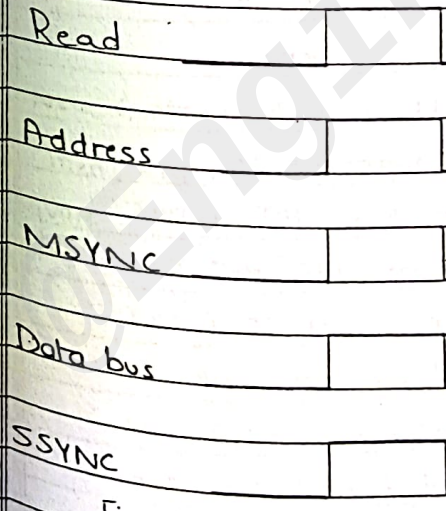


Fig: Asynchronous Bus

8254 Programming

1. Write a subroutine program to generate 1KHz square wave from counter-1 of 8254 PIT.

Soln:

Counter = Counter 1

Mode of operation = Mode 3: Square Wave Generator

Frequency of square wave = 1KHz

Assume,

Frequency of clock = 2MHz (max 10MHz)

Address of Counter - 1 = 81H

Address of counter 2 = 82H

Address of control register = 83H.

We know,

$$\text{Count} = \frac{\text{Frequency of clock}}{\text{Frequency of square wave}}$$

$$= \frac{2\text{MHz}}{1\text{KHz}} = \frac{2\times10^{6}\text{Hz}}{1\times10^{3}\text{Hz}}$$

$$= 2000$$

Now; Control word = 01110111 = 77H

Then, Required subroutine is as follows.

```
Square  MVI A, 77H ; Load control word in Accumulator
Wave:   OUT 83H ; Store control word in control Register
        MVI A, 00 ; Load lower byte of count in Accumulator
        OUT 81H ; Store lower byte of count in counter-1
        MVI A, 20 ; Load higher byte of count in Accumulator
        OUT 81H ; Store higher byte of count in counter-1
        RET ; Return to main program.
```

2. Write a subroutine program to generate a pulse of width 50 microseconds using 8254 PIT. (Take a frequency of clock as 2MHz and the counter 0).

Soln;

Counter = Counter 0

mode of operation = mode 1: Hardware Retriggerable one Shot.

Time Period of Pulse = 50 microseconds

Frequency of clock = 2MHz.

Assume,

Address of counter 0 = 80H

Address of counter 1 = 81H

Address of counter 2 = 84H

Address of control Register = 83H

We have,

$$\text{Frequency of pulse} = \frac{1}{\text{(Time Period of Pulse)}} = \frac{1}{50 \times 10^{-6}}$$

$$= \frac{10^6}{50} \text{ Hz}$$

Now,

$$\text{Count} = \frac{\text{(Frequency of clock)}}{\text{Frequency of pulse}}$$

$$= \frac{2\text{MHz}}{\left(\frac{10^6}{50}\right)\text{Hz}} = \frac{2 \times 10^6}{\left(\frac{10^6}{50}\right)} = 100$$

Thus, Count = 0100

Now,

Control Word = 00110011 = ~~35H~~ 33H

Hence,

→

Required Subroutine is as follows:

```
Pulse: MVI  A, 35H
       OUT  83H ; Store control word in Control Register
       MVI  A, 00
       OUT  80H ; Store lower byte of a count in
                  counter 0
       MVI  A, 01
       OUT  80H ; Store higher byte of a count in
                  counter 0
       RET
```

Chapter 5 - Interrupts

→ An interrupt is a signal that a peripheral board sends to the central processor in order to request attention
→ In response to the interrupt, the processor stops what it is currently doing and executes the Interrupt Service Routine (ISR) in order to handle the interrupt.
→ When the execution of service routine is terminated, the original process may resume its previous operation.
→ The interrupt is initiated by an external device and is asynchronous i.e it can be initiated at any time without reference to the system clock. However, the response to an interrupt request is directed or controlled by the microprocessor.
→ Interrupts are primarily issued on:
- Initiation of I/O operation
- Completion of I/O operation.
- Occurance of Hardware / Software Errors.

Sources of Interrupts
1. Processor (Internal) Interrupts,
2. Hardware (External) Interrupts, and
3. Software Interrupts.

1. Processor (Internal) Interrupts
→ This interrupt is caused by some error conditions produced in the processor internally during the execution of instructions.
→ For example, the various conditions like - divide by zero, register overflow, stack overflow, etc generate processor interrupts.

2 Hardware (External) Interrupts :

→ This interrupt is caused by external signal applied to the Interrupt pins.

→ For example, in 8086 microprocessors, the external requests are caused through NMI (Non-Maskable Interrupts) and INTR (Interrupt Request) Pins.

3 Software Interrupts

→ This interrupt is caused by the available interrupt instructions.

→ For example, in 8086 microprocessor. INT instructions eg: INT 21H instruction causes DOS interrupts to execute. Similarly in 8085 RST0 - RST7 instructions cause software interrupts.

Classification of Interrupts

→ Interrupts can be broadly classified into :

1. Maskable and non-maskable Interrupts,
2. Vectored and non-vectored Interrupts

1 Maskable and non-maskable Interrupts

i) Maskable Interrupts

→ These interrupts can be enabled or disabled by executing instructions such as EI (Enable Interrupts) DI (Disable Interrupts) and SIM (Set Interrupt Masked) Instructions.

→ Thus those interrupts which can be disabled or delayed are called maskable interrupts.

→ In 8085 microprocessor, the interrupt requests occured through INTR, RST 7.5, RST 6.5 and RST 5.5 are maskable interrupts. Similarly, in 8086 microprocessor,

INTR is the maskable interrupt pin.

ii) Non-maskable Interrupts :

→ These interrupts cannot be enabled or disabled using instructions.

→ Thus, those interrupts which cannot be disabled or delayed or blocked by the microprocessor are called the maskable interrupts.

→ For example, in 8085 microprocessor, TRAP is the only non-maskable interrupt pin and in 8086 microprocessor NMI is non-maskable interrupt pin.

→ Normally, the emergency circuits like-power protection circuits are connected to non-maskable interrupt pins.

2. Vectored and Non-Vectored Interrupts

i) Vectored Interrupts.

→ Those interrupts whose ISR address is already known to the microprocessor are called vectored interrupts.

→ For example, TRAP, RST 7.5, RST 6.5 and RST 5.5 are vectored interrupts in 8085.

8085 Interrupts	Call location (or) Interrupt vector
TRAP	0024H
RST 7.5	003CH
RST 6.5	0034H
RST 5.5	002CH

ii) Non-vectored Interrupts.

→ In this interrupts, the address of the Interrupt Service Routine (ISR) is not already known to the microprocessor and needs to be supplied externally.

→ Normally, the interrupting device or an external hardware needs to supply the interrupt vector (i.e starting address of ISR) to the microprocessor.

→ For example, in 8085 and 8086 microprocessor INTR (Interrupt Request) is the non-vectored interrupt pin.

8085 Interrupts

→ 8085 microprocessor has 5 interrupts pins. They are

1. INTR
2. RST 7.5
3. RST 6.5
4. RST 5.5
5. TRAP

1. INTR (Interrupt Requests)

→ It is the maskable and non-vectored pin in 8085 microprocessor.

→ The interrupt request in IMR can be enabled or disabled using EI (Enable Interrupt) and DI (Disable Interrupt) instructions.

→ The request in INTR pin is acknowledged by microprocessor using INTA (Interrupt Acknowledge) pin.

2. RST 7.5 :-

→ It is the hardware reset pin in 8085 microprocessor.
→ It is highest priority reset pin among all reset pins.
→ It is maskable and vectored interrupt, that can be enabled or disabled using SIM (Set Interrupt Mask) instruction.

3. RST 6.5 :-

→ It is the reset pin in 8085 microprocessor with second highest priority.
→ It is also maskable and vectored interrupt and also can be enabled or disabled using SIM instruction.

4. RST 5.5 :-

→ It is also the reset pin in 8085 microprocessor with lowest priority.
→ It is also maskable and vectored interrupt and can be enabled or disable using SIM instruction.

5. TRAP

→ It is the only non-maskable interrupts in 8085 ~~Pri~~ up.
→ It has highest priority among all interrupt pins.
→ It is the vectored interrupt and is generally used for critical events such as power failure, emergency shut off, etc.

Priority of 8085 Interrupts

→ In 8085, TRAP pin has the highest and INTR pin has the lowest priority. The priority of all interrupt pins is as follows:

1. TRAP	Priority decreases ↓
2. RST 7.5	
3. RST 6.5	
4. RST 5.5	
5. INTR	

8086 Interrupts and Interrupt Vector Table (IVT):

imp Interrupt Vector Table (IVT)

- In 8086, the first 1kB of memory from 00000H to 003FFH is set aside as a table for storing the starting address of the Interrupt Service Routine (ISR) or Interrupt Service Procedures. This table is called Interrupt Vector Table (IVT)
- → Since 4 bytes are required to store the CS and IP values for each interrupt service procedures, the table can hold the starting addresses for upto 256 interrupt procedures.
- → The starting address of an interrupt-service procedure is often called the interrupt vector or the interrupt pointer, so the table is referred to as interrupt-vector table (IVT) or interrupt-pointer table (IPT).
- Figure shows the IVT indicating the arrangement of 256 interrupt vectors.

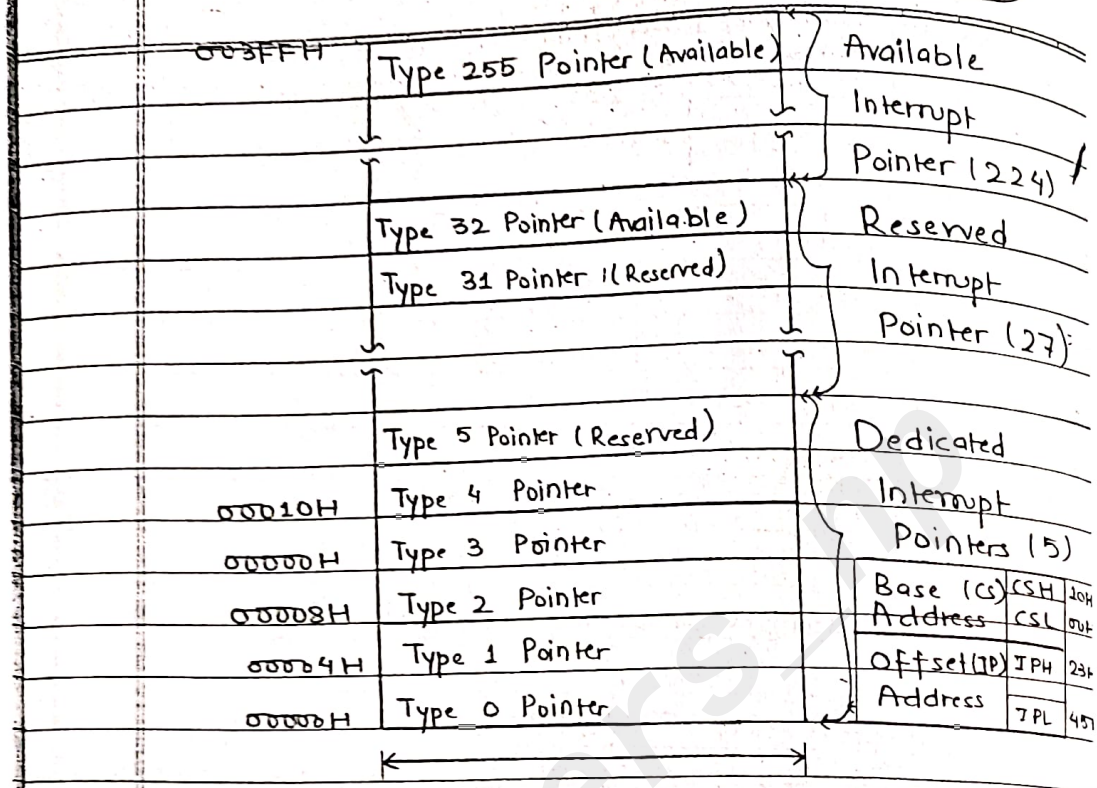


Fig: 8086 Interrupt Vector Table

8086 Interrupt Types

1. Software Interrupts

→ These interrupts are initiated by executing an interrupt instruction. For example, INT 21H is the software interrupt in 8086 that is the DOS interrupt and provides more than 80 different services.

2. Hardware Interrupts.

→ These interrupts are initiated by applying an electrical signal to the interrupt pins of the processor.

→ For example, INTR and NMI pin of 8086 microprocessor for providing external hardware interrupts to 8086 microprocessor.

3. Pre-defined or Dedicated Interrupts:

→ These include first five interrupts in 8086 Interrupt Vector Table and are internal interrupts.

ⓐ Type 0: Divide-Error Interrupt

→ This interrupt is issued by INT 00H instruction.

→ Type 0 interrupt occurs whenver the result of the division overflows or whenever an attempt is made by divide by zero.

ⓑ Type 1: Single-step Interrupt.

→ In this interrupt, microprocessor executes in single-step mode, i.e goes for debugging after execution of single step of instruction.

→ For this type, Trap Flag (TF) should be set.

→ Thus, this type of interrupt is useful for monitoring & debugging programs.

ⓒ Type 2: Non mask-able Interrupt:

→ This interrupt is initiated when NMI pin of MPU receives a low to high transition. This interrupt is normally used for catastrophic conditions such as power failure.

ⓓ Type 3: Break-point Interrupt:

→ In this interrupt, a break-point is set and is non-maskable.

→ When we insert a breakpoint and then goes to the desired breakpoint subroutine for debugging.

(e) Type 4: Overflow Interrupt

→ Overflow interrupt can be generated using INT 4 or INTO (Interrupt overflow) instruction.

→ Overflow flag should be set in order to execute INTO instruction.

8086 Interrupts Priority:

→ The priority of various 8086 interrupts are as follows

1. Divide Error, INTO
2. NMI
3. INTR
4. Single-Step
5. Internal Interrupts (Except Single Step)

Priority decrease.

Servicing Multiple Interrupts (Prioritising Multiple Interrupt

→ There are mainly two ways of servicing multiple interrupts. They are:

1. Polled Interrupts (Equal Priority Devices)
2. Chained (Vectored) Interrupts

1. Polled Interrupts (Equal Priority Devices)

→ In this method, there is one-common branch address for all interrupts. The program that takes care of interrupts begins at the branch address and polls the interrupt sources in sequences.

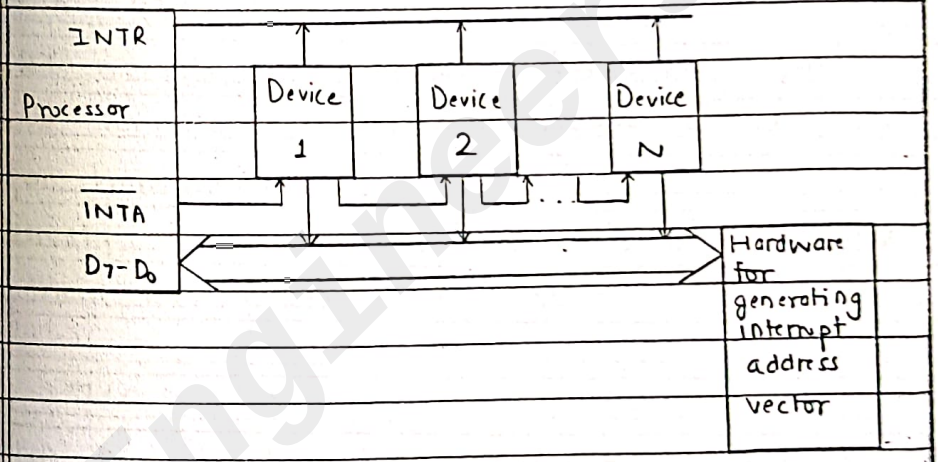
→ The order in which they are ~~bored~~ tested determines the priority of each interrupt.

→ The highest priority is tested first, and if its interrupt signal is on, control branches to the service routine for this source. Otherwise the next lower priority source is tested and so on.

→ Polled interrupts are very simple but for large number of devices, polled interrupts are slower. Moreover, as these interrupts are handled using software, they are slower than vectored interrupts.

2. Chained (Vectored) Interrupts

→ This is the hardware concept of handling multiple interrupts. In this technique, the devices are connected in a chain function as shown in figure below for setting up the priority.



→ Here, the device with highest priority is placed in the first position, followed by the lowest priority devices.

→ Suppose that one or more devices interrupt the processor at a time. In response, the processor saves its current status and then generates an interrupt acknowledge ($\overline{INTA}$) signal to the highest priority device, which is device 1 in above fig. If this device has generated interrupt it will

accept the $\overline{INTA}$ from the processor, otherwise it passes $\overline{INTA}$ on to the next devices until the $\overline{INTA}$ is accepted by the interrupting device. Once accepted, the device provides a means to the processor for finding the interrupt address vector using external hardware.

→ It is faster than polled interrupt.

Chapter - 6 : Input - Output Interfaces

① Serial I/O Standards : 8251A USART
② 8259A PIC
③ 8255A PPI
④ 8254 PIT
⑤ DMA

1. Serial I/O Standards : 8251A USART (Universal Synchronous Asynchronous Receiver Transmitter)

→ The 8251A is a programmable chip designed for synchronous and asynchronous serial data communication, packaged in a 28-pin DIP.

→ The USART accepts data characters from CPU in parallel format and converts them into a continuous serial data stream for transmission.

→ Simultaneously, it can receive serial data streams and converts them into parallel data character for CPU.

* Serially I/O :

→ Specially, there are two basic approaches for serial data communication. They are:

Ⓐ Software-controlled I/O (eg: using SID, SOD pins, RIM & SIM instructions in 8085).

Ⓑ Hardware-controlled I/O (using 8250 UART, 8251 USART)

* Serial Data Standards (protocols)

1. RS232C
2. BISYNC

Qn. Explain RS232C standard with necessary conditions.

Block Diagram of 8251A USART:

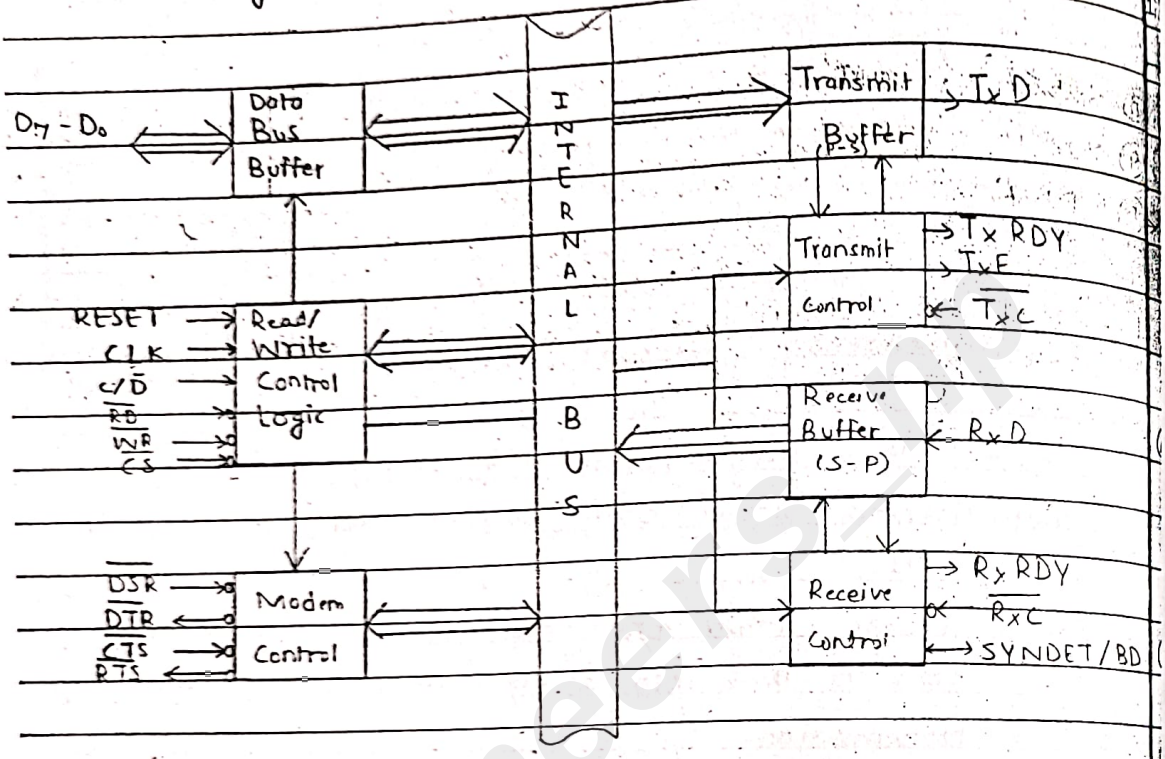


Fig: Functional Block Diagram of 8251A USART

Explanation:

Above functional block diagram of 8251A USART includes 5 selections which are as follows:

1. Read/Write Control Logic and Registers.
2. Transmitter Section
3. Receiver Section
4. Data Bus Buffer Section
5. Modem Control Section.

1. Read/Write Control Logic and Registers:

→ This control logic interfaces the chip with the mpu, determines the functions of the chip according to the control word in its register, and monitors the data flow.

→ This section includes R/W control logic, six input signals, control logic and three buffer registers: data registers, control register and status register.

The various signals to control logic are as follows:

Ⓐ $\overline{CS}$ (Chip Select)

- When this signal goes low, the 8251A is selected for by the mpu for communication.

Ⓑ C/$\overline{D}$ (Control/Data)

- When this signal is high, the control register or the status register is addressed, when it is low the data buffer is addressed.

Ⓒ $\overline{WR}$ (Write)

- When this signal goes low, the mpu either writes in the control register or sends output to the data buffer.

Ⓓ $\overline{RD}$ (Read)

- When this signal goes low, the mpu either reads a status from the status register or accepts data from the data buffer.

CS	C/D	RD	WR	Function
0	1	1	0	MPU writes instructions in the ctrl reg
0	1	0	1	MPU reads status from the status reg
0	0	1	0	MPU (works) outputs data to the data buffer (read)
0	0	0	1	MPU accepts data from data buffer
1	x	x	x	8251A USART is not selected

(e) RESET (Reset)

- A high on this input resets the 8251A and forces it into the idle mode.

(f) CLK (Clock):

- This is the clock input, usually connected to the system clock.
- This clock doesnot control either the transmission or the reception rate.

*Control Register

- This 16-bit control register for a control word consis of two independent bytes, the first byte is called the mode instruction (word) and the second byte is called the command instruction (word).
- This register can be accessed as an output port when C/D pin is high.

* Status Register
- This input register checks the ready status of the peripheral.
- This register is addressed as an input port when C/D̄ pin is high.

② Transmitter Section:
- This section accepts parallel data from the mpu and converts them into serial data.
- It has two registers: a buffer registers to hold eight bits and an output register to convert eight bits into a stream of serial bits.
- This section transmits data on the TxD pin with the appropriate framing bits (start and stop bit).
- Different signals associated with Transmitter Section are as follows:

@ TxD (Transmit Data):-
- Serial bits are transmitted on this line.

ⓑ TxC (Transmitter Clock):-
- This signal controls the rate at which bits are transmitted by the 8251A USART.
- The clock frequency can be 1, 16 or 64 times the baud.

ⓒ TxRDY (Transmitter Ready):-
- When this output signal is high, it indicates that the buffer register is empty and the USART is ready to accept the byte.

bit rate → bits per second
baud rate → symbols per second

ⓓ TxE (Transmitter Empty)
- When this output signal is high, it indicates that the output register is empty.

③ Receiver Section:
- This section accepts the serial data on the RxD line from a peripheral and converts them into parallel data.
- This section has two registers: the receiver input register and the buffer register.
- When the RxD line goes low, the control logic assumes it as start bit, waits for half time, and samples the line again.
- If the line is still low, the input register accepts the following bits, forms a character, and loads it into the buffer register.
- The various signals on receiver section are:

ⓐ RxD (Receive Data):
→ Bits are received serially on this line and converted into a parallel byte in the receiver input register.

ⓑ RxC (Receive Clock):
→ This is a clock signal that contains the rate at which bits are received by the USART.
→ In the asynchronous mode, the clock can be set to 1, 16 or 64 times the baud.

⑤ RxRDY (Receiver Ready):

→ This output signal goes high when the USART has a character in the buffer register and is ready to transfer it to the mPU.

Initializing the 8251A:

~~To implement the serial communication, the mPU must~~

④ Data Bus Buffer Section:

→ This section holds 8-bit data (D_7-D_0) before data is written to transmitter section or after data is read from receiver section.

→ It transmits data to and from mPU and 8251A USART.

⑥ Modem Control Section: and

→ This section checks for the handshake signals, controls the communication by checking the readiness, termination of the data transfer.

Initializing the 8251A:

- To implement the serial communication, the mPU must inform the 8251A of all details, such as mode, baud, stop bits, parity, etc.
- Items set by mode instructions are;
  - → Synchronous / Asynchronous mode,
  - → Stop bit length (asynchronous mode)
  - → character length,
  - → parity bit
  - → Baud rate factor (asynchronous mode), etc.

- Similarly items set by common word are:
  - → Transmit Enable / Disable,
  - → Receive Enable / Disable,
  - → DTR, RTS Output of Data,
  - → Resetting of error flag, etc.
- Thus, before data transfer, a set of control words must be loaded into 16-bit control register of 8251A.
- The control words are divided into two formats:
  1. Mode word,
  2. Common word

1. Mode word

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S_2	S_1	EP	PEN	L_2	L_1	B_2	B_1

Stop-bit length

0	1	0	1
0	0	1	1
Inhibit	1 bit	1.5 bits	2 bits

Baud rate factor

0	1	0	1
0	0	1	1
SYNC Mode	1X	16X	64X

Character length

0	1	0	1
0	0	1	1
5 bits	6 bits	7 bits	8 bits

Parity Check

0	1	0	1
0	0	1	1
Disable	Odd Parity	Disable	Even Parity

Fig: Bit Configuration of Mode Word

Date:
Page 3

2 Command word

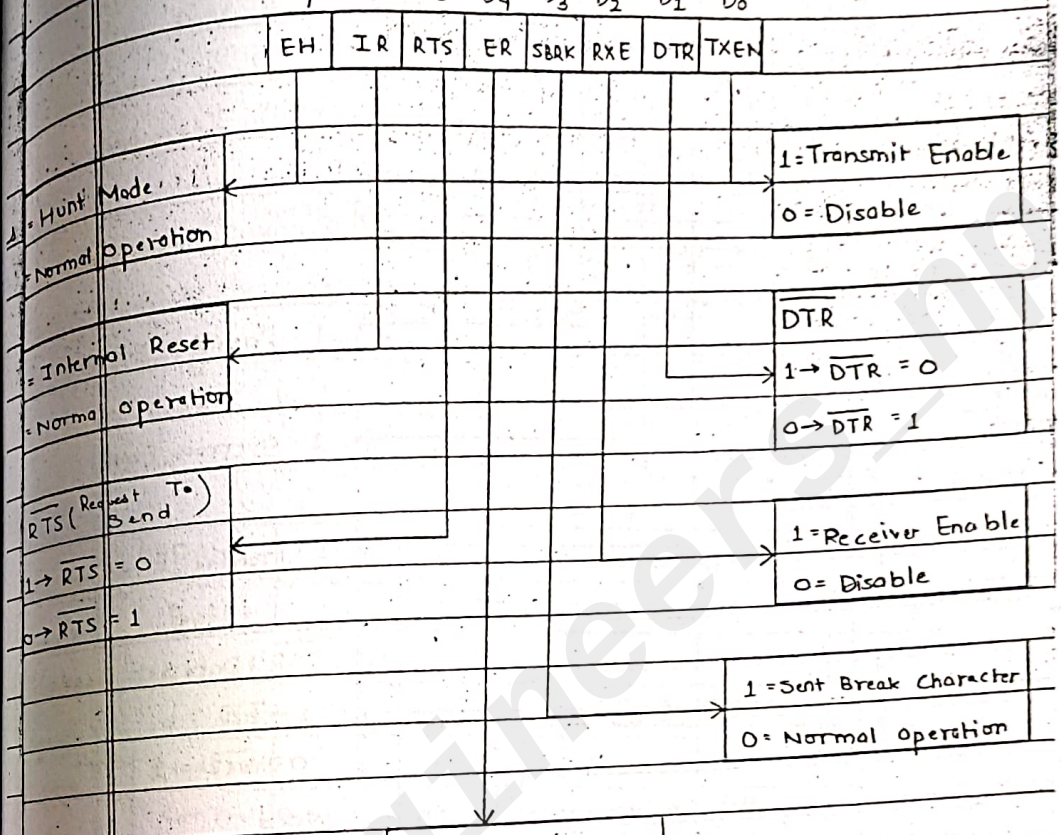


Fig: Bit configuration of Command Word.

Status Word

It is possible to see the internal status of 8251A by reading a status word.

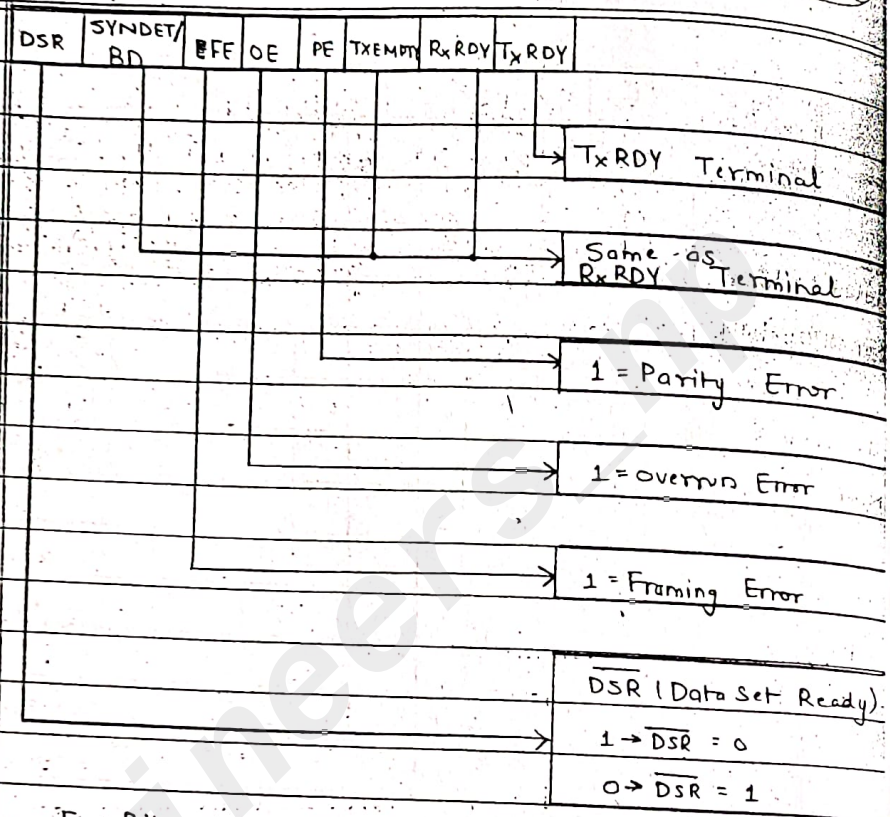


Fig: Bit configuration of Status Word

2.

V.Imp # 8259A PIC (Programmable / Priority Interrupt Controller)

- The 8259A is a programmable / priority interrupt controller that is used to manage and resolve the interrupts issued from a number of input, output devices at the same time.
- It is a 28-pin IC that is designed to work with Intel 8085, 8086 and 8088 microprocessors.

Normally, most of the microprocessors have limited number of pins for handling the interrupt request from peripheral devices. For example, if we are working with 8086, we have only two interrupt input pins: NMI and INT. If we save NMI for power failure interrupt, this leaves only one interrupt pin for all other application which is difficult to manage.

8259A is a most common PIC that accepts interrupt requests from upto 8 I/O devices (in single mode) and upto 64 devices in cascaded mode.

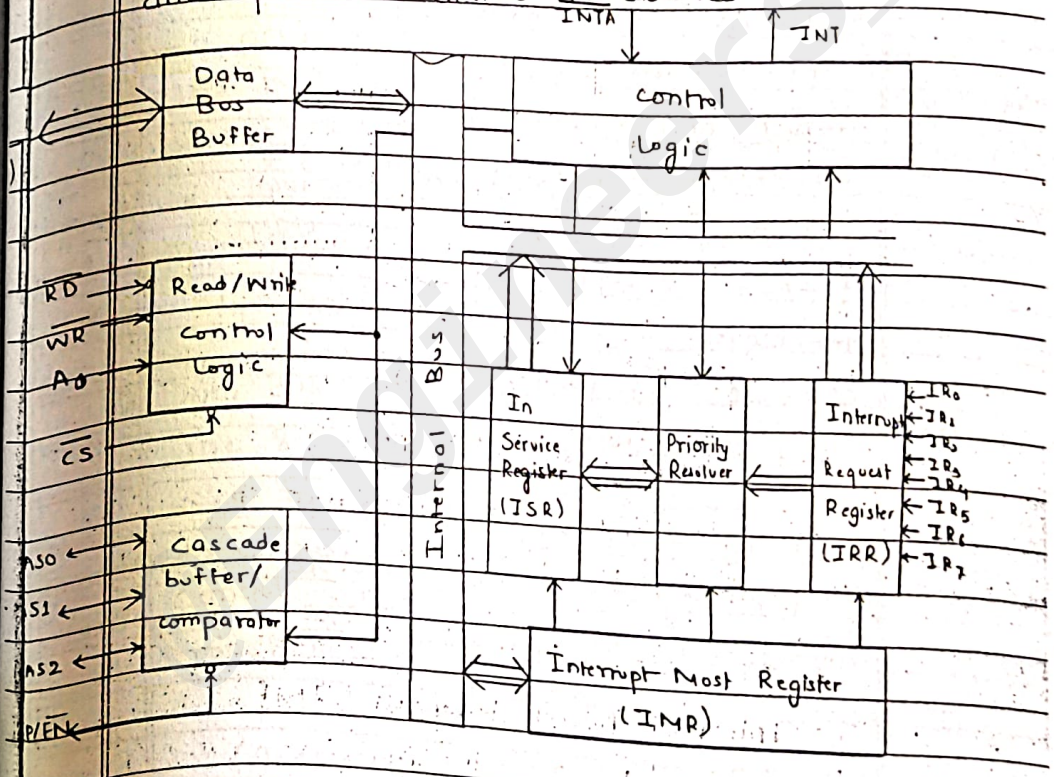


Fig: Block diagram of 8259A PIC

8259A Block Diagram explanation :

→ The functional block diagram of 8259A consists of following four sections :

1. Interrupt and Control Logic Section,
2. Data Bus Buffer Section,
3. Read/Write Control Logic Section, and
4. Cascade Buffer / Comparator Section

1. Interrupt and control Logic Section

- This section consists of 5 sub-sections which are as follows:

ⓐ Interrupt Request Register (IRR):

→ This 8 bit register is used to store the status of all 8 interrupt input lines (IR_0-IR_7) which are requesting service. The 8 interrupt inputs are set corresponding bits of interrupt request registers.

ⓑ In-Service Register (ISR):

→ This register is used to store the information about the interrupt levels that are currently being serviced.

ⓒ Interrupt Mask Register (IMR):

→ This register stores the masking bits of interrupt lines.

→ Masking of higher priority input will not affect the request lines of lower priority.

→ This register can be programmed by OCW1 (Operation Command word)

Date
Page 4

ⓓ Priority Resolver:

→ This logic block determines the priorities of the bit set in the IMR.

→ The bit corresponding to the highest priority interrupt is selected and stored into corresponding bit of ISR during $\overline{INTA}$ pulse.

ⓔ Control Logic Block:

→ This block has two pins: INT (Interrupt) as an output and $\overline{INTA}$ (Interrupt Acknowledgement) as an input.

→ The INT pin is connected to Interrupt Request pin (eg: INTR in 8085) and $\overline{INTA}$ pin is connected to Interrupt Acknowledgement pin (eg: $\overline{INTA}$ in 8085) of MPU.

2. Data Bus Buffer Section:

→ The 8-bit tri-state, bidirectional 8-bit buffer is used to interface the 8259A to the system data bus of MPU.

→ Control words and status informations are transferred through Data Bus Buffer.

3. Read/Write Control Logic:

- The function of this block is to accept output commands to from the CPU.
- It contains the Initialization Command Word (ICW) and Operation Command Word (OCW) registers which store the various control formats for the device operation and setup 8259A to operate in various modes.

- The different input lines to Read/Write control logic are as follows:

① $\overline{CS}$ (Chip Select):

- A low on this input enables the 8259A. NO reading or writing will occur unless the device is selected

② $\overline{WR}$ (Write):

- A low on this input enables the CPU to write control words (ICWs and OCWs) to the 8259A.

③ $\overline{RD}$ (Read):

- A low on this input enables the 8259A to send the status of IRR, ISR, IMR or the interrupt level on the data bus.

(4) A_0

→ This signal is directly tied to one of the address lines of MPU.

→ This signal is used in conjunction with $\overline{WR}$ and $\overline{RD}$ signals to write commands into various command registers, as well as reading the various status registers of the chip.

4) Cascade Buffer / Comparator Section:

→ This section is used to operate 8259A in cascaded mode, in which it can handle interrupt requests from maximum upto 64 I/O devices.

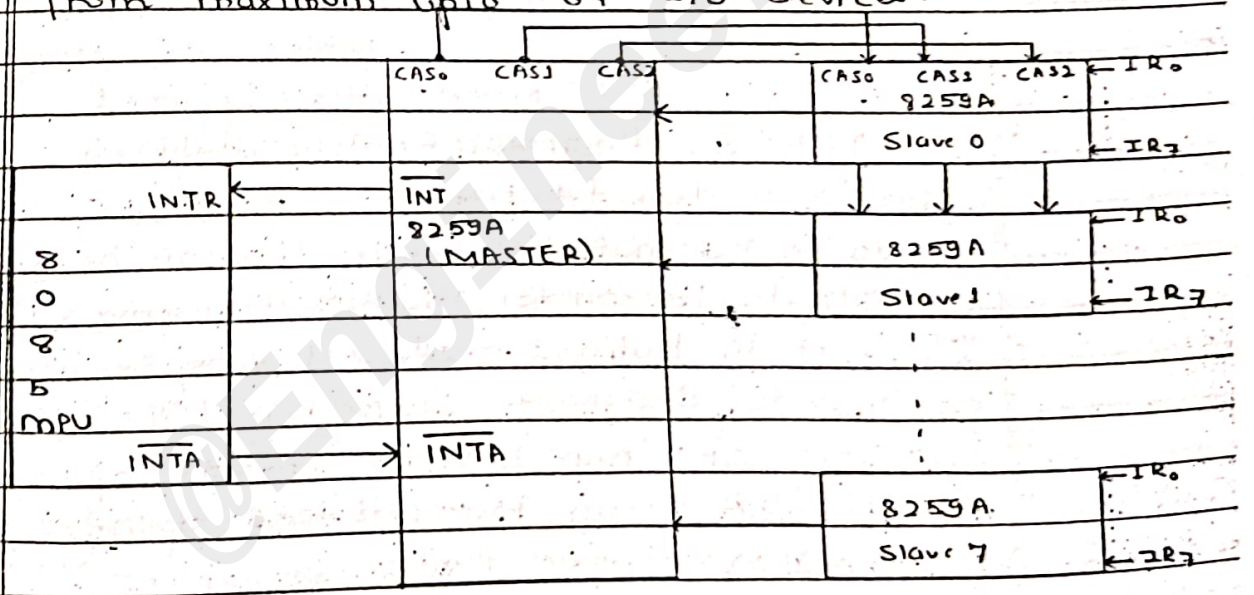


Fig: Cascading 8259As

As shown in above fig to operate in cascade mode, multiple 8259As chip are configured in master-slave pattern.

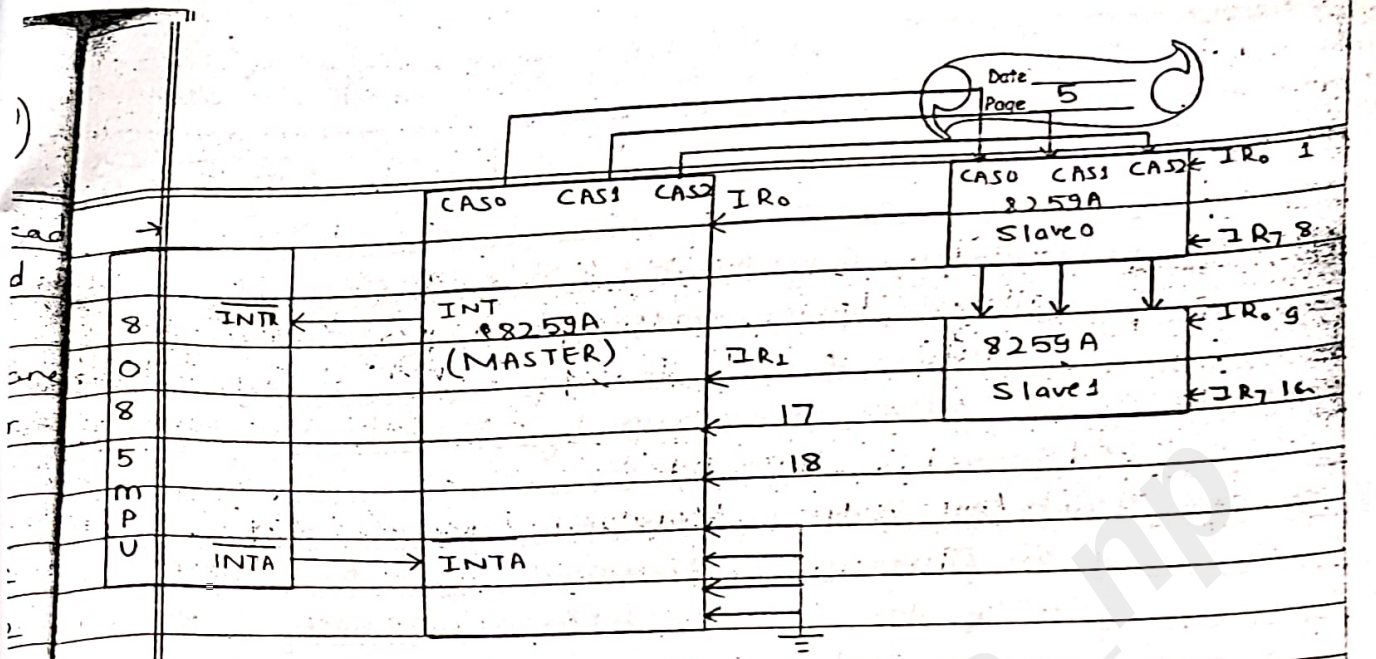
The associated three pins: CAS2 - CAS0 are outputs when the 8259A is used to master and are inputs when 8259A is used as slave.

The master puts out an identification code of 3 bits (8 devices) on the CAS0 to CAS2 lines and as per the combn the corresponding slave is selected. Eg: If CAS2-CAS0 have value 000 resp then 8259A slave 0 is selected.

* $\overline{SP}/\overline{EN}$ (Slave Program / Enable Buffer):
  - → This is a dual function pin.
  - → When in the buffered mode, it can be used as an output to control buffer transceivers ($\overline{EN}$).
  - → When not in buffered mode, it can be used as an input to designate a master. (SP = 1 / SP = 0)
  - → Buffered and non-buffered mode of operation of 8259A can be specified during initialization.

Q. How can you handle 18 interrupt requests using 8259A · PICs ?

Important



Modes of operation of 8259A

There are various operating modes of 8259A which are as follows:

1. Fully nested Mode (FNM),
2. Specially Fully Nested Mode (SFNM),
3. Rotation Priority Mode
   - ⓐ Automatic Rotation
   - ⓑ Specific Rotation
4. Special Mask Mode.
5. Polled Mode.

1. Fully Nested Mode:

- It is the default mode set after initialization i.e this mode is entered after initialization, unless another mode is programmed or changed through OCW.
- The interrupt requests are ordered in priority from 0 to 7 (0 highest and 7 lowest priorities).

2. Specially Fully Nested Mode (SFNM):
- used in cascade connection,
- allows further interrupt requests from slave
- SFNM is set up by ICW4 during initial

3. Rotation Priority Mode:
@ Automatic Rotation (Equal Priority Devices)
- In this mode, a device after being serviced receives the lowest priority, so a device requesting an interrupt will have to wait, in the worst case until each of 7 other devices are serviced almost once.
Example, if before rotation IR4 has the highest priority & currently serviced.

IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
0	0	0	1	0	0	0	0

Now, After rotation (IR4 was serviced, all other priorities rotated correspondingly) as,

IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
2	1	0	7	6	5	4	3

(b) Specific Rotation (Specific Priority)
- The programmer can change priorities by programming the bottom priority and thus fixing all other priorities i.e if IR5 is programmed as the bottom priority device, then IR6 will have the highest one.

- The set priority command is issued in OCW2, where R=1, S=1, L0-L2 is the binary priority level code of the bottom priority device.

3. Special Mask Mode:
   - In this mode, when mask bit is set in OCW1, it inhibits further interrupts at that level and enables interrupts from all other levels (lower as well as higher) that are not masked.
   - Thus, any interrupts maybe selectively enabled or masked by loading the mask register.
   - The special mask mode is set by OCW3.

4. Polled Mask:
   - The poll command is issued by setting P="1" in OCW3.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
I=1	–	–	–	–	W2	W1	W0

Here, W0-W2: Binary code of the highest priority level requesting service.
I=1, sets in Polled mode, if there is interrupt.

3. 8255A Programmable Peripheral Interface (PPI)

- The 8255A is a widely used, programmable, parallel I/O device that provides three 8-bit input/output ports in one 40-pin IC-package.
- It can be programmed to transfer data under various conditions, from simple I/O to interrupt.
- It is an important general-purpose I/O device that can be used with almost any microprocessor.
- It has 24 I/O pins that can be grouped primarily into two 8-bit parallel ports A and B with the remaining 8-bits as port C.
- The eight bits of port C can be used as an individual bits or can be grouped into two 4-bit ports: Port ~~Copper~~ C_{UPPER} (C_U) & Port C_{LOWER} (C_L).
- Basically the functions of 8255A can be classified into two modes:
  1. Bit Set/Reset (BSR) Mode; and
  2. I/O Mode
- The BSR mode is used to set or reset the bits in port C.
- The I/O mode is further divided into three modes:
  ① Mode 0, ② Mode 1, and ③ Mode 2
- In Mode 0, all port functions as simple I/O ports.
- In Mode 1, ports A and B use some bits from port C as handshake signals. In the handshake mode, two types of I/O data transfer can be implemented: status check and interrupt.

- In Mode 2, port A can be setup for bidirectional data transfer using handshake signals, from port C and port B can be setup either in mode 0 or mode 1.

Various Applications :

1. Printer Interface,
2. Keyboard and display interface,
3. A/D Converter and D/A converter interface,
4. Floppy disk interface, etc.

Why 8255A is used ?

- to expand the I/O ports.
- no use of external combinational logic circuit.

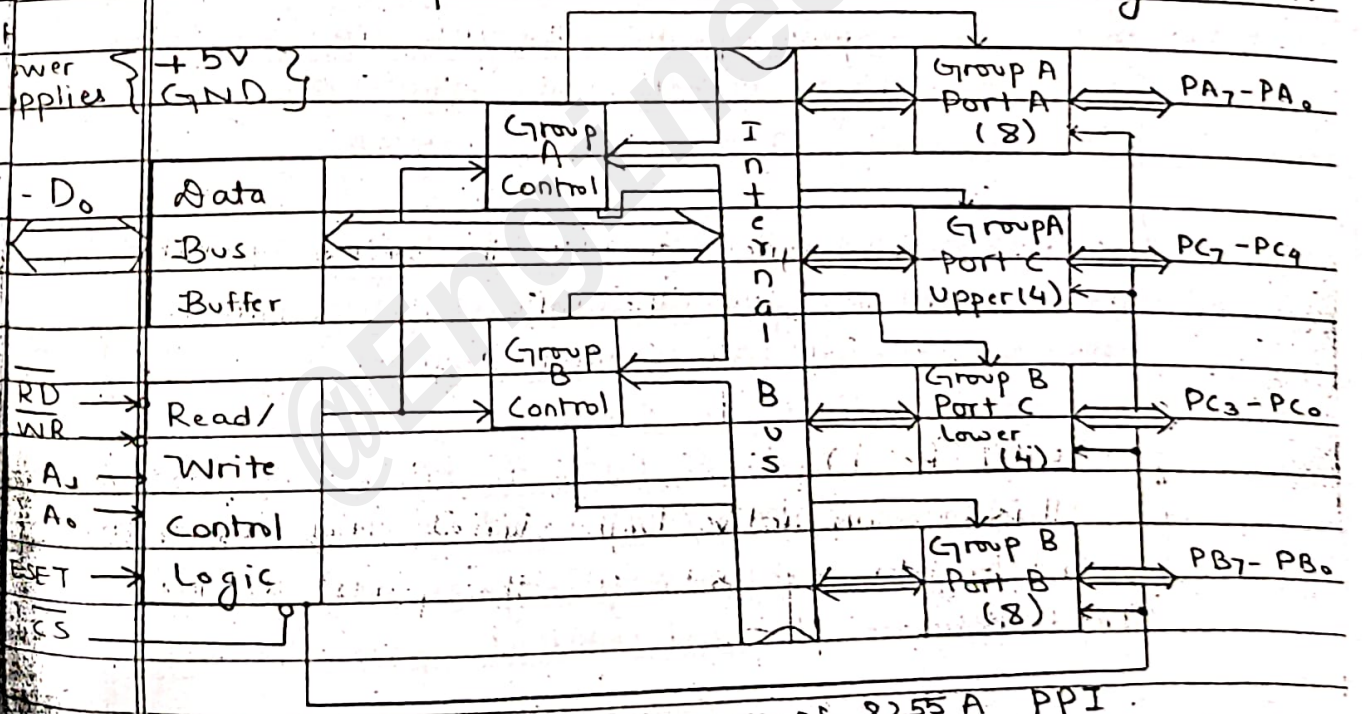


Fig: Functional Block Diagram of 8255A PPI.

Internal Block Diagram of 8255A PPI

Description

→ The different blocks included in the functional block diagram of 8255A are as follows:

1. DATA BUS BUFFER:

→ This is 8-bit bidirectional buffer used to interface the internal data bus of 8255A with system data bus of MPU.

2. READ/WRITE CONTROL LOGIC:

→ It accepts address and control signals from the microprocessor.

→ The control section has 6 lines which are follo

. $\overline{RD}$ (Read):

The control signal enables the Read operation. When this signal is low, the MPU reads data from a selected I/O port of 8255A.

. $\overline{WR}$ (Write):

The control signal enables the write operation. When this signal goes low, the MPU writes into a selected I/O Port or Control Register.

. RESET (Reset):

This is an active high signal that clears the control register and sets all the ports in the Input mode.

- $\overline{CS}$, A_0 and A_1 : Select

These are the device signals. $\overline{CS}$ is connected to decoded address and should be low to select 8255A chip and A_0 and A_1 lines are address lines generally connected to mpu address lines A_0 and A_1 respectively.

$\overline{CS}$	A_1	A_0	Selection
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control Register
1	x	x	8255A is not selected

3. Group A Control:
- This control block controls port A and port C upper
- It accepts control signals from the control word and forwards them to respective ports.

4. Group B Control:
- This control block controls port B and port C lower.
- It accepts control signals from the control word and forwards them to the respective ports.

5 Port A, Port B and Port C :
- These are 8-bit bidirectional ports.
- They can be programmed to operate in various modes as follows:

Port	Mode 0	Mode 1	Mode 2
Port A	Yes	Yes	Yes
Port B	Yes	Yes	No (Mode 0 or
Port C	Yes	No (Handshake Signals)	No (Handshake Signals)

Control Word of 8255A PPI :

- The content of control register is called control word.
- It specifies I/O functions for each port.
- The control register can be accessed to write a control word when $A_1A_0 = 11$. This register is not accessible for Read operation.

- Control Word for I/O mode :-

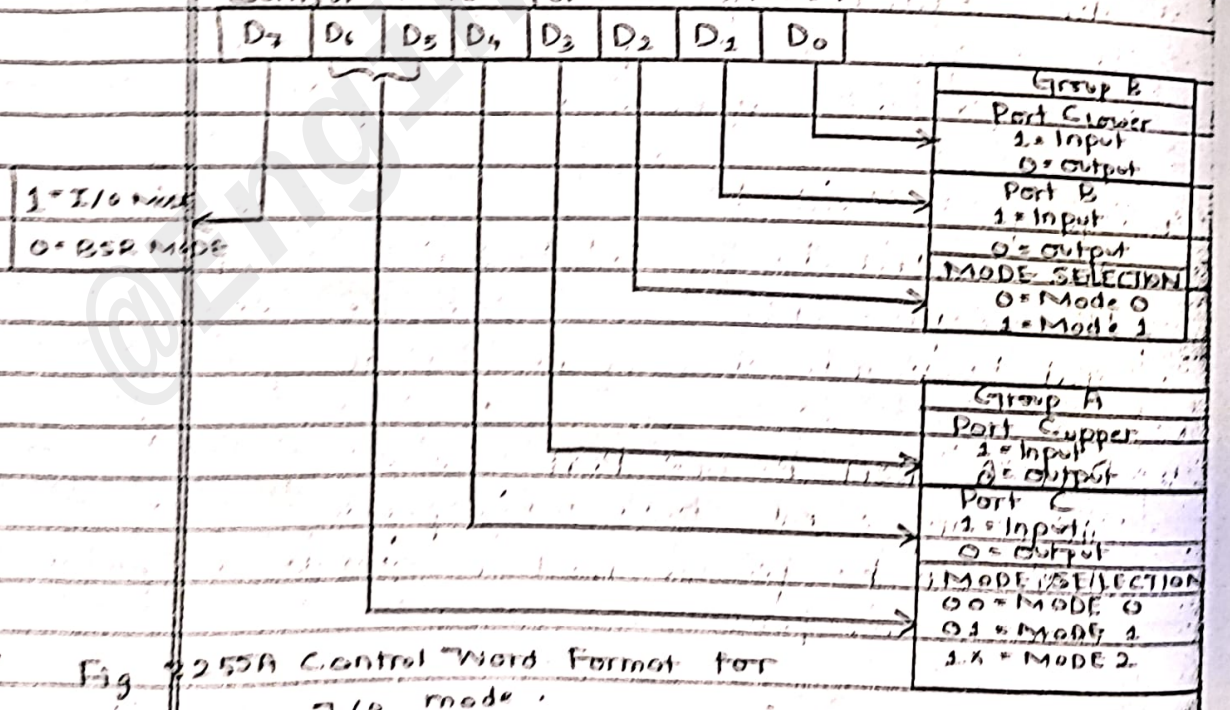


Fig: 8255A Control Word Format for I/O mode.

Date
Page 7

The various I/O modes of 8255A are as follows:

1. Mode 0: Simple Input or Output:

- In this mode, ports A and B are used as two simple I/O ports and port C as two 4-bit ports.
- Each port (or half port) in port C can be programmed to function as input or output port.
- The input/output features in mode 0 are as follows:
  1. Outputs are latched.
  2. Inputs are not latched.
  3. Ports donot have interrupt or handshake capability.
- A common example is the keyboard used for data entry.

2. Mode 1: Input or output with Handshake:

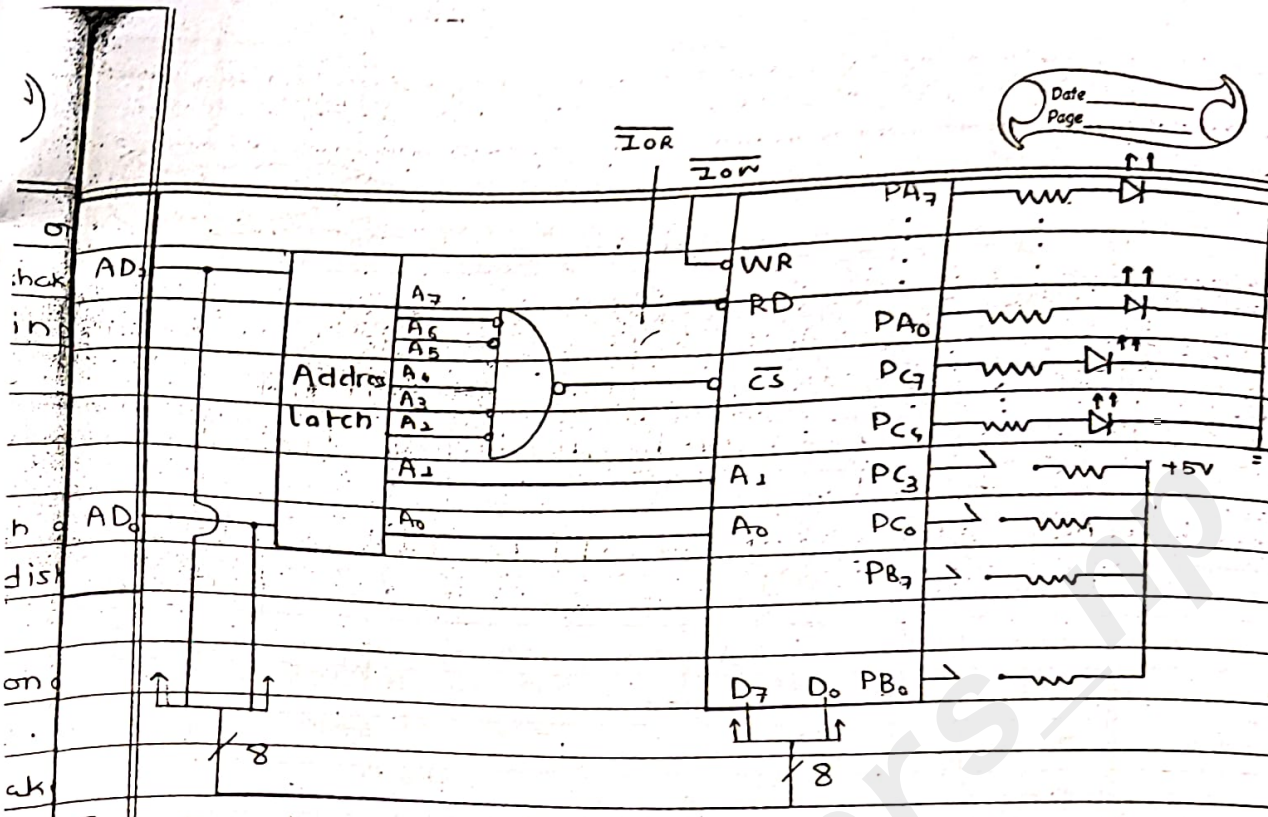
- In mode 1, handshake signals are exchanged betn mpu and peripheral prior to data transfer.
- The features of this mode include the following:
  (a) Two ports (A and B) function as 8-bit I/O ports. They can be configured either as input port or output port.
  (b) Each port uses three lines from port C as handshake signals. The remaining two lines of port C can be used for simple I/O functions.
  (c) Input and output data are latched.
  (d) Interrupt logic is supported.
- If Port B is initialized in mode 1 for either input or output pins PC_0, PC_1 and PC_2 function as handshake lines.

- If Port A is initialized in mode 1 for either input o output, pins PC_3, PC_4 and PC_5 function as handsh
- Remaining lines PC_6 and PC_7 are available for I/O or output lines.

3. Mode 2: Bidirectional data transfer
- This mode is used primarily in applications such data transfer between two computers or floppy dis controller interface.
- In mode 2, port A can be configured as bidirection port and port B either in mode 0 or mode 1.
- Port A uses 5 signals from Port C for handshak
- The remaining three signals from Port C can be us for simple I/O or handshake for Port B.
- If port B is in mode 0, then PC_0, PC_1 and PC_2 a used for I/O and if port B is in mode 1, then PC_0, PC_1 and PC_2 are used as handshak lines.

8255 Program:

Q.1. Write a program for 8255A PPI to read the DIP switches and display the reading from Port B at Port B and from Port C lower at Port C upper using the given circuit.
(Assume; Port Address : Port A = 30H)



Solution :→

Basic Addressing :

here, Port B and Port C lower = Input

Port A and Port C upper = output

Now, Control word for this condition is,

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
1	0	0	0	0	0	1	1	= 83H

We have Address calculation from above circuit can be done as :

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Port Selection	Port Address
0	0	1	1	0	0	0	0	Port A	30H
0	0	1	1	0	0	0	1	Port B	31H
0	0	1	1	0	0	1	0	Port C	32H
0	0	1	1	0	0	1	1	Control Register	33H

Now, Required program is as follows:

```
MVI A, 83H ; Load control word in Accumulator
OUT 33H ; Store control word in control Register
IN 31H ; Read switches at Port B
OUT 30H ; Display the reading at Port A
IN 32H ; Read switches at Port C
ANI 0FH ; Mask the upper 4-bits of Port C
RLC ; Rotate and place data in upper half of
      Accumulator
RLC
RLC
RLC
OUT 32H ; Display data at Port Cupper
HLT ; Terminate the program.
```

4. BSR (Bit Set/Reset) mode:

- Any bit of port-C can be set or reset using this mode.
- BSR mode doesn't alter any previously transmitted control word with bit $D_7 = 1$.
- In BSR mode, individual bits of port C can used for applications such as on/off switch.

Date
Page 8

BSR Control Word:

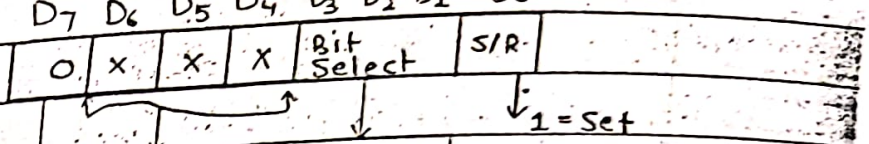


Fig: 8255A Control Word for BSR mode

Qn. Write BSR Control Word sub-routine to set bits PC_7 and PC_3 and reset them after 10 ms. (Assume, control register address = 83H and delay subroutine is available.

Solution: Here,

BSR Control Words.

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
To set bit PC_7	0	0	0	0	1	1	1	1	= 0FH
To reset bit PC_7	0	0	0	0	1	1	1	0	= 0EH
To set bit PC_3	0	0	0	0	0	1	1	1	= 07H
To reset bit PC_3	0	0	0	0	0	1	1	0	= 06H

Now,

Port address of Control Register = 83H

Required Subroutine is as follows:

```
BSR: MVI A, 0FH ; Load byte in Accumulator to set PC
     OUT 83H ; Store control word to set PC7 on control
     MVI A, 07H ; Load byte in accumulator to set PC
     OUT 83H ; Store control word to set PC3 on control re
     CALL Delay ; Call to available subroutine 'Delay' for
                                              1oms delay
     MVI A, 0EH ; Load byte in accumulator to reset PC7
     OUT 83H ; Store control word to reset PC7 on control
     MVI A, 06H ; Load byte in accumulator to reset PC3.
     OUT 83H ; Store control word to reset PC3 on control r
     RET ; Return to main program.
```

8237 DMA Controller:

Direct Memory Access (DMA):

- DMA is an I/O technique commonly used for high speed for data transfer. For example; data transfer b system memory and a floppy disk.
- In status check I/O and interrupt, I/O, data transf is relatively slow because each instruction needs to be fetched and executed.
- In DMA, mPU releases the control of the buses to device called DMA controller.
- The DMA Controller manages data transfer betn memory and a peripheral under its control, thus bypassing the mPU.

DMA Transfer uses two signals: HOLD and HLDA in 8085 microprocessor.

In Normal operation DMA Controller acts as a slave and mpu treats it as normal peripheral device but during DMA operation, DMA Controller acts as a master bypassing CPU.

Concept of DMA :-

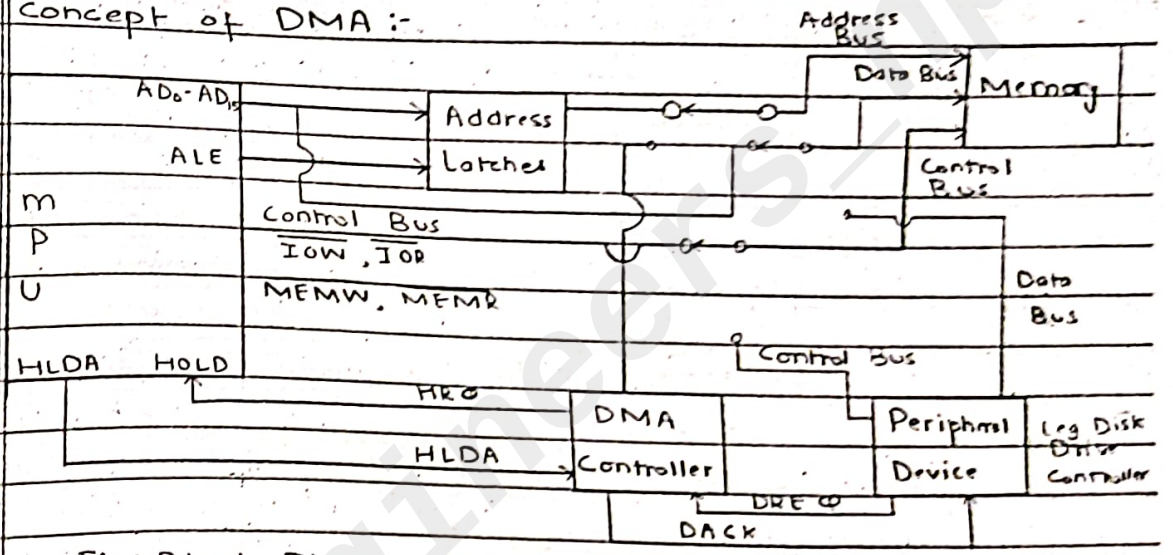


Fig: Block Diagram showing DMA Transfer.

Explanation.

The sequence of DMA Transfer as shown in above block diagram are as explained below:

1. Originally, microprocessor is connected to memory as shown in figure above with switches closed for address, data and control buses to mpu. When pheripheral wants to transfer data using DMA Transfer it sends DMA Request, DREQ signal to the DMA Controller.

2. If the input (channel) of DMA controller is unmasked the DMA Controller will send a hold-request (HRQ) signal to the microprocessor's HOLD input.
3. The MPU finishes the current machine cycle and floats its buses, sending out hold-acknowledge signal (HLDA) to the DMA Controller.
4. When DMA Controller receives HLDA signal, it will send a control signal which connects the three bus switches to DMA Controller and the peripheral. This disconnects the processor from buses and connects DMA Controller to ~~have~~ the buses. Now, DMA Controller sends out the address of the byte to be transferred and sends out DMA-Acknowledge (DACK) signal to the peripheral device.
5. Then the DMA transfer begins and finally when the data transfer is complete, the DMA Controller unasserts its HOLD-Request signal i.e HRQ=0 and releases the buses to MPU.

The 8237 DMA controller:

- The 8237 is a programmable DMA Controller packaged in 40 pin IC package.
- It has 4-independent channels with each channels capable of transferring 64KB.
- It must interface with two types of devices: the MP and the peripherals such as floppy disks and be able to operate in two modes: master mode (Dur DMA Transfer) and a slave mode (during normal op
- Many of the signals that are input in the I/O mode becomes output in the processor mode.

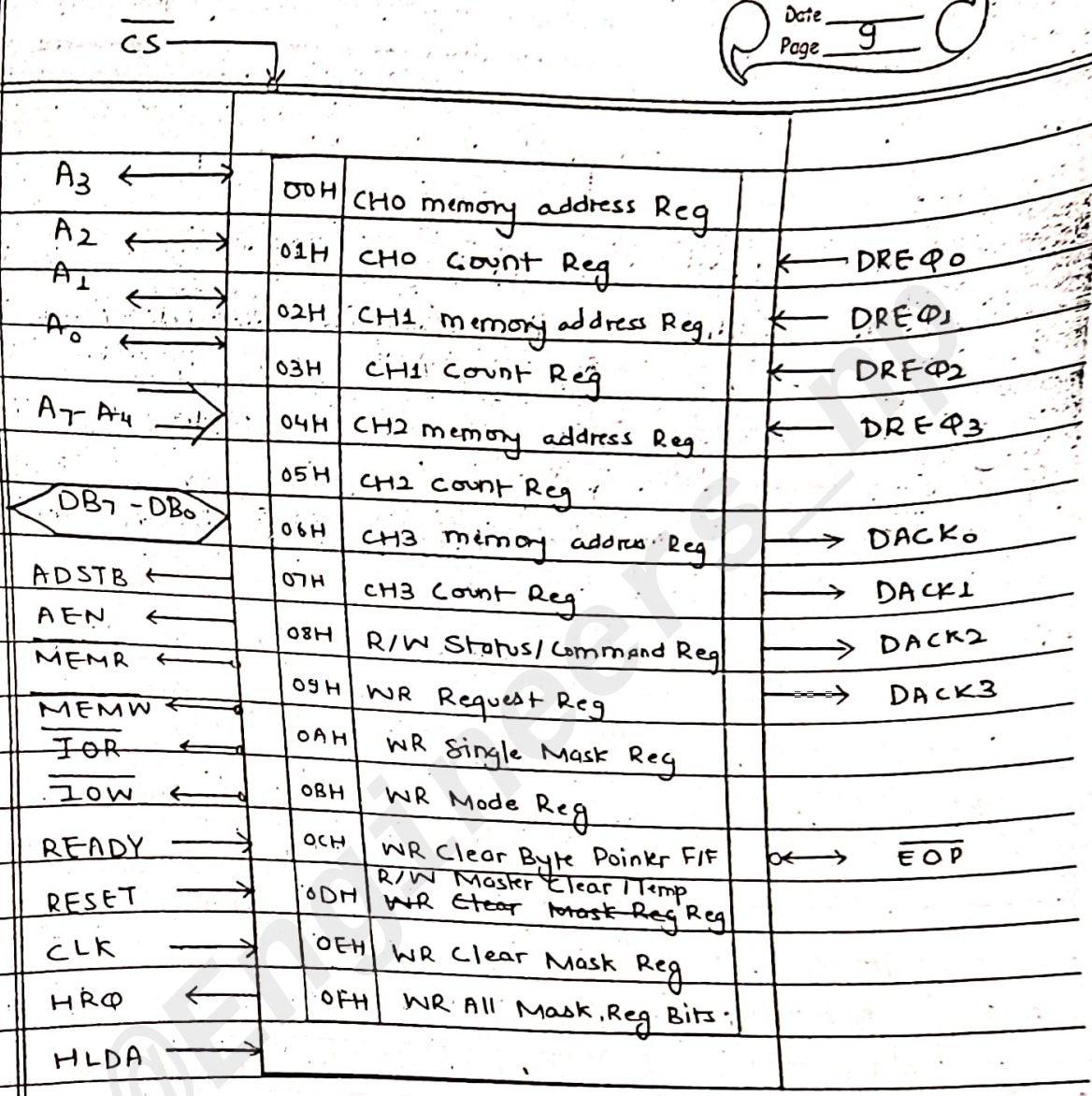


Fig: Block Diagram of 8237A with Internal Registers:

DMA CHANNELS AND INTERACTIONS:

- The 8237 has 4 independent channels: CH0 to CH3.
- Internally, two 16-bit registers are associated with each channel:

@Count Register:

- It is used to load a count of the number of bytes to be copied.

(b) Memory Address Register (MAR):

- It is used to load the starting address of the byte to be copied.
- The address of these registers are determined by address lines: A_3 to A_0 and the chip select ($\overline{CS}$) sig
- For example, Address 0000 on Address lines A_3-A_0 selects CH0 memory Address Register (MAR) or address 0001 on Address lines A_3-A_0 selects CH0 Count Register.
- Similarly, all the remaining registers are selected in sequential order.
- The address of internal registers range from 00H to 0FH.

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
0	0	0	0	0	0	0	0	= 00H (CH0 M
↓				↓			↓	
Chip Select ($\overline{CS}$)				1	1	1	1	= 0FH (ma re

DMA SIGNALS:

- The different signals request to understand DMA Transf are as explained below:

DREQ0 - DREQ3 (DMA Request)

- These are 4 independent asynchronous signal input to the DMA Channels from peripherals such as floppy disks, hard disks, etc.
- To obtain DMA service, a request is generate by activating the DREQ lines of the channel.

DACK0 - DACK3 (DMA Acknowledge)

→ These are output lines to inform the individual peripherals that a DMA is granted.

→ DREQ and DACK are equivalent to handshake signals in I/O device.

AEN and ADSTB (Address Enable and Address Strobe):

→ These are active high output signals that are used to latch higher-order address byte to generate 16-bit address.

$\overline{MEMW}$ and $\overline{MEMR}$ (Memory Write and Memory Read)

→ These are output signals used during DMA cycle to write and read data from memory.

$\overline{IOW}$ and $\overline{IOR}$ (Input output Write and I/O Read)

→ These are output signals used during DMA cycle to write & read data to and from I/O device.

$A_3 - A_0$ and $A_7 - A_4$ (Address lines):

→ $A_3 - A_0$ are bidirectional Address lines. They are used as inputs to access control registers.

→ During DMA cycles, these lines are used at output lines to generate a low order address that is combined with the remaining address lines $A_7 - A_4$.

HRQ and HLDA (Hold Request and Hold Acknowled

→ HRQ is an output signal used to request the MPU control of the system bus.

→ After receiving the HRQ, the MPU completes the bus cycle in process and issues the HLDA signal.

Modes of Operation of 8237:

→ The 8237 chip operates in two major cycles 1. Idle Cycle and 2. Active Cycle.

1. Idle Cycle.

→ When no channel is requesting service, the 8237 will enter in idle cycle. In this cycle, the 8237 sample DREQ lines every clock cycle to determ if any channel is requesting service.

2. Active Cycle.

→ When 8237 is in idle cycle and channel requests DMA service, the device will output an HRQ to the microprocessor and enter the active cycle. In active cycle, the DMA transfer occurs in one of the 4 modes.

1. Single Transfer Mode (or Byte Transfer Mode
2. Block Transfer Mode.
3. Demond Transfer Mode, and
4. Cascade Mode.

1 Single (Byte) Transfer Mode:

→ In this mode, the device is programmed to make only one transfer ie only one byte is transferred. The word count will be decremented by one and the address is also incremented or decremented following each transfer.

2 Block Transfer Mode:

→ In this mode, the device is activated by DREQ to continue making transfer the multiple bytes, until Terminal Count (TC) becomes zero or an external End of Process (EOP) is encountered.

3 Demand Transfer Mode:

→ In this mode, the device is programmed to continue making transfer until a TC becomes zero or EOP = 0 or until DREQ lines goes inactive. Thus, transfer may continue until the I/O device has exhausted its data capacity.

4 Cascade Mode:

→ This mode is used to cascade more than one 8237 together for simple system expansion.

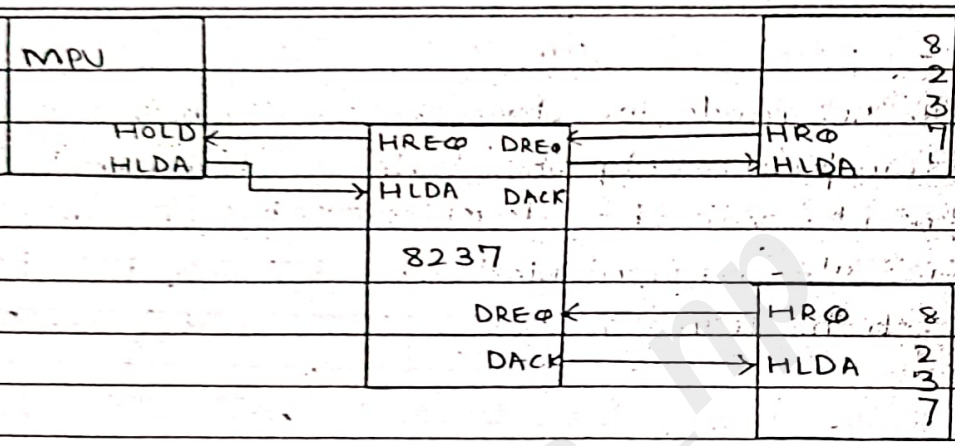


Fig: Cascading multiple 8237

5. # 8254 Programmable Interval Times (PIT):

→ The Intel 8254 is a counter/timer device designed to solve the common timing & counter problems in microprocessor system design.

→ It provides three independent 16-bit counters, each capable of handling clock inputs upto 10MHz.

→ Instead of setting up timing loops in software, the programmer configures the 8254 to match his/her requirements and programs one of the counters for the desired delay. After the desired delay, the 8254 will interrupt the CPU.

→ It operates in +5V regulated power supply and 24 signal pins.

→ The 8254 is the advanced version of 8253, and includes the feature read back command that is not available in 8253.

Applications :

- Real time clock,
- To generate accurate time delays,
- As an event-counter,
- Square wave generator,
- Programmable Rate Generator,
- Complex Waveform Generator.

Block Diagram of 8254 PIN

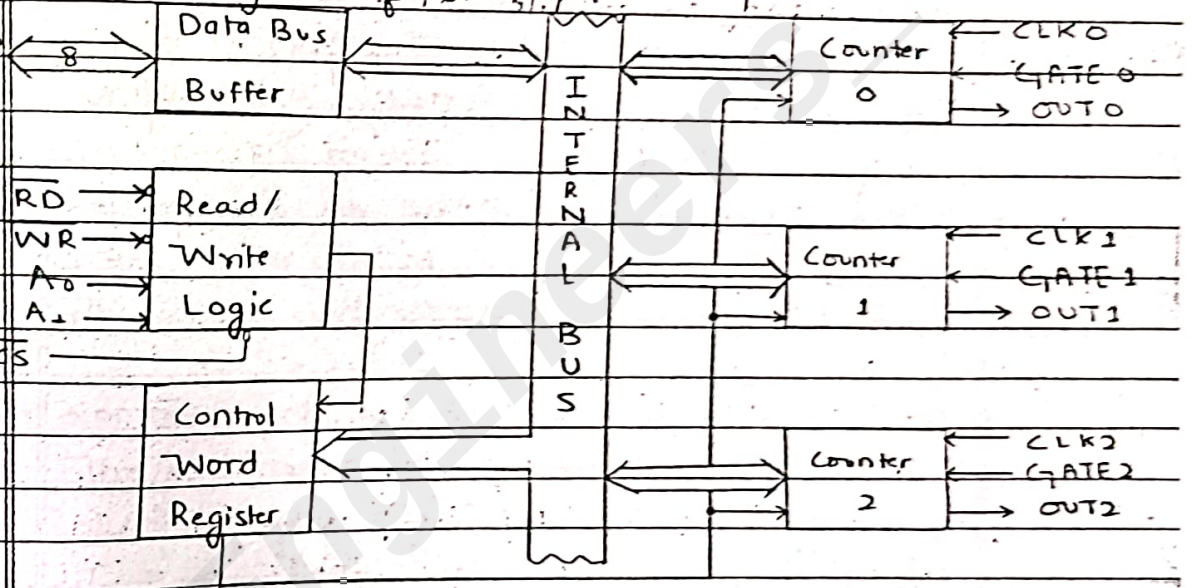


Figure : Block Diagram of 8254

Description :

1. Data Bus Buffer

→ This 3-state, bidirectional 8-bit buffer is used to interface the 8254 to the system bus of MPU.

2 Read / Write Logic.

→ The Read/Write Logic accepts iputs from the system bus and generates control signals for the other functional blocks of the 8254.

→ This section has five signals : $\overline{RD}$ (Read), $\overline{WR}$ (Write), $\overline{CS}$ (Chip Select) and address lines A_0 and A_1.

→ $\overline{RD}$ (Read) and $\overline{WR}$ (Write) are connected to $\overline{IOR}$ (Input output Read) and $\overline{WR}$ (Write) are connected to $\overline{MEMR}$ (Memory Read) and $\overline{MEMW}$ (Memory Write) respectively in Memory Mapped I.O.

→ Address lines A_1 and A_0 are connected to the address lines A_1 and A_0 of MPU and $\overline{CS}$ is used to select the device connected to the decoded address.

→ $\overline{CS}$ is used to select the device. If $\overline{CS} = 0$, the 8254 chip is selected for operation.

→ The counters and control word register are selected according to signals on the lines A_1 and A_0 as shown in table below :

A_1	A_0	Selection
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Control Register

3 Control Word Register

→ The Control Word Register is selected by the Read/write Logic when $A_1 A_0 = 11$.

→ If the MPU does the write operation to the 8254, the data is stored in the control word register and is interpreted as a Control Word used to define the operation of the Counters.

4 Counters

→ 8254 includes three identical 16-bit counters: Counter 0, Counter 1 and Counter 2 that can operate independently in any six modes of operation.

→ To operate as a counter, a 16-bit count is loaded into its register.

→ The counter can count either in BCD or Binary.

Control Word Format of 8254:

→ The 8-bit control word format in Control Register of 8254 is as shown below:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RW1	RW0	M_2	M_1	M_0	BCD

where, SC → Select Counter

SC1	SC2	Selection
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Read-Back command

RW → Read / Write.

RW1	RW2	
0	0	Counter Latch Command
0	1	Read/Write least Significant Byte only
1	0	Read / Write most significant Byte only
1	1	Read / Write least significant byte first, then most significant byte.

M → Mode

M_2	M_1	M_0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

BCD (X=0)

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

Fig: Control Word Format of 8254 PIT

Q Generate an 8254 control word to operate it as a BCD Counter in mode 2 using Counter 0 with 16-bit value of count.

Soln;

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	1	1	0	1	0	1

∴ Control Word = 35H

Modes of Operation of 8254:

→ 8254 can operate in 6 different values of operation.

Mode 0: Interrupt on Terminal Count:

- Mode 0 is typically used for event counting.
- After the control word is written, OUT is initially low, and will remain low until the counter reaches zero.
- OUT then goes high and remains high until a new count or a new Mode 0 control word is written in the counter.
- GATE = 1 enables counting and GATE = 0 disables counting. GATE has no effect on OUT.

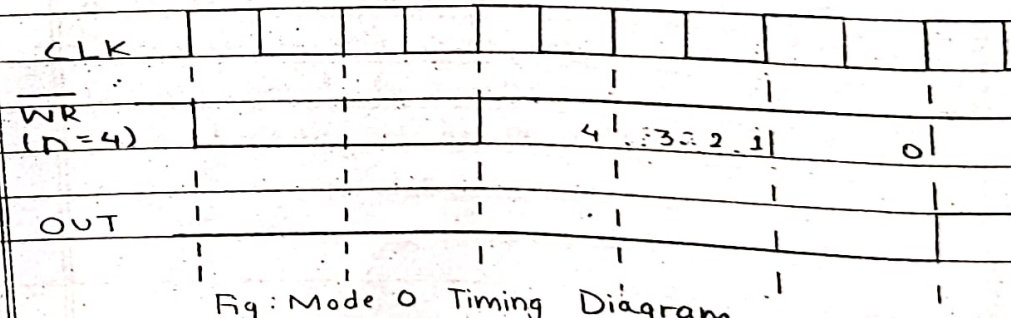


Fig: Mode 0 Timing Diagram

Mode 1 : Hardware re-triggable one-shot :

- OUT is initially high.
- OUT will go low on the CLK pulse following a trigger to begin the one-shot pulse, and will remain low until counter reaches zero.
- OUT will then go high and remain high until the CLK pulse after the next trigger.
- In this mode, GATE signal is used to trigger.
- The one-shot is re-triggerable, hence OUT will remain low for N CLK pulses after any trigger.

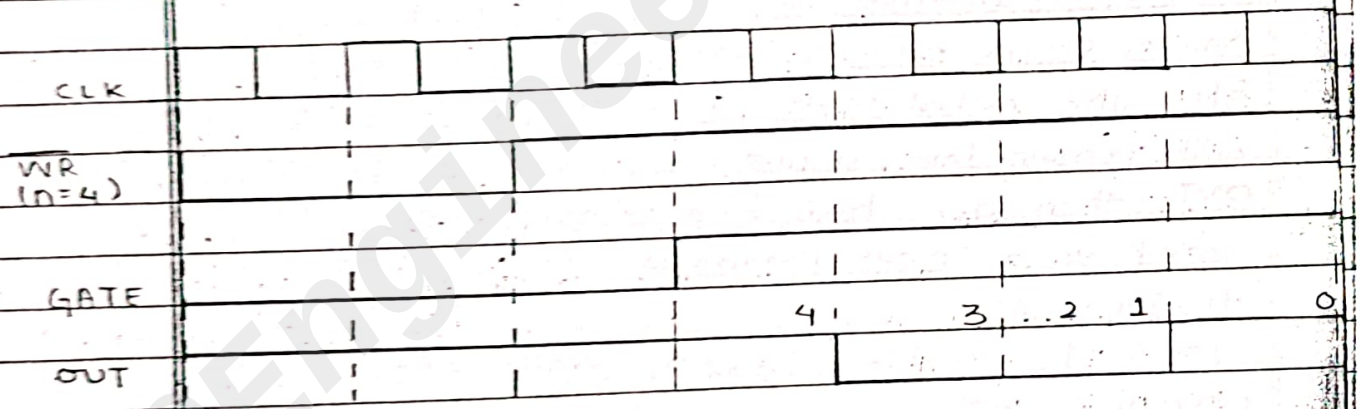


Fig: Mode 1 Timing Diagram

Date
Page 32

Mode 2 : Rate Generator

→ This mode is typically used to generate a Real Time Clock Interrupt.
→ OUT will initially be high. When the initial count has decremented to 1, OUT goes ~~high~~ low for one clock pulse.
→ OUT then goes high again, the counter re-loads the initial count and the process is repeated.

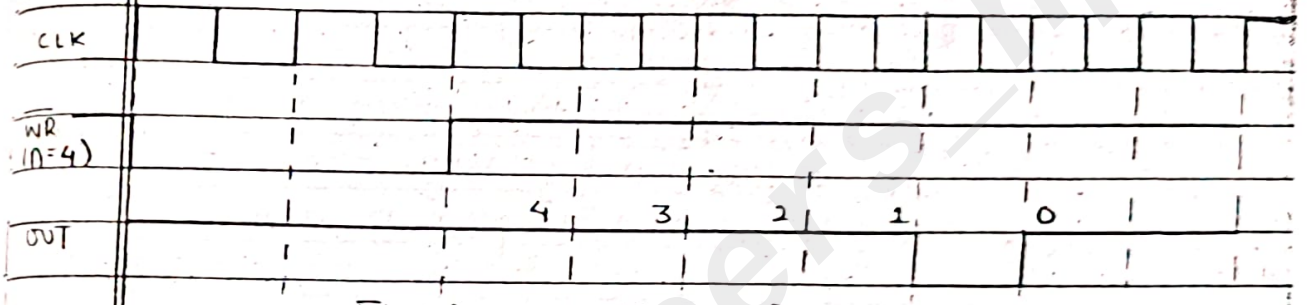


Fig: Mode 2 Timing Diagram.

Mode 3 : Square Wave Generator.

→ Mode 3 is typically used for Baud Rate Generation.
→ In this mode, when a count is loaded, the OUT goes high.
→ The count is decremented by every two clock cycle and when it reaches zero OUT goes low and the count is released again.
→ This is repeated continuously, thus a square wave with period equal to the period of count is generated.
i.e frequency of square wave = $\frac{\text{(Frequency of clock)}}{\text{count}}$

सुगम स्टेसनरी सप्लायर्स एण्ड फोटोकपी सर्भिस
बालकुमारी, ललितपुर ९८४१५९९५९२
NCIT College

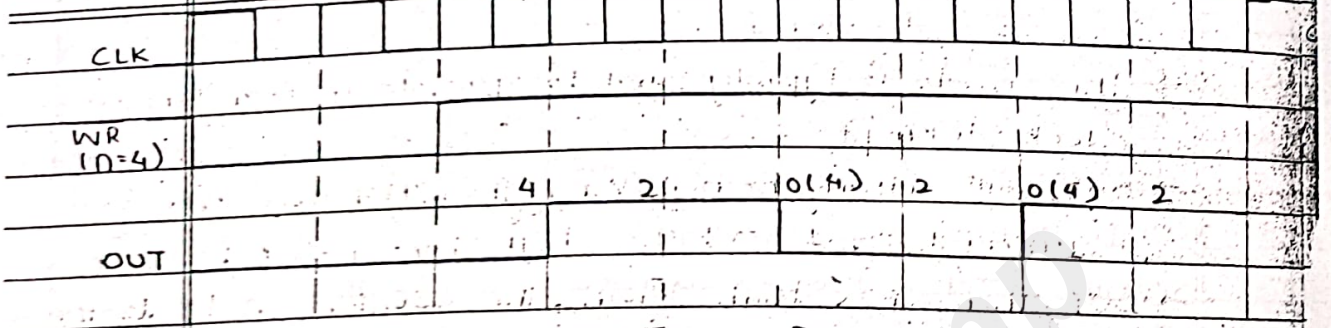


Fig Mode 3 : Timing Diagram

Mode 4 : Software Triggered Strobe

→ In this mode, OUT is initially high.

→ OUT goes low for one clock period at the end of the count.

→ The count must be recorded for subsequent outputs.

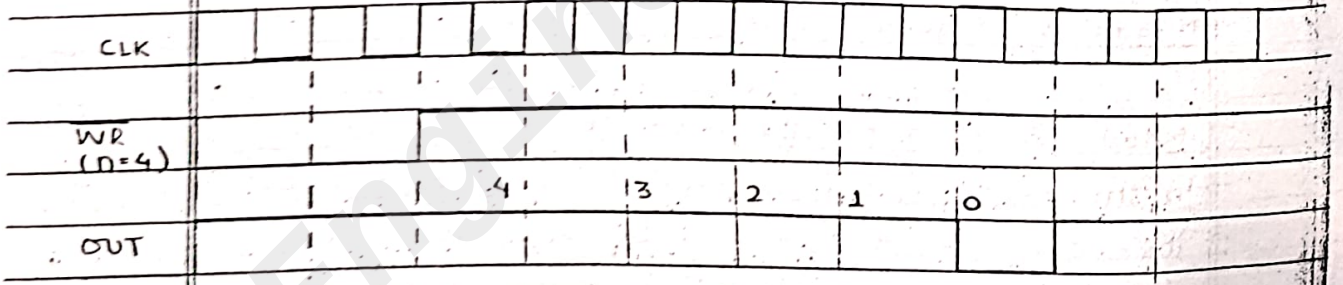


Fig Mode 4 : Timing Diagram

Mode 5 : Hardware Triggered Strobe

→ This mode is similar to mode 4 except that it is triggered by the rising pulse at the GATE.

→ Initially, OUT is high and when GATE pulse is triggered, the count begins.

→ At the end of the count, the OUT goes low for one clock cycle.

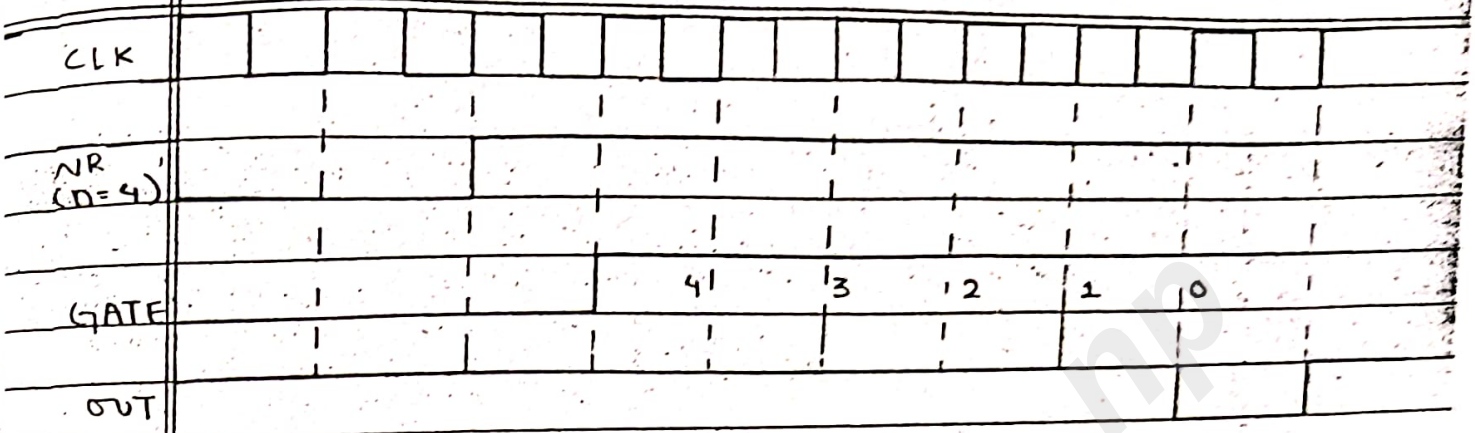


Fig: Mode 5 Timing Diagram