CHAPTER 1
INTRODUCTION TO MICROPROCESSORS

MICROPROCESSORS

- A Microprocessor is a multipurpose, Programmable clock-driven, register based electronic device that read binary instruction from a storage device called memory, accepts binary data as input and processes data according to those instructions and provides results as outputs.
- A Microprocessor is a clock driven semiconductor device consisting of electronic circuits manufactured by using either a LSI or VLSI technique.
- A typical programmable machine can be represented with three components : MPU, Memory and I/O as shown in Figure

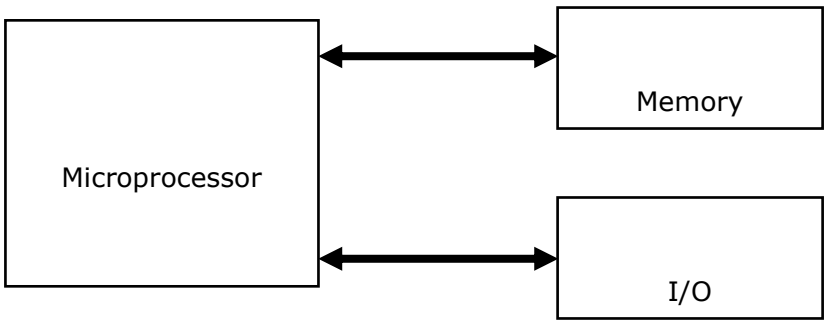


Figure: A Programmable Machine

- These three components work together or interact with each other to perform a given task; thus they comprise a system
- The machine (system) represented in above figure can be programmed to turn traffic lights on and off, compute mathematical functions, or keep trace of guidance system.
- This system may be simple or sophisticated, depending on its applications.
- The MPU applications are classified primarily in two categories : reprogrammable systems and embedded systems
- In reprogrammable systems, such as Microcomputers, the MPU is used for computing and data processing.
- In embedded systems, the microprocessor is a part of a final product and is not available for reprogramming to end user.

MICROCOMPUTER

- As the name implies, Microcomputers are small computers
- They range from small controllers that work directly with 4-bit words to larger units that work directly with 32-bit words

- Some of the more powerful Microcomputers have all or most of the features of earlier minicomputers.
- Examples of Microcomputers are Intel 8051 controller-a single board computer, IBM PC and Apple Macintosh computer.

MICRO CONTROLLER

- Single-chip Microcomputers are also known as Microcontrollers.
- They are used primarily to perform dedicated functions.
- They are used primarily to perform dedicated functions or as slaves in distributed processing.
- Generally they include all the essential elements of a computer on a single chip: MPU, R/W memory, ROM and I/O lines.
- Typical examples of the single-chip microcomputers are the Intel 8051, AT89C51, AT89C52 and Zilog Z8.
- Most of the micro controllers have an 8-bit word size, at least 64 bytes of R/W memory, and 1K byte of ROM
- I/O lines varies from 16 to 40

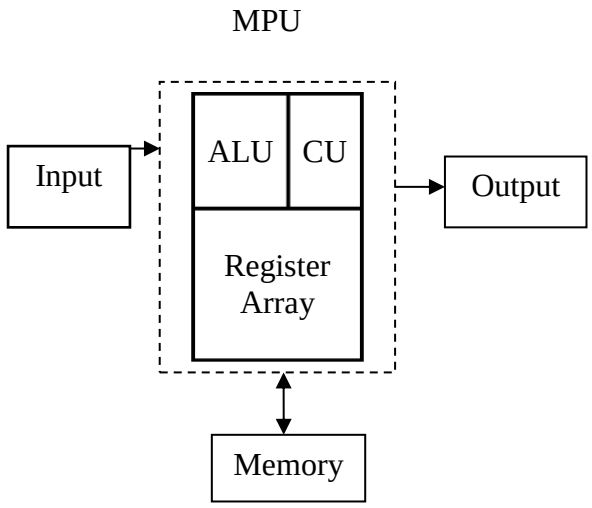
APPLICATIONS OF MICROPROCESSOR

- Microcomputers
- Industrial Control
- Robotics
- Traffic Lights
- Washing Machines
- Microwave Oven
- Security Systems
- On Board Systems

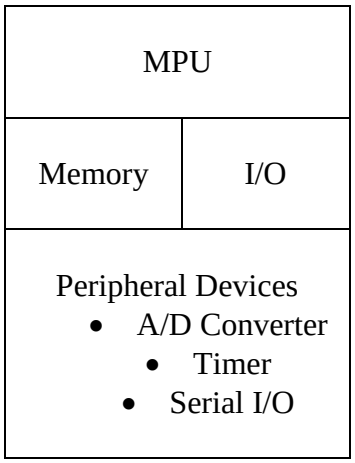
Difference between Microprocessors and Microcontrollers

Although both microprocessor and microcontrollers have been designed for real time applications and they share many common features, they have significant differences which are as follows:

Microprocessor	Microcontroller
1. Microprocessor is a silicon chip which includes ALU, register circuit and control circuits. 2. The general block diagram to show microprocessor is as shown below:	1. Microcontroller is a silicon chip which includes microprocessor, memory and I/O in a single package. 2. The general block diagram of microcontroller is as shown below:



3. Normally used for general purpose computers as CPU.
4. The performance speed, i.e. clock speed of microprocessor is higher ranging frequency from MHz to GHz.
5. Addition of external RAM, ROM and I/O ports makes these systems bulkier and much more expensive.
6. Microprocessors are more versatile than microcontrollers as the designers can decide on the amount of RAM, ROM and I/O ports needed to fit the task at hand. E.gs. Intel 8085, 8086, Motorola 68000, Intel Core i7, etc.



3. Normally microcontrollers are used for specific purposes (embedded system) e.g. traffic light controller, printer, etc.
4. The performance speed of microcontroller is relatively slower than that of microprocessors, with clock speed from 3-33 MHz.
5. Has fixed memory and all peripherals are embedded together on a single chip, so are not bulkier and are cheaper than microprocessors.
6. As microcontrollers have already fixed amount of RAM, ROM and I/O ports, so are not versatile as the user cannot change the amount of memory and I/O ports. E.gs. AT89C51, ATmega32, AT89S52, etc.

Evolution of Intel Microprocessors

4 bit Microprocessors

4004

- Introduced in 1971
- First microprocessor by Intel
- It was a 4-bit microprocessor
- Its clock speed was 740 KHz
- It had 2,300 transistors
- It could execute around 60,000 instructions per seconds
- Used in calculators

4040

- Introduced in 1974
- 4-bit microprocessor
- 3,000 transistors were used
- Clock speed was 740 KHz
- Interrupt features were available

8 Bit Microprocessors

8008

- Introduced in 1972 it was first 8 bit microprocessor
- Its clock speed was 500 KHz
- Could execute 50,000 instruction per second
- Used in: Computer terminals, Calculator, Bottling Machines, industrial Robots

8080

- Introduced in 1974
- It was also 8-bit microprocessor
- Its clock speed was 2 MHz
- It has 6,000 transistors
- 10 times faster than 8008
- Could execute 500,000 instructions per second
- Used In: Calculators, Industrial Robots

8085

- Introduced in 1976
- It was also 8-bit microprocessor
- Its clock speed was 3 MHz
- Its data bus is 8 bit and address bus is 16 bit
- It has 6,500 transistors
- It could execute 769,230 instructions per second
- It could access 64KB of memory
- It has 246 instructions
- Used In: early PC, On-Board Instrument Data Processors

16 Bit Microprocessors

8086

- Introduced in 1978
- First 16-bit microprocessor
- Clock speed is 5 to 10 MHz
- Data bus is 16-bit and address bus is 20-bit
- It had 29,000 transistors
- It could execute 2.5 million instructions per second
- Could access 1MB of memory
- It had 22,000 instructions
- Used In: CPU of Microcomputers

8088

- Introduced in 1979
- It was also 16-bit microprocessor
- It was creates as cheaper version of Intel's 8086
- 16-bit processor with an 8-bit data bus
- Could execute 2.5 million instructions per second
- The chip become the most popular in the computer industry when IBM used it for its first PC

80286

- Introduced in 1982
- It was 16-bit microprocessor
- Its clock speed was 8 MHz
- Data bus is 16-bit and address bus is 24-bit
- Could address 16 MB of memory
- It has 134,000 transistors
- Could execute 4-million instructions per second

32 Bit Microprocessors

80386

- Introduced in 1986
- First 32-bit microprocessor
- Data bus is 32 bit and address bus is 32-bit
- It could address 4GB of memory
- It has 275,000 transistors
- Clock speed varied from 16 MHz to 33 MHz depending upon different versions
- Different Versions
    - 80386DX
    - 80386SX
    - 80386SL

80486

- Introduced in 1989
- 32-bit microprocessor
- Had 1.2 million transistors
- Clock speed varied from 16 MHz to 100 MHz depending upon the various versions
- It had five different versions
    - 80486DX
    - 80486SX
    - 80486DX2
    - 80486SL
    - 80486DX4
- 8KB of cache memory was introduced

Pentium

- Introduced in 1993
- It was also 32-bit microprocessor
- Clock speed was 66 MHz
- Data bus is 32-bit and address bus is 32-bit
- Could address 4GB of memory
- Could execute 110 million instructions per second
- Cache memory
    - 8KB for Instruction
    - 8KB for data
- Upgraded Version: Pentium Pro

Pentium II

- Introduced in 1997
- 32-bit microprocessor
- Clock speed was 233 to 450 MHz
- MMX technology was supported
- L2 cache and processor were on one circuit
- Upgraded Version: Pentium II Xenon

Pentium III

- Introduced in 1999
- It was 32-bit microprocessor
- Clock speed varied from 500 MHz to 1.4 GHz
- It had 9.5 million transistors

Pentium IV

- Introduced in 2000
- 32-bit microprocessor
- Clock speed was from 1.3 GHz to 3.8 GHz
- L1 cache was 32 KB and L2 cache was 256 KB
- It had 42 million transistors

Intel Dual Core

- Introduced in 2006
- It is 32-bit or 64 bit Microprocessor
- It has 2-cores
- Both cores have their own internal bus and L1 cache but share the external bus and L2 cache
- Support SMT (Simultaneously Multithreading Technology)

64 Bit Microprocessors

Intel Core 2

- Introduced in 2006
- 64-bit microprocessor
- Clock speed is from 1.2 GHz to 3GHz
- It has 291 million transistors
- L1 cache- 64 KB per core
- L2 cache- 4 MB
- Versions:
  - Intel Core 2 Duo
  - Intel Core 2 Quad
  - Intel Core 2 Extreme

Intel Core i7

- Introduced in 2008
- 64-bit microprocessor
- It has 4 physical cores
- Clock speed is from 2.66 GHz to 3.33 GHz
- It has 781 million transistors
- L1 cache- 64 KB per core
- L2 cache- 256 KB
- L3 cache- 4 MB

Intel Core i5

- Introduced in 2009
- It is a 64-bit microprocessor
- It has 4 physical cores
- Its clock speed is from 2.40 GHz to 3.60 GHz
- It has 781 million transistors
- L1 cache- 64 KB per core
- L2 cache- 256 KB
- L3 cache- 8 MB

Intel Core i3

- Introduced in 2010
- 64-bit microprocessor
- It has 2 physical cores
- Clock speed is from 2.93 GHz to 3.33 GHz
- It has 781 million transistors
- L1 cache- 64 KB per core
- L2 cache- 512 KB
- L3 cache- 4 MB

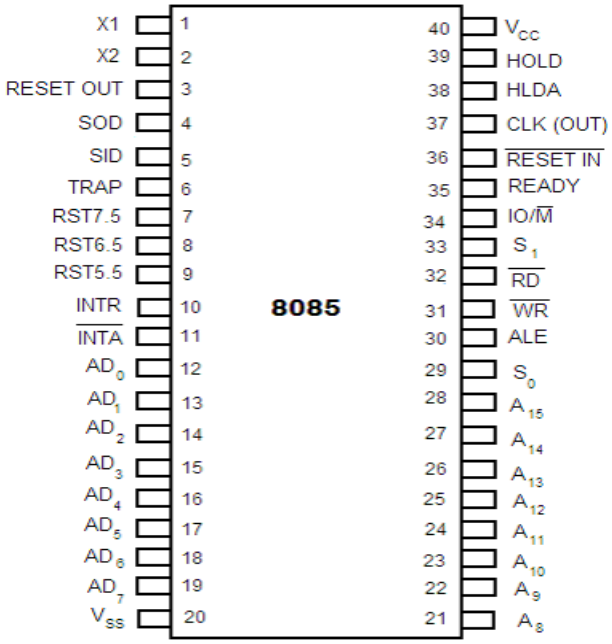
CHAPTER 2/ CHAPTER 3
8085, 8086 Microprocessors and ALP

8085 INTRODUCTION

The features of INTEL 8085 are:

- It is an 8 bit processor.
- It is a single chip N-MOS device with 40 pins.
- It works on 5 Volt dc power supply.
- The maximum clock frequency is 3 MHz while minimum frequency is 500 KHz.
- It provides 74 instructions with 5 different addressing modes.
- It has multiplexed address and data bus (AD0-AD7).
- It provides 16 address lines so it can access 2^{16} =64K bytes of memory.
- It generates 8 bit I/O address so it can access 2^{8}=256 input ports.

8085 Pin Diagram



Some important pins are:

- **AD0-AD7**: Multiplexed Address and data lines.
- **A8-A15**: Tri-stated higher order address lines.
- **ALE**: Address latch enable is an output signal. It goes high when operation is started by processor.
- **S0, S1**: These are the status signals used to indicate type of operation.
- **$\overline{\text{RD}}$:** Read is active low input signal used to read data from I/O device or memory.
- **$\overline{\text{WR}}$**: Write is an active low output signal used write data on memory or an I/O device.
- **READY**: This an output signal used to check the status of output device. If it is low, µP will WAIT until it is high.

- **TRAP**: It is an Edge triggered highest priority, non-maskable interrupt. After TRAP, restart occurs and execution starts from address 0024H.
- **RST 5.5, 6.5, 7.5**: These are maskable interrupts and have low priority than TRAP.
- **$\overline{\text{INTR}}$ & INTA**: INTR is an interrupt request signal after which µP generates INTA or interrupt acknowledge signal.
- **IO/$\overline{\text{M}}$**: This is output pin or signal used to indicate whether 8085 is working in I/O mode (IO/M=1) or Memory mode (IO/M=0).
- **HOLD & HLDA**: HOLD is an input signal .When µP receives HOLD signal it completes current machine cycle and stops executing next instruction. In response to HOLD µP generates HLDA that is HOLD Acknowledge signal.
- **$\overline{\text{RESETIN}}$**: This is input signal. When RESETIN is low µp restarts and starts executing from location 0000H.
- **SID**: Serial input data is input pin used to accept serial 1 bit data.
- **SOD**: Serial output data is output pin used to send serial 1 bit data.
- **X1, X2**: These are clock input signals and are connected to external LC or RC circuit. These are divide by two so if 6 MHz is connected to X1X2, the operating frequency becomes 3 MHz.
- **VCC & VSS**: Power supply VCC=+ -5Volt& VSS=-GND reference.

8085 ARCHITECTURE

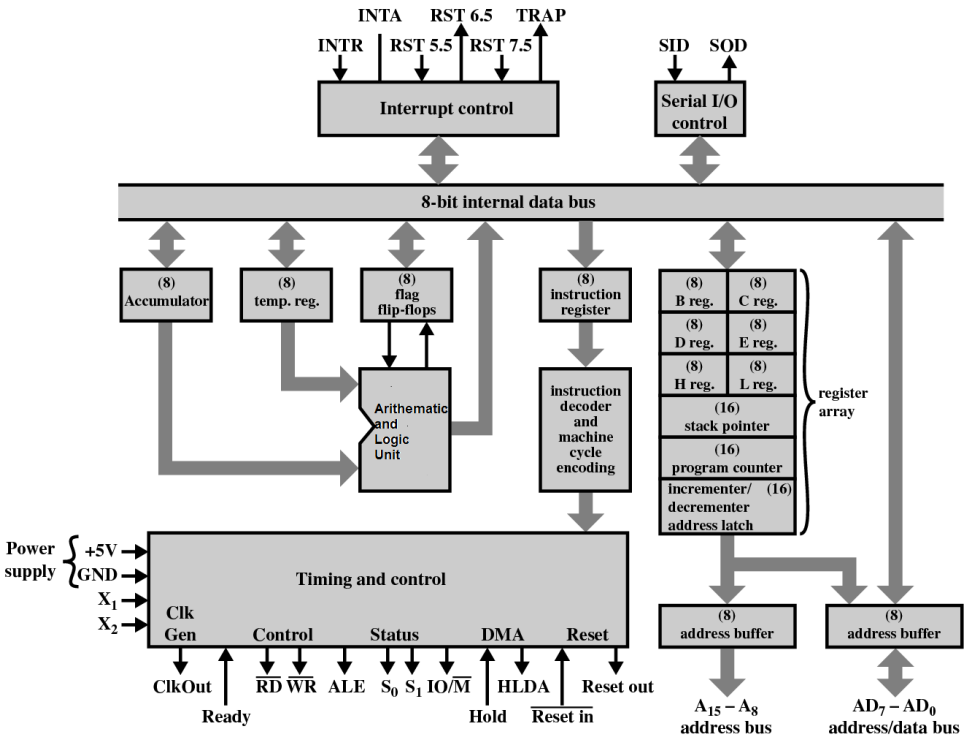


Figure: Block Diagram

ARITHMETIC AND LOGIC UNIT (ALU)

Accumulator:

It is 8 bit general purpose register. It is connected to ALU. So, most of the operations are done in Accumulator (A).

Temporary register:

It is not available for user. All the arithmetic and logical operations are done in the temporary register but user can't access it.

Flag Register: It is an 8-bit register which consists of 5 flip flops used to know status of various operations done.

S	Z	-	AC	-	P	-	CY

Fig. *8085 Flag Register*

S: Sign flag is set when result of an operation is negative.

Z: Zero flag is set when result of an operation is 0.

AC: Auxiliary carry flag is set when there is a carry out of lower nibble or lower four bits of the operation.

CY: Carry flag is set when there is carry generated by an operation.

P: Parity flag is set when result contains even number of 1's.

Rest are don't care flip flops and reserved for future use.

REGISTER ARRAY

Temporary registers (W, Z): These are not available for user. These are loaded only when there is an operation being performed.

General purpose: There are six 8-bit general purposes register in 8085 namely B, C, D, E, H and L. These are used for various data manipulations. They can be used in pairs as 16-bit registers. The register pairs are: BC pair, DE pair and HL pair.

Special purpose: There are two special purpose registers in 8085:

SP (Stack Pointer): It is a 16-bit register used to hold the address of stack during stack operation i.e PUSH and POP operations.

PC (Program Counter): It is a 16-bit register which holds the address of next instruction to be fetched. When a single byte instruction is executed PC is automatically incremented by 1. Upon reset PC contents are set to 0000H.

TIMIMG AND CONTROL UNIT

This unit synchronizes all the microprocessor operations with the clock and generates the control signals necessary for communication between the microprocessor and peripherals. The $\overline{RD}$ and $\overline{WR}$ signals are sync pulse indicating the availability of data on the data bus.

INSTRUCTION REGISTER AND DECODER

The instruction register and decoder are part of ALU. When an instruction is fetched from memory, it is loaded in the instruction register. The decoder decodes the instruction and establishes the sequence of events to flow. The instruction register is not programmable and cannot be accessed through any instructions.

INTERRUPT CONTROL

It accepts different interrupts like TRAP, RST 5.5, RST 6.5, RST 7.5 and INTR. INTA is interrupt acknowledgement signal.

SERIAL IO CONTROL

It is used to accept and send the serial 1 bit data by using SID and SOD signals and it can be performed by using SIM & RIM instructions.

8085 INSTRUCTION SETS

a) **Data Transfer Instructions**
b) **Arithmetic Instructions**
c) **Logical Instructions**
d) **Rotate Instructions**
e) **Branching Instructions**
f) **Control Instructions**

a) Data Transfer instructions

Mnemonics	Description	Example
MOV Rd, Rs	• Copies the content of source register Rs into destination register Rd • Rs and Rd can be A, B, C, D, E, H, L	MOV A, B
MOV Rd, M	• Copies the content of memory location M into the destination register Rd • Rd can be A, B, C, D, E, H, L • The memory location M is specified by HL pair	MOV A, M
MOV M, Rs	• Copies the content of register Rs into memory location M • Rs can be A, B, C, D, E, H, L • The memory location M is specified by HL pair	MOV M, A
MVI Rd, 8-bit	• The 8-bit data is stored in the destination register Rd • Rd can be A, B, C, D, E, H, L	MVI A, 32H
MVI M, 8-bit	• The 8-it data is stored in memory location M • M is specified by HL pair	MVI M, 32H
LDA 16-bit	• Copies the content of memory	LDA 2015H

(Load Accumulator Direct)	location specified by 16-bit address into A	A← [2015]
STA 16-bit (Store Accumulator Direct)	• Copies the content of A into 16-bit memory address	STA 2015H A→[2015]
LDAX Rp (Load Accumulator Inirect)	• Copies the content of memory location specified by register pair Rp into A • Rp can be B or D i.e. BC pair or DE pair	LDAX B
STAX Rp (Store Accumulator Indirect)	• Copies the content of A into 16-bit memory address specified by register pair Rp • Rp can be B or D i.e. BC pair or DE pair	STAX B
LXI Rp, 16-bit (Load Register Pair)	• Loads 16-bit data into register pair • Rp can be B, D or H i.e. BC pair, DE pair or HL pair	LXI H, 2015H L←15 H←20
IN 8-bit	• The data from i/p port specified by 8-bit address is transferred into A	IN 40H A←[40] 40H is address of input port
OUT 8-bit	• The data of A is transferred into output port specified by 8-bit address	OUT 10H A→[10] 10H is address of output port
XCHG	• Exchange the content of HL pair with DE pair i.e. the content of H and D are exchanged whereas content of L and E are exchanged	XCHG H←→D L←→E

b) Arithmetic Instructions

Mnemonics	Description	Example
ADD R/M (add register/memory)	• The content of register /memory (R/M) is added to the A and result is stored in A • The memory M is specified by HL pair	ADD B A←A+B ADD M A←A+M
ADC R/M (add with carry)	• The content of register /memory (R/M) is added to the A along with carry flag CF and result is stored in A • The memory M is specified by HL pair	ADC B A←A+B+CF ADC M A←A+M+CF
ADI 8-bit (add immediate)	• The 8-bit data is added to A and result is stored in A	ADI 32H A←A+32
ACI 8-bit	• The 8-bit data is added to A	ACI 32H

(add immediate with carry)	along with carry flag CF and result is stored in A	A←A+32+CF
SUB R/M (subtract register/memory)	• The content of register /memory (R/M) is subtracted from A and result is stored in A • The memory M is specified by HL pair	SUB B A←A-B SUB M A←A-M
SBB R/M (subtract with borrow)	• The content of register /memory (R/M) is subtracted from A along with borrow flag BF and result is stored in A • The memory M is specified by HL pair	SBB B A←A-B-BF SBB M A←A-M-CF
SUI 8-bit (subtract immediate)	• The 8-bit data is subtracted from A and result is stored in A	SUI 32H A←A-32
INR R/M (Increment Register/Memory)	• Increment the content of register/memory by 1 • Memory is specified by HL pair	INR B B←B+1 INR M M←M+1
DCR R/M (Decrement Register/Memory)	• Decrement the content of register/memory by 1 • Memory is specified by HL pair	DCR B B←B-1 DCR M M←M-1
INX Rp (Increment Register Pair)	• Increment the content of register pair Rp by 1	INX H HL←HL+1
DCX Rp (Decrement Register Pair)	• Decrement the content of register pair Rp by 1	DCX H HL←HL-1

c) Logical Instructions

Mnemonics	Description	Example
CMP R/M (Compare Register/Memory)	• Compares the content of register/memory with A • The result of comparison is: If A< R/M : Carry Flag CY=1 If A= R/M : Zero Flag Z=1 If A> R/M : Carry Flag CY=0	CMP B CMP M
CPI 8-bit (Compare Immediate)	• Compares 8-bit data with A • The result of comparison is: If A< 8-bit : Carry Flag CY=1 If A= 8-bit : Zero Flag Z=1 If A> 8-bit : Carry Flag CY=0	 CPI 32H

ANA R/M (logical AND register/memory)	• The content of A are logically ANDed with the content of register/memory and result is stored in A • Memory M must be specified by HL pair	ANA B A←A.B ANA M A←A.M
ANI 8-bit (AND immediate)	• The content of A are logically ANDed with the 8-bit data and result is stored in A	ANI 32H A←A.32H
ORA R/M (logical OR register/memory)	• The content of A are logically ORed with the content of register/memory and result is stored in A • Memory M must be specified by HL pair	ORA B A←A or B ORA M A←A or M
ORI 8-bit (OR immediate)	• The content of A are logically ORed with the 8-bit data and result is stored in A	ORI 32H A←A or 32H
XRA R/M (logical XOR register/memory)	• The content of A are logically XORed with the content of register/memory and result is stored in A • Memory M must be specified by HL pair	XRA B A←A xor B XRA M A←A xor M
XRI 8-bit (XOR immediate)	• The content of A are logically XORed with the 8-bit data and result is stored in A	XRI 32H A←A xor 32H

d) Rotate Instructions

Mnemonics	Description	Example
RLC (Rotate Accumulator Left)	• Each bit of A is rotated left by one bit position. • Bit D7 is placed in the position of D0.	RLC
RRC (Rotate Accumulator Right)	• Each bit of A is rotated right by one bit position. • Bit D0 is placed in the position of D7.	RRC
RAL (Rotate Accumulator Left with Carry)	• Each bit of A is rotated left by one bit position along with carry flag CY • Bit D7 is placed in CY and CY in the position of D0.	RAL
RAR (Rotate Accumulator Right with Carry)	• Each bit of A is rotated right by one bit position along with carry flag CY. • Bit D7 is placed in CY and CY in the position of D0.	RLC

e) Branching Instructions

Mnemonics	Description	Example
JMP 16-bit (Unconditional Jump)	The program sequence is transferred to the memory location specified by 16-bit address	JMP C000H
JC	Jump on Carry (CY=1)	
JNC	Jump No Carry (CY=0)	
JP	Jump on Positive (S=0)	
JM	Jump on Negative (S=1)	
JZ	Jump on Zero (Z=1)	
JNZ	Jump No Zero (Z=0)	
JPE	Jump on Parity Even (P=1)	
JPO	Jump on Parity Odd (P=0)	
CALL 16-bit	The program sequence is transferred to the subroutine at memory location specified by the 16-bit address	CALL C000H
RET	The program sequence is transferred from the subroutine program to calling program	RET

f) Control Instructions

Mnemonics	Description	Example
NOP	No operation is performed	NOP
HLT	The CPU finishes executing the current instruction and stops any further execution	HLT

8085 ADDRESSING MODES

The ways by which operands are specified in an instruction are called addressing modes. The different addressing modes of 8085 are:

1. Immediate Addressing Mode

If the data is present within the instruction itself, then it is called immediate addressing mode.

Examples:
```
MVI A, 05H
ADI 55H
LXI H, C000H
```

2. Register Addressing Mode

If the data is present in the register and the register are specified in an instruction, than it is called register addressing mode.

Example:
```
MOV A, B
ADD B
ANA C
```

3. Direct Addressing Mode

If the address of the data is specified in the instruction itself, than it is called direct addressing mode.

Example: LDA 2000H
STA 2000H
IN 10H
OUT 01H

4. Register Indirect Addressing Mode

If the register pair which contains the address of the data is specified in the instruction, than it is called register indirect addressing mode.

Example: LDAX B
STAX D
MOV M, A
MOV B, M

5. Implied Addressing Mode

If the opcode in an instruction tells about the operand, than it is called implied addressing mode.

Example: RAL
RRC

TIMING DIAGRAMS OF 8085 INSTRUCTIONS

a) Related Terms

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

Machine Cycle: The time taken by the processor to access memory location, IO ports or to acknowledge an interrupt once is called as machine cycles. It is the combinations of T-states.

T-States: It is the sub-division of operation performed in one clock cycle of processor's clock.

b) Status Signals for Various Machine Cycles

Machine Cycles	Status Signals		
	IO/ $\overline{M}$	**S_1**	**S_0**
Opcode Fetch	0	1	1
Memory Read	0	1	0
Memory Write	0	0	1
IO Read	1	1	0
IO Write	1	0	1

c) Types of Instruction On the Basis of Size

One Byte Instructions

These Instruction use a total memory of one-byte

These Instructions include the opcode and operand in the same byte.

Examples: MOV C, A
ADD B
RLC

Two Byte Instructions

These instructions use a total memory of two bytes.

The first byte specifies the opcode and second byte specifies the operand

Examples: MVI A, 32H
ADI 30H
IN 40H

Three Byte Instructions

These instructions use a total memory of three bytes.

The first byte specifies the opcode and the remaining two bytes specifies the 16-bit address i.e. second byte specifies the lower order address and third byte specifies the higher order address.

Example: LDA 2070H
STA 2050H
LXI H, 2070H

8086 MICROPROCESSOR

INTERNAL ARCHITECTURE OF 8086

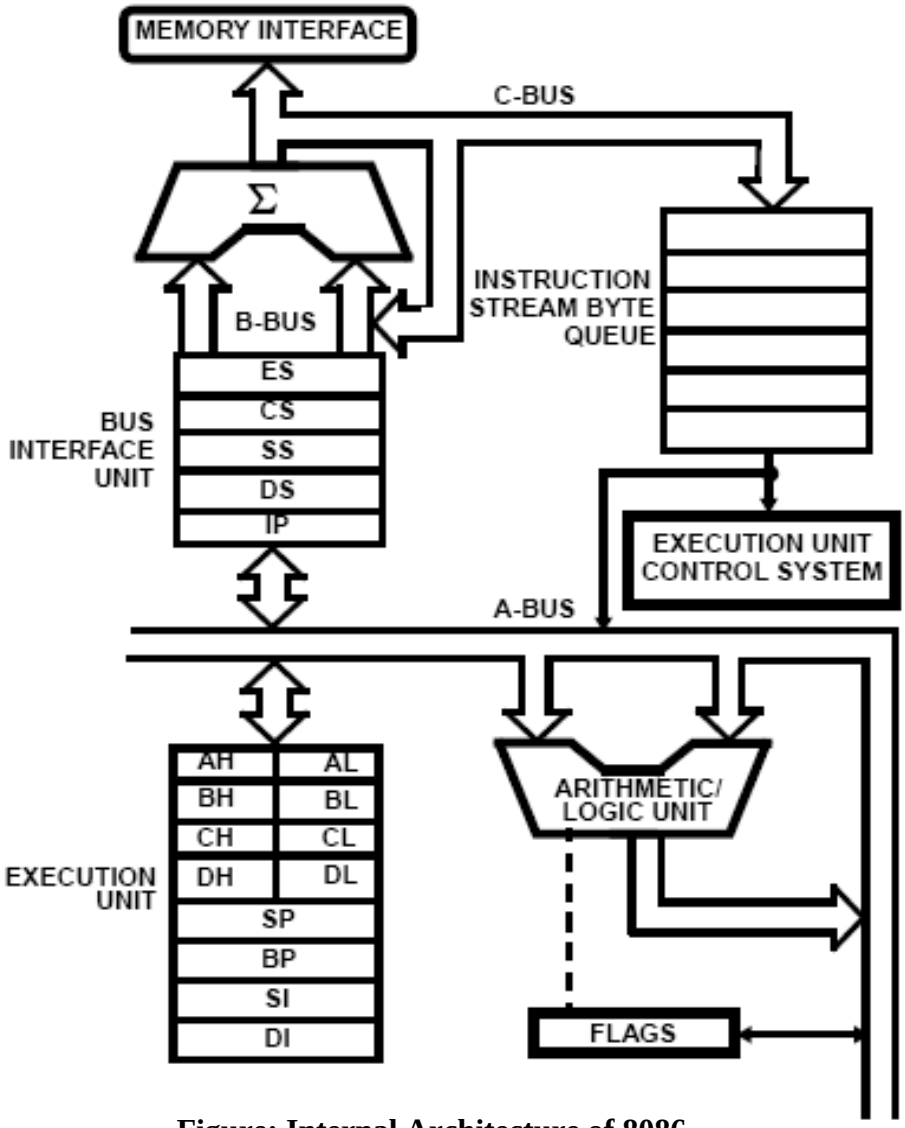


Figure: Internal Architecture of 8086

- The 8086 CPU is divided into two independent functional parts : BIU (Bus Interface Unit) and EU (Execution Unit)
- Dividing the work between these units speed up the processing.

A. THE EXECUTION UNIT (EU)

- The EU of the 8086 tells the BIU where to fetch the instructions and data from, decodes instructions and executes instructions.
- The EU has a 16 bit ALU which can add subtract, ND, OR, increment, decrement, complement or shift binary numbers.

1. GENERAL PURPOSE REGISTERS

- The EU has eight general purpose registers, labeled AH, AL, BH, BL, CH, CL, DH and DL.
- These registers can be used individually for temporary storage of 8 bit data.
- The AL register is also called accumulator
- It has some features that the other general purpose registers do not have.
- Certain pairs of these general purpose registers can be used together to store 16 bit words.
- The acceptable register pairs are AH and AL,BH and BL,CH and CL,DH and DL
- The AH-AL pair is referred to as the AX register, the BH-BL pair is referred to as the BX register, the CH-CL pair is referred to as the CX register, and the DH-DL pair is referred to as the DX register.

AX = Accumulator Register
BX = Base Register
CX = Count Register
DX = Data Register

2. FLAG REGISTER

- A Flag is a flip-flop which indicates some condition produced by the execution of an instruction or controls certain operations of the EU.
- A 16 bit flag register in the EU contains 9 active flags.
- Figure below show shows the location of the nine flags in the flag register.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
U	**U**	**U**	**U**	**OF**	**DF**	**IF**	**TF**	**SF**	**ZF**	**U**	**AF**	**U**	**PF**	**U**	**CF**

Figure: 8086 Flag Register Format

U = UNDEFINED

CONDITIONAL FLAGS

CF = CARRY FLAG [Set by Carry out of MSB]
PF = PARITY FLAG [Set if Result has even parity]
AF= AUXILIARY CARRY FLAG FOR BCD
ZF = ZERO FLAG [Set if Result is 0]
SF = SIGN FLAG [MSB of Result]
OF = OVERFLOW FLAG

CONTROL FLAG

TF = SINGLE STEP TRAP FLAG
IF = INTERRUPT ENABLE FLAG
DF = STRING DIRECTION FLAG

- The six conditional flags in this group are the CF,PF,AF,ZF,SF and OF
- The three remaining flags in the Flag Register are used to control certain operations of the processor.
- The six conditional flags are set or reset by the EU on the basis of the result of some arithmetic or logic operation.
- The Control Flags are deliberately set or reset with specific instructions you put in your program.
- The three control flags are the TF, IF and DF.
- Trap Flag is used for single stepping through a program.
- The Interrupt Flag is used to allow or prohibit the interruption of a program.
- The Direction Flag is used with string instructions.

3. POINTER REGISTERS

- The 16 bit Pointer Registers are IP,SP and BP respectively
- SP and BP are located in EU whereas IP is located in BIU

3.1 STACK POINTER (SP)

- The 16 bit SP Register provides an offset value, which when associated with the SS register (SS:SP)

3.2 BASE POINTER (BP)

- The 16 bit BP facilitates referencing parameters, which are data and addresses that a program passes via the stack.
- The processor combines the addresses in SS with the offset in BP.
- BP can also be combined with DI and SI as a base register for special addressing.

4. INDEX REGISTERS

- The 16 bit Index Registers are SI and DI

4.1 SOURCE INDEX (SI) REGISTER

- The 16 bit Source Index Register is required for some string handling operations
- SI is associated with the DS Register.

4.2 DESTINATION INDEX (DI) REGISTER

- The 16 bit Destination Index Register is also required for some string operations.
- In this context, DI is associated with the ES register.

B. THE BUS INTERFACE UNIT

- The BIU send out address, fetches instructions from memory, reads data from ports and memory, and writes data to ports and memory.
- In other words BIU handles all transfers of data and addresses on the buses for the execution unit.

1. SEGMENT REGISTERS

1.1 CODE SEGMENT REGISTER (CS)

- It contains the starting address of a program's code segment.
- This segment address plus an offset value in the IP register indicates the address of an instruction to be fetched for execution
- For normal programming purpose, you need not directly reference this register.

1.2 DATA SEGMENT REGISTER (DS)

- It contains the starting address of a program's data segment
- Instruction uses this address to locate data.
- This address plus an offset value in an instruction causes a reference to a specific byte location in the data segment.

1.3 STACK SEGMENT REGISTER (SS)

- Permits the implementation of a stack in memory
- It stores the starting address of a program's stack segment the SS register.
- This segment address plus an offset value in the Stack Pointer (SP) register indicates the current word in the stack being addressed.

1.4 EXTRA SEGMENT REGISTER (ES)

- It is used by some string operations to handle memory addressing.
- ES Register is associated with the DI Register.

2. INSTRUCTION POINTER (IP)

- The 16 bit IP Register contains the offset address of the next instruction that is to execute.
- IP is associated with CS register as (CS:IP)
- For each instruction that executes, the processor changes the offset value in IP so that IP in effect directs each step of execution.

3. THE QUEUE

- While the EU is decoding an instruction or executing an instruction which does not require use of the buses, the BIU fetches up to six instructions bytes for the following instructions.
- The BIU Stores pre-fetched bytes in First in First out register set called a queue.
- When the EU is ready for its next instruction, it simply reads the instruction bytes for the instruction from the queue in the BIU.
- This is much faster than sending out an address to the system memory and waiting for memory to send back the next instruction bytes or bytes.
- Fetching the next instruction while the current instruction executes is called pipelining.

ADDRESSING MODES

- The different ways in which a processor can access data are referred to as its addressing modes.
- In assembly language statements, the addressing mode is indicated in the instruction itself.
- The various addressing modes are

1. REGISTER ADDRESSING MODE

- It is the most common form of data addressing.
- Transfers a copy of a byte/word from source register to destination register.

INSTRUCTION	SOURCE	DESTINATION
MOV AX,BX	REGISTER BX	REGISTER AX

- It is carried out with 8 bit registers AH,AL,BH,BL,CH,CL,DH & DL or with 16 bit registers AX,BX,CX,DX,SP,BP,SI and DI.
- It is important to use registers of same size.
- Never mix an 8 bit register with a 16 bit register i.e. MOV AX, BL

EXAMPLES

MOV AL, BL : Copies BL into AL
MOV ES, DS : Copies DS into ES
MOV AX, CX : Copies CX into AX

2. IMMEDIATE ADDRESSING MODE

- The term immediate implies that the data immediately follow the hexadecimal opcode in the memory.
- Note that immediate data are constant data.
- It transfers the source immediate byte/word of data in destination register or memory location.

INSTRUCTION	SOURCE	DESTINATION
MOV CH,3AH	DATA 3AH	REGISTER AX

EXAMPLES

MOV AL, 90 : Copies 90 into AL
MOV AX, 1234H : Copies 1234H into AX
MOV CL, 10000001B : Copies 100000001 binary value into CL

3. DIRECT ADDRESSING MODE

- In this scheme, the address of the data is defined in the instruction itself.
- When a memory location is to be referenced, its offset address must be specified

INSTRUCTION	SOURCE	DESTINATION
MOV AL,[1234H]	ASSUME DS=1000H 1000**0** H + 1234 H 11234H MEMORY LOCATION 11234 H	REGISTER AL

EXAMPLES

MOV AL, [1234H] : Copies the byte content of data segment memory location 11234H into AL.

MOV AL, NUMBER : Copies the byte content of data segment memory location NUMBER into AL.

4. REGISTER INDIRECT ADDRESSING MODE

- Register Indirect Addressing allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI and SI.
- It transfers byte/word between a register and a memory location addressed by an index or base registers.
- The symbol [ ] denote indirect addressing.

INSTRUCTION	SOURCE	DESTINATION
MOV CL,[BX]	ASSUME DS=1000H ASSUME BX=0300H 1000**0** H + 0300 H 10300H MEMORY LOCATION 10300H	REGISTER CL

EXAMPLES

MOV CX, [BX] : Copies the word contents of the data segment memory location addressed by BX into CX.

MOV [DI], BH : Copies BH into the data segment memory location addressed by DI.

MOV [DI], [BX] : Memory to Memory moves are not allowed except with string instructions.

5. BASE PLUS INDEX ADDRESSING MODE

- Base plus index addressing is similar to indirect addressing because it indirectly addresses memory data
- This type of addressing uses one base register (BP or BX) and one Index Register (DI or SI) to indirectly address memory.

INSTRUCTION	SOURCE	DESTINATION
MOV [BX+SI],CL	REGISTER CL	ASSUME DS=1000H ASSUME BX=0300H ASSUME SI =0200H 1000**0**H + 0300H + 0200H 10500H MEMORY LOCATION 10500H

EXAMPLES

MOV CX, [BX+DI] : Copies the word contents of the data segment memory location addressed by BX plus DI into CX.

MOV CH, [BP+SI] : Copies the byte contents of the stack segment memory location addressed by BP plus SI into CH.

6. REGISTER RELATIVE ADDRESSING MODE

- In this case, the data in a segment of memory are addressed by adding the displacement to the content of base or an index register (BP, BX ,DI or SI).
- Transfers a byte/word between a register and the memory location addressed by an index or base register plus a displacement.

INSTRUCTION	SOURCE	DESTINATION
MOV [BX+4],CL	REGISTER CL	ASSUME DS=1000H ASSUME BX=0300H 1000**0**H + 0300H + 4H 10304H MEMORY LOCATION 10304H

EXAMPLES

MOV [SI+4], BL : Copies BL into the data segment memory location addressed by SI plus 4

7. BASE RELATIVE PLUS INDEX ADDRESSING MODE

- The base relative plus index addressing mode is similar to the base plus index addressing mode but it adds a displacement to form a memory address.

- Transfers a byte or word between a register and the memory location addressed by a base and an index register plus a displacement.

INSTRUCTION	SOURCE	DESTINATION
MOV [BX+SI+05],CL	REGISTER CL	ASSUME DS=1000H ASSUME BX=0300H ASSUME SI =0200H 1000**0**H + 0300H + 0200H +05H 10505H MEMORY LOCATION 10505H

EXAMPLES

MOV DH, [BX+DI+20H] : Copies the byte contents of the data segment memory location addressed by the sum of BX, DI and20H into DH

INSTRUCTION SETS

1. DATA TRANSFER INSTRUCTIONS

1.1 GENERAL PURPOSE BYTE OR WORD TRANSFER INSTRUCTIONS

INSTRUCTIONS	COMMENTS
MOV MOV Destination,Source Eg : MOV CX,04H	Copy byte or word from specified source to specified destination.
PUSH PUSH Source Eg: PUSH BX	Copy specified word to top of stack.
POP POP Destination Eg: POP AX	Copy word from top to stack to specified location.
XCHG XCHG Destination,Source Eg: XCHG AX,BX	Exchange word or byte.

1.2 SIMPLE INPUT AND OUTPUT PORT TRANSFER INSTRUCTIONS

INSTRUCTIONS	COMMENTS
IN IN AX,Port_Addr Eg:IN AX,34H	Copy a byte or word from specified port to accumulator.
OUT OUT Port_Addr,AX Eg: OUT 2CH,AX	Copy a byte or word from accumulator to specified port.

1.3 SPECIAL ADDRESS TRANSFER INSTRUCTIONS

INSTRUCTIONS	COMMENTS
LEA LEA Register,Source Eg: LEA BX,PRICE	Load effective address of operand into specified register.
LDS LDS Register,Source Eg: LDS BX,[4326H]	Load DS register and other specified register from memory.

2. ARITHMETIC INSTRUCTIONS

2.1 ADDITION INSTRUCTIONS

INSTRUCTIONS	COMMENTS
ADD ADD Destination,Source Eg: ADD AL,74H	Add specified byte to byte or word to word.
ADC ADC Destination,Source Eg: ADC CL,BL	Add byte + byte + carry flag Add word+word + carry flag
INC INC Register Eg: INC CX	Increment specified byte or word by 1.
AAA	ASCII adjust after addition.
DAA	Decimal adjust after addition.

2.2 SUBTRACTION INSTRUCTIONS

INSTRUCTIONS	COMMENTS
SUB SUB Destination,Source Eg: SUB CX,BX	Subtract byte from byte or word from word.
SBB SBB Destination,Source Eg: SBB CH,AL	Subtract byte and carry flag from byte. Subtract word and carry flag from word.
DEC DEC Register Eg: DEC CX	Decrement specified byte or word by 1.
NEG NEG Register Eg: NEG AL	Form 2's complement.

CMP CMP Destination,Source Eg: CMP CX,BX **CF ZF SF** CX = BX 0 1 0 CX > BX 0 0 0 CX < BX 1 0 1	Compare two specified bytes or words.
AAS	ASCII adjust after subtraction.
DAS	Decimal adjust after subtraction.

2.3 MULIPLICATION INSTRUCTION

INSTRUCTIONS	COMMENTS
MUL MUL Source MUL CX	Multiply unsigned byte by byte or unsigned word by word. When a byte is multiplied by the content of AL, the result is kept into AX. When a word is multiplied by the content of AX, MS Byte in DX and LS Byte in AX.
IMUL IMUL Source IMUL CX	Multiply signed byte by byte or signed word by word.
AAM	ASCII adjust after multiplication. It converts packed BCD to unpacked BCD.

2.4 DIVISION INSTRUCTIONS

INSTRUCTIONS	COMMENTS
DIV	Divide unsigned word by byte Divide unsigned word double word by byte. When a word is divided by byte, the word must be in AX register and the divisor can be in a register or a memory location. After division AL (quotient) AH (remainder) When a double word is divided by word, the

DIV Source DIV BL DIV CX	double word must be in DX:AX pair and the divisor can be in a register or a memory location. After division AX (quotient) DX (remainder)
AAD	ASCII adjust before division BCD to binary convert before division.
CBW	Fill upper byte of word with copies of sign bit of lower byte.
CWD	Fill upper word of double word with sign bit of lower word.

3. BIT MANIPULATION INSTRUCTIONS

3.1 LOGICAL INSTRUCTIONS

INSTRUCTIONS	COMMENTS
NOT NOT Destination Eg: NOT BX	Invert each bit of a byte or word.
AND AND Destination,Source Eg: AND BH,CL	AND each bit in a byte/word with the corresponding bit in another byte or word.
OR OR Destination,Source Eg: OR AH,CL	OR each bit in a byte or word with the corresponding bit in another byte or word.
XOR XOR Destination,Source Eg: XOR CL,BH	XOR each bit in a byte or word with the corresponding bit in another byte or word.

3.2 SHIFT INSTRUCTIONS

INSTRUCTIONS	COMMENTS
SHL/SAL SHL Destination,Count CF<-MSB LSB<-0	Shift Bits of Word or Byte Left, Put Zero(s) in LSB.
SHR SHR Destination,Count 0->MSB LSB->CF	Shift Bits of Word or Byte Right, Put Zero(s) in MSB.
SAR SAR Destination,Count MSB->MSB LSB->CF	Shift Bits of Word or Byte Right, Copy Old MSB into New MSB.

3.3 ROTATE INSTRUCTIONS

INSTRUCTIONS	COMMENTS
ROL	Rotate Bits of Byte or Word Left,MSB to LS and to CF.
ROR	Rotate Bits of Byte or Word Right,LSB to MSB and to CF.
RCL	Rotate Bits of Byte or Word Left,MSB to CF and CF to LSB.
RCR	Rotate Bits of Byte or Word Right,LSB TO CF and CF TO MSB.

4. PROGRAM EXECUTION TRANSFER INSTRUCTIONS

4.1 UNCONDITIONAL TRANSFER INSTRUCTION

INSTRUCTIONS	COMMENTS
CALL	Call a Subprogram/Procedure.
RET	Return From Procedure to Calling Program.
JMP	Goto Specified Address to Get Next Instruction (Unconditional Jump to Specified Destination).

4.2 CONDITIONAL TRANSFER INSTRUCTION

INSTRUCTIONS	COMMENTS
JC	Jump if Carry Flag CF=1.
JE/JZ	Jump if Equal/Jump if Zero Flag (ZF=1).
JNC	Jump if No Carry i.e. CF=0
JNE/JNZ	Jump if Not Equal/Jump if Not Zero(ZF=0)
JNO	Jump if No Overflow.
JNP/JPO	Jump if Not Parity/Jump if Parity Odd.
JNS	Jump if Not Sign(SF=0)
JP/JPE	Jump if Parity/Jump if Parity Even (PF=1)
JS	Jump if Sign (SF=1)
LOOP	Loop Through a Sequence of Instructions Until CX=0.
JCXZ	Jump to Specified Address if CX=0.

ASSEMBLY LANGUAGE PROGRAMMING

INTRODUCTION

- Assembly Language uses two, three or 4 letter mnemonics to represent each instruction type.
- Low level Assembly Language is designed for a specific family of Processors : the symbolic instruction directly relate to Machine Language instructions one for one and are assembled into machine language
- To make programming easier, many programmers write programs in assembly language
- They then translate Assembly Language program to machine language so that it can be loaded into memory and run.

ADVANTAGES OF ASSEMBLY LANGUAGE

- A Program written in Assembly Language requires considerably less Memory and execution time than that of High Level Language.
- Assembly Language gives a programmer the ability to perform highly technical tasks.
- Resident Programs (that resides in memory while other programs execute) and Interrupt Service Routine (that handles I/P and O/P) are almost always developed in Assembly Language.
- Provides more control over handling particular H/W requirements.
- Generates smaller and compact executable modules.
- Results in faster execution.

TYPICAL FORMAT OF AN ASSEMBLY LANGUAGE INSTRUCTION

LABEL	OPCODE FIELD	OPERAND FIELD	COMMENTS
NEXT:	ADD	AL,07H	; Add correction factor

- Assembly language statements are usually written in a standard form that has 4 fields.

- A **LABEL** is a symbol used to represent an address. They are followed by colon. Labels are only inserted when they are needed so it is an optional field.

- The **OPCODE FIELD** of the instruction contains the mnemonics for the instruction to be performed. The instruction mnemonics are sometimes called as operation codes.

- The **OPERAND FIELD** of the statement contains the data, the memory address, the port address or the name of the register on which the instruction is to be performed.

- The final field in an assembly language statement is the **COMMENTS** which start with semicolon. It forms a well-documented program.

ASSEMBLY LANGUAGE PROGRAM DEVELOPMENT TOOLS

1. EDITOR

- An Editor is a Program which allows you to create a file containing the Assembly Language statements for your Program.

2. ASSEMBLER

- An Assembler Program is used to translate the assembly language mnemonics for instruction to the corresponding binary codes.

3. LINKER

- A Linker is a Program used to join several files into one large .obj file. It produces .exe file so that the program becomes executable.

4. LOCATOR

- A Locator is a program used to assign the specific address of where the segment of object code are to be loaded into memory.
- It usually converts .exe file to .bin file.

5. DEBUGGER

- A Debugger is a program which allows you to load your .obj code program into system memory, execute program and troubleshoot.
- It allows you to look at the content of registers and memory locations after your program runs.
- It allows to set the breakpoint.

6. EMULATOR

- An Emulator is a mixture of hardware and software.
- It is used to test and debug the hardware and software of an external system such as the prototype of a Microprocessor based system.

TYPES OF ASSEMBLERS

1. **One Pass Assembler**
    - This assembler goes through or scans the assembly language program once and translates the assembly language program into binary codes.
    - This assembler has the program of defining forward references i.e a JUMP instruction using an address that appears later in the program must be defined by the programmer.

2. **Two Pass Assembler**
    - This assembler goes through or scans the assembly language program twice.
    - In the first scan, the assembler the table of symbols which consists of labels with their corresponding addresses.
    - In second scan, the assembler translates the assembly language program into the binary codes.
    - No address calculations are needed to be done by the programmer for JUMP instructions
    - It is more efficient and easier to use

MACRO ASSEMBLER

A macro assembler translates a program written in macro language into the binary codes (machine language). A macro language is the one in which all the instruction sequences can be defined using macros. A macro is an instruction sequence having specific name that appears repeatedly in a program. The macro assembler replaces a macro name with the appropriate instructions sequence, each time it encounters a macro name.

Example: a macro to add two 16 bit numbers

```
ADDITION MACRO
        MOV AX, num1
        MOV BX, num2
        ADD AX, BX
      ENDM
```

ASSEMBLER DIRECTIVES

- Assembly Language supports a number of statements that enable to control the way in which a source program assembles and lists. These Statements are called Directives.
- They act only during the assembly of a program and generate no machine executable code.
- The most common Directives are:

1. PAGE DIRECTIVE

- The PAGE Directive helps to control the format of a listing of an assembled program.
- It is optional Directive.
- At the start of program, the PAGE Directive designates the maximum number of lines to list on a page and the maximum number of characters on a line.
- Its format is

  PAGE [LENGTH],[WIDTH]
- Omission of a PAGE Directive causes the assembler to set the default value to PAGE 50,80

2. TITLE DIRECTIVE

- The TITLE Directive is used to define the title of a program to print on line 2 of each page of the program listing.
- It is also optional Directive.
- Its format is

  TITLE [TEXT]

TITLE "PROGRAM TO PRINT FACTORIAL NO"

3. SEGMENT DIRECTIVE

- The SEGMENT Directive defines the start of a segment.
- A Stack Segment defines stack storage, a data segment defines data items and a code segment provides executable code.
- The format (including the leading dot) for the directives that defines the stack, data and code segment are

```
.STACK [SIZE]
.DATA
    ……. Initialize Data Variables
.CODE
```

- The Default Stack size is 1024 bytes.
- To use them as above, Memory Model initialization should be carried out.

4. MEMORY MODEL DEFINTION

- The different models tell the assembler how to use segments to provide space and ensure optimum execution speed.
- The format of Memory Model Definition is

  .MODEL [MODEL NAME]

- The Memory Model may be TINY, SMALL, MEDIUM, COMPACT, LARGE AND HUGE.

MODEL TYPE	DESCRIPTION
TINY	All DATA, CODE & STACK Segment must fit in one Segment of Size <=64K.
SMALL	One Code Segment of Size <=64K. One Data Segment of Size <=64 K.
MEDIUM	One Data Segment of Size <=64K. Any Number of Code Segments.
COMPACT	One Code Segment of Size < =64K. Any Number of Data Segments.
LARGE	Any Number of Code and Data Segments.
HUGE	Any Number of Code and Data Segments.

5. THE PROC DIRECTIVE

- The Code Segment contains the executable code for a program, which consists of one or more procedures, defined initially with the PROC Directive and ended with the ENDP Directive.
- Its Format is given as:

```
PROCEDURE NAME    PROC
                ……..
                …….
                …….
PROCEDURE NAME    ENDP
```

6. END DIRECTIVE

- As already mentioned, the ENDP Directive indicates the end of a procedure.
- An END Directive ends the entire Program and appears as the last statement.
- Its Format is

  END [PROCEDURE NAME]

7. THE EQU DIRECTIVE

- It is used for redefining symbolic names

EXAMPLE
DATAX DB 25
DATA EQU DATAX

8. THE .STARTUP AND .EXIT DIRECTIVE

- MASM 6.0 introduced the .STARTUP and .EXIT Directive to simplify program initialization and Termination.
- .STARTUP generates the instruction to initialize the Segment Registers.
- .EXIT generates the INT 21H function 4ch instruction for exiting the Program.

DEFINING TYPES OF DATA

The Format of Data Definition is given as
[NAME] DN [EXPRESSION]

EXAMPLES

```
STRING DB 'HELLO WORLD'
NUM1 DB 10
NUM2 DB 90
```

DEFINITION	DIRECTIVE
BYTE	DB
WORD	DW
DOUBLE WORD	DD
FAR WORD	DF
QUAD WORD	DQ
TEN BYTES	DT

- Duplication of Constants in a Statement is also possible and is given by

[NAME] DN [REPEAT-COUNT DUP (EXPRESSION)]

EXAMPLES

```
DATAX DB 5 DUP(12)          ; 5 Bytes containing hex 0c0c0c0c0c
DATA DB 10 DUP(?)           ; 10 Words Uninitialized

DATAZ DB 3 DUP(5 DUP(4))    ; 44444 44444 44444
```

1. CHARACTER STRINGS

- Character Strings are used for descriptive data.
- Consequently DB is the conventional format for defining character data of any length
- An Example is
  DB 'Computer City'
  DB "Hello World"
  DB "NCIT College"

2. NUMERIC CONSTANTS

#BINARY : VAL1 DB 10101010B
#DECIMAL : VAL1 DB 230
#HEXADECIMAL : VAL1 DB 23H

8086 MODES OF OPERATION

There are two available modes of operation for 8086.

1. Minimum mode of Operation
2. Maximum Mode of Operation

Minimum mode is obtained by connecting the mode selection *MN*/ $\overline{MX}$ to +5V.
Maximum mode is obtained by connecting *MN*/ $\overline{MX}$ to ground.

1. Minimum Mode of Operation

- In minimum mode of operation, 8086 provides all control signals needed to implement the memory and I/O interfacing.
- For minimum mode 8288 bus controller is not required
- The minimum mode signals can be divided into the following basic groups:
  a) Address/Data Bus
  b) Status Signals
  c) Control Signals
  d) Interrupt and DMA Signals

a) Address/ Data Bus

- Serves two functions
- 20 bit address bus (A_0-A_{19})
- 16 bit data lines (D_0-D_{15}) multiplexed with address lines (A_0-A_{15})

b) Status Signals

- 4 most significant address lines (A_{16}-A_{19}) are multiplexed with status signals (S_3-S_6)
- Bit S_3 and S_4 are used to specify 8086 internal segment registers.

S_4	S_3	Segment Register
0	0	Extra
0	1	Stack
1	0	Code
1	1	Data

- Bit S_5 is the logic level of the internal enable flag
- Bit S_6 is always at logic '0' level.

c) Control Signals

The following are the control signals associated with 8086 minimum mode

- **ALE (Address Latch Enable):** latch address and data
- **$\overline{\text{BHE}}$ (Bus High Enable):** $\overline{\text{BHE}}$ must be made low for read or write operation and it acts as a status signal S_7
- **$\overline{\text{M}}$ /IO (Memory/IO):** IO operation if $\overline{\text{M}}$ /IO=1
  Memory operation if $\overline{\text{M}}$ /IO=0
- **DT/ $\overline{\text{R}}$ (Data Transmit/Receive):** Data Transmitting if DT/ $\overline{\text{R}}$ =1
  Data Receiving if DT/ $\overline{\text{R}}$ =0
- **$\overline{\text{RD}}$ (Read):** indicates read bus cycle when $\overline{\text{RD}} = 0$
- **$\overline{\text{WR}}$ (Write):** indicates write bus cycle when $\overline{\text{WR}} = 0$
- **$\overline{\text{DEN}}$ (Data Enable):** tells the external device when to put data
- **READY :** used to insert wait states into the bus

d) Interrupt and DMA Signals

- The key interrupt signals are **INTR** (Interrupt Request) and **INTA** (Interrupt Acknowledge)
- The DMA interface of the 8086 minimum mode consists of **HOLD** and **HLDA** signals

Minimum Mode Timing Diagrams

a) Read Cycle Timing Diagram for Minimum Mode (Input)

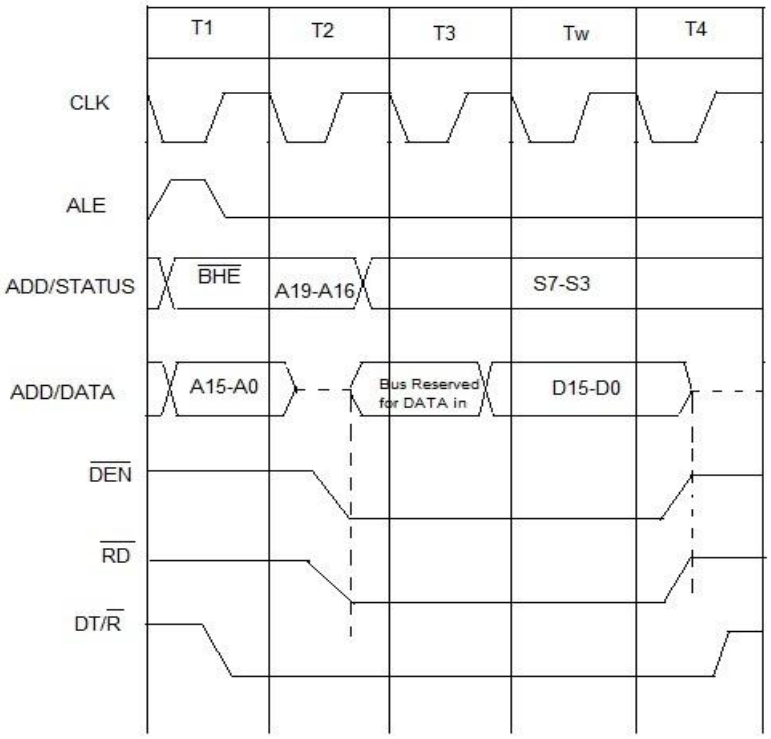


Fig: Read Cycle Timing Diagram

b) Write Cycle Timing Diagram for Minimum Mode (Output)

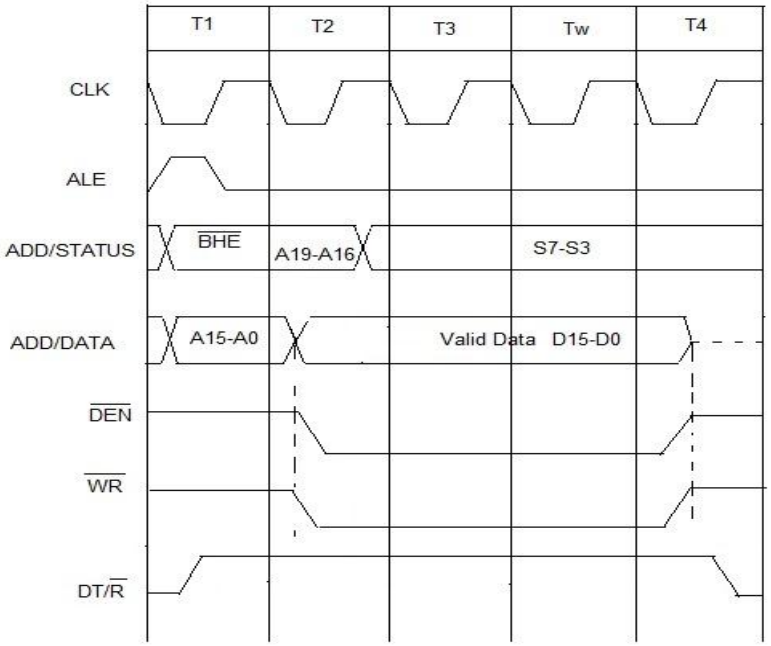


Fig: Write Cycle Timing Diagram

2. Maximum Mode

- In maximum mode, the 8086 is operated by connecting the MN/$\overline{MX}$ pin to ground.
- In this mode, the processor derives the status signals S_2, S_1, S_0. Another chip called Bus Controller derives the control signals using these status signals.
- In maximum mode, there may be more than one microprocessors (co-processors) in the system configuration.
- The Bus Controller receives the three status signals S_2, S_1, S_0 from 8086 and generates the signals that are needed to control the memory, IO and Interrupt Interfaces.

Status Inputs			CPU Cycles	8288 Command
$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$		
0	0	0	Interrupt Acknowledge	$\overline{INTA}$
0	0	1	Read I/O Port	$\overline{IORC}$
0	1	0	Write I/O Port	$\overline{IOWC}$, $\overline{AIOWC}$
0	1	1	Halt	None
1	0	0	Instruction Fetch	$\overline{MRDC}$
1	0	1	Read Memory	$\overline{MRDC}$
1	1	0	Write Memory	$\overline{MWTC}$, $\overline{AMWC}$
1	1	1	Passive	None

- **$\overline{\text{INTA}}$** is used to issue interrupt acknowledge pulses to the interrupt controller or to the interrupting device
- **$\overline{\text{IORC}}$** (IO Read Command), **$\overline{\text{IOWC}}$** (IO write Command) enable an IO interface to read or write data from or to the address port
- **$\overline{\text{MRDC}}$** (Memory Read Command), **$\overline{\text{MWTC}}$** (Memory Write Command) are used to read from or write into memory locations
- For both IO and Memory Write Command signals, the advance signals namely **$\overline{\text{AIOWC}}$** (Advance IO Write Command) and **$\overline{\text{AMWTC}}$** (Advance Memory Write Command) are available.

Maximum Mode Timing Diagrams

a) Read Cycle Timing Diagram

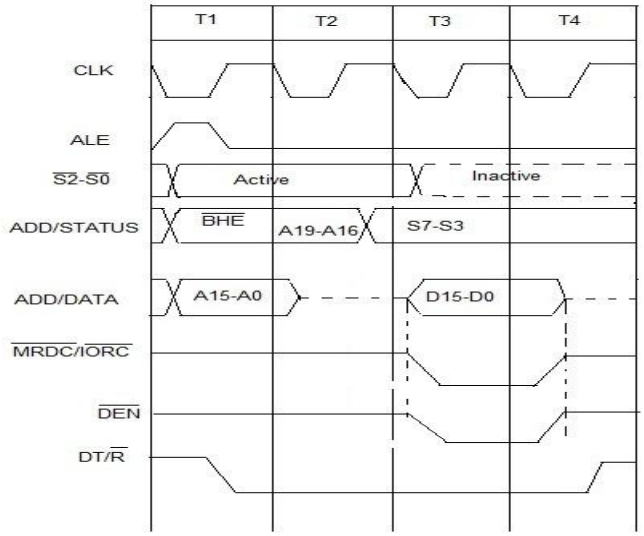


Fig: Read Cycle Timing Diagram

b) Write Cycle Timing Diagram

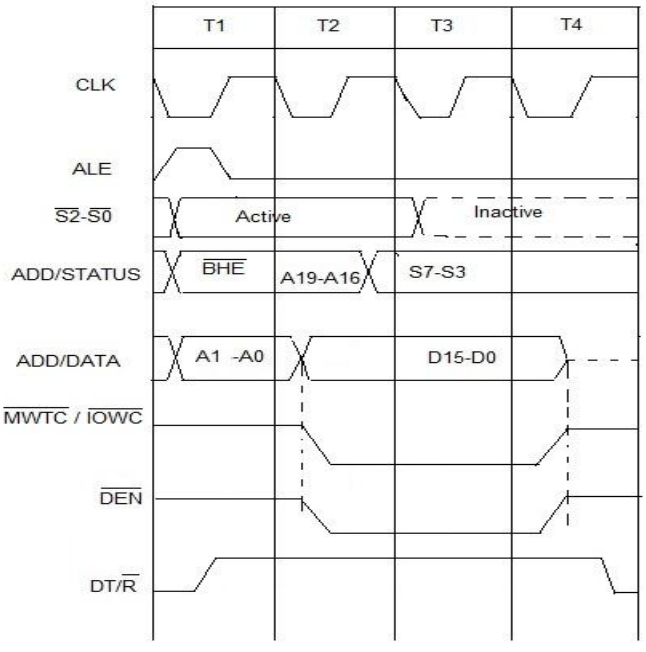


Fig: Write Cycle Timing Diagram

MODULAR PROGRAMMING

1. LINKING AND RELOCATION

Program is composed from several smaller modules. Modules could be developed by separate teams concurrently. The modules are assembled producing .OBJ modules (Object modules). The DOS linking program links the different object modules of a source program and function library routines to generate an integrated executable code of the source program.

The main input to the linker is the .OBJ file that contains the object modules of the source programs. The linker program is invoked using the following options.

C> LINK

or

C>LINK MS.OBJ

The .OBJ extension is a must for a file to be accepted by the LINK as a valid object file.

The first object may generate a display asking for the object file, list file and libraries as inputs and an expected name of the .EXE file to be generated. The output of the link program is an executable file with the entered filename and .EXE extension. This executable filename can further be entered at the DOS prompt to execute the file. Linking is necessary because of the number of codes to be linked for the final binary file.

The linked file in binary for ***run*** on a computer is commonly known as executable file or simply '.exe.' file. After linking, there has to be re-allocation of the sequences of placing the codes before actually placement of the codes in the memory. The loader program performs the task of reallocating the codes after finding the physical RAM addresses available at a given instant. The ***loader*** is a part of the operating system and places codes into the memory after reading the '.exe' file. This step is necessary because the available memory addresses may not start from 0x0000, and binary codes have to be loaded at the different addresses during the run. The loader finds the appropriate start address.

In a computer, the loader is used and it loads a program that is ready to run, into a section of RAM. A program called *locator* reallocates the linked file and creates a file for permanent location of codes in a standard format.

2. STACK

- The stack is a block of memory that may be used for temporarily storing the contents of the registers inside the CPU.
- It is a top-down data structure whose elements are accessed using the stack pointer (SP) which gets decremented by two as we store a data word into the stack and gets incremented by two as we retrieve a data word from the stack back to the CPU register.

- The process of storing the data in the stack is called 'pushing into' the stack and the reverse process of transferring the data back from the stack to the CPU register is known as 'popping off' the stack.
- The stack is essentially *Last-In-First-Out* (LIFO) data segment. This means that the data which is pushed into the stack last will be on top of stack and will be popped off the stack first.
- The stack pointer is a 16-bit register that contains the offset address of the memory location in the stack segment. Stack Segment register (SS) contains the base address of the stack segment in the memory.

3. PROCEDURES/SUBROUTINES

- Procedure or a subroutine or a function is a key concept for modular programming, the essential way to reduce complexity.
- A procedure is a reusable set of instructions that has a name.
- Only one copy of the procedure is stored in the memory; and it can be called as many times as needed.
- As only one copy is stored, it saves memory; but has execution time overhead for the CALL and RETURN operations.
- CALL transfers control to the procedure just like in JUMP; but unlike a JUMP, procedure has a RETURN instruction which returns control to the instruction following the CALL instruction
- In order to implement such a return, the necessary information is stored on a stack, before transferring control to the procedure.
- In program, procedure starts with **PROC** directive and ends with **ENDP** directive.
- **PROC** directive is followed by the type of procedure: **NEAR** or **FAR**.

1.3.1 Calling a Procedure

- CALL instruction followed by procedure name is used in a program to call a procedure.
- Procedure calls are of two types: Near CALL and Far CALL.

Near CALL

- A procedure may be in the same code segment as that of the main program (**Intra segment**). In such a case, we specify only IP. This is known as **NEAR CALL.**

Example:

```
Main program:
….
…..
CALL DISPLAY      ; calling a procedure. Transfer control to procedure named as DISPLAY
…..
…….. Procedure Definition……..
DISPLAY PROC NEAR
MOV DX, OFFSET string1
MOV AH, 02H
INT 21H
RET
DISPLAY ENDP
```

FAR CALL

- A procedure may be in a different code segment (**Inter segment)**. In such a case, we need to specify both IP and CS (directly or indirectly). This is known as a **FAR CALL.**

Example:

```
DISPLAY PROC FAR
. . . . . .
DISPLAY ENDP
```

Now, a CALL to DISPLAY is assembled as FAR CALL.

1.3.2 Return from Procedure

- We use a **RET** instruction to return from the called procedure.
- The control returns to the instruction following the CALL instruction in the calling program. Corresponding to the two varieties of CALL instructions (near & far), two forms of RET instructions (near & far) exist.
- Near RET instruction POPS 16-Bit word from the stack and places it in the IP.
- Far RET instruction POPS two 16-Bit words from the stack and places them in IP & CS.
- In ALP, RET is written within the procedure, before the ENDP directive and the Assembler will automatically select the proper (Near or Far) RET instruction.

4. MACRO

- **MACRO** is a group of instructions with a name.
- When a macro is invoked, the associated set of instructions replaces the macro name in the program. This macro expansion is done by a Macro Assembler and it happens before assembly. Thus the actual Assembler sees the expanded source.

MACRO Definition:

A macro has a name. The body of the macro is defined between a pair of directives, **MACRO** and **ENDM**.

Examples of Macro Definitions:

```
DISPLAY MACRO          ; Definition of a Macro named DISPLAY
MOV DX, OFFSET string1
MOV AH, 09H
INT 21H
ENDM                                ; end of Macro
```

MACROS Vs PROCEDURES

1) Procedure:

- Only one copy exists in memory. Thus memory consumed is less.
- Called when required
- Return address (IP or CS:IP) is saved (PUSH) on stack before transferring control to the subroutine through CALL instruction. It should be popped (POP) again when control comes back to calling program with RET instruction.
- Execution time overhead is present because of the call and return instructions.
- If more lines of code, better to write a procedure than a macro.

2) Macro:

- When a macro is invoked, the corresponding code is inserted into the source. Thus multiple copies of the same code exist in the memory leading to greater space requirements.
- However, there is no execution overhead because there are no additional call and return instructions.
- No use of stack for operation.
- Good if few lines of code are in the Macro body.

CHAPTER 5: INTERRUPTS

INTRODUCTION

Interrupt is considered as an emergency signal to which the MP responds as soon as possible. When the microprocessor receives an interrupt signal, it suspends the current executing program and jumps to an interrupt service routine (ISR) to respond to the incoming interrupt When a device interrupts, it actually wants the microprocessor to give a service which is equivalent to asking the microprocessor to call a subroutine. This subroutine is call Interrupt Service Routine (ISR).

SOURCES OF INTERRUPTS

There are three sources of interrupts and they are as follows:

1. Processor Interrupt
2. Software Interrupt
3. Hardware Interrupt

Processor Interrupt

These interrupts are generated by processor itself, usually in response to an error condition. For example: In 8086 Type 0 interrupt occurs when attempt to divide by zero which is a processor interrupt.

Software Interrupt

These are special instructions that trigger an interrupt response to processor.
In 8086 the general form of software interrupt instruction is INT nnH (eg: INT 21H)

Hardware Interrupt

Hardware interrupts are interrupt request initiated by external hardware.
8086 have two pins reserved for hardware interrupts. They are NMI and INTR

CLASSIFICATIONS OF INTERRUPTS

Interrupts can be classified as:

- Maskable Interrupt or Non-Maskable Interrupt
- Vectored Interrupt or Non-Vectored Interrupt

Maskable Interrupt

- The interrupts which can be blocked or delayed by using instructions are called maskable interrupts.
- In 8085, the RESET interrupts (RST 5.5, RST 6.5 and RST 7.5) and INTR are maskable interrupts. They can be enabled/disabled by using instructions EI/DI.
- In 8086, INTR is maskable interrupt. It can be enabled/disabled by using instructions STI/CLI.

Non-Maskable Interrupt

- Those interrupts which cannot be blocked by instructions are termed as non-maskable interrupts.
- In 8085, TRAP is only non-maskable interrupt and it is used for power failure and emergency cutoff.
- In 8086, NMI is non-maskable interrupt.

Vectored Interrupt

- The interrupts for which address of ISR is already known to MP are called vectored interrupts.
- In 8085, RESET interrupts (RST 5.5, RST 6.5 and RST 7.5) are vectored interrupts.

Interrupt	Vector Address (Hex)
RST 5.5	002C
RST 6.5	0034
RST 7.5	003C

Non-Vectored Interrupt

- In non-vectored interrupts, the interrupting device needs to supply the address of the ISR to the microprocessor.
- In 8085, INTR is non-vectored interrupt.

8085 INTERRUPTS

8085 microprocessor consists of five interrupt signals: INTR, RST 5.5, RST 6.5, RST 7.5 and TRAP

1. **INTR (Interrupt Request)**
   - The INTR is only non-vectored interrupt.
   - INTR is maskable interrupt and can be masked by using EI (Enable Interrupt)/DI (Disable Interrupt) instruction pair.
   - INTR can be used for external hardware interrupts for various applications.
2. **RESET Interrupts**
   - 8085 consists of three reset interrupts: RST 5.5, RST 6.5 and RST 7.5
   - They are vectored interrupts whose address are defined in interrupt vector table
   - They are maskable interrupts and can be enabled or disabled individually by using RIM and SIM instructions.

Interrupt	Vector Address (Hex)
TRAP	0024
RST 5.5	002C
RST 6.5	0034
RST 7.5	003C

Fig: 8085 Interrupt Vector Table

3. **TRAP**
   - It is only non-maskable interrupt of 8085 MP.
   - It is a vectored interrupt whose address is defined in interrupt vector table.
   - It is used as external hardware interrupt source and is used for power failure and emergency shutoff instruction.

8086 INTERRUPTS

- An 8086 interrupt can come from any one of 3 sources.
- One source is an external signal applied to the Nonmaskable Interrupt (NMI) or to the (INTR) input pin. An interrupt caused by a signal applied to one of these inputs (NMI or INTR) is referred as **Hardware Interrupt**.
- A second source of an interrupt is execution of the interrupt instruction INT nn. This is referred as Software **Interrupt.**
- The third source of an interrupt is some error conditions produced in 8086, by the execution of an instruction, referred as processor interrupt.

INTERRUPT VECTOR TABLE (IVT)

- 8086 consists of 256 interrupts stored from memory location 00000H to 003FFH i.e. 1024 Bytes (1KB) of memory.
- These 256 interrupts are stored in memory location forming a table which is called Interrupt Vector Table (IVT).
- An Interrupt Vector contains the address (segment CS and offset IP) of the Interrupt Service Routine.
- Each vector is a 4 byte long and contains the starting address of the Interrupt Service Routine.
- The first 2 bytes of the vector contain the offset address (IP) and the last two bytes contain the segment address (CS).

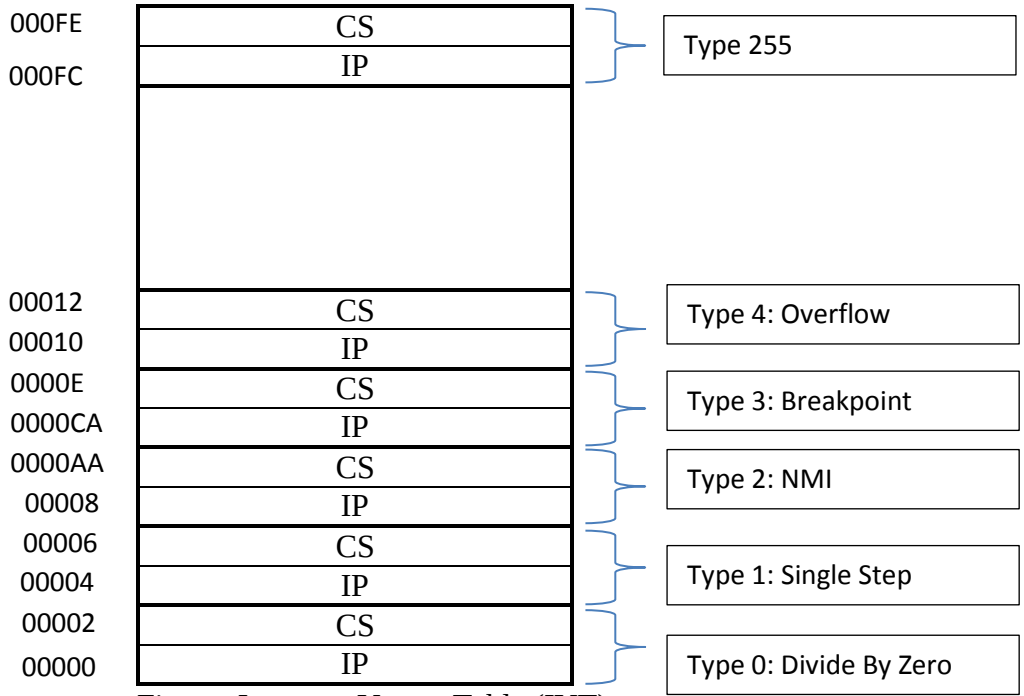


Figure: Interrupt Vector Table (IVT)

8086 PREDEFINED INTERRUPT TYPES

1. DIVIDE BY ZERO INTERRRUPT: TYPE 0

- Divide error occurs when the result of division overflow or whenever an attempt is made to divide by zero.
- The 8086 type 0 is automatic and cannot be disabled in any way

2. SINGLE STEP INTERRUPT : TYPE 1

- If the 8086 trap flag is set, the 8086 will automatically do a type 1 interrupt after each instruction executes.
- When a MP is interrupted using Type 1 interrupt, it will execute one instruction and stop so that we can then examine the contents of registers and memory locations.
- In other words, in single step mode, a system will stop after it executes each instruction and waits for further direction from us.

3. NON MASKABLE INTERRUPT : TYPE 2

- A result of placing logic 1 on the NMI input pin causes type 2 interrupt.
- This input is non-maskable, which means that it cannot be disabled.
- Another common use of type 2 interrupt is to save program data in case of a system power failure or to deal with some other catastrophic failure conditions.
- Some external circuitry detects when the AC power to the system fails and sends an Interrupt signal to the NMI.

4. BREAKPOINT INTERRUPT : TYPE 3

- Used to insert a breakpoint into the program. When a break point is inserted, the system executes the instructions up to the breakpoint, and then stops execution.
- The INT 3 instruction is often used to store a breakpoint in a program for debugging.

5. OVERFLOW INTERRUPT : TYPE 4

- A special vector used with the INTO instruction.
- The 8086 overflow flag, OF, will be set if the signed result of an arithmetic operation on two signed numbers is too large to be represented in the destination register or memory location
- The INTO instruction interrupts the program if an overflow condition exists as reflected by overflow flag (OF).

PRIORITY OF 8086 INTERRUPTS

INTERRUPTS	PRIORITY
DIVIDE ERROR, INT n , INTO	HIGHEST
NMI	
INTR	
SINGLE STEP	LOWEST

PRIORITISING INTERRUPTS

1. Polled Interrupts

A polling procedure is used to identify the interrupt source having the highest priority.
Only one branch address is used for all interrupts.
The priority of each interrupt source determines the order in which it is polled. The source with the highest priority is tested first, and if its interrupt signal is on, control branches to a routine that services that source. Otherwise, the source with the next lower priority is tested, and so on.

2. Vectored Interrupt

A vectored interrupt unit functions as an overall manager in an interrupt system environment. The unit accepts interrupt requests from many sources, determines which request has the highest priority, and issues an interrupt request to the computer based on this determination. To speed up the operation, each interrupt source has its own interrupt vector address to access its own service routine directly.

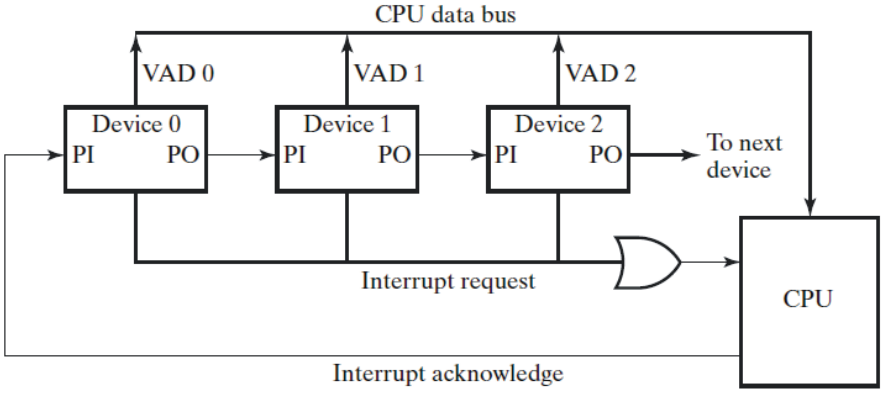


Figure: Vectored Interrupt

Operation:

1. All devices that can request an interrupt are connected serially and in priority order with the highest priority device placed first on the daisy chain, farthest from the CPU, and the lowest priority device placed last and closest to the CPU.
2. First, any (or all) of the devices signal an interrupt on the Interrupt Request line.
3. Next, the CPU acknowledges the interrupt on the Interrupt Acknowledge line.
4. A device on the line passes the Interrupt Acknowledge signal to the next lower priority device only if it has NOT requested service.
5. The first device on the priority chain requiring service asserts it interrupt vector address (VAD) on the CPU data bus.
6. The CPU services the device
7. The Interrupt Acknowledge signal is passed to the next lower priority device and steps 2 – 6 are performed for the next lower priority device.

INTERRUPT PROCESSING

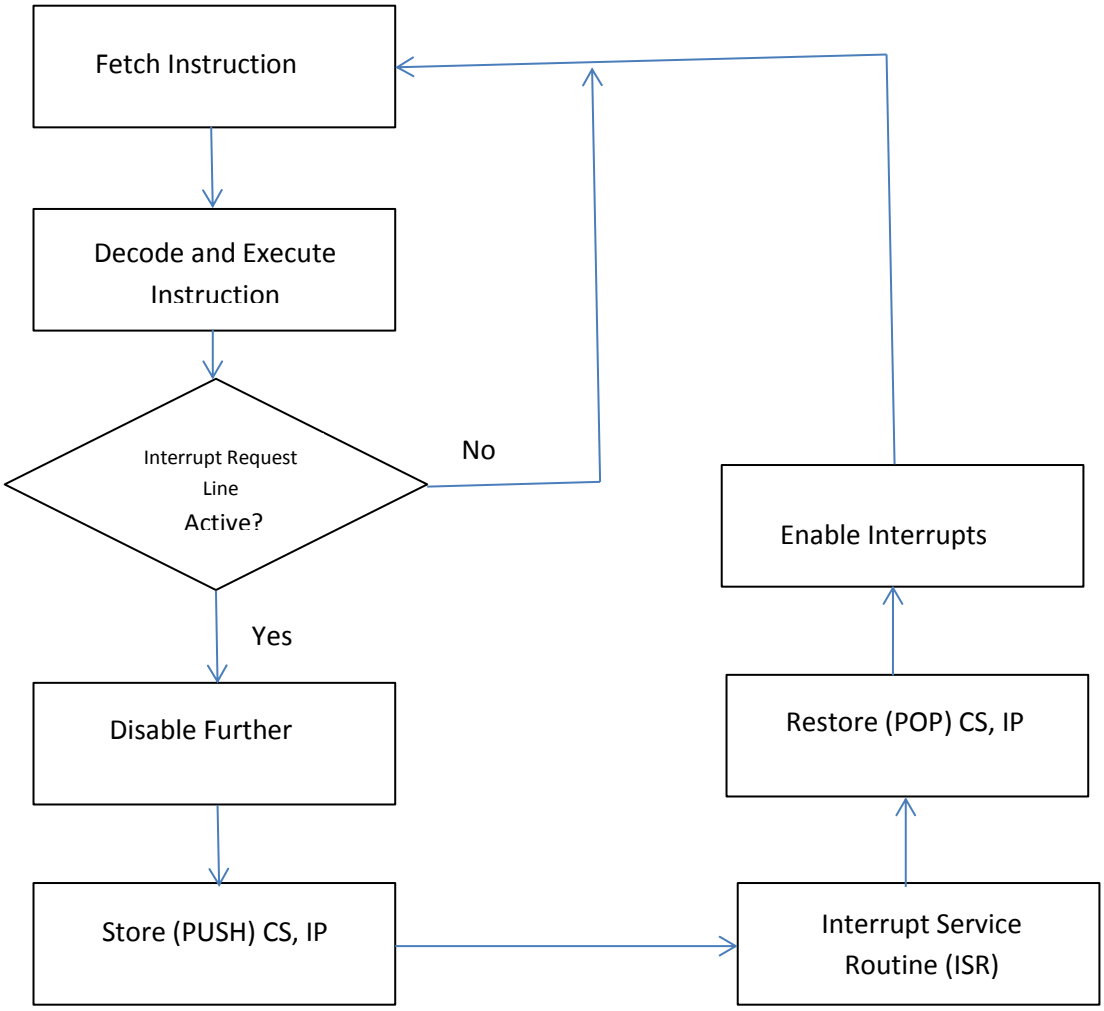


Figure: Interrupt Processing

- The processor checks for interrupts. If interrupt has occurred, processor will complete the instruction currently being executed.
- The processor will disable the further interrupts.
- The processor stores the current state of program by PUSH operation i.e. the value of flag register and CS: IP will be stored into stack by PUSH operation.
- The processor will load the address of the ISR and execute the ISR. At the end of ISR, instruction IRET is used which makes the processor return from the ISR to the original program.
- After the execution of ISR, the processor restores the previous state of program i.e. it will restore the value of flag register and CS: IP from stack by POP operation.
- The processor enables the interrupts and then starts program execution from where it has been interrupted.

CHAPTER-4
BUS STRUCTURE AND MEMORY DEVICES

BUS STRUCTURE

In any microprocessor system, the system bus consists of a number of separate lines. Each line is assigned a particular function. Fundamentally in any system, the system bus can be classified into three functional groups: the address, the data and control lines or buses.

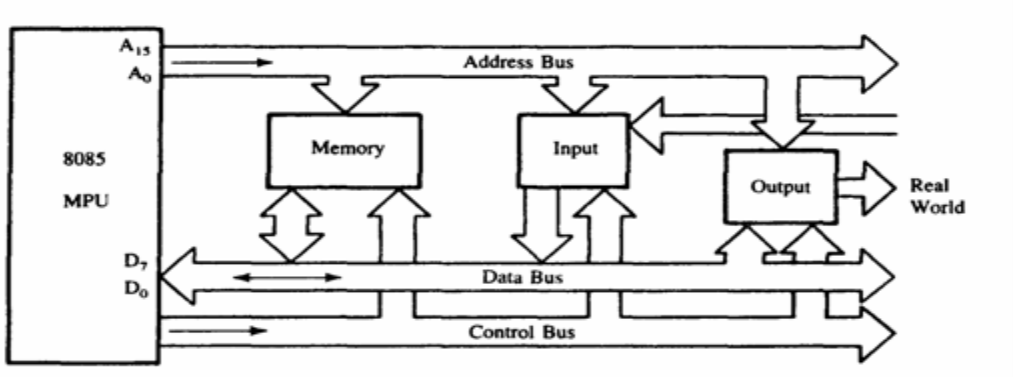


Fig: Bus Structure

Data Bus

- The data bus provides path for transferring data between the microprocessor system and the peripherals.
- The data bus consists of a number of separate lines, generally 8, 16, 32 or 64. The number of lines is referred as the width of the data bus.
- Since, each line carry only one bit at a time, the number of lines determines how many bits can be transmitted at a time.
- The width of data bus is a key factor in determining the overall system performance.

Address Bus

- The address bus which consists of a number of separate lines, are used to designate the source or destination of the data on data bus. For example, if the CPU requires reading a word (8, 16, 32 or 64 bits of data) from memory, it put the address of the desired word on the address bus.
- The width of address bus (i.e number of lines) determines the maximum possible memory capacity of the system.
- The address bus is also used to address IO ports.

Control Bus

- The control bus is a group of lines used to control the access to and the use of the data and address bus since the data and address bus are shared by all components of microcomputer system. Hence control bus provides a means of controlling their use.
- The control bus carries the control signals. The control signals transmit both command and timing information between the system modules.
- The timing signals indicate the validity of data and address information; whereas the command signals specify the operations to be performed.
- Some control signals are: Memory Read, Memory Write, IO Read, IO Write, Interrupt Request, Interrupt Acknowledge, Bus request, Bus Grant etc.

SYNCHRONOUS AND ASYNCHRONOUS BUS

Synchronous Bus

In a synchronous bus, the occurrence of the events on the bus is determined by a clock. The clock transmits a regular sequence of 0's and 1's of equal duration. A single 1-0 transition is called clock cycle or bus cycle and defines a time slot. All other devices on the bus can read the clock live, and all events start at the beginning of the clock cycle. In synchronous bus, all devices are tied to a fixed rate, and hence the system cannot take advantage of device performance. It is easier to implement.

Asynchronous Bus

In an asynchronous bus, the timing is maintained in such a way that occurrence of one event on the bus follows and depends on the occurrence of previous event. Asynchronous bus are faster than the synchronous bus as the events are independent of the processor timing.

MEMORY DEVICES

- In a microcomputer system, the memory is used to store both instructions and data. A memory can be volatile or non-volatile. The contents of a volatile memory are lost if power is turned off; whereas the non-volatile memory retains its contents after the power is switched off.
- A microcomputer's memory system can be divided into three groups:
  a) Processor Memory
  b) Primary or Main Memory
  c) Secondary Memory

a) Processor Memory

- Microprocessor's registers and cache memory are referred to as processor memory.
- The registers are used to hold results temporarily when computation is in progress. The speed of registers is equal to the speed of microprocessor, since they are fabricated using the same technology. Although the use of registers enhances the execution speed, the cost involved in the approach forces the microprocessor designers to include only few registers inside the processors.
- Beside registers, these days microprocessor also consists of a separate memory in order to store the frequently needed information, known as cache memory. The speed and efficiency of program execution has been significantly improved by the use of cache memory.

b) Primary/Main Memory

It is the storage area where all programs are executed. The microprocessors can directly access only those items that are stored in primary memory. Hence, all programs and data must be in primary memory prior to execution.

Primary memories can be divided into two main groups:

- Random Access Memory (RAM)
- Read Only Memory (ROM)

Random Access Memory (RAM)

From its name itself, it is obvious that this memory uses random access mode. It is read/write memory i.e information can be read from it or can be written on it. However, it is volatile in nature. Being random access mode, it is faster with access time in nano seconds. The Random Access Memories are basically of two types:

- Dynamic RAM (DRAM)
- Static RAM (SRAM)

Dynamic RAM: Dynamic RAM stores data in capacitors; it can hold data for few milliseconds. Thus dynamic RAM need to be refreshed by using external refresh circuitry. The charging of capacitor and refreshing is done by MOS transistors. Due to its small storage cell, DRAM exhibits compactness i.e. greater number of bits can be stored in small chips. Thus, DRAMs are cheaper than SRAM. However the interfacing circuit of DRAM is complicated because of refreshing circuit.

Static RAM: The static RAM is made up of flip-flops. A single flip-flop stores one bit information i.e. 0 or 1. Each flip-flop is called storage cell of the memory device. The individual storage cell is addressed with the help of row and column decoder in the SRAM.

Read Only Memory (ROM)

- ROM can only be read. This is non-volatile in nature. ROMs are divided into two types: Bipolar ROMs and MOS ROMs. The bipolar ROMs are faster than MOS ROMs. Each type is further divided into two common types: Mask ROM and Programmable ROM (PROM). MOS ROMs contain one more type; erasable ROM (EPROM and EEPROM).
- Mask ROMs are programmed by a masking operation performed on the chip during the manufacturing process. The content of Mask ROMs are permanent and cannot be changed by the user.
- Programmable ROMs (PROMs) can be programmed by user by means of proper equipment. The content of this type of memory cannot be changed unless it is erased by special device.
- EPROMs can be programmed by using PROM programmer. Its memory can be erased by exposing the chip via a window on a chip to ultraviolet light. EPROMs chip should be remove from the microcomputer system for programming.
- EEPROMs can be erased and programmed without removing the chip from the microcomputer system. The content of EEPROMs can be erased electrically.

c) Secondary Memory

Secondary Memory refers to the storage medium comprising slow devices such as hard disks. These devices are used to hold large data files and huge programs such as compilers and database management systems that are needed by the processor frequently. The secondary memories are also referred to as auxiliary or backup storage. On the basis of access modes, secondary memories are of three types:

- Random Access Secondary Memory
- Sequential Access Memory
- Semi random Access Memory

Random Access Secondary Memory: A random access secondary memory uses the random access mode. In random access mode, any location of the memory can be accessed at random. In this mode, the memory access time is

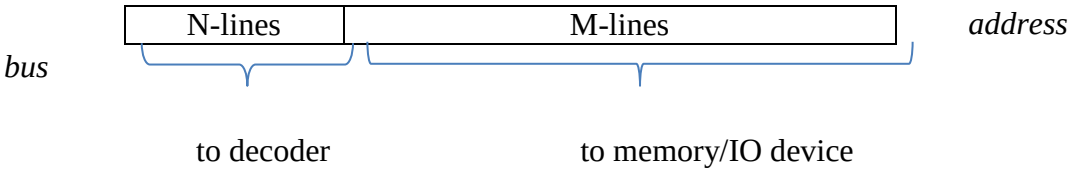
independent of the location from which data is accessed. Example: Semiconductor Memories

Sequential Access Memory: In these types of memory, the access method is strictly sequential. In sequential access memory, the memory access time depends on the location in which data is stored. They are also called serial access memories. Example: Magnetic Tapes

Semi random Access Memory: These memories combine both random and sequential access mode. This is done in order to achieve a compromise between random access and sequential access memories because the random access is more expensive than sequential access though it is much faster. Example: Floppy, Hard Disks

ADDRESS DECODING

- The process of generating chip select signal ($\overline{CS}$) using the address lines of the microprocessor and a decoder or logic-gates, to interface memory devices or IO devices with microprocessor is called address decoding.
- During address decoding, the address lines of a microprocessor is divided into two parts. N most significant lines and M list significant bits. The N most significant lines are passed to decoder/logic-gates to generate $\overline{CS}$ signal, whereas the M list significant lines are passed directly to the device (memory/IO).



- Address decoding is of two types:
  Full Address Decoding (Unique/Absolute Address Decoding)
  Partial Address Decoding (Non-Unique Address decoding)

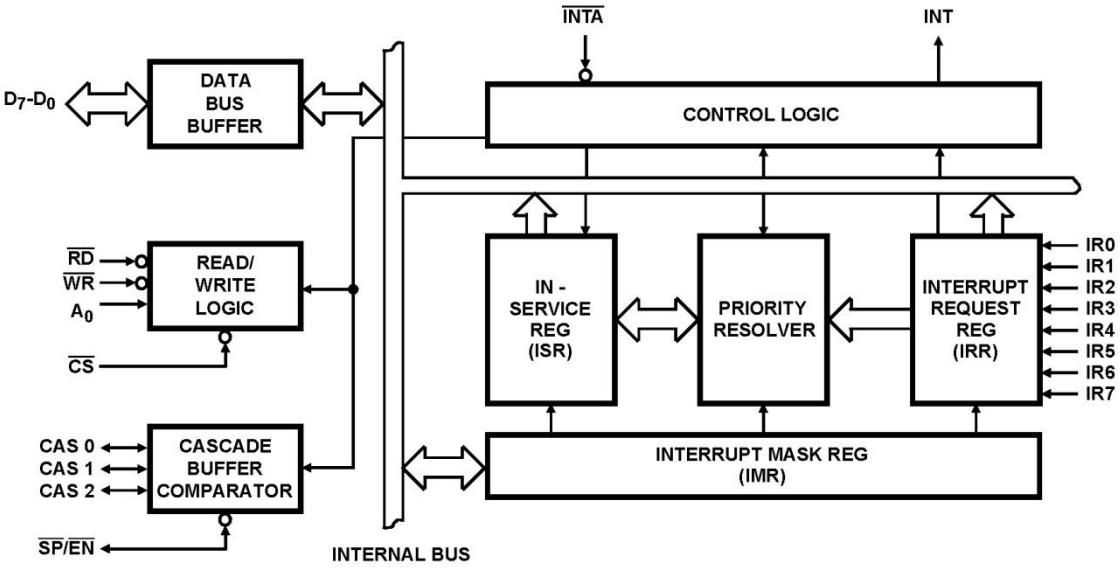
Full Address Decoding: If all the address lines of a microprocessor system is used to address a memory or IO devices, such address decoding is called full address decoding. The address of the memory/IO in this scheme is unique so it is also called unique address decoding or absolute address decoding.

Partial Address Decoding: If all the address lines of microprocessor is not used i.e. only some address lines are used to address a memory location or IO device then it is called partial address decoding. The address of the memory location/IO device is not unique i.e. the memory or IO device may have two or more address. It is also called non-unique address decoding.

CHAPTER 5
INPUT OUTPUT INTERFACES

8259A PROGRAMMABLE INTERRUPT CONTROLLER (PIC)

Block Diagram



Data Bus Buffer

- This tri-state, bidirectional 8-bit buffer is used to interface 8259A with the system data bus
- Control words and status information are transferred through data bus buffer.

Read/Write Control Logic

- The function of this block is to accept commands from CPU and control the device.
- It contains the initialization command word register (ICW) and operational command word register (OCW) to store various control formats for device operation.
- It has the following control signals-

  $\overline{CS}$ (Chip Select**):** A low on this input enables the 8259A.

  $\overline{WR}$ (Write): A low on this input enables the CPU to write control words to 8259A.

  $\overline{RD}$ (Read**):** A low on this input enables 8259A to send its status to the CPU.

  A_0: This line is connected to address line A_0 of CPU. It is used to select various command registers.

  INT (Interrupt): This pin is directly connected to INTR of MPU. This pin goes high whenever a valid interrupt request occurs. It is used to interrupt CPU.

  $\overline{INTA}$ (Interrupt Acknowledgement): This pin is used to enable 8259A interrupt-vector data onto the data bus. This pin is connected to INTA of CPU.

Status Registers

Interrupt Request Register (IRR)

- 8-bit register that indicates which interrupt request inputs are active

In-Service Register (ISR)

- 8-bit register that contains the level of interrupt being serviced

Interrupt Mask Register (IMR)

- 8-bit register that stores the interrupt mask bits and indicates which interrupts are masked off.

Priority Resolver

- This block determines the priority of the bits set in IRR.

Cascade Buffer

- This block is used for cascading multiple 8259s to handle more than 8 interrupts.
- CAS_2-CAS_0 cascade lines are used as output from master to slave 8259s
- SP/ $\overline{EN}$ (Slave Program / Enable) pin is used to identify master or slave 8259A. For master SP/ $\overline{EN}$ =1 and for slave SP/ $\overline{EN}$ =0.

Interfacing 8259A with Microprocessor

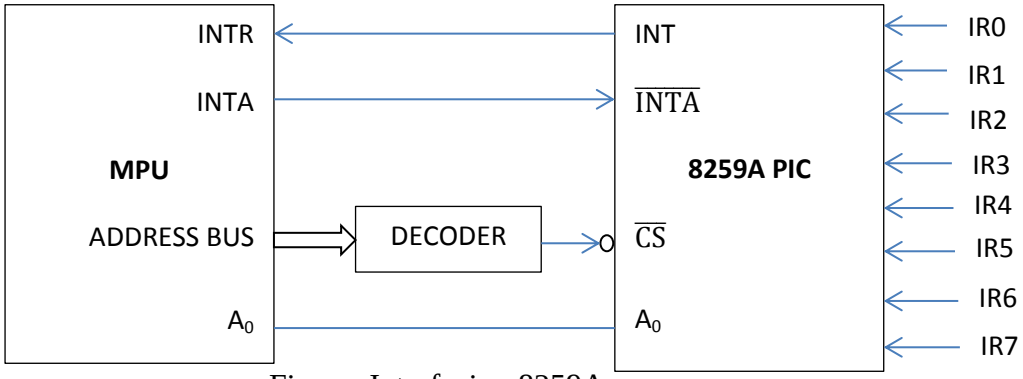


Figure: Interfacing 8259A

8259A MODES OF OPERATION

Commonly used modes of operation of 8259A are-

- Fully Nested Mode
- Rotating Priority Mode
- Special Masked Mode
- Polled Mode

1. Fully Nested Mode

This is general purpose mode in which all IRs (interrupt requests) are arranged from highest to lowest priority, with IR_0 as the highest and IR_7 as the lowest.

IR_0	IR_1	IR_2	IR_3	IR_4	IR_5	IR_6	IR_7
0	1	2	3	4	5	6	7

Highest Priority ... Lowest Priority

In addition any IR can be assigned the highest priority in this mode. The priority sequence will begin from that IR.

IR_0	IR_1	IR_2	IR_3	IR_4	IR_5	IR_6	IR_7
5	6	7	0	1	2	3	4
		↑	↑				
		Lowest Priority	Highest Priority				

2. **Rotating Priority Mode**
   It includes two modes: Automatic Rotation Mode and Specific Rotation Mode
   Automatic Rotation Mode
   In this mode, a device or IR_X after being serviced receives the lowest priority and the $IR_{(X+1)}$ receives the highest priority. Assume that IR_2 has just been serviced, then priority order will be

IR_0	IR_1	IR_2	IR_3	IR_4	IR_5	IR_6	IR_7
5	6	7	0	1	2	3	4
		↑	↑				
		Lowest Priority	Highest Priority				

   Specific Rotation Mode
   This mode is similar to the automatic rotation mode except that the user can select any IR for lowest priority, thus fixing all other priorities.

3. **Special Mask Mode**
   - This mode can be enabled by making ESMM and SMM bits as '1' in operation control word OCW3.
   - In this mode, by programming the mask register, it is possible to selectively enable the interrupts.

4. **Polled Mode**
   - Polled mode is enabled by making P=1 in OCW3.
   - In this mode of operation INT output of the 8259A is either not connected to INTR of MPU or the system interrupts are disabled by the software.
   - The peripherals are serviced by MPU by polling the interrupt request.

Q. How can you handle 18 interrupts using 8259A PIC?

- We can handle 18 interrupts using 8259A PIC by connecting three 8259As in cascade. Here, one 8259A is made master by connecting SP/ $\overline{EN}$ pin to +5V and remaining two 8259As are made slave by connecting SP/ $\overline{EN}$ pin to ground.
- The INT pin of each slave is connected to interrupt request (IR) pin of the master 8259A.
- The cascade buffer lines (CAS0-CAS2) of master 8259A must be connected to cascade buffer lines (CAS0-CAS2) of each slave.
- The INT pin of master 8259A is connected to INTR pin of the microprocessor whereas the $\overline{INTA}$ pin of master 8259A is connected to INTA pin of microprocessor

and the address line A_o of master 8259A is connected to address line A_o of the microprocessor.

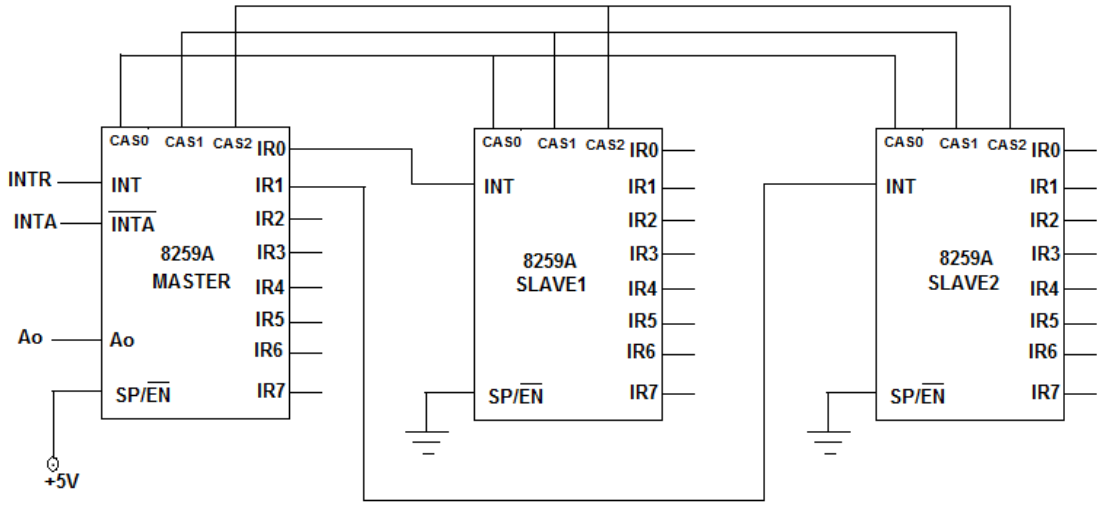
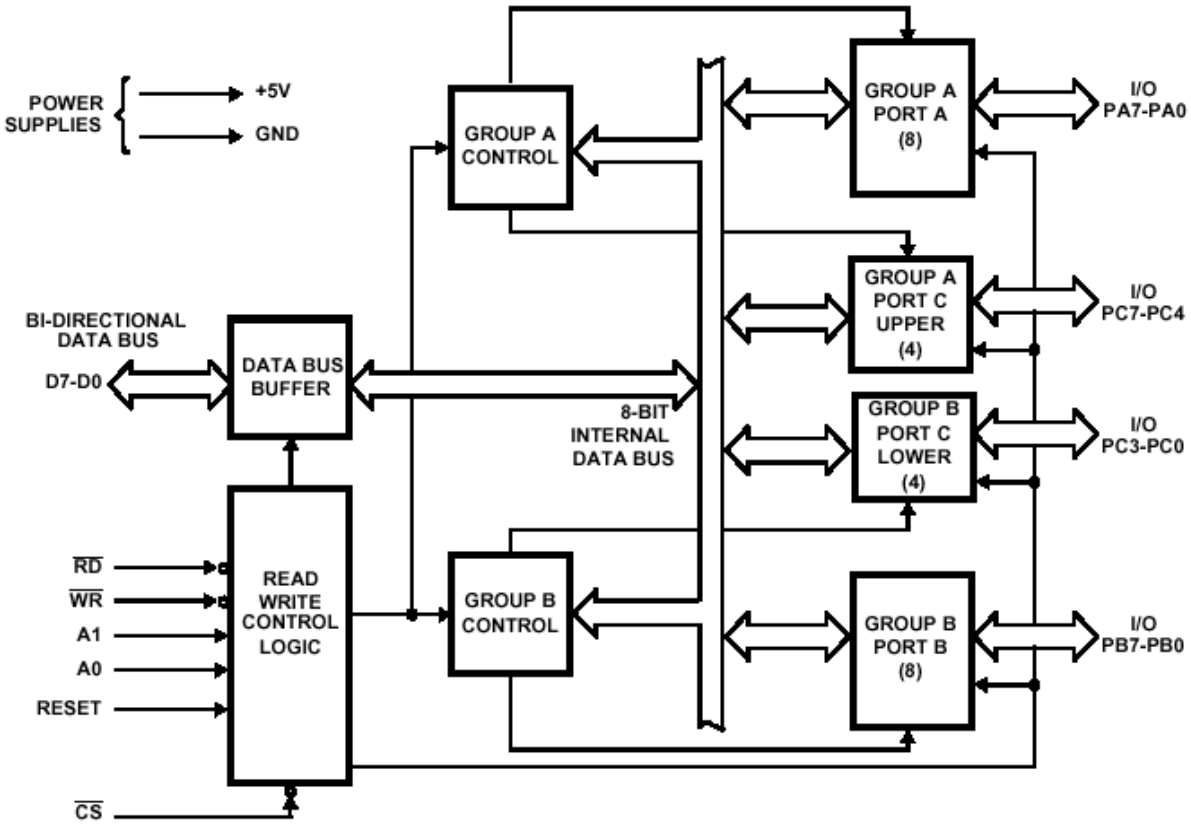


Fig: Cascading multiple 8259As

8255A PROGRAMMABLE PHERIPHERAL INTERFACE (PPI)

- 8255A I widely used programmable parallel I/O device. It can be programmed to transfer data under various conditions from simple I/O to interrupt I/O.
- It is a general purpose device that can be interfaced with almost any microprocessor for I/O interface.

BLOCK DIAGRAM



Data Bus Buffer

- This tri-state bidirectional 8-bit buffer is used to interface 8255A to the system data bus.
- Data, control words and status information are transferred through data bus buffer.

Read/Write Control Logic

- The function of this block is to manage all the internal and external transfer of both data and control words.
- It has six lines:
  - **$\overline{RD}$ (**Read**):** When the signal is low, the MPU reads data from the selected I/O port of 8255A.
  - **$\overline{WR}$ (**Write**):** When the signal is low, the MPU writes data into the selected I/O port or writes control word into control register.
  - **RESET** (Reset): This is an active high signal. It cleans the control register and set all ports in the input mode.
  - **$\overline{CS}$ (**Chip Select**), A_O** and **A_1:** These are the device select signals. $\overline{CS}$ is connected to decoded address; A_O and A_1 are generally connected to MPU address lines A_O and A_1 respectively.
    If $\overline{CS}$=0, 8255A is selected and if $\overline{CS}$=1, 8255A is not selected.
    A_O and A_1 specify the selection of I/O ports as shown below.

A_1	A_O	Port Selected
0	0	Port A
0	1	Port B
1	0	Port C
1	1	Control Register

Group A and Group B Controls

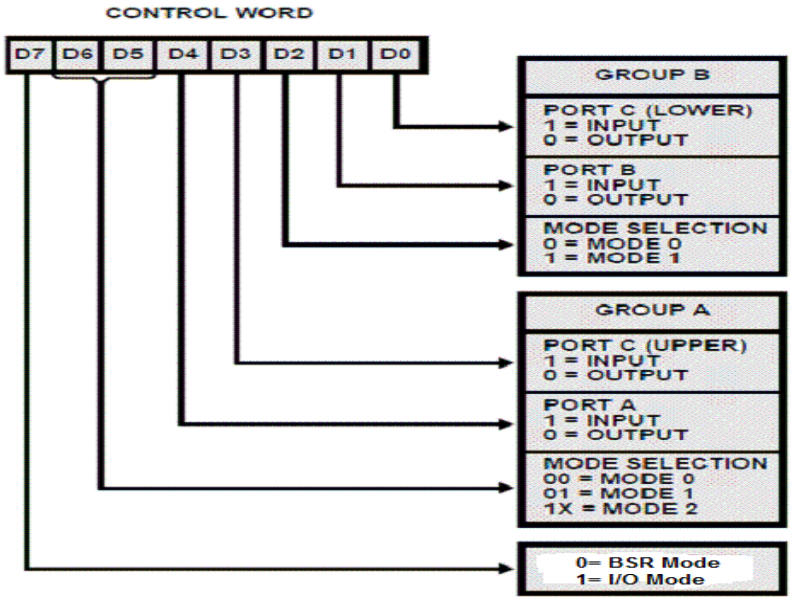
- Each of the control blocks accepts commands from read/write control logic and issues proper command to its associated ports.
  Group A Control: Port A and Port C Upper
  Group B Control: Port B and Port C Lower

I/O Ports

- The 8255A has 24 I/O pins that are grouped into three 8-bit parallel I/O ports: Port A (PA) Port B (PB) and Port C (PC)
- The 8-bit of port C can be used as individual bits or grouped into two 4-bit ports: Port C Upper (PC_U) and Port C Lower (PC_L)
- The functions of I/O ports are defined by writing a control word into control register.

CONTROL WORD OF 8255A

8255A consists of 8-bit control register as shown in figure below. The content of control register is called control word. The control word specifies an I/O function for each port. Control register can be accessed to write control word when A_0=1 and A_1=1. Control register is not accessible for read operation.



Q. Derive a control word for 8255A to configure:
Port A= i/p port
Port B= o/p port
Port C Lower= i/p port
Port C Upper= o/p port
In simple I/O mode with Group A in mode 0 and Group B in mode 1.

Solution:

Control Word =	1	0	0	1	0	1	0	1

=10010101
= 95H

8255A MODES OF OPERATION

8255A consists of two modes of operation.

1. I/O Mode (Input/Output Mode)
2. BSR Mode (Bit set/reset 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: Mode 0, Mode 1 and Mode 2.

Mode 0: Simple Input or Output Mode

In this mode, ports A and B are used as two simple 8-bit I/O port and port C as two 4-bit ports. Each port can be programmed to function simply as input port or output port. The I/O features in Mode 0 are:

- Output are latched
- Inputs are not latched
- Ports do not have handshake or interrupt capacity

Mode 1: Input or Output with Handshake

In this mode, handshake signals are exchanged between the MPU and peripherals during data transfer. The features of this mode are:

- Two ports PA and PB function as 8-bit I/O ports. They can be configured either as input or output.
- Each port PA and PB uses three lines from Port C as handshake signals. The remaining two lines of PC can be used for Simple I/O function (mode 0).
- Input and Output data are latched.
- Interrupt logic is supported.

Mode 2: Bidirectional Data Transfer

- This mode is used primarily in applications such as data transfers between two computers.
- In this mode Port A can be configured as the bidirectional port (data bus) and Port B either in mode 0 or mode 1.
- Port A uses five signals from Port C as handshake signals for data transfer. The remaining three signals form Port C can be used as Simple I/O or as handshake for Port B.

BSR Mode (Bit Set/Reset)

- The BSR control word is only connected with 8-bits of Port C, which can be set or reset by writing appropriate control word in the control register.
- I/O operations of port A and port B are not affected by a BSR control word.
- In BSR mode, individual bits of port C can be used for applications such as ON/OFF switch.

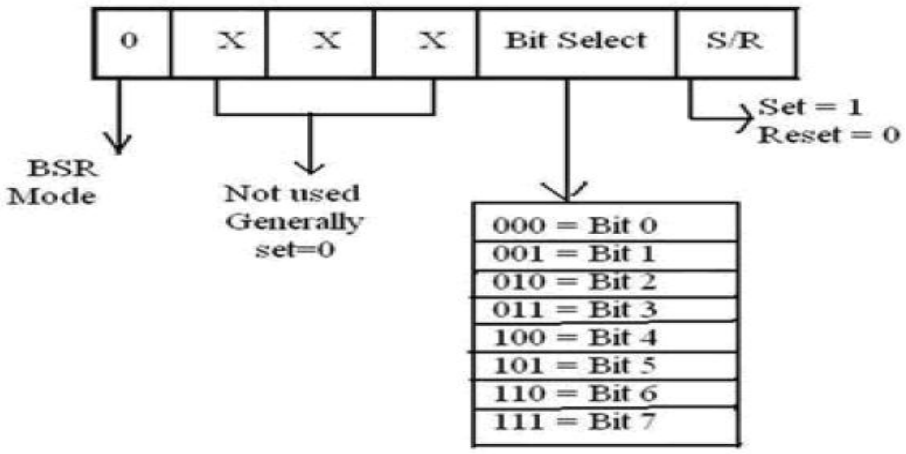


Fig: BSR Control Word

Q. Generate BSR control words to set bit PC_7 and PC_3 and reset PC_0 and PC_2.

Solution:

Control Word to Set PC_7=

0	0	0	0	1	1	1	1

= 00001111

=0FH

Control Word to Set PC_3= 00000111

= 07H

Control Word to Reset PC_0= 00000000

= 00H

Control Word to Reset PC_2= 00000100

=04H

8254 PROGRAMMABLE INTERVAL TIMER (PIT)

8254 PIT is functionally similar to software designed counters and timers.

It generates accurate time delays and can be used for the applications such as real-time clock, event counter, square wave generator and a complex waveform generator.

BLOCK DIAGRAM

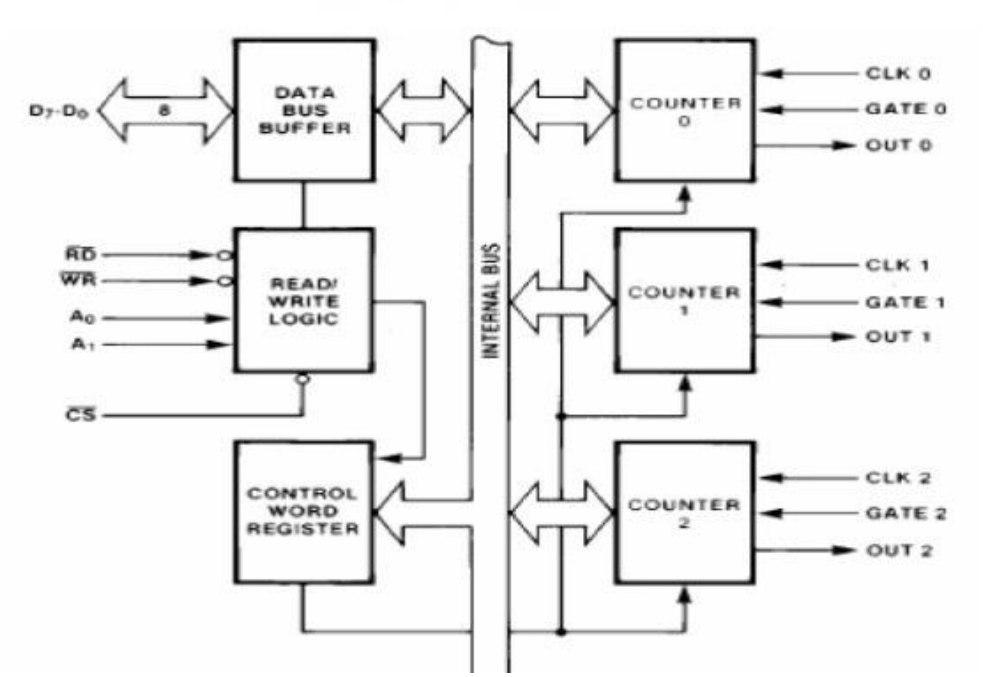


Fig: Block Diagram of 8254/8253 PIT

Data Bus Buffer

- This tri-state bidirectional 8-bit buffer is used to connect 8254 PIT to the data bus of MPU.

Control Logic

- The control section has five signals: $\overline{RD}$ (Read), $\overline{WR}$ (Write), $\overline{CS}$ (Chip Select) and address lines A_O, A_1.
- $\overline{RD}$ (Read) and $\overline{WR}$ (Write) are connected to $\overline{IOR}$ (IO Read) and $\overline{IOW}$ (IO Write) respectively in IO mapped IO and $\overline{RD}$ (Read) and $\overline{WR}$ (Write) are connected to $\overline{MEMR}$ (Memory Read) and $\overline{MEMW}$ (Memory Write) respectively in Memory mapped IO.
- Address lines A_O and A_1 are connected to the address lines A_O and A_1 of MPU and $\overline{CS}$ is connected to the decoded address.
- $\overline{CS}$ is used to enable or select the device. If $\overline{CS} = 0$ the chip is selected.
- The counters and control word register are selected according to signals on lines A_O and A_1 as shown in table below-

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

Control Word Register

- This register is accessed when A_O and A_1 are at logic 1.
- It is used to write a command /control word which specifies the counter to be used, its mode and either read or write operation.

Counters

- 8254 includes three identical 16-bit counters that can operate independently in any six modes of operation.
- To operate a counter, a 16-bit count is loaded into its register.
- The counter can count either in BCD or Binary.

CONTROL WORD OF 8254

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC—Select Counter

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Read-Back Command (see Read Operations)

M—Mode

M2	M1	M0	
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

RW—Read/Write

RW1	RW0	
0	0	Counter Latch Command (see Read Operations)
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

BCD

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

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

Solution:

Control Word=

0	0	1	1	0	1	0	1

=00110101

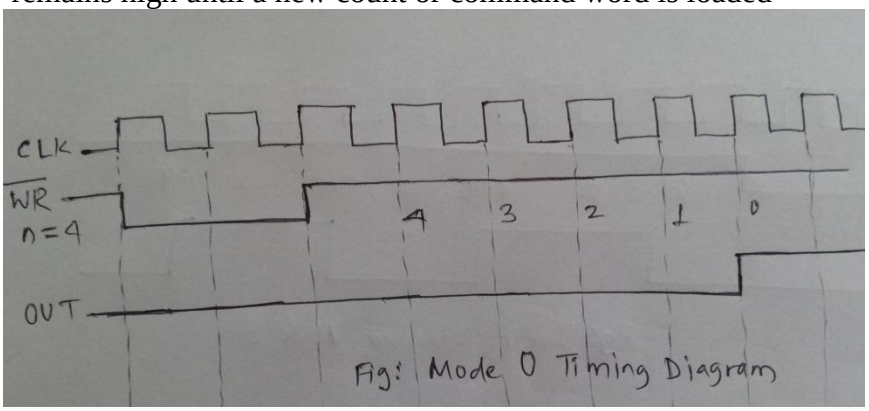
=35H

MODES OF OPERATION

8254 can operate in six different modes of operation.

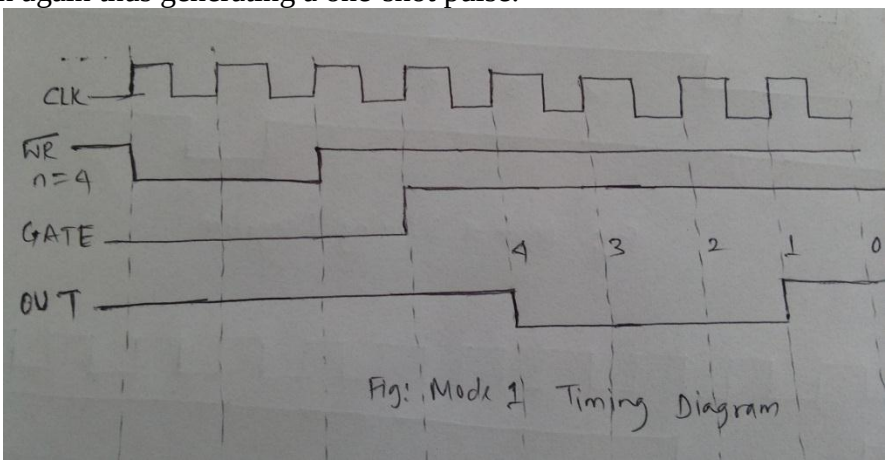
Mode 0: Interrupt on Terminal Count

- Initially OUT is high
- Once count is loaded in the register, the counter is decremented every cycle and when the count reaches zero, the OUT goes high
- The OUT remains high until a new count or command word is loaded



Mode 1: Hardware Retriggerable One-shot

- OUT is initially high
- When the GATE is triggered, the OUT goes low and at the end of count, the OUT goes high again thus generating a one-shot pulse.



Mode 2: Rate Generator

- This mode is used to generate a pulse equal the clock period at a given interval
- When count is loaded the OUT stays high until the count reaches 1 and then OUT goes low for one clock period
- The count is reloaded automatically and the pulse is generated continuously

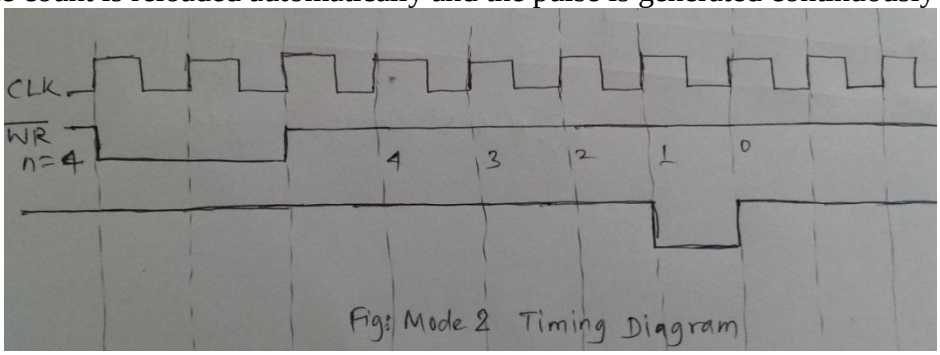


Fig: Mode 2 Timing Diagram

Mode 3: Square Wave Generator

- In this mode, when a count is loaded, the OUT goes high
- The count is decremented by two every clock cycle and when it reaches zero OUT goes low and the count is reloaded 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 = $\frac{\text{Frequency of Clock}}{\text{count}}$

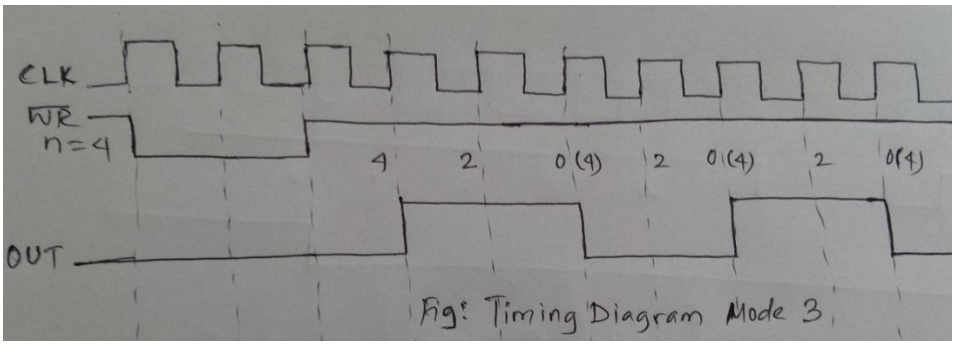


Fig: Timing Diagram Mode 3

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 reloaded for subsequent outputs

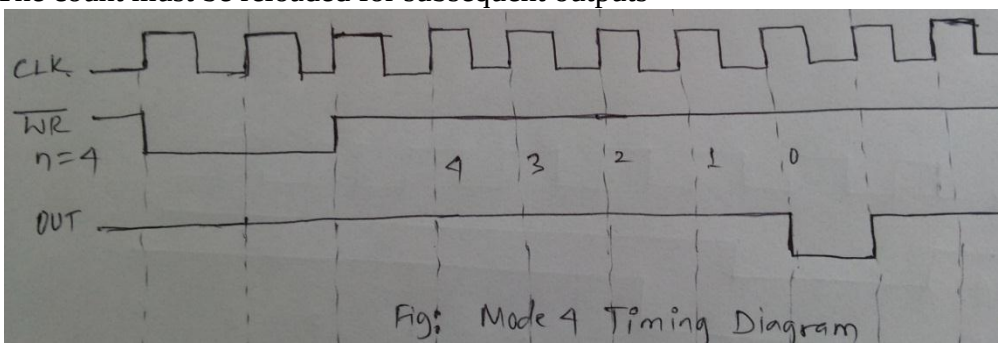
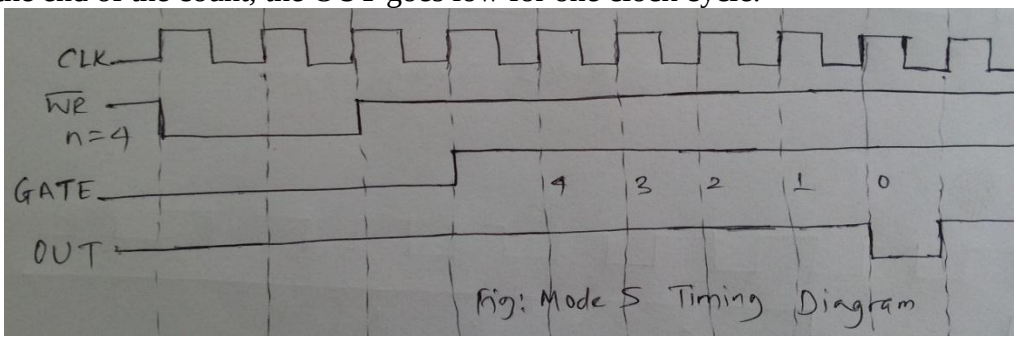


Fig: Mode 4 Timing Diagram

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.



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

Solution:

Given:

Counter= Counter 1

Mode of Operation= Mode 3: Square Wave Generator

Frequency of square wave= 1 KHz

Assume: Frequency of Clock be 2 MHz

Address of Counter 0= 80H

Address of Counter 1= 81H

Address of Counter 2= 82H

Address of Counter 3= 83H

We know that: Count $= \frac{\text{Frequency of Clock}}{\text{Frequency of Square Wave}}$

$= \frac{2\ MHz}{1\ Mhz}$

$= \frac{2000\ KHz}{1\ KHz}$

$= 1000$

Control Word =

0	1	1	1	0	1	1	1

= 01110111

=77H

Subroutine Program

```
SQUAREWAVE:   MVI A, 77H      ;move value of control word in A
              OUT 83H         ;load control word in A into control register
              MVI A, 00       ; move lower byte of count into A
              OUT 81H         ; move lower byte of count in A into Counter 1
              MVI A, 10       ; move higher byte of count into A
              OUT 81H         ; move higher byte of count in A into Counter 1
              RET             ; return from subroutine program
```

Q. Write a subroutine program to generate a pulse of width 50 microseconds using 8254 PIT. Take frequency of clock as 2 MHz and use Counter 0.

Solution:

Given:

Counter= Counter 0

Mode of Operation= Mode 1: Hardware Retriggerable One Shot

Time period of pulse= 50 µS

Frequency of Clock= 2 MHz

Assume: Address of Counter 0= 80H

Address of Counter 1= 81H

Address of Counter 2= 82H

Address of Counter 3= 83H

We know that: Count $= \frac{\text{Frequency of Clock}}{\text{Frequency of Pulse}}$

Count $= \frac{\text{Frequency of Clock}}{\frac{1}{\text{time period of pulse}}}$

$= \frac{2000000Hz}{\frac{1}{50\times10^{-6}}}$

$= 100$

$=0100$

Control Word =

0	0	1	1	0	1	0	1

= 00110101

=35H

Subroutine Program

```
PULSE:  MVI A, 35H      ; move value of control word in A
        OUT 83H         ; load control word in A into control register
        MVI A, 00       ; move lower byte of count into A
        OUT 80H         ; move lower byte of count in A into Counter 0
        MVI A, 01       ; move higher byte of count into A
        OUT 80H         ; move higher byte of count in A into Counter 0
        RET             ; return from subroutine program
```

DMA CONTROLLER

DMA (Direct Memory Access) and 8237 DMA CONTROLLER

- Direct memory access is an I/O technique commonly used for high speed data transfer.
- In DMA, the MPU releases the control of the buses to a device called DMA controller.
- The controller manages the data transfer between memory and peripherals under its control, thus bypassing the MPU.
- For all practical purposes, DMA controller is a processor capable of copying data at high speed from one location to another location.

THE 8237 DMA CONTROLLER

- The 8237 is a programmable DMA controller used as a peripheral interface circuit for microprocessor system.
- It is designed to improve system performance by allowing external devices to directly transfer information to or from system memory. Memory to memory transfer capability is also improved.
- It has four independent channels with each channel capable of transferring 64K bytes.
- The 8237 is designed to be used in conjunction with external 8-bit address register such as 8282.

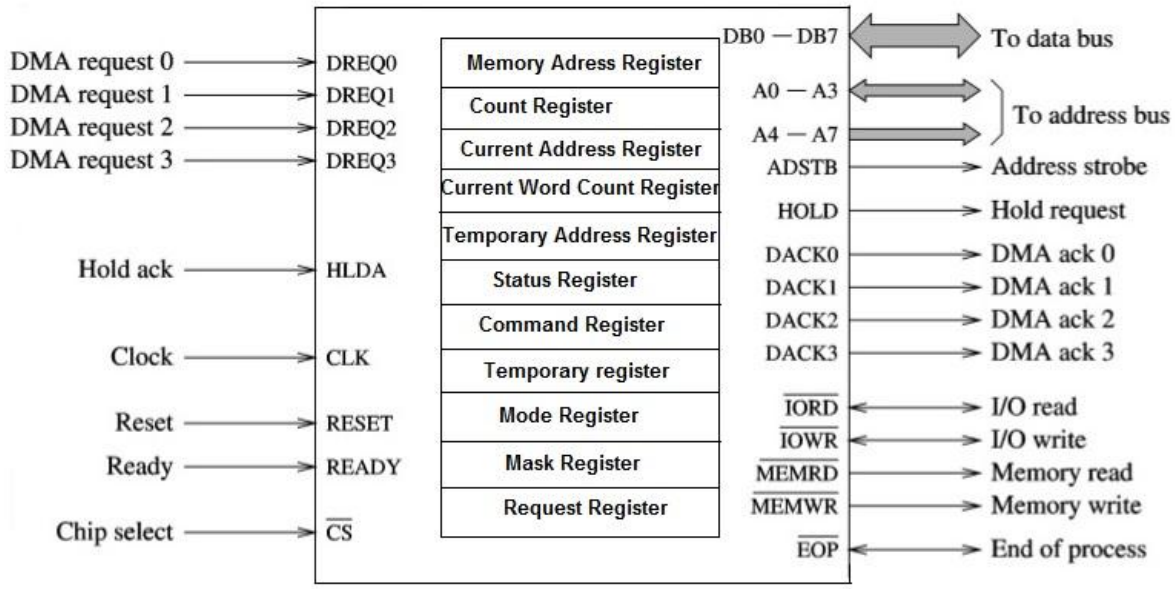


Figure: 8237 DMA Controller

- 8237 is a complex device. We can study its properties under two sections: DMA Channels and DMA Signals.

DMA Channels

- 8237 has four DMA Channels: CH0 to CH3
- Two 16-bit registers are associated with each channel.
- Memory Address Register: used to load starting address of byte to be copied
- Count Register: used to load the count of number of bytes to be copied.
- The address of these register are determined by four address lines A_3 to A_0

A_3	A_2	A_1	A_0	Channel	Register
0	0	0	0	CH0	Memory Address Register
0	0	0	1	CH0	Count Register
0	0	1	0	CH1	Memory Address Register
0	0	1	1	CH1	Count Register

and so on.

DMA Signals

- DMA signals can be divided into two groups:
  i) Signals required to interface with peripheral devices
  ii) Signals required to interface with MPU

i) Signals required to interface with peripheral devices:

DREQ0-DREQ3 (DMA Request)

- These are four independent, asynchronous input signals to the DMA channels from peripheral devices.
- To obtain DMA service, a request is generated by activating the DREQ line of the channel.

DACK0-DACK3 (DMA Acknowledgement)

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

ii) Signals required to interface with peripheral devices:

AEN (Address Enable) and ADSTB (Address Strobe)

- These are active high output signals that are used to latch a high order address byte from data lines to generate 16-bit address.

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

- These are output signals used during the DMA cycle to write and read from memory.

A_3-A_0 Address Lines

- These are bidirectional address lines.
- They are used to access various internal registers.

HRQ (Hold Request) and HLDA (Hold Acknowledgement)

- HRQ is an output signal used to request MPU to receive control of the system bus.
- After receiving the HRQ, the MPU completes the bus cycle and issues the HLDA signal.

DMA Transfer

The sequence of DMA transfer is as follows:

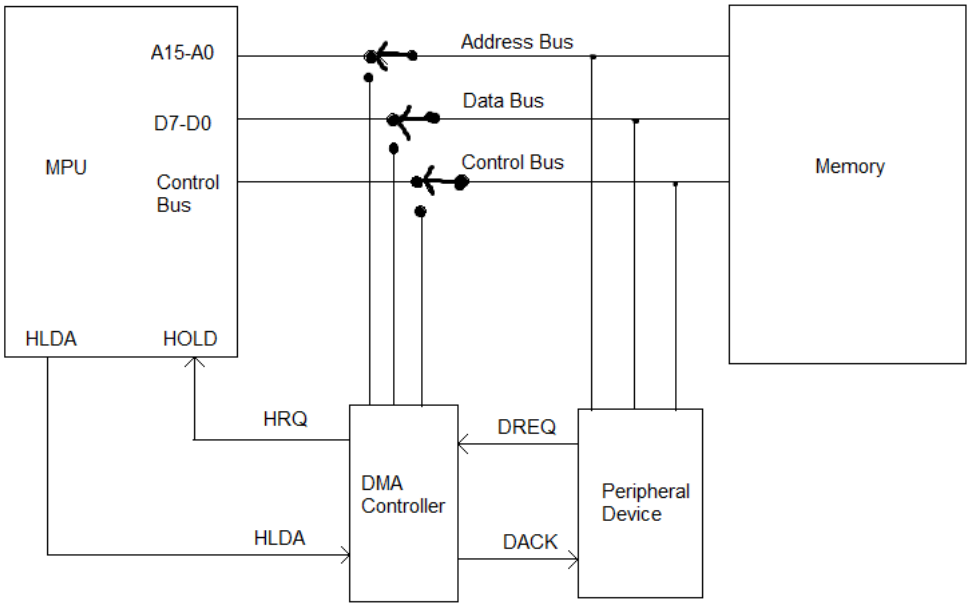


Figure: DMA transfer

1. The I/O devices request the DMA controller to perform DMA transfer through DREQ (DMA request) line.
2. The DMA controller in turn sends a request signal (HRQ) to MPU through the HOLD line.
3. The MPU finishes the current machine cycle and releases the system bus. It also acknowledges the receiving of HOLD signal through HLDA line.
4. The DMA controller acquires control of the system bus. The DMA controller sends the DACK signal to the peripheral I/O and the DMA transfer begins.
5. At the end of the transfer, the system bus is released by the DMA controller. The MPU takes control of the system bus and continues the operation.

DMA Operation

The 8237 is designated to operate in two major cycles. These are called Ideal and Active cycle.

A) Idle Cycle:

- When no channel is requesting service, the 8237 will enter the idle cycle. In this cycle the 8237 will sample DREQ lines every clock cycle to determine if any channel is requesting DMA service. The device will also sample $\overline{CS}$, looking for an attempt by MPU to write or read the internal register of 8237.
- When $\overline{CS}$ is low and HRQ is low, the 8237 enters the program condition.
- Special software commands can be executed by 8237 in program condition.

B) Active Cycle:

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

i) Single Transfer Mode
ii) Block transfer Mode
iii) Demand Transfer Mode
iv) Cascade Mode

Single Transfer Mode

- In this mode, the device is programmed to make one transfer only.
- The word count will be decremented and the address decremented or incremented following each transfer.

Block Transfer Mode

- In this mode the device is activated by DREQ to continue making transfer during the service until Terminal Count (TC), caused by word count going to zero or an external End of Process ($\overline{EOP}$) is encountered.

Demand Transfer Mode

- In demand transfer mode the device is programmed to continue making transfer until a TC or external $\overline{EOP}$ is encountered or until DREQ line goes inactive.
- Thus transfer may continue until the I/O device has exhausted its data capacity. After the I/O device has had a chance to catch up, the DMA service is re-established by means of a DREQ.

Cascade Mode

- This mode is used to cascade more than one 8237 together for simple system expansion.
- The HRQ and HLDA signals from the additional 8237 are connected to the DREQ and DACK signals of a channel of the initial 8237.

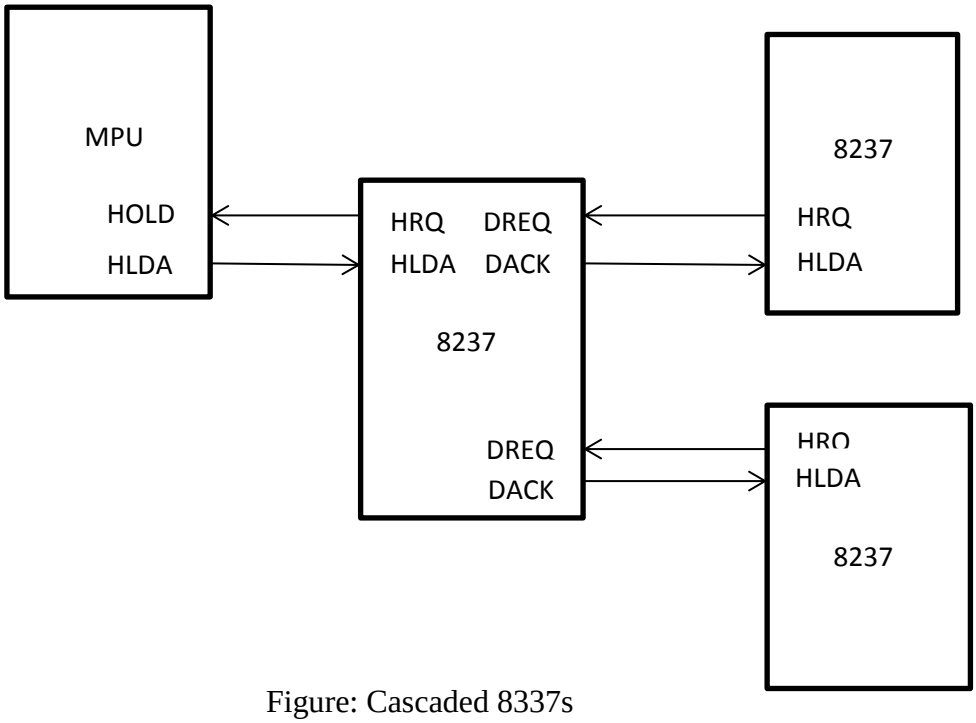


Figure: Cascaded 8337s

8251A UNIVERSAL SYNCHRONOUS ASYNCHRONOUS RECEIVER TRANSMITTER (USART)

The 8251A is a USART (Universal Synchronous Asynchronous Receiver Transmitter) for serial data communication. As a peripheral device of a microcomputer system, the 8251A receives parallel data from the CPU and transmits serial data after conversion. This device also receives serial data from the outside and transmits parallel data to the CPU after conversion.

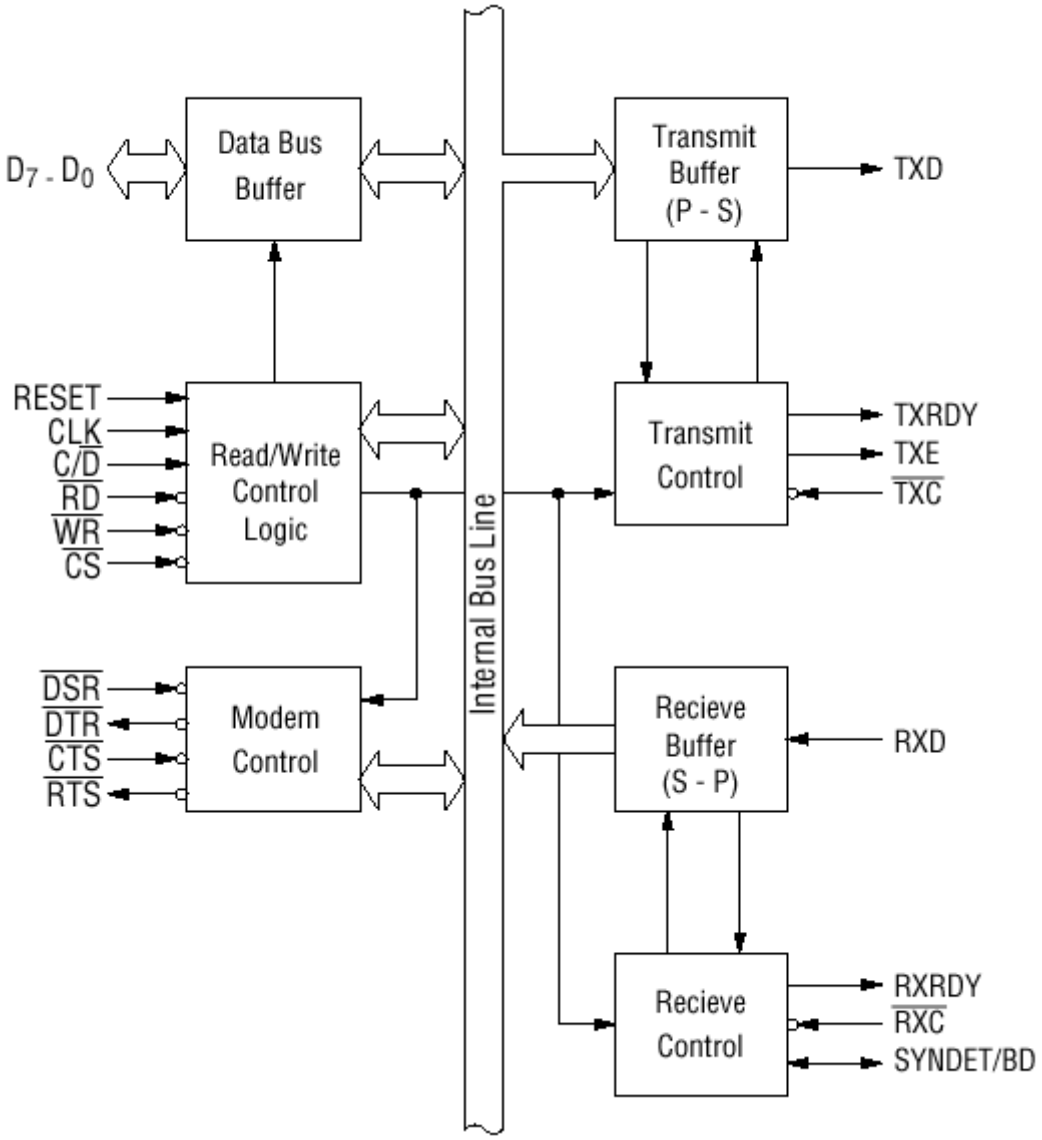


Figure: Block diagram of the 8251A USART

Data Bus Buffer
This 3-state bidirectional, 8-bit buffer is used to interface 8251A to the system data bus. Date is transmitted or received by the buffer upon execution of Input/Output instruction of CPU.

Read/Write Control Logic
This block contains following signals-

- **RESET**
  A "High" on this input forces the 8251A into "reset status." The device waits for the writing of "mode instruction."
- **CLK(Clock)**
  CLK signal is used to generate internal device timing.
- **WR(Write)**
  This is the "active low" input terminal which receives a signal for writing transmit data and control words from the CPU into the 8251A.
- **RD(Read)**
  This is the "active low" input terminal which receives a signal for reading receive data and status words from the 8251A.
- **C/D (Command/Data)**
  This is an input terminal which receives a signal for selecting data or command words and status words when the 8251A is accessed by the CPU. If C/D = 0, data will be accessed. If C/D = 1, command word or status word will be accessed.
- **CS (Chip Select)**
  This is the "active low" input terminal which selects the 8251A at low level when the CPU accesses.

Modem Control
The 8251A has the set of control inputs and outputs that can be used to simply interface to almost any modem. This block has following signals-

- **DSR (Data Set Ready)**
  This is an input port for MODEM interface. The input status of the terminal can be recognized by the CPU reading status words.
- **DTR (Data Terminal Ready)**
  This is an output port for MODEM interface. It is possible to set the status of DTR by a command.
- **CTS (Clear to Send)**
  This is an input terminal for MODEM interface which is used for controlling a transmit circuit. The terminal controls data transmission if the device is set in "TX Enable" status by a command. Data is transmittable if the terminal is at low level.
- **RTS (Request to Send)**
  This is an output port for MODEM interface. It is possible to set the status RTS by a command.

Transmit Buffer

The Transmit Buffer accepts parallel data from the Data Bus Buffer, converts it to serial bit stream, inserts the appropriate character or bits and outputs a serial stream of data on the TXD output pin.

Transmit Control
The Transmit Control manages all the activities associated with the transmission of serial data. It has following signals-

- **TXRDY (Transmitter Ready)**
  This is an output terminal which indicates that the transmitter is ready to accept a data character.
- **TXE(Transmitter Empty)**
  This is an output terminal which indicates that the 8251A has transmitted all the characters and had no data character.
- **TXC (Transmitter Clock)**
  The Transmitter Clock controls the rate at which the character is to be transmitted.

Receive Buffer

The Receive Buffer accepts serial data, converts this serial data to parallel format and sends to the CPU. Serial Data is input to RXD pin.

Receive Control
This functional block manages all receiver-related activities and consists of following signals-

- **RXRDY (Receiver Ready)**
  This is a terminal which indicates that the 8251A contains a character that is ready to ready by CPU.
- **RXC (Receiver Clock)**
  This is a clock input signal which determines the transfer speed of received data.
- **SYNDET/BD (SYNC detect/ Break Detect)**
  This pin is used in Synchronous Mode for SYNDET and may be used as either input or output and is programmable through the Control Word. In Asynchronous mode it can be used for BD which goes HIGH whenever the receiver remains low through two consecutive stop bits.

CONTROL WORDS
There are two types of control word.
1. Mode instruction
2. Command

1) Mode Instruction
The bit configuration of mode instruction is shown in Figures 2 and 3. In the case of synchronous mode, it is necessary to write one-or two byte sync characters. If sync characters were written, a function will be set because the writing of sync characters constitutes part of mode instruction.

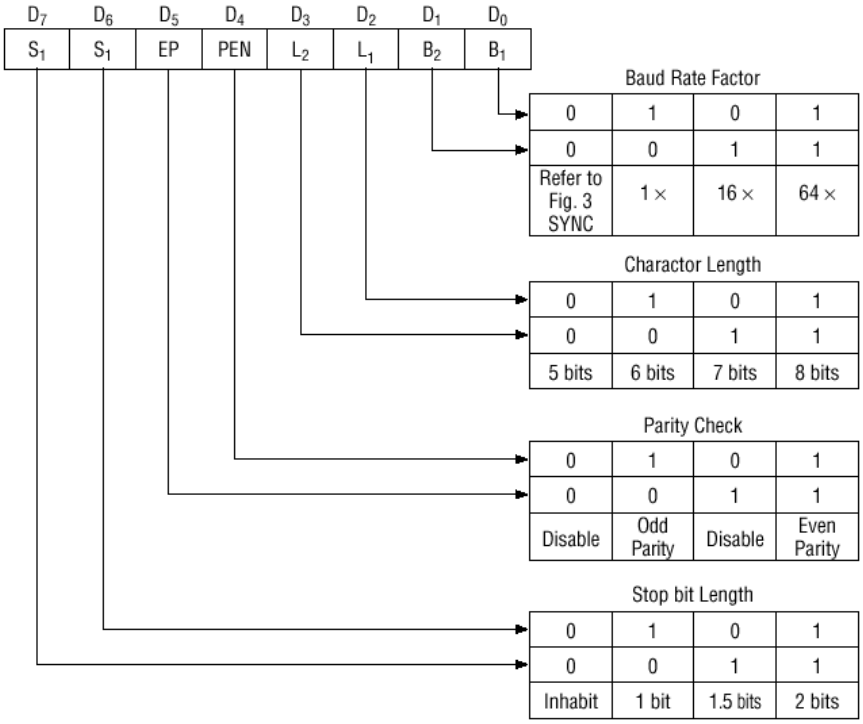


Fig. 2 Bit Configuration of Mode Instruction (Asynchronous)

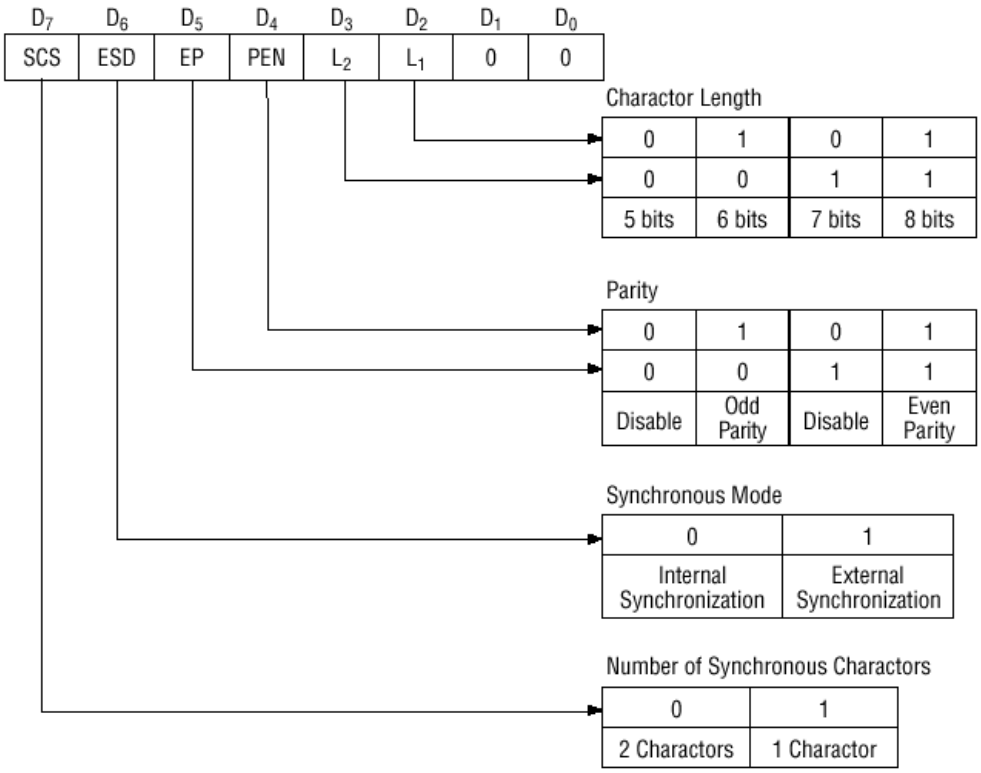


Fig. 3 Bit Configuration of Mode Instruction (Synchronous)

2) Command

Command is used for setting the operation of the 8251A. It is possible to write a command whenever necessary after writing a mode instruction and sync characters.

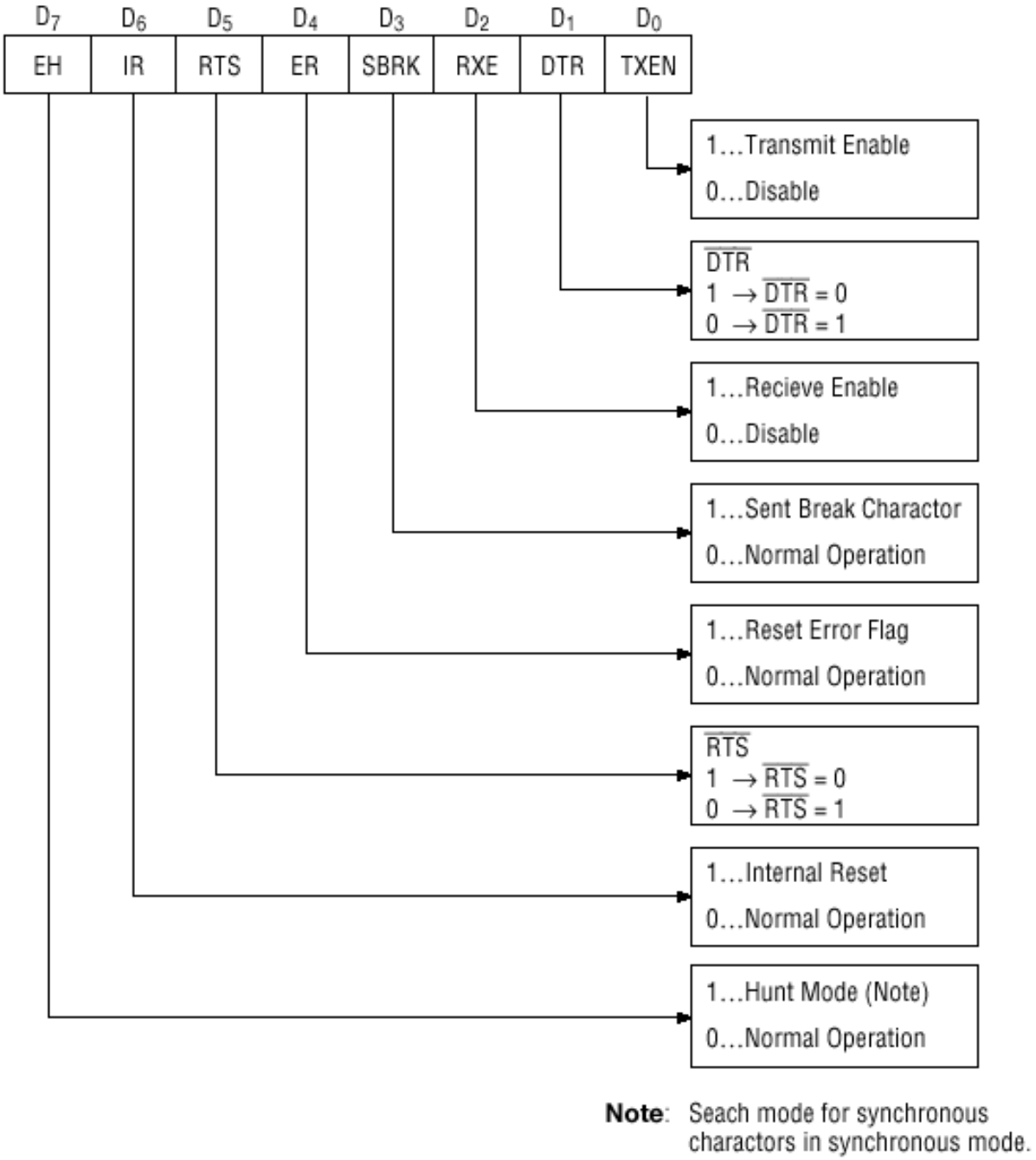


Fig. 4 Bit Configuration of Command

STATUS WORD

It is possible to see the internal status of the 8251A by reading a status word. The bit configuration of status word is shown in Fig. 5.

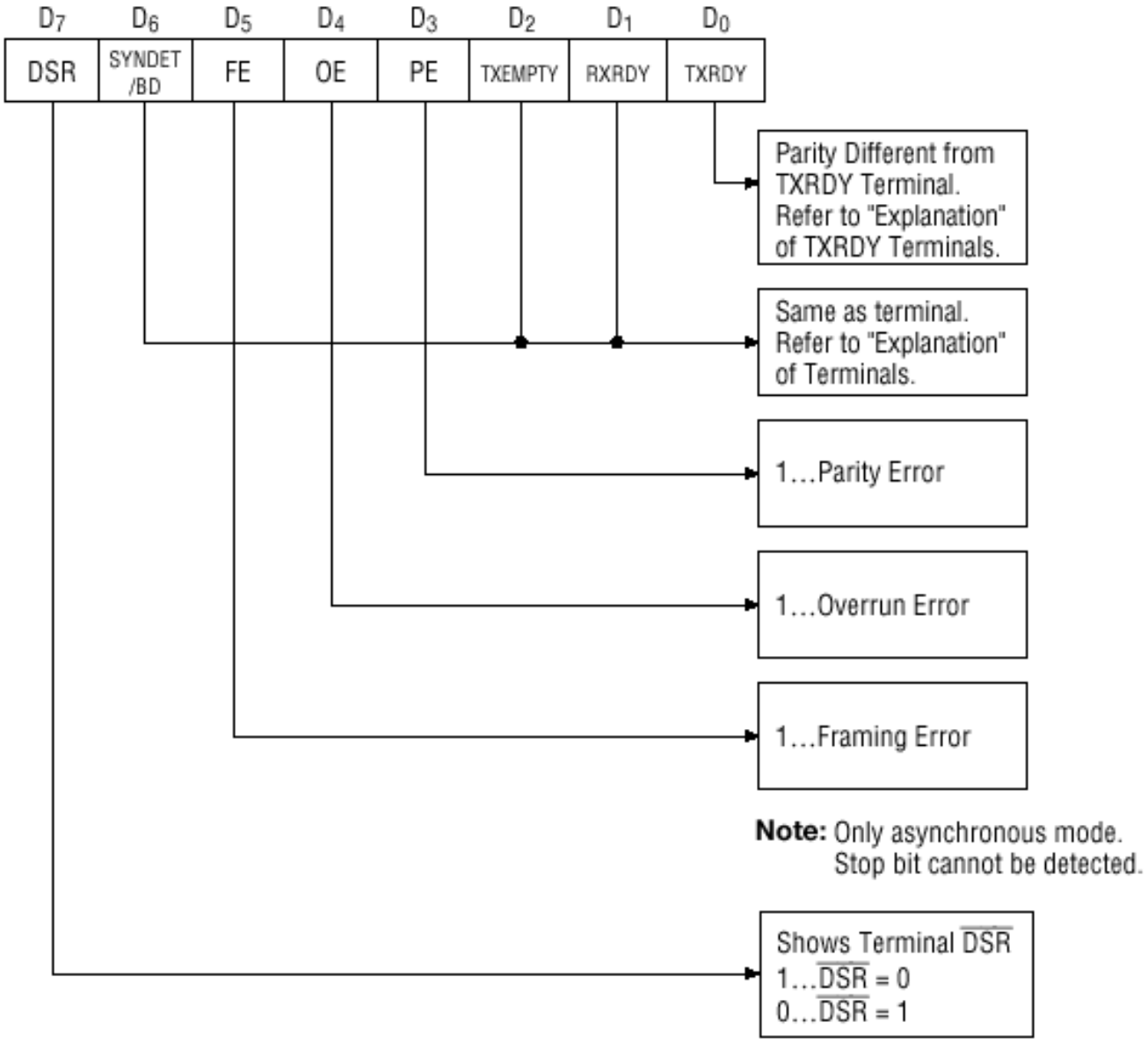


Fig. 5 Bit Configuration of Status Word

--------Program to add numbers input from keyboard------

```
.model small
.stack
.data
.code
    main proc

        .startup

        mov ah,01h
        int 21h
        sub al,30h
        mov bl,al              ;store 1st number in bl

        mov ah,01
        int 21h
        sub al,30h

        add al,bl
        push al
        jnc SKIP

        mov dl, '1'
        mov ah,02h
        int 21h

  SKIP: pop al
        add al,30h
        mov dl,al
        mov ah,02h
        int 21h

        .exit
     main endp
    end main
```

----------Program to concatenate two Strings------------

```
.model small
.stack
.data
    string1 db 'Microprocessor$'
    string2 db 'Assembly Language$'
    string3 db  ?
```

```
.code
    main proc

        .startup

        mov di,offset string3
        mov si,offset string1
        mov cx,14
AGAIN1:
       mov bx,[si]
       mov [di],bx
       inc si
       inc di
       loop AGAIN1

       mov si,offset string2
       mov cx,18
AGAIN2:
       mov bx,[si]
       mov [di],bx
       inc si
       inc di
       loop AGAIN2

        .exit
     main endp
    end main
```

----------Program to check password. If Password is wrong print 'Invalid User' and if right print 'Welcome'---------

```
.model small
.stack
.data
    password db 'secret$'
    msg1 db 'Enter the Password:$'
    msg2 db 'Welcome$'
    msg3 db 'Invalid User$'
    val  db  ?
.code
    main proc
        .startup
```

```
        mov dx,offset msg1
        mov ah,09h
        int 21h

        mov bh,0          ; to count no. of character matched
        mov cx,5          ; length of password
        mov si,offset password
 AGAIN: mov ah,01h
        int 21h
        mov val,al
        mov bl,[si]
        cmp val,bl
        jne SKIP          ; jump if not equal
        inc bh
   SKIP:inc si
        loop AGAIN


       cmp bh,5
       jne INVALID

       mov dx,offset msg2
       mov ah,09h
       int 21h
       jmp EXIT

 INVALID:
       mov dx,offset msg3
       mov ah,09h
       int 21h

   EXIT:
        .exit
     main endp
    end main
```

---------Program to Print two strings in different lines------

```
.model small
.stack
.data
    name1 db 'pokhara university',13,10,'$'
    name2 db 'Kaski Nepal',13,10,'$'
```

```
.code
    main proc
        .startup

        mov dx,offset name1
        mov ah,09h
        int 21h

        mov dx,offset name2
        mov ah,09h
        int 21h

        .exit
     main endp
    end main
```

--------Program to change LOWERCASE string into UPPERCASE-----

```
.model small
.stack
.data
    name1 db 'pokhara university$'
    .code
    main proc

        .startup

        mov dx,offset name1
        mov ah,09h
        int 21h

        mov cx,18
        mov si, offset name1
UPPERCASE:
        sub [si],20h        ;note: to change UPPERCASE into lowercase add 20h
        mov dl,[si]
        mov ah,02h
        int 21h
        inc si
        loop UPPERCASE

        .exit
     main endp
    end main
```

---Program to ADD two 8-bit BCD numbers and display result----

```
.model small
.stack
.data
    num1 db 32
    num1 db  51
.code
    main proc

        .startup
         mov al, num1
         mov ah, num2
         add al, ah
         aam
         add ax, 3030h

         mov bl,al

         mov dl,ah
         mov ah,02h
         int 21h

         mov dl,bl
         mov ah,02h
         int 21h

        .exit
     main endp
    end main
```

----------Program to input a string from keyboard and print it in reverse order----------

```
    .model small
    .stack
    .data
          string db ?
          len db 10           ; length of string to be input
    .code
     main proc
          .startup
```

```
          mov cx, len
          mov si, offset string

AGAIN:
          mov ah, 01h
          int 21h
          mov [si], al
          inc si
          loop AGAIN

          mov cx, len
REPEAT:
          mov dl, [si]
          mov ah, 02h
          int 21h
          dec si
          loop REPEAT

          .exit
     main endp
end main
```

------Program to convert hexadecimal (or binary) to ASCII-----
------ number must be less than 100----------

```
.model small
.stack
.data
    num db 32h
.code
    main proc

        .startup
         mov al, num
         mov ah, 0
         aam
         add ax, 3030h

         mov bl,al

         mov dl,ah
         mov ah,02h
         int 21h
```

```
        mov dl,bl
        mov ah,02h
        int 21h

       .exit
     main endp
    end main
```

------Program to print number of vowels in given string-------

```
.model small
.stack
.data
    string db 'microprocessor$'
.code
    main proc
        .startup

        mov bh,0          ; to count no. of vowels
        mov cx,14         ; length of string
        mov si,offset string
 AGAIN:
        mov al,[si]

        cmp al, 'a'
        je COUNT          ; jump if equal

        cmp al, 'e'
        je COUNT          ; jump if equal

        cmp al, 'i'
        je COUNT          ; jump if equal

        cmp al, 'o'
        je COUNT          ; jump if equal

        cmp al, 'u'
        jne SKIP          ; jump if not equal
COUNT:  inc bl
SKIP:   inc si
        loop AGAIN
```

```
       add bl,30h   ;convert count to ASCII to print
       mov dl,bl
       mov ah,02h
       int 21h

        .exit
     main endp
    end main
```

------Program to convert ASCII to Hexadecimal (or Binary)-----
------Result stored in AX---------

```
.model small
.stack
.data
     string db 'Enter two digit Number:$'
    num db ?
.code
    main proc

        .startup
         mov cx, 10                ;multiplier of 10
         mov dx, offset string
         mov ah, 09h
         int 21h

         mov ah, 01h
         int 21h
         sub al, 30h
         mov ah, 0
         mul cx     ;multiplies ax by cx
         push ax    ;save result
         mov ah, 01h
         int 21h
         sub al, 30h
         mov bl, al
         mov bh, 0
         pop ax
         add ax, bx
        .exit
     main endp
    end main
```