

# **19MZC12&19RAC12 / MICROPROCESSOR AND APPLICATIONS**

**Prepared by**

Mr.C.RAMKUMAR,

Assistant Professor,

Department of Electrical and Electronics Engineering,

Muthayammal Engineering College (Autonomous),

Rasipuram – 637 408.

# Course Objective

- 1.To study the basic concept of 8085 microprocessor
- 2.To study the basic concept of 8051 microcontroller
- 3.To introduce the PIC microcontroller and programming of 8085 and 8051
- 4.To study the basic concept of interfacing
5. To study the application of Processors and controllers

# DIFFERENCE BETWEEN MICROPROCESSOR AND MICROCONTROLLER

MICROPROCESSOR	MICROCONTROLLER
A microprocessor is a general purpose device is called a CPU	A microcontroller is a dedicated chip is called a single chip
Microprocessor based system design is complex	Microcontroller based system design is simple
It is flexible which means we can change the size	It is not flexible
Microprocessor do not have power saving features	Microcontrollers have the power saving mode
Size of RAM / ROM can vary	Size of the RAM / ROM fixed
It does not contain internal memory	It contain internal memory
Cost is high	Cost low

# UNIT 1 – 8085 PROCESSOR

## INTRODUCTION

A Microprocessor is known as a Central Processing Unit.

It is fabricated on a single chip

The first microprocessor was Intel 4004 introduced in 1971. It is a 4 bit processor .

The first microprocessor to make into a home computer was Intel 8080. It is a 8 bit computer on one chip introduced in 1974.

In 1976 updated the intel 8080 design with 8085 by adding two instruction to enable / disable the interrupt and serial port

In 1978 introduced the intel 8086. It is a 16 bit processor.

In 1980 introduced the intel 8087.

# **INTRODUCTION TO MICROCOMPUTER SYSTEM**

Microcomputer system is the interconnection of CPU , memory and input / output device

## **CPU**

Central Processing unit is the brain of the microcomputer system. It coordinate the entire microcomputer operation.

It accept input and output device depending on the instruction permanent programs stored in ROM or temporary program stored in RAM execute the program and finally result will be displayed through output device .

The CPU chip of the microcomputer is know as microprocessor

# **MEMORY**

It is a two types of Memory

RAM – Random Access Memory

ROM – Read Only Memory

RAM is used for temporarily storing the program

ROM is used for permanently storing the program

# **PERIPHERAL DEVICE**

The input and output devices called peripheral device. The input device is used to feed the data and the output device is used to display the result

# **BUSES**

The input output and memory device are connected to CPU by a group of line is called buses.

They can be divided into three types

1.Address bus

2.Data bus

### 3. Control bus

The address bus carries a address of the memory location or I/O devices that the CPU want to access.

Data bus is used to transfer the data between CPU, memory and I/O device.

Control bus is used to control the signal between CPU, memory and I/O device.

## **INTRODUCTION TO MICROPROCESSOR**

The CPU built on a single IC is called microprocessor.

The internal architecture of microprocessor determine how and what operation can be performed with the data. To perform any operation microprocessor requires

1. Arithmetic Logic unit
2. Control Logic Unit
3. Register
4. Instruction register
5. Program counter

## 6.Internal bus

1.ALU is the computational unit of microprocessor which performs arithmetic and logic operation on binary unit

Register array is the internal storage device

The control unit is generating the control signal for internal and external operation of microprocessor

Program counter is used to hold the memory address for the next instruction to be executed

## **ADVANTAGE OF MICROPROCESSOR**

1.Low cost

2.Low size

3.Low power consumption

4.High reliability



# FEATURES OF 8085

8085 is an 8 bit general purpose microprocessor.

It is a 40 pin dual in line package single chip integrated circuit

Only one +5V power supply is needed for operation

It can operate with a 3MHz single phase clock

It provides Serial Input data and Serial Output Data Line for simple serial interface

In 8085 microprocessor has the following register

1. One 8-bit Accumulator (ACC) register A
2. Six 8-bit general purpose register B,C,D,E,H,L
3. One 16 – bit stack pointer SP
4. One 16-bit program counter , PC
5. Instruction register
6. Temporary register

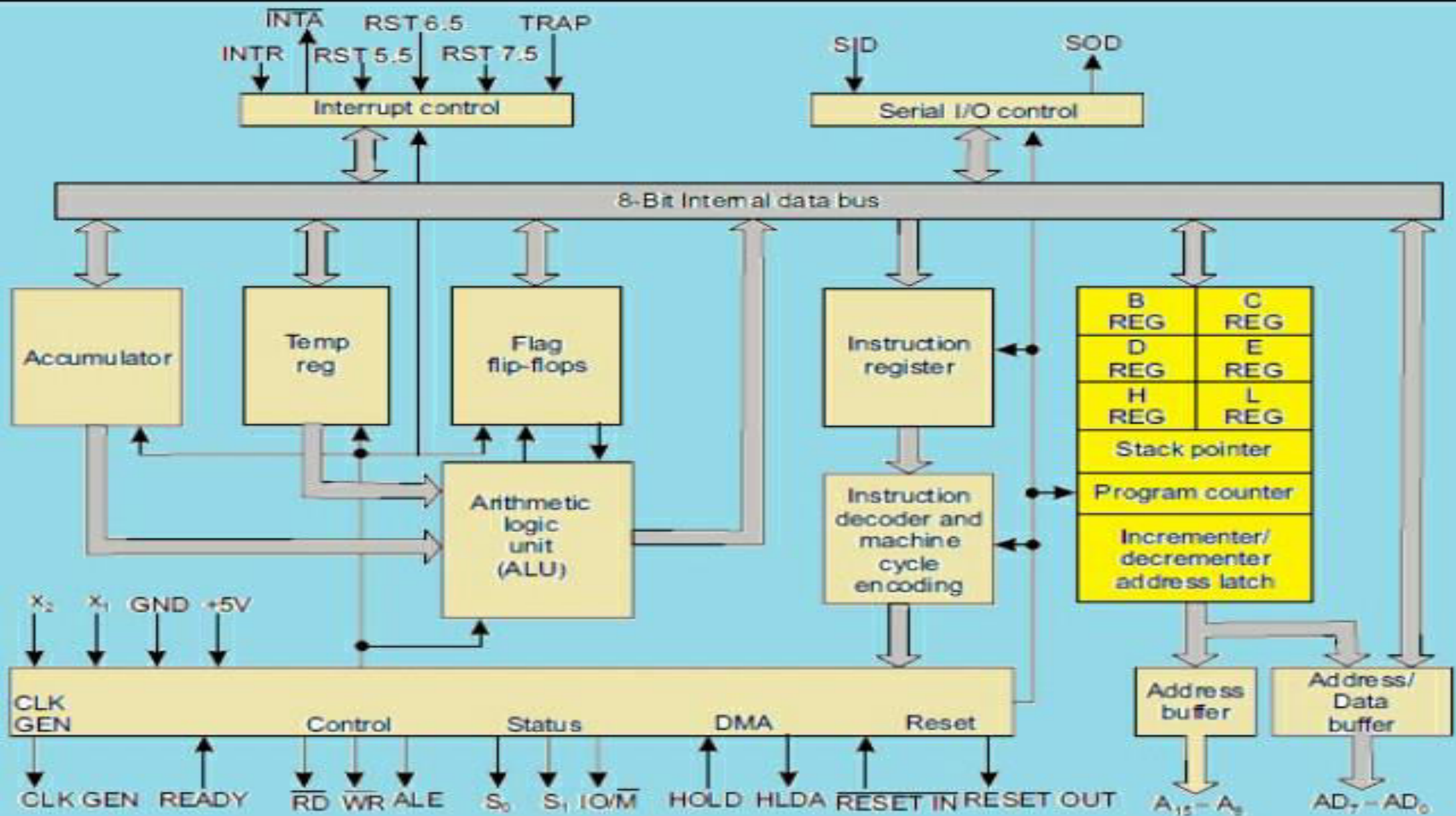
# **ARCHITECTURE OF 8085 MICROPROCESSOR**

The architecture of 8085 has the following components

1. Accumulator
2. General Purpose register
3. Program Counter
4. Stack pointer
5. Arithmetic and Logic Unit
6. Temporary register
7. Flags
8. Instruction register and Decoder
9. Timing and Control unit
10. Interrupt control
11. Serial I/O control

12. Address buffer and Address Data buffer

13. Address bus and data bus



Architecture of 8085

# **ACCUMULATOR**

Accumulator is an 8-bit register.

It hold one of the data to be processed by Arithmetic and Logic unit

It also stores the result of the operation

The accumulator is also called a A-register

It is connected to 8\_bit internal data bus

The bi-directional arrow between the accumulator and the bus.

It allows the accumulator to send and receive the data

# **GENERAL PURPOSE REGISTERS**

The 8085 microprocessor contain six 8 bit general purpose register

They are B,C,D,E,H,L register

A combination of two 8 bit registers is a 16 bit data

The combination og two 8 bit registers is known as register pair

The register pair in 8085 are B-C,D-E and H-L pair

H-L pair is act as a memory pointer

## **PROGRAM COUNTER(PC)**

It is a 16-bit special-purpose register.

It is used to hold the memory address of the next instruction to be executed

The microprocessor increments the content of program counter during the execution of instruction

## **STACK POINTER(SP)**

It is a 16-bit register

It is used as a memory pointer

Stack pointer is decremented each time when data is loaded into the stack and incremented when the data is retrieved from stack

## **ARITHMETIC AND LOGIC UNIT(ALU)**

It is an 8-bit register

ALU carries only the arithmetic and logic operation

Arithmetic operation such as addition, subtraction, multiplication, division

Logic operation such as AND, OR and EXOR.

ALU results are stored back in the accumulator

## **TEMPORARY REGISTER**

It is an 8-bit register

It is associated with ALU

It hold data during an arithmetic and logical operation

## **FLAG**

Flag register is a group of five individual flip-flops

The five status flag of intel 8085 are

1. Carry flag (CY)

2. Parity flag(P)

3. Auxiliary Carry flag(AC)

4. Zero flag(Z)

Sign flag(S)

## **INSTRUCTION REGISTER AND DECODER**

It is a 8-bit register.

The instruction decoder decodes the content of the instruction register

During fetch cycle,opcode of an instruction is stored in the instruction register

# **TIMING AND CONTROL UNIT**

This unit has the following signal

Control signal : READY,RD,WR,ALE

Status signal : S0,S1,IO/N

DMA signal : HOLD

RESET signal : RESET IN RESET OUT

## **INTERRUPT**

It has two types

1.Maskable

2.Non maskable

Maskable contain contain RST 5.5,6.6,7.5

Non maskable contain TRAP,INTR

## **SERIAL I/O CONTROL**

Serial Input Data

Serial Output Data



The Serial input data enter into the instruction then the Serial Output data can be leave

## **ADDRESS BUFFER AND ADDRESS DATA BUFFER**

The Stack pointer and program counter can be loaded into the address buffer and address data buffer

The 8-bit internal data bus is also connected to the address data buffer

The bidirectional arrow indicates the three connection that allow address data buffer t send or receive data from 8-bit internal data

## **ADDRESS BUS AND DATA BUS**

The Intel 8085 requires a 16 bit address.

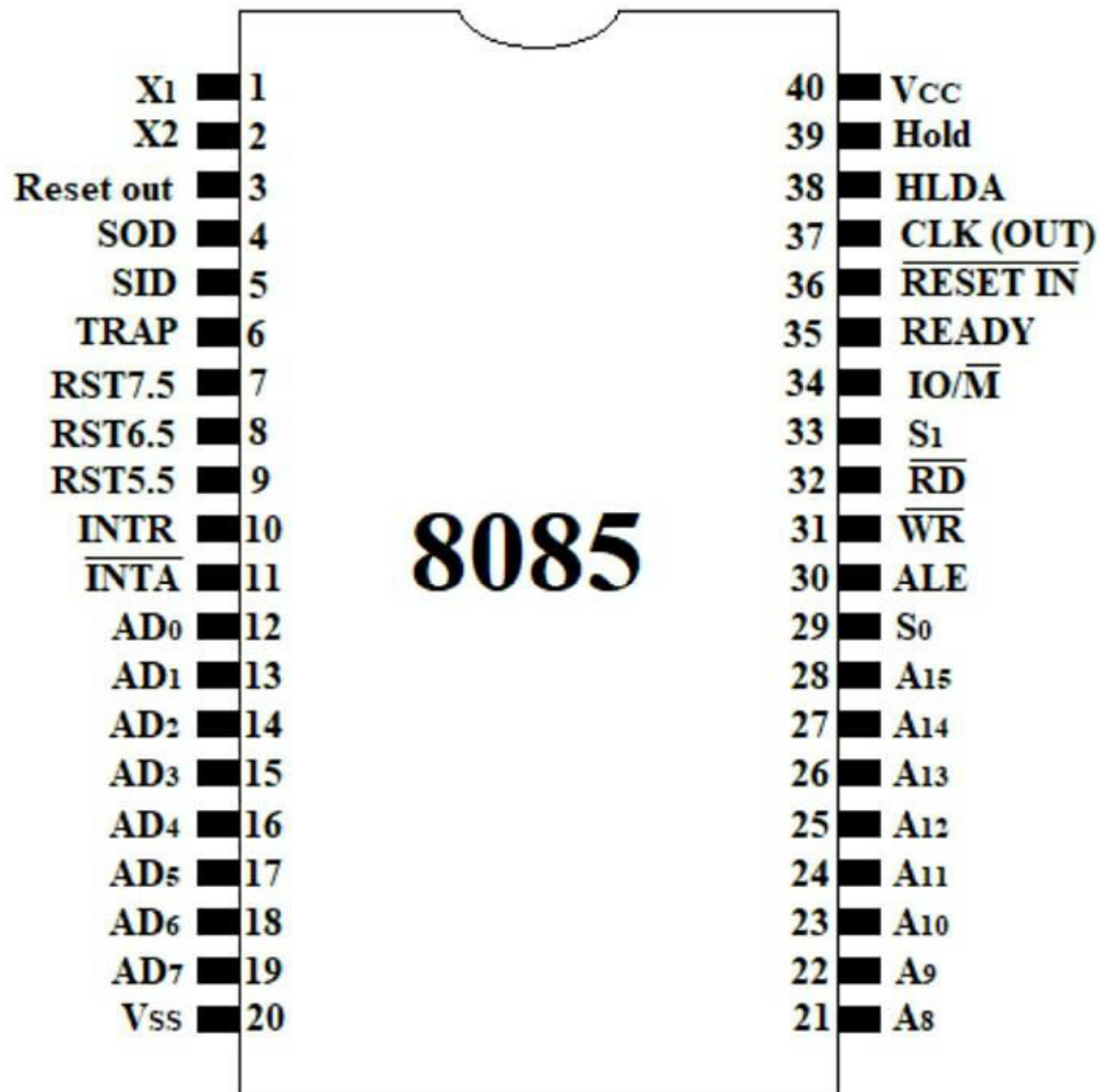
The 8 most significant bit are address are transmitted by the address bus

The 8 least significant bit are address are transmitted by address data bus

# **PIN DIAGRAM OF 8085**

The Pin of 8085 can be classified as given below

- 1.Power supply and frequency signal
- 2.Address signals
- 3.Data signals
- 4.Control and status signal
- 5.Interrupts
- 6.Serial I/O signals
- 7.Acknowledgement signals



## **Address bus**

A8-A15 are address bus and are used for most significant bits of the memory address

## **Address/Data buffer**

AD0-AD7 are the address /data bus. They are used least significant 8 bits of the memory address

## **Address Latch enable (ALE)**

It goes High during first clock of a machine cycle and enables the lower 8-bit of the address to be latched

## **IO/M**

It is a status signal which distinguish whether the address is for memory or I/O.

When it goes high the address on the address bus is for an I/O device.

When it goes low the address on the address bus is for a memory location.

## **WR**

This active LOW signal is used to control WRITE operation.

When it goes low, the data on the data bus is written into the selected memory or I/O location.

## **READY**

It is used by the microprocessor to sense whether a peripheral is ready to transfer data or not.

A peripheral may be connected to the microprocessor through READY line.

If READY is high the peripheral is ready. If it is low the microprocessor waits till it goes high.

## **HOLD**

It indicates that another device is requesting for the use of the address and data bus.

Having received a HOLD request the microprocessor relinquishes the use of the buses as soon as the current machine cycle is completed

## **HLDA**

It is a signal for HOLD acknowledgement.

It indicates that the HOLD request has been received.

After the removal of a HOLD request the HLDA goes low.

## **INTR**

It is an interrupt request signal. Among interrupts it has the lowest priority. When it goes high the program counter does not increment its content.

After completing the instruction at hand it attends the interrupting device.

## **INTA**

It is an interrupt acknowledgement sent by the microprocessor after interrupt (INTR) is received.

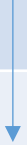
## **RST 5.5, 6.5, 7.5**

RST 7.5, RST 6.5 and RST 5.5 are the restart interrupts. They cause an internal restart to be automatically inserted. Each of them has a programmable mask.

## **TRAP**

TRAP is the highest priority interrupts it is a non mask able interrupts

TRAP	HIGHEST PRIORITY
RST 7.5	
RST 6.5	
RST 5.5	LOWEST PRIORITY



## **RESET IN**

It reset the program counter to zero

It also interrupt enable and HLDA flip-flop

## **RESET OUT**

It indicates that the CPU is being reset.

## **X1, X2**

These are terminals to be connected to an external crystal oscillator which drives internal circuitry of the microprocessor to produce a suitable clock for the operation of microprocessor.

## **CLK**

It is a clock output for user, which can be used for other digital ICs. Its frequency is same at which processor operates



## **SID**

It is data line for serial input. The data of this line is loaded into the 7th bit of the accumulator when RIM instruction is executed.

## **SOD**

It is a data line for serial output. The 7th bit of the accumulator is output on SOD when SIM instruction is executed

## **Vcc**

It is +5 Volts supply pin

## **Vss**

It is a ground pin.

# **INSTRUCTION SET**

## Instruction

An instruction is a binary pattern designed inside a microprocessor to perform a specific

## Function

## Instruction Set

The entire group of instructions, called the instruction set, determines what functions the microprocessor can perform

## 8085 Instruction Set

The 8085 instructions can be divided into five different groups based on the function the instruction carry out. They are

1. Data Transfer Instructions
2. Arithmetic Instructions
3. Logical Instructions
4. Branching Instructions
5. Machine Control Instructions

## **Data Transfer Instructions**

This group of instructions are used for loading data into registers, moving data in between registers and moving data between registers and memory location.

Examples:

MOV B,D

LDA 4000 H

STA B

MOV M,C

XCHG

## **Arithmetic Instructions**

This group of instructions are used for addition, subtraction, increment or decrement or data in registers or in memory.

Example

ADD M

SUB C

INR M

DCX B

DAA

## **Logical Instructions**

This group of instructions execute the logical operations like AND, OR, XOR or comparing data between registers or between register and memory, rotating or complementing data in registers or memories.

Examples

ANA B

ORA M

RAL

RRc

CPI A5 H

## **Branch Instructions**

The group of instructions perform

Subroutine CALL

Conditional and unconditional JUMP Return and

Restart

Examples

JMP 9000H

JNC LOOP 1

CC LOOP 2

RET

RST 0

## **Machine Control Instructions**

This group of instructions are used for transferring data between stack and the registers exchanging the contents of SP and HL with the stack top moving data to and from I/O ports enabling or disabling or masking interrupts

Examples

PUSH D

SPHL

IN 80H

OUT 07H

RIM

## **Data Transfer Instructions**

MOV

Move the content from source to Destination

MOV destination , source

The content of the source register is moved to the destination register . The content of the source register are not altered . If one of the operand is memory location. It is specified by the contents of HL registers

MOV r1,r2  $[r1] \leftarrow [r2]$

MOV r,M  $[r] \leftarrow [[HL]]$

MOV M,r  $[[HL]] \leftarrow [r]$

## Examples

1. MOV B,A Before execution After execution

A B A B

### MVI

Move Immediate 8 bit data

MVI destination, data

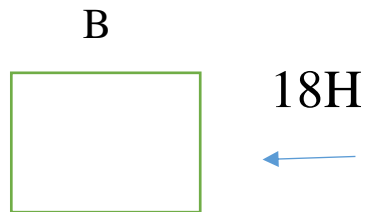
The given data is moved to the destination register or memory.

MVI r,data [r] data

MVI M,data [[HL]] data

Example

MVI B,18 H





# LDA

Load Accumulator Direct

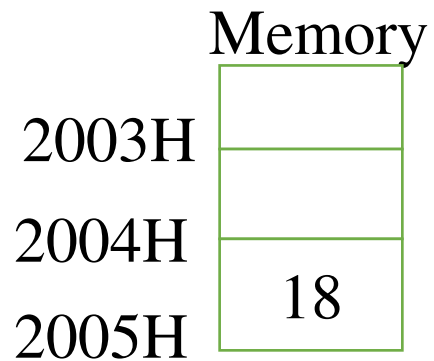
LDA address

The content of the memory location, whose address is specified by the 2nd and 3rd bytes of the instruction is loaded into the accumulator. The contents of the source are not altered.

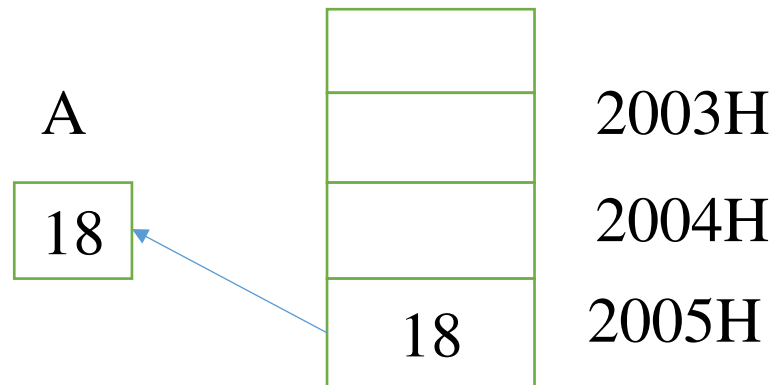
$[A] \leftarrow [\text{address}]$

Examples

LDA 2005 H



Before execution



After execution

# STA

Store Accumulator Direct

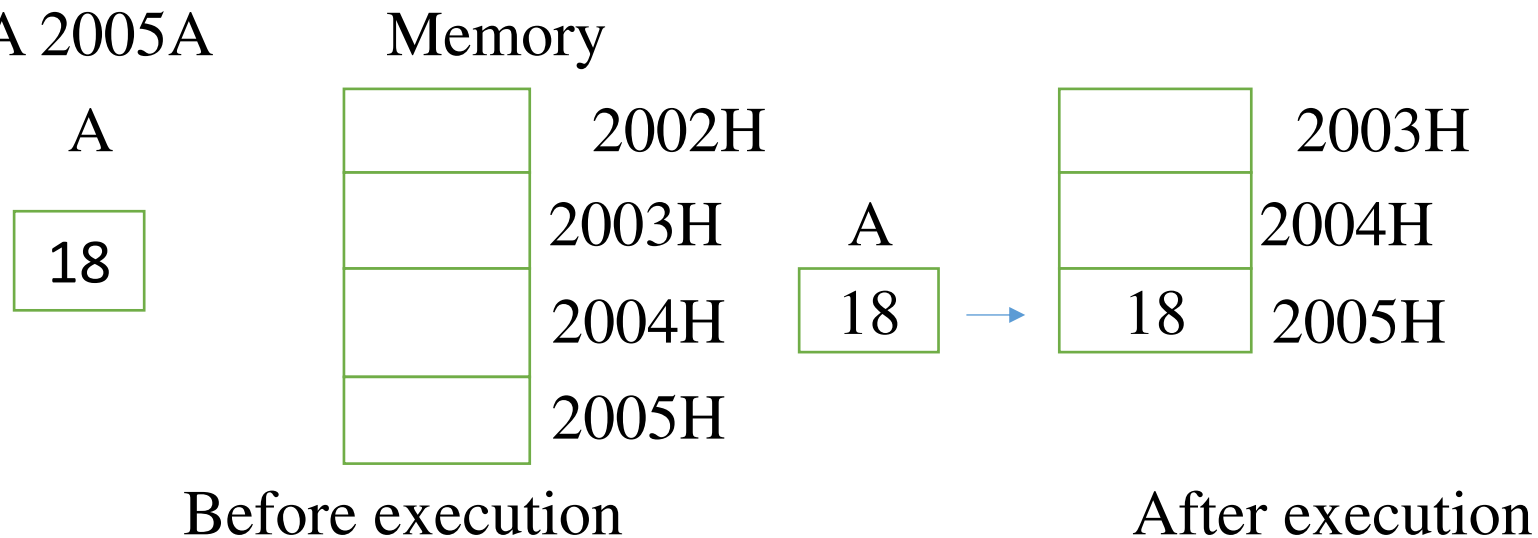
STA address

The contents of the accumulator are stored to a memory location specified by the 2nd and 3rd byte of the instruction.

$[\text{address}] \leftarrow [A]$

Example

STA 2005A



# LDAX

Load Accumulator Indirect

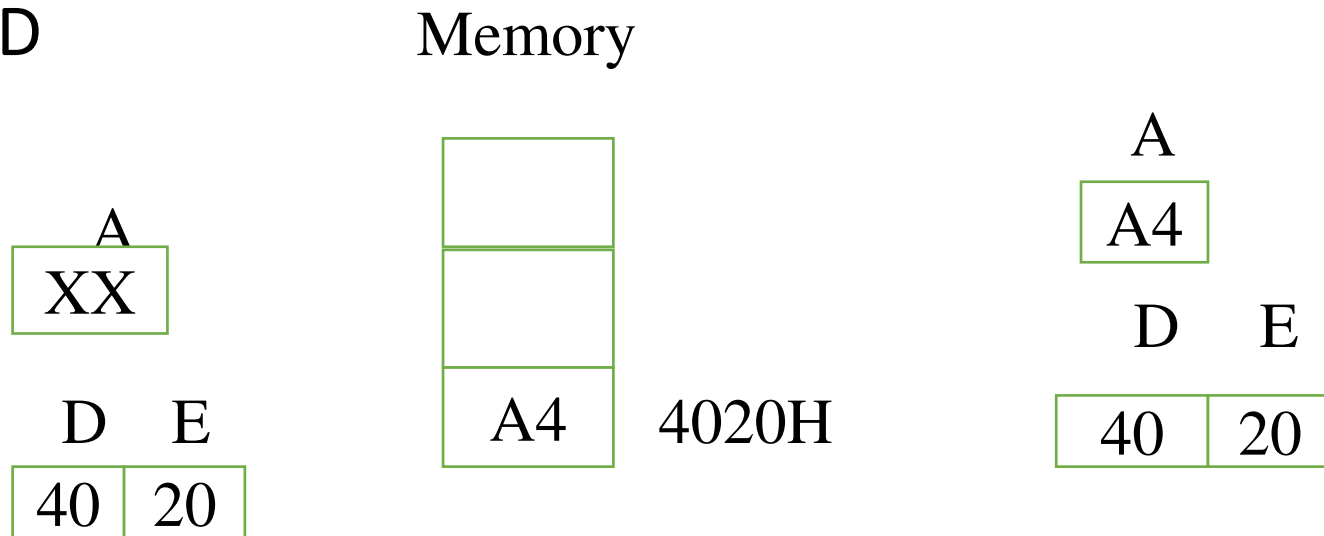
LDAX register pair

The content of the memory location, whose address is in the register pair (B-C or D-E) is loaded into the accumulator.

$[A] \leftarrow [[rp]]$

Example

LDXA D



# STAX

Stored Accumulator Indirect

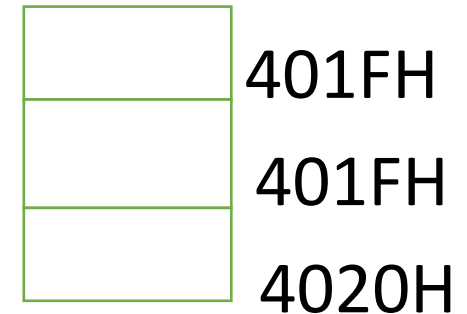
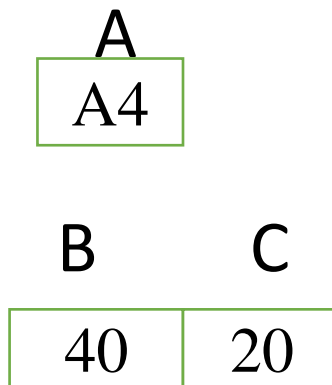
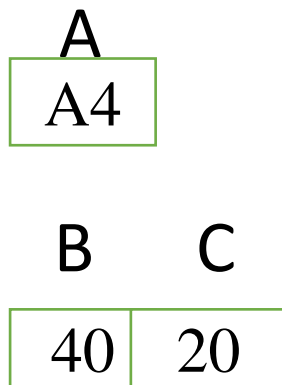
STAX register pair

The content of accumulator is stored in the memory location whose address is in the register pair(B-C,D-E)

[[rp]]-[A]

Example

STAX B



# LHLD

Load H and L registers Direct

LHLD address

The content of the memory location, whose address is specified in the instruction, is loaded into register L. The content of next memory location is loaded into register H

$[L] \leftarrow [\text{address}]$

$[H] \leftarrow [\text{address}+1]$

Example

LHLD 2005H

Before execution

2005	18
2006	F3
2007	

After execution

H	L
F3	18

## SHLD

Store H and L registers Direct

## SHL

The content of register L is stored in the memory location whose address is specified in the instruction. The content of register H is stored in the next memory location.

$[\text{Address}] \leftarrow [L]$

$[\text{address}+1] \leftarrow [H]$

Example

SHLD 2005 H

Before execution

H	L
F0	B3

After execution

Memory	
B3	2005 H
F0	2006 H

# LXI

Load Register Pair Immediate

**LXI rp**, 16 bit data

This instruction loads 16 bit immediate data into the register pair (B-C, D-E, H-L). This is a 3 byte instruction . The second byte specifies the low-order byte and the third byte specifies the high-order byte.

[rh] ← 8 MSBs of data

[rl] ← 8 LSBs of data

Example

LXIB,1122H

After execution

B      C

11	12
----	----

# XCHG

Exchange H and L with D and E

The contents of register H are exchanged with the contents of register D and the contents of register L are exchanged with the contents of register E.

$[H-L] \leftrightarrow [D-E]$

Example

## XCHG

Before execution

H	L
10	12

D	E
30	40

After execution

H	L
30	40

D	E
10	12



# Arithmetic Instruction

## ADD

Add register or Memory or Accumulator

ADD register

The content of the register or memory is added to the content of the accumulator and the result is stored in the accumulator

$$[A] \leftarrow [A] + [r]$$

Example

ADD B

A    A    B

$$33 \leftarrow 11 + 22$$

# ADC

Add Register to Accumulator with Carry

ADC register

The content of the register or memory and the carry flag are added to the content of the accumulator and the result is stored in the accumulator

$$[A] \leftarrow [A] + [r] + [CY]$$

Example

ADC B

$$\begin{array}{r} A \\ 21 \leftarrow 10 + 10 + 01 \end{array}$$

# ADI

Add Immediate data to accumulator

ADI 8 bit data

The immediate 8 bit data is added to the content of the accumulator and the result is stored in the accumulator

$[A] \leftarrow [A] + \text{data}$

Example

ADI B2H

A    A

C4  $\leftarrow$  12 + B2

# ACI

Add Immediate data to accumulator with carry

ACI 8 bit data

The 8 bit data and the carry flag are added to the content of the accumulator and the result is stored in the accumulator

$$[A] \leftarrow [A] + [\text{data}] + [CY]$$

Example

ACI 15H

A      A              CY

38 ← 23 + 15 + 00

## **DAD**

Add register pair to H-L register

DAD register pair

The 16 bit contents of the specified register pair are added with the content of the H-L register pair and the result is stored in the H-L register

$$[H-L] \leftarrow [H-L] + [rp]$$

Example

**DAD B**

H L      H L    B    C

33 33 ← 11 11 + 22 22

## **SUB B**

Subtract register/memory from Accumulator

Sub register/memory

The content of register or memory location is subtracted from the content of the accumulator

and the result is stored in the accumulator

$$[A] \leftarrow [A] - [r]$$

Example

SUB B

A    A    B

$$10 \leftarrow 20 - 10$$

**SBB**

Subtract register/memory and borrow from accumulator

SBB register/memory

The content of register or the memory and the borrow flag are subtracted from the content of the accumulator and the result is stored in the accumulator

$$[A] \leftarrow [A] - [r] - [CY]$$

Example

SBB B

$[A] \leftarrow [A] - [r] - [CY]$

A    A    B    CY

$IF \leftarrow 40 - 20 - 01$

**SUI**

Subtract Immediate Data from accumulator

SUI 8-bit data

The 8-bit data is subtracted from the content of accumulator and the result is stored in the accumulator

$[A] \leftarrow [A] - \text{data}$

Example

SUI 13 H

A    A

$05 \leftarrow 18 - 3$

## **SBI**

Subtract Immediate data with Borrow from Accumulator

The 8 bit data and the content of borrow flag (carry flag) are subtracted from the content of the accumulator and the result is stored in the accumulator.

$$[A] \leftarrow [A] - \text{data} - [CY]$$

Example :

SBI 13H

A    A        CY

04 ← 18 - 13 - 01

## **INR**

Increment contents of register/memory by 1

INR register /memory

The content of register or memory location pointed by H-L pair is incremented by 1 and the result is stored in same place



$[r] \leftarrow [r] + 1$

$[[HL]] \leftarrow [[HL]] + 1$

Example

INR B

B    B

$IC \leftarrow IB + 1$

**DCR**

Decrement register/ memory content by 1

Decrement register/memory

The content of the register is decremented by 1 and the result is stored in the same place

$[r] \leftarrow [r] - 1$

$[HL] \leftarrow [HL] - 1$

Example

DCR C

C C

$A2 \leftarrow A3 - 1$

**INX**

Increment register pair by 1

INX rp

The content of register is incremented by 1

$[rp] \leftarrow [rp] + 1$

Example

INX B

B C B C

A1 04  $\leftarrow$  A1 03 +1

# DCX

Decrement register pair by 1

DCX rp

The content of register is decremented by 1

$[rp] \leftarrow [rp] - 1$

Example

DCX D

D	E		D	E
FF	FE	←	FF	FF -1

## Logical instruction

ANA

Logical AND with accumulator

AND register/memory

The contents of the accumulator are logically anded with the contents of the register or memory and the results is stored in the accumulator.

$$[A] \leftarrow [A] \wedge [r]$$
$$[A] \leftarrow [A] \wedge [[HL]]$$

Example

AND B

$$[A] \leftarrow 0110\ 0011$$
$$[B] \leftarrow 0100\ 1010$$

Result 0100 0010 stored in A

## **ANI**

AND immediate data with Accumulator

ANI 8 bit data

The contents of the accumulator are logically anded with the 8-bit data and the results is stored in the accumulator

$[A] \leftarrow [A] \wedge \text{data}$

Example

ANI 15 H

$[A] \leftarrow 0110\ 0011$

Data  $\leftarrow 0001\ 0101$

Result  $\leftarrow 0000\ 0001$  stored in A

# ORA

Logically OR with Accumulator

ORA register/memory

The contents of the accumulator are logically ORed with the contents of the register or memory and the results is stored in the accumulator.

$$[A] \leftarrow [A] \vee [r]$$
$$[A] \leftarrow [A] \vee [[HL]]$$

Example ORA C

$$[A] = 1100\ 0011$$
$$[C] = 0101\ 0101$$

Result = 1101 0111 stored in accumulator

## **ORI**

Logically OR Immediate data

ORI 8 bit data

The contents of the accumulator are logically ORed with the 8-bit data and the results is stored in the accumulator

$[A] \rightarrow [A] \vee \text{data}$

Example ORI 71 H

$[A] = 0110\ 1110$

Data = 0111 0001

Result = 0111 1111

# XRA

Exclusive OR with Accumulator

XRA register/memory

The contents of the accumulator are logically ORed with the contents of the register or memory and the results is stored in the accumulator.

$[A] \leftarrow [A] \vee [r]$

$[A] \leftarrow [A] \vee [[HL]]$

Example XRA E

$[A] = 1110 \ 1110$

$[E] = 0101 \ 1010$

Result=1011 0100 stored in accumulator



## **XRI**

Exclusive OR immediate with accumulator

XRI 8 bit data

The exclusive ORed 8-bit data with the content of accumulator and the results is stored in the accumulator

$[A] \leftarrow [A] \vee \text{data}$

Example XRI 18 H

$[A] = 1100\ 0011$

Data = 0001 1000

Result = 1101 1011

## **CMA**

Complement Accumulator

The content of accumulator are complemented(1's complement)

$[A] \leftarrow \overline{[A]}$

Example

**CMA**

Before execution

10011001

After execution

01100110

**CMC**

Complement carry

The carry flag is complemented

$[CY] \leftarrow \overline{[CY]}$

Example CMC

Before execution

CY

1

After execution

CY

0

# CMP

Compare with Accumulator

CMP reg/memory

The content of the register or memory is subtracted from the content of the accumulator and the status flag are set according to the result of the subtraction .But the result is not stored .

The content of the accumulator remains unchanged.

If  $[A] < [r]$  ;  $CY = 1$ ,  $Z = 0$

If  $[A] = [r]$ ;  $CY = 0$ ,  $Z = 1$

Example CMP B

Before execution

After execution

A B

A B

45 75

45 75

FLG  $CY = 1$   $Z = 0$

## **CPI**

Compare Immediate data with Accumulator

CPI 8 bit data

The 8 bit data is compared with the contents of the accumulator . The status flag are set according to the result of subtraction . But the result is discarded

[A]- data

Example

CPI 45 H

## **STC**

Set Carry

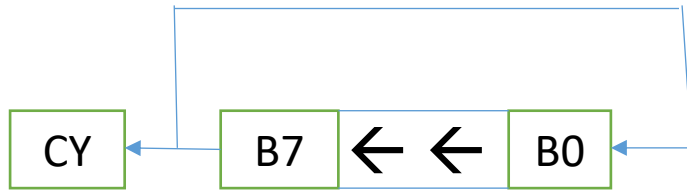
The carry flag is set to 1

[CY]←1

# RLC

## Rotate Accumulator Left

The content of the accumulator is rotated left by one bit. The seventh bit of the accumulator is moved to carry bit as well as to the zero bit of the accumulator



Example

CY    A

Before execution : 0    10001011

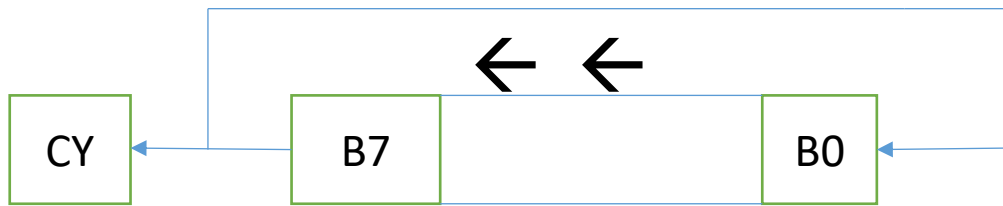
CY    A

After execution    : 1    00010111

# RRC

## Rotate Accumulator Right

The content of the accumulator is rotated right by one bit. The zero bit of the accumulator (B0) is moved to the seventh bit (B7) as well as to carry bit.



Example

CY    A

Before execution :    10100111

CY    A

After execution    :    11010011

# RAL

Rotate Accumulator Left through carry

The content of the accumulator is rotated left one bit through carry. The seventh bit of the accumulator (B7) is moved to carry and the carry bit is moved to the zero bit (B0) of the Accumulator



Example

CY    A

Before execution :    11100001

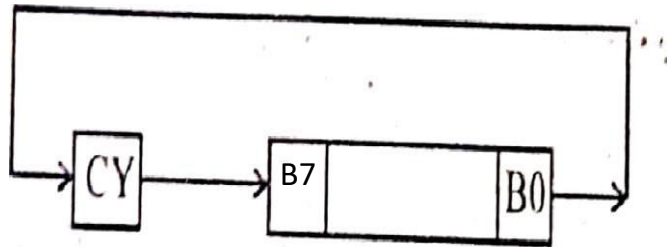
CY    A

After execution :    11000010

# RAR

Rotate Accumulator Right through carry

The content of the accumulator is rotated right one bit through carry. The zero bit of the accumulator (B0) is moved to carry and the carry bit is moved to the seventh bit (B7) of the Accumulator



Example

CY    A

Before execution : 0 00011100

CY    A

After execution : 0 00001110



## **Branch Instruction**

### **JMP**

#### **(i)Jump Unconditionally**

JMP 16-bit address

2<sup>nd</sup> byte and 3<sup>rd</sup> byte of the instruction give the address of the label where the program jumps. The address of the label is the address of the memory location of next instruction to be executed. The program jumps to the instruction specified by the address unconditionally

$[PC] \leftarrow \text{Label}$

Example

JMP 4000H

#### **(i)Jump conditionally**

After the execution of the condition jump instruction the program jumps to the instruction specified by the address(Label) if the specified condition is fulfilled. Otherwise the program proceeds further in the normal sequence

Instruction	Operation
JC	Jump on carry
JNC	Jump on No carry
JP	Jump on positive
JM	Jump on Minus
JPE	Jump on Parity Even
JPO	Jump on Parity Odd
JZ	Jump on zero
JNZ	Jump on No zero

# CALL

## i) Unconditional Subroutine Call

CALL 16-bit address

CALL instruction is used to call a subroutine. Before the control is transferred to the subroutine the address of the next instruction of the main program is saved in the stack. The content of SP is decremented by 2 to indicate the new stack top.

$$[[SP]-1] \leftarrow PCH$$
$$[[SP]-2] \leftarrow PCL$$
$$[SP] \leftarrow [SP]-2$$
$$[PC] \leftarrow \text{Label (address)}$$

Example

CALL 5000H

## ii) Conditional Subroutine call

Instruction	Operation
CC	Call on carry
CNC	Call on No carry
CP	Call on Positive
CM	Call on Minus
CPE	Call on Parity even
CPO	Call on Parity Odd
CZ	Call on zero
CNZ	Call on non zero

## **RET**

i) Return from subroutine Unconditionally

RET is used at the end of subroutine. Before the execution of subroutine the address of the next instruction of the main program saved the main program is saved in the stack. The content of SP is incremented by 2 to indicate the new stack top.

$$PCL \leftarrow SP$$
$$PCH \leftarrow [[SP]+1]$$
$$[SP] \leftarrow [SP]+2$$

## ii) Conditional return

Instruction	Operation
RC	Return on carry
RNC	Return on No carry
RP	Return on Positive
RM	Return on Minus
RPE	Return on Parity even
RPO	Return on Parity Odd
RZ	Return on zero
RNZ	Return on non zero

## RST

### Restart

Restart is a one-word CALL instruction. The content of PC is saved in the stack. The Program jumps to the instruction starting at restart location

Instruction	Restart Locations (Hex)
R8T0	0000
RST1	0008
RST2	0010
RST3	0018
RST4	0020
RST5	0028
RST6	0030
RST7	0038

## **PCHL**

Load PC with HL contents

The contents of registers H and L are copied into the program counter. The contents of H are placed as a high order byte and the contents of L are placed as a low order byte.

$[PCH] \leftarrow [H]$

$[PCL] \leftarrow [L]$

## **Stack, I/O and Machine Control Instructions**

### **IN**

Input Data to Accumulator from a Port

IN 8-bit Port address

The contents of the input port specified in, the operand are read and loaded into the accumulator.

$[A] \leftarrow [\text{Port}]$

Example IN 80 H



## **OUT**

Output Data from Accumulator to a Port

OUT 8-bit Port address

The contents of the accumulator are copied into the output port specified by the operand.

$[\text{Port}] \leftarrow [A]$

Example :OUT 30 H

## **PUSH**

Push Register pair onto stack

PUSH rp

The contents of the register pair are pushed into the stack. The SP is decremented and the contents of the high order register (B, D, H, A) are copied into that location. The SP is decremented again and the contents of the low order register (C, E, L) are copied to that location.

[[SP]—1] ← [rh]

[[SP]—2] ← [rl]

[SP] ← [SP]—2

Example PUSH B

Before execution

B C

23 45

SP

2099 2097

After execution

B C

23 45

SP

Stack

	2097
45	2098
23	2099
XX	

**POP**

POP OF Stack to Register Pair

The contents of the memory location pointed out by the SP are copied to low-order register (C, E, L) of the operand. The SP is incremented by 1 and the contents of that memory location are copied to high-order register (A, B, D, H) of the operand. The SP is again incremented by 1.

$[rL] \leftarrow [SP]$

$[rH] \leftarrow [[SP]+1]$

$[SP] \leftarrow [SP]+2$

Example POP H

Before execution

H L

XX XX

SP

2090

**HLT**

Halt

After execution

H L

18 5A

SP

2092

Stack

	2090H
5A	2091H
18	2092H

The execution of the instruction HLT stops the microprocessor. The MPU finishes executing the current instruction and halts any further execution. The registers and Status flags remain unaffected.

# XTHL

Exchange H and L with Stack TOP

The content of register L are exchanged with the byte of the stack top. The content of register H are exchanged with the byte below the Stack top

$[L] \leftarrow [SP]$

$[H] \leftarrow [SP]+1$

Before execution

H	L
11	22

Stack

44	3000
33	3001

After execution

H	L
33	44

Stack

22	3000
11	3001

## **SPHL**

Copy H and L to the SP

The contents of HL pair are transferred to the stack pointer register.

$[SP] \leftarrow [HL]$

## **EI**

Enable Interrupts

When this instruction is executed, the interrupts are enabled.

## **DI**

Disable Interrupts

When this instruction is executed, the interrupts are disabled.

## **NOP**

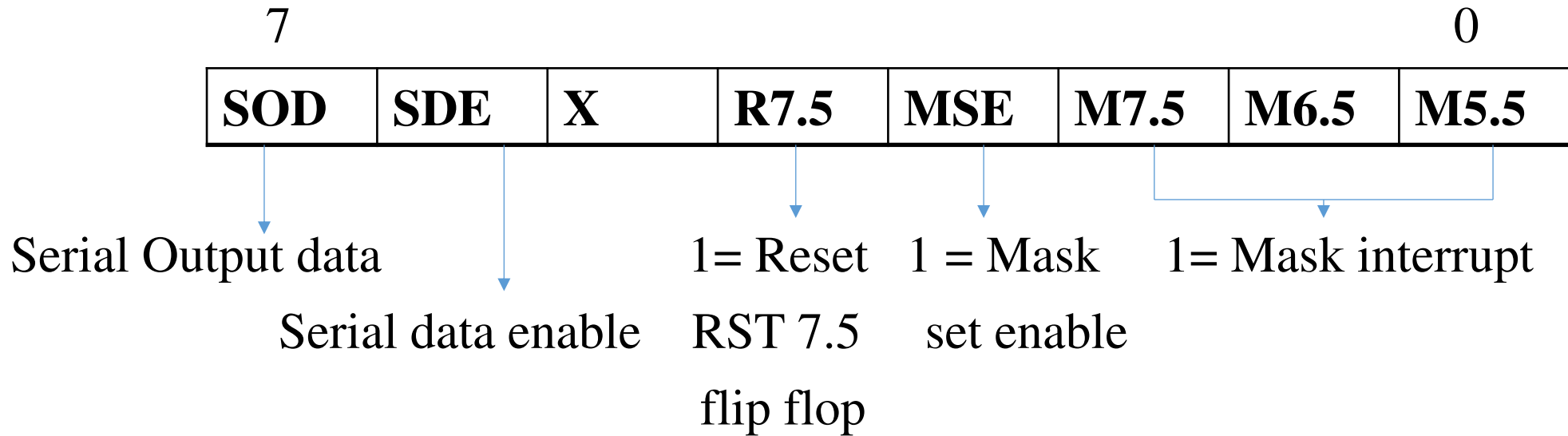
NO operation

When this instruction is executed, no operation is performed. The register and flags remain unaffected. The instruction is used to fill in time delays or to delete or insert instruction with trouble shooting

# SIM

## Set Interrupt Mask

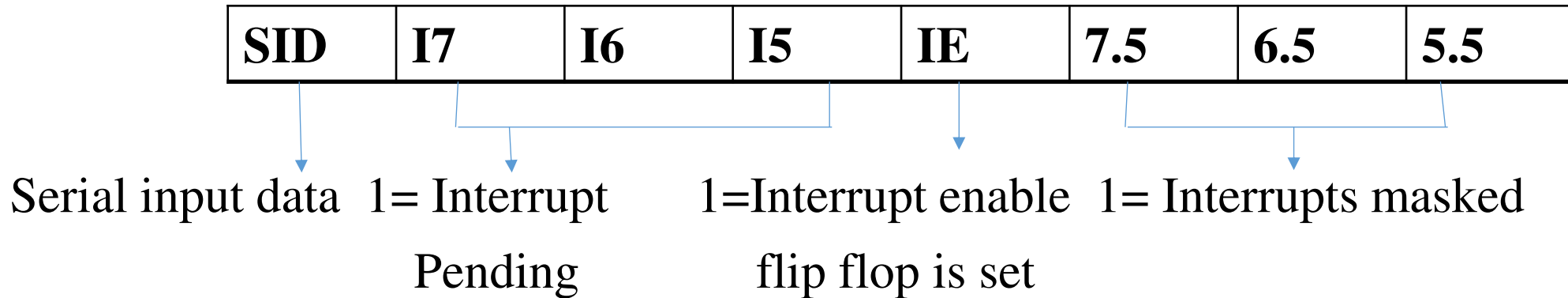
SIM is a multipurpose instruction and used to implement the 8083 interrupts (RST 7.3,6.5 and S.5) 2nd serial data output. This instruction interprets the accumulator contents as follows:



# RIM

## Read Interrupt Mask

RIM is a multipurpose instruction used to read the status of interrupts (RST 7.5, 6.5 and 5.5) and to read serial data input bit. The instruction interprets the accumulator contents, as follows:



# ADDRESSING MODES

Every instruction of a program has to operate on a data. The method of specifying the data to be operated by the instruction is called Addressing.

Addressing mode is the way the microprocessor identifies the operands for the instruction.

The 8085 has the following addressing modes.

- Immediate addressing mode
- Direct addressing mode
- Register addressing mode
- Register indirect addressing mode
- Implicit addressing mode

## **Immediate Addressing Mode**

In immediate addressing mode, the data (Operand) is specified within the instruction itself.

Examples :

MVI A, 18 H



ADI 09 H

LXI H, 50AB H

MVI A, 18 M

Accumulator

18 →

Move the data 18H to the Accumulator

### **Direct Addressing Mode**

In direct addressing mode, the address of the data (operand) is specified in the instruction itself.

Examples :

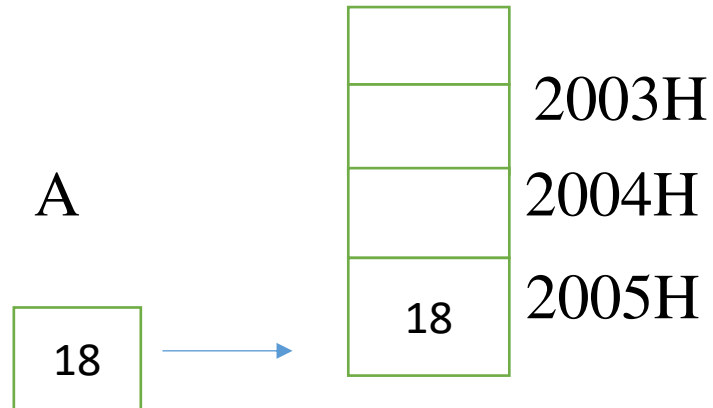
STA 2005 H

OUT 05 H

LDA 4100 H

STA 2005 H

Memory Location



Store the content of the accumulator in the memory location 2005 H. 2005 H is it memory address where the data is to be stored. It is given in the instruction itself.

### **Register Addressing Mode**

In register addressing mode, the instruction specifies the name of the register in which the data is available.

The opcode specifies the address of the register in addition to the, operation to be performed.

Examples

MOV A, B

ANA B

SUB H

MOV A,B

A

B



Move the content of register B to A

## Register Indirect Addressing Mode

In register addressing mode, the instruction specifies the name of the register in which the data is available.

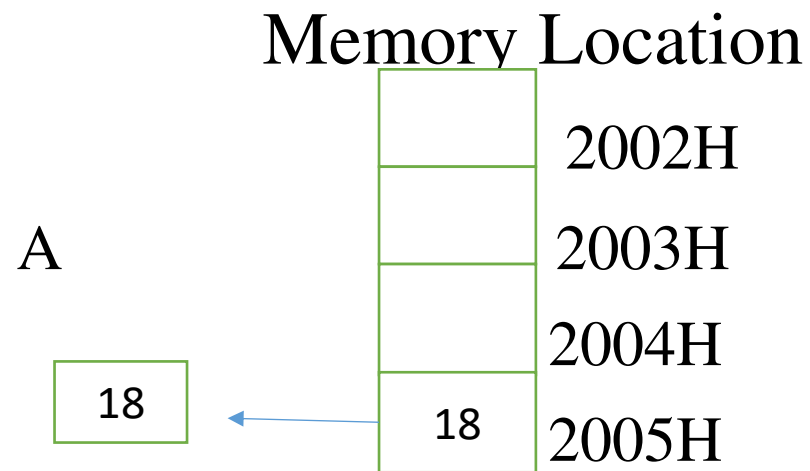
Examples

MOV A, M

SUB M

DCR M

MOV A,M



## **Implicit addressing Mode**

Some instruction operate on the content of the accumulator. Such instruction do not requires the address of operand.

Example

CMA

RAL

RAR

## 8085 Addressing mode

Type	Instruction
Direct	STA 2005H
Register	MOV A,B
Register Indirect	MOV A,M
Immediate	MVIA,18H
Implicit	CMA

## **Timing diagram**

The timing diagram provides the information about the various condition of the signal in which the machine cycle is executed

The timing diagram matches the peripheral devices like memories, ports etc.. Can be selected to form a system with microprocessor as CPU

## **Instruction cycle**

It is defined as the time required to complete the execution of an instruction.

The 8085 instruction cycle consist of one to six machine cycles.

## **Machine cycle**

It is defined as the time required to complete one operation of accessing memory , I/O or acknowledgement an external request . This cycle may consist of 3 to 6 T-states

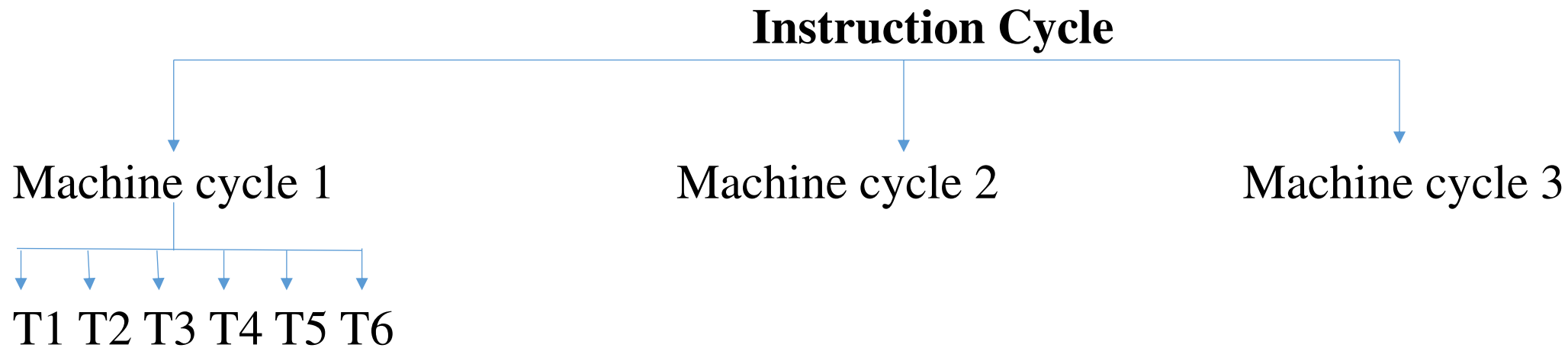
## **T-state**

It is defined as one subdivision of the operation performed in one clock period

These subdivision are internal state synchronized with system clock.

One T – state – One clock period

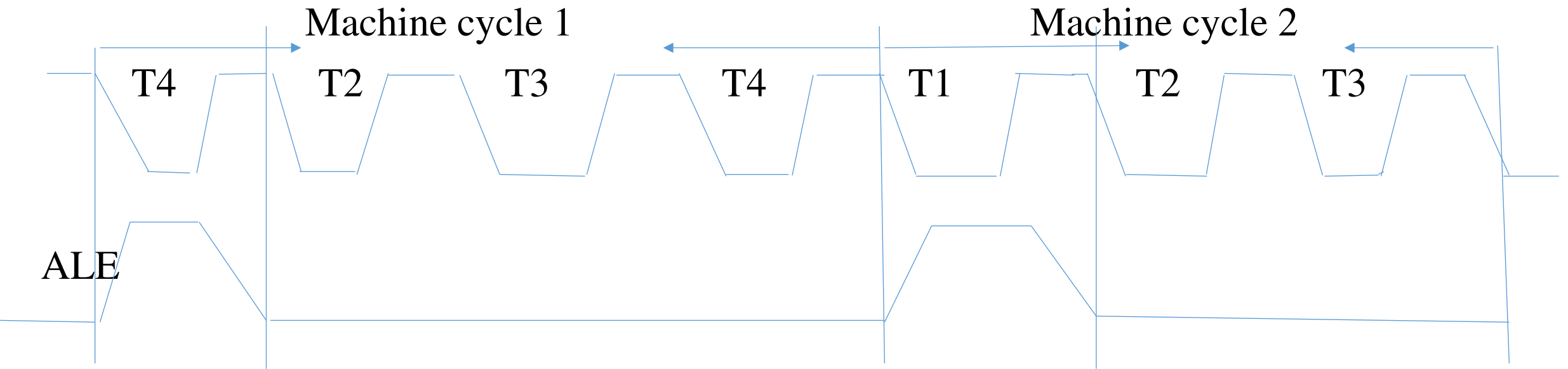
Each instruction of the 8085 microprocessor can be divided into a few basic operation called machine cycle and each machine cycle can be divided into T-states



### **ALE(Address Latch Enable)**

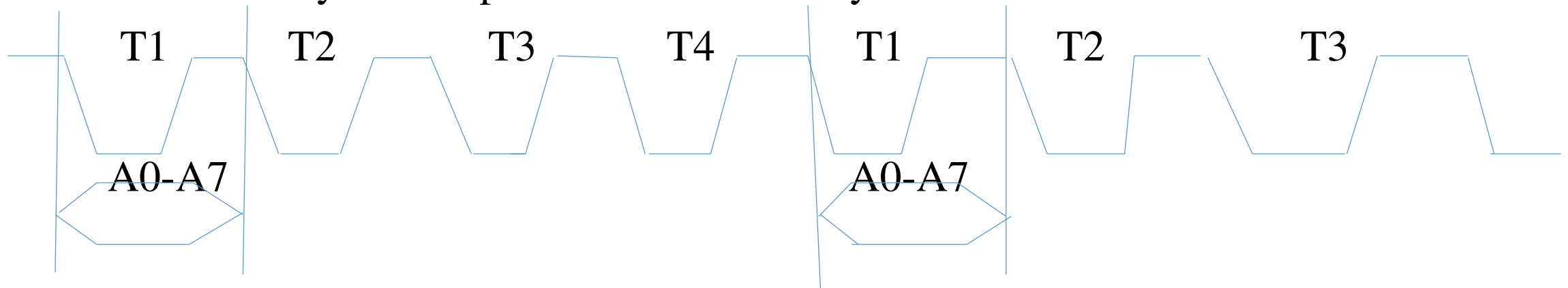
This signal is high active signal. It is activated in the beginning of the T1 state of each machine cycle , except bus idle machine cycle , and it remains active in the T1 state





### **A0-A7(Lower byte address)**

The lower byte of address is available on the multiplexed addressed / data bus during T1 state of each machine cycle except bus idle machine cycle

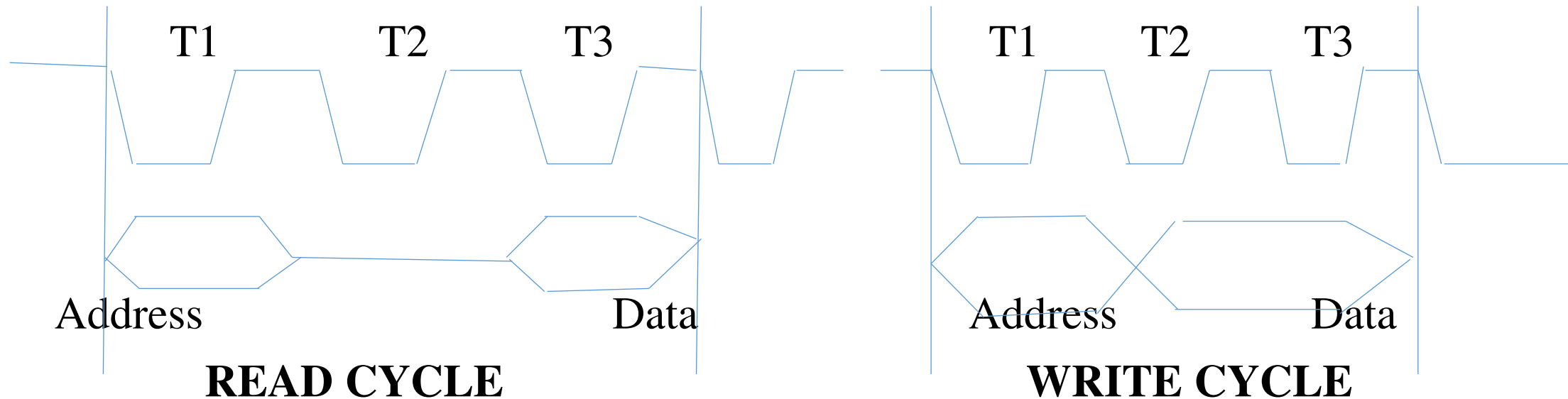


## D4-D7(Data bus)

The data from memory or I/O device and from microprocessor to memory or I/O device is transferred during T2 and T3 states.

It is important to note that in read machine cycle, data will appear on the data bus during the later part of the T2-state whereas in write cycle data will appear on the data bus at the beginning of the T2-state

To read data from memory or I/O device it is necessary to select memory or I/O device



## **A8-A15**

The higher byte of address is available on A8-A15 bus during T1,T2 and T3 states of each machine cycle , except bus idle machine cycle

### **During T2 state**

At the beginning of T2 states the microprocessor sends the control signal RD goes low to enable memory

The data which is opcode D0-D7 of the selected memory location are placed on A/D bus

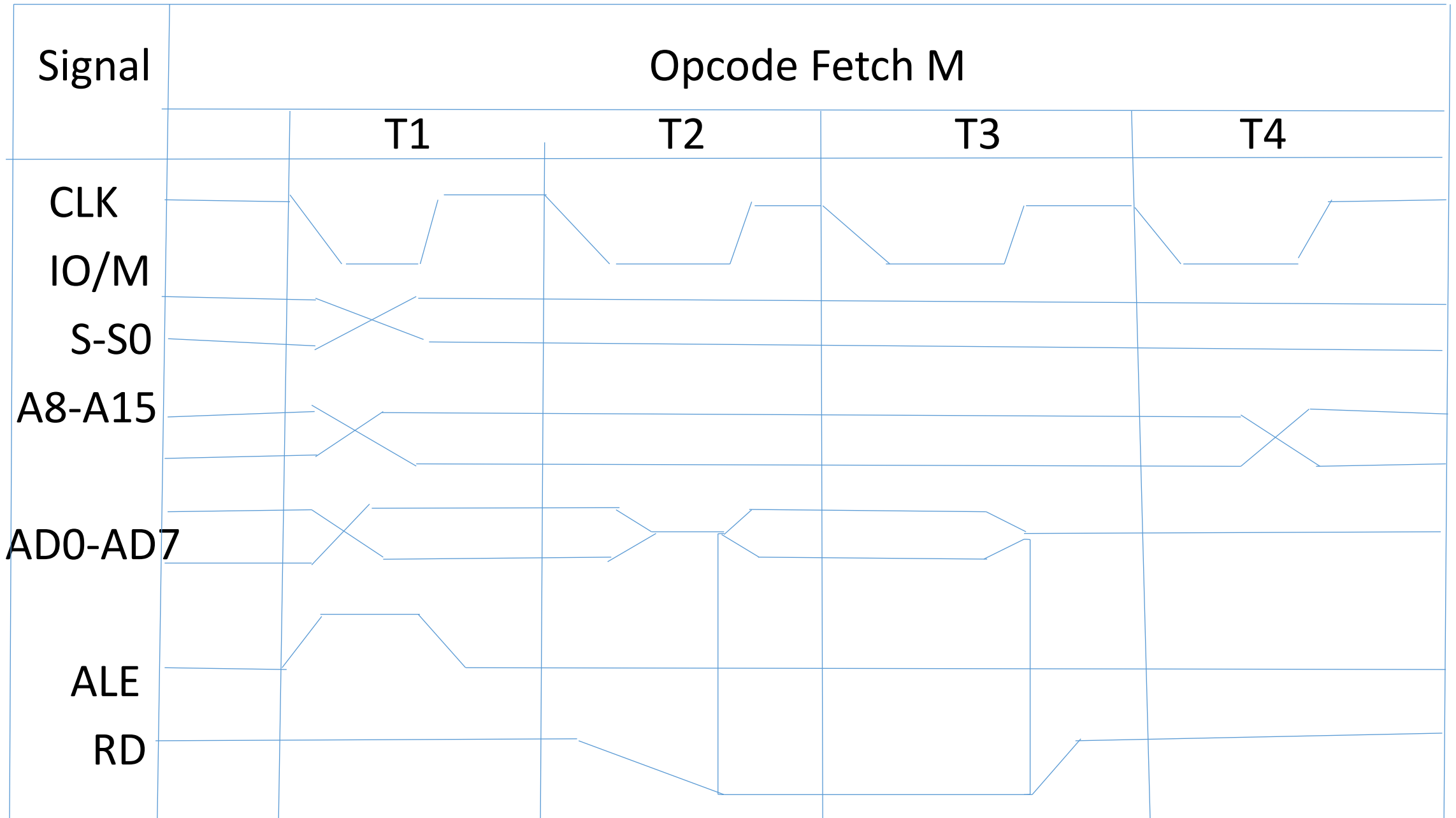
### **During T3 state**

The opcode of A/D bus is transferred to the instruction register

The RD is high thus disabling the memory from A/D bus

### **During T4 state**

The fetched opcode is decoded



## **Memory Read Machine cycle**

If instruction is one byte in length opcode fetch cycle only one machine cycle is required for completion

When the instruction is more than one byte to transfer the operand from memory or I/O device may require a few more machine cycle

Usually these cycles have 3 states

For example

`MVI D,18H`

This instruction is a memory read cycle which consist of two machine cycle .

First machine cycle is to fetch the opcode and the second machine cycle is the memory read machine cycle which transfer the operand – 18 H from memory

### **During T1 state**

The content of program counter is placed on Address or Address / Data bus

The ALE signal goes high and at middle of T1 ALE goes low

The microprocessor recognizes the memory read machine cycle by the status signal

For memory read cycle IO/M=0 S1=1,S0=0

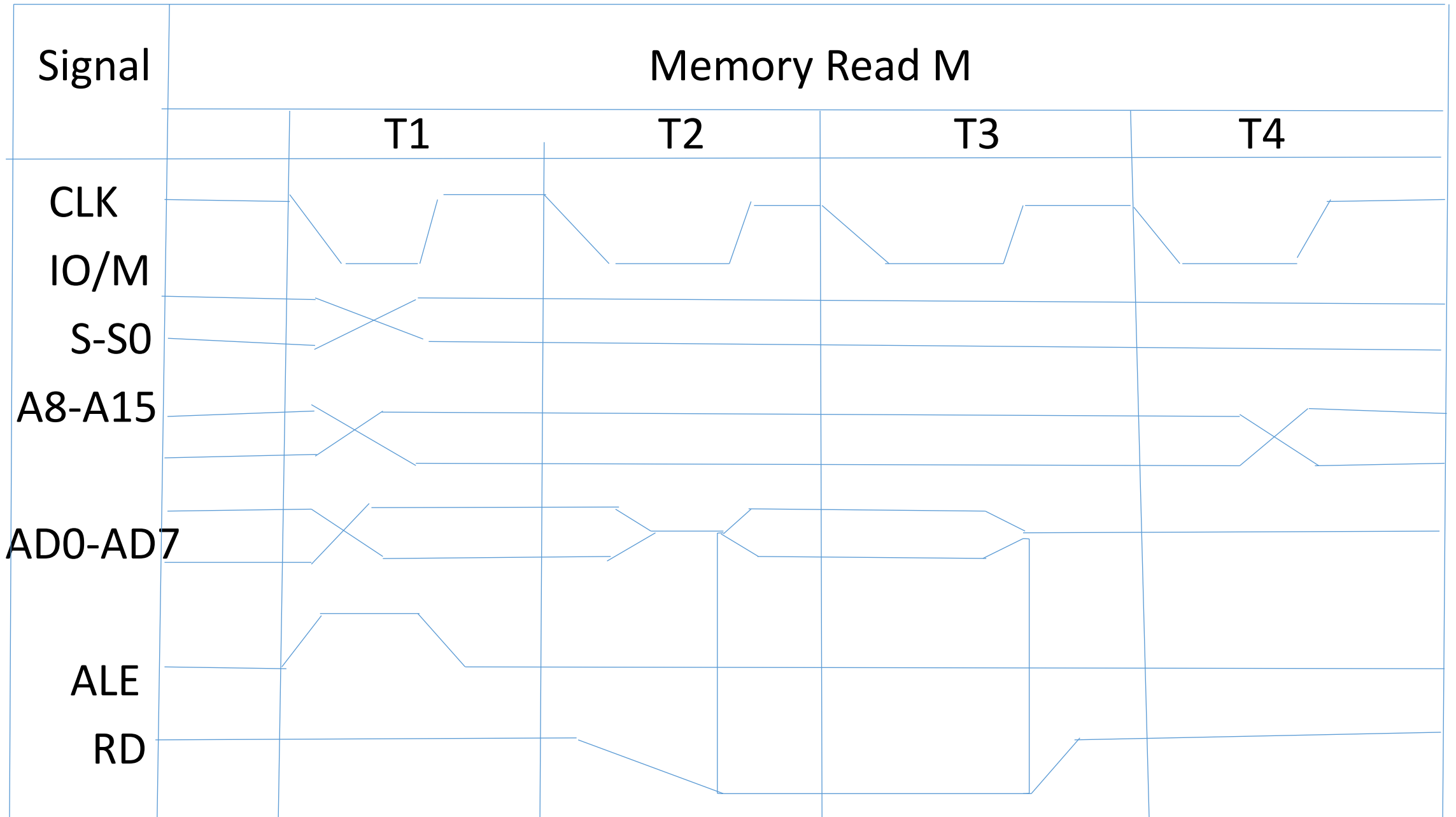
### **During T2 state**

The content of (D0-D7) of selected memory location are placed on the A/D bus

### **During T3 state**

The data loaded on the A/D bus is moved to microprocessor

In the middle of the T3 state RD goes high and disables the memory read operation



## **I/O Read Machine Cycle**

It is similar with read operation

For example IN 80 H

It is a two byte I/O read instruction

The first machine cycle is opcode and fetch and the second is I/O read machine cycle is used where content of the port address is transferred to the microprocessor. The status information for I/O read is  $IO/M = 1, S1=0, S0=0$

## **Memory write machine cycle**

A memory write or I/O write cycle is used for sending data from one of the microprocessor register to the memory or I/O device

For example

MOV M,A

This requires two machine cycle

The first is opcode fetch machine cycle . The second is the write machine cycle used for



Transferring the content of accumulator to the memory location.

### **During T1 state**

The microprocessor place 16-bit address of the memory location on the address or Address / data buffer

The ALE goes high and at the middle of T1 state it goes low which is used to latch the address bit in the memory

The microprocessor recognizes the memory write machine cycle , from the status signal

For the memory write machine cycle  $IO/M = 0$  ,  $S1=0,S0=1$

### **During T2 state**

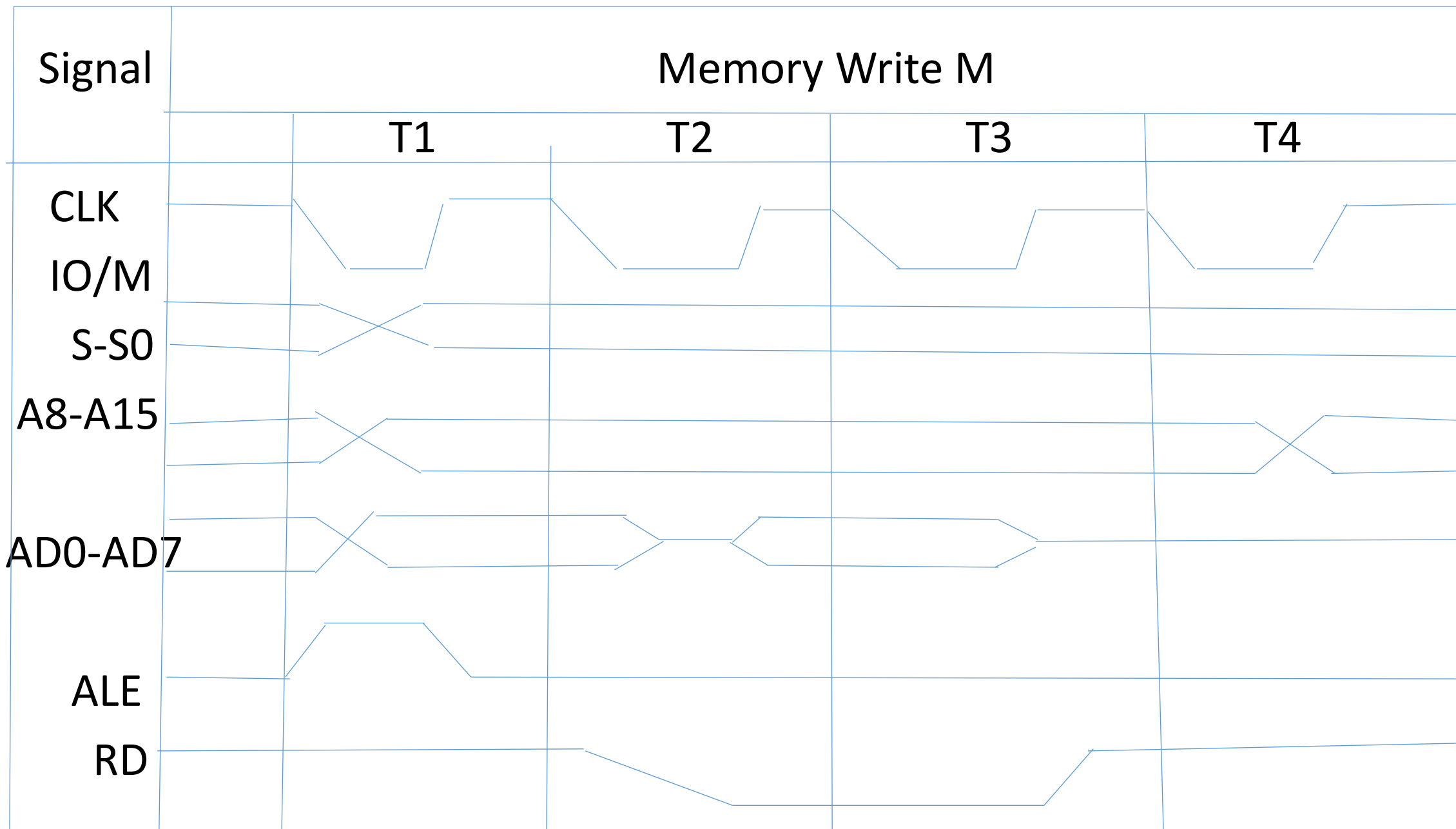
At the beginning of T2 state WR goes low to enable write operation

In this state the content of the register C is placed on data bus

### **During T3 state**

The data on the data bus is transferred to the specific memory location

In the middle of T3 state the control signal WR goes high and disable the memory



## I/O Write Machine cycle

It is similar with memory write operation

Example OUT 03 H

The I/O write machine cycle is used for sending data from accumulator to the output device whose address is 03H

The microprocessor recognize the machine cycle from the status signal . For I/O write machine cycle  $IO/M = 1$  ,  $S1=0, S0=1$

## **INTA Machine cycle**

When microprocessor is executing a program it checks the INTR line during execution of each instruction.

If INTR is high the microprocessor completes the current instruction and send a signal INTA.

## **Bus Idle Machine cycle**

It is a machine cycle during which data bus of the microprocessor is not used . It is in idle state

During bus idle machine cycle

READY line is not sampled by the microprocessor

No memory or I/O device is communicating with the microprocessor

No WAIT cycle are possible

The data bus is not used it is tri stated

## **INTERRUPTS**

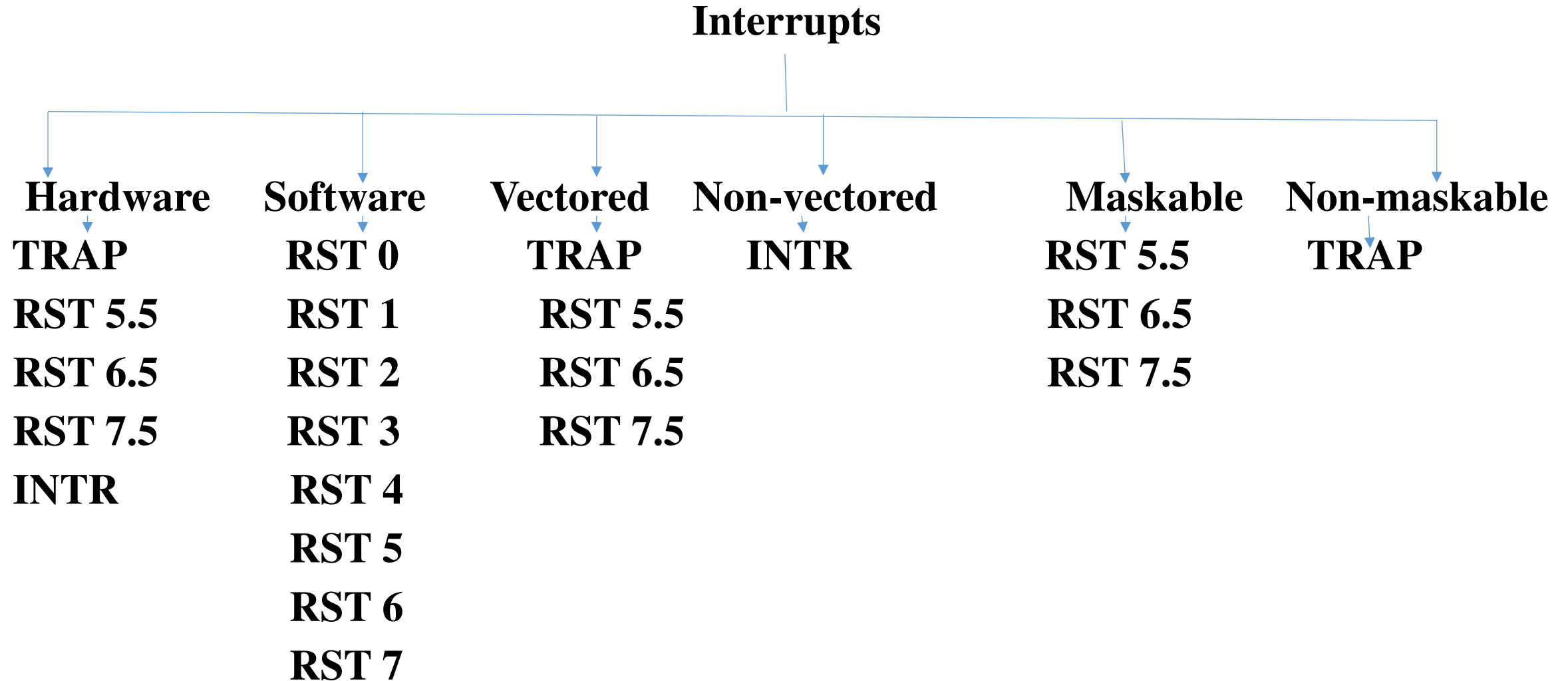
The process of interrupting the normal program execution to carry out a specific task / work is referred to as interrupt

The microprocessor can be interrupted

1. By an external signal generated by peripheral
2. By an internal signal generated by instruction
3. By an internal signal generated due to an exceptional condition which occurs while
4. executing an instruction.

When microprocessor receives an interrupt signal it stops executing the main program, uses the status of all registers in stack then processor executes the interrupt service routine (ISR) in order to perform the specific work requested by the interrupt.

## Classification of interrupts



## **Interrupt Vectored Table**

It is not possible to give each interrupt Service Routine (ISR) a fixed slot in the memory since the different routines may be of different size.

When an interrupt occurs the control does not go to the ISR directly.

Instead a CALL instruction to a predetermined location on the 00 page of the memory is executed.

## **Software Interrupt**

The syntax for software interrupt instruction is RST n

When n range from 0 to 7 (RST1,RST2...)

These are single byte instruction and the Hex code with the address in the Vectored Interrupt table

<b>Instruction Op-code</b>	<b>Hex code in Hex</b>	<b>Vector address</b>
RST0	C7	0000
RST1	CF	0008
RST2	D7	0010
RST3	DF	0018
RST4	E7	0020
RST5	EF	0028
RST 6	F7	0030
RST 7	FF	0038

When any of these instruction are executed a CALL to the specified address is executed.

The content of the program counter is saved in stack memory.

Vector address of the instruction are spaced by 8 byte.

Usually 3 byte JMP instruction to the corresponding ISR is stored

## **Hardware Interrupts**

The 8085 CPU has 5 hardware interrupts

<b>Interrupt</b>	<b>Vector address</b>
TRAP	0024
RST 5.5	002C
RST 6.5	0034
RST 7.5	003C
INTR	-



The space between RST 5 (software interrupt) and RST 5.5 (hardware interrupt) is only 4 byte

The predetermined addresses of the hardware interrupts are in the first page of memory area.

Note that the vector addresses of the hardware interrupt are spaced only 8 bytes

## **Vectored and Non-Vectored Interrupts**

The first 4 hardware interrupt signals (TRAP, RST5.5, RST6.5, RST7.5) are directly vectored to the address specified in the interrupt vector table

They are also called **vectored interrupts**.

However the interrupt request INTR is handled slightly in different way

When INTR interrupt is activated the CPU produce an Interrupt Acknowledgement INTA signal

INTR is called **non vectored interrupt**

## **Maskable and Non-maskable Interrupts**

The hardware interrupts can be classified as maskable and nonmaskable interrupts.

Except TRAP, the others are **maskable interrupts**.

TRAP is a **non maskable interrupt**.

The maskable interrupts may be enabled and disabled by software instructions.

When they are disabled the CPU ignores them when they occur.

## **Priority interrupts**

When a hardware interrupt is activated it is sensed only after completing the present cycle.

If more than one interrupt are actuated at the same time, they are serviced basis.

TRAP has the highest priority.

It is always serviced first then RST 7.5, then RST 6.5 and finally RST 5.5.

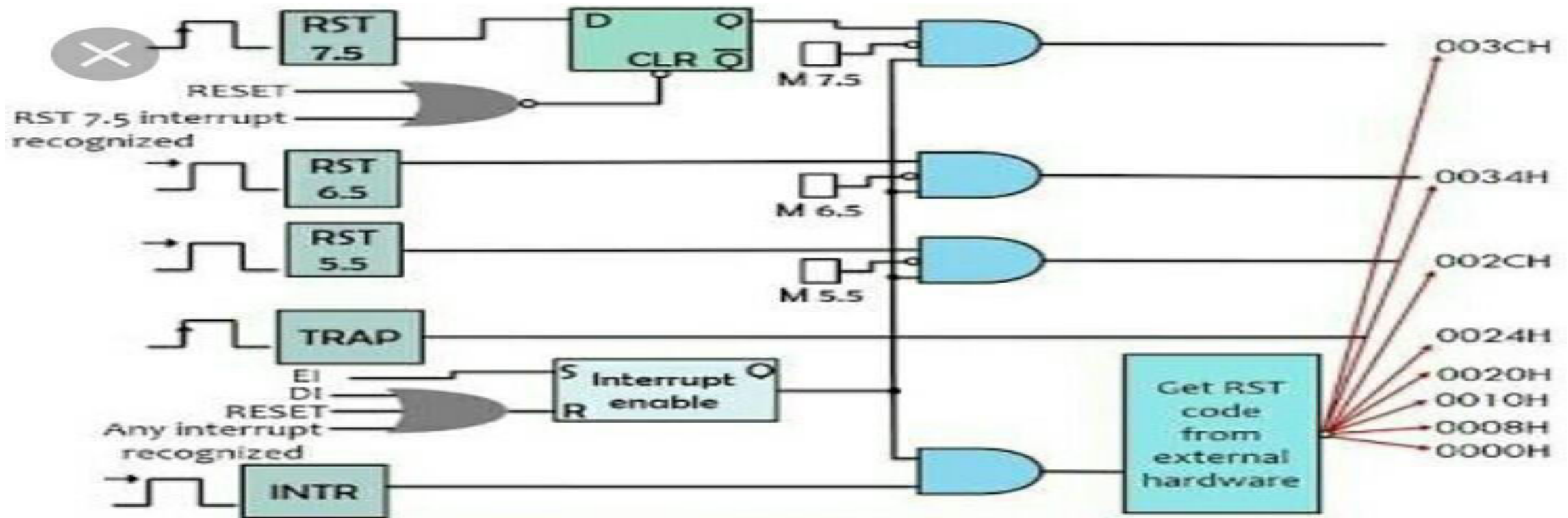
At the end of every instruction cycle, internally, the interrupt are scanned in priority

- TRAP - Highest Priority
- RST 7.5
- RST 6.5
- RST 5.5
- INTR Lowest Priority

## **Interrupt Service Sequence**

- A hardware interrupt occurs or a RST n instruction is encountered in the program.
- CPU completes the execution of the current instruction and scans the interrupt lines.
- If there is any interrupt, the contents of program counter is stored in the stack memory
- The CPU sends out an INTA signal (for hardware interrupt only) CPU branches to the vector table location.

- Execute JMP to specified ISR
- Carriers out ISR
- When RET instruction is encountered the CPU loads the return address the stack into the program counter and returns to the main program



Interrupt structure of 8085 microprocessor

# Working of Interrupt Control

## TRAP

Trap being non-maskable it is used to handle the very important function

In general it is used to take care of sudden power failures. It may execute a routine to transfer the content of the main memory to the back up memory.

It is edge and level sensitive which means input has to go high

The rising edge triggers the D flip-flop

The D flip-flop and the AND output triggers the TRAP input.

The TRAP is cleared by

The RESET signal or

Internal TRAP Acknowledgement signal

As soon as Internal TRAP Acknowledgement signal will reset the D-flipflop

## **RST 7.5,6.5,5.5**

RST 7.5 is edge sensitive. The D-flipflop is actuated by the rising edge and its output become high.

The RST 7.5 interrupt is cleared by internal RST 7.5 interrupt

RST 6.5 and RST 5.5 are level sensitive interrupt

## **MEMORY ORGANIZATION**

A Memory unit is an integral part of any microcomputer and its primary purpose is to hold the instructions and data

The major design goal of a memory unit is it allow it to operate at a speed close to that of microprocessor.

The cost of a memory unit is prohibitive and practically it is not feasible to design large memory unit with one technology.

The memory system is usually designed with different technologies such as solid state, magnetic and optical

The memory system can be divided into three groups

Microprocessor memory

Primary or main memory

Secondary memory

### **Microprocessor memory**

It comprises a set of microprocessor register. The registers are used to hold temporary memory .

There is no speed between the register and microprocessor

### **Main memory**

It is the storage area in which all the program are executed.

Primary memory is the computer memory that is accessed directly by the CPU

These are several types of memory such as processor cache and system ROM

Primary memory is also called as primary storage

The primary storage may also refer to internal storage device like internal hard drives.

The operating system and applications are loaded into primary memory since RAM can be accessed much faster than storage device.

The data can be transferred between CPU and RAM.

### **Secondary memory**

It refers to storage device like hard drives and solid state

It also refers to removable storage such as USB flash drives, Ds and DVD

Unlike primary memory, secondary memory is not accessed directly by CPU

RAM plays an important role it provides fast data to access the speed

Secondary memory is slower than primary memory

Secondary memory is to store permanent data



# **19MZC12&19RAC12 / MICROPROCESSOR AND APPLICATIONS**

**Prepared by**

**Mr.C.RAMKUMAR,**

**Assistant Professor,**

**Department of Electrical and Electronics Engineering,**

**Muthayammal Engineering College (Autonomous),**

**Rasipuram – 637 408.**

# Unit – II 8051 Microcontroller

## Microcontroller

1. 8051 Microcontroller is an 8-bit microcontroller created in 1981. It is most popular and commonly used microcontroller.
2. An 8-bit microcontroller has an 8-bit data bus and 16-bit address bus.
3. It is an integrated chip designed under very large scale integration techniques .
4. It consists of a processor with other peripheral devices like memory, I/O ports and timer. A microcontroller contains all these components in a single chip
5. A microcontroller does not require much additional interfacing ICs for operation and its functions
6. A microcontroller clock speed is limited only to a few tens of MHz

# Classification of microcontrollers

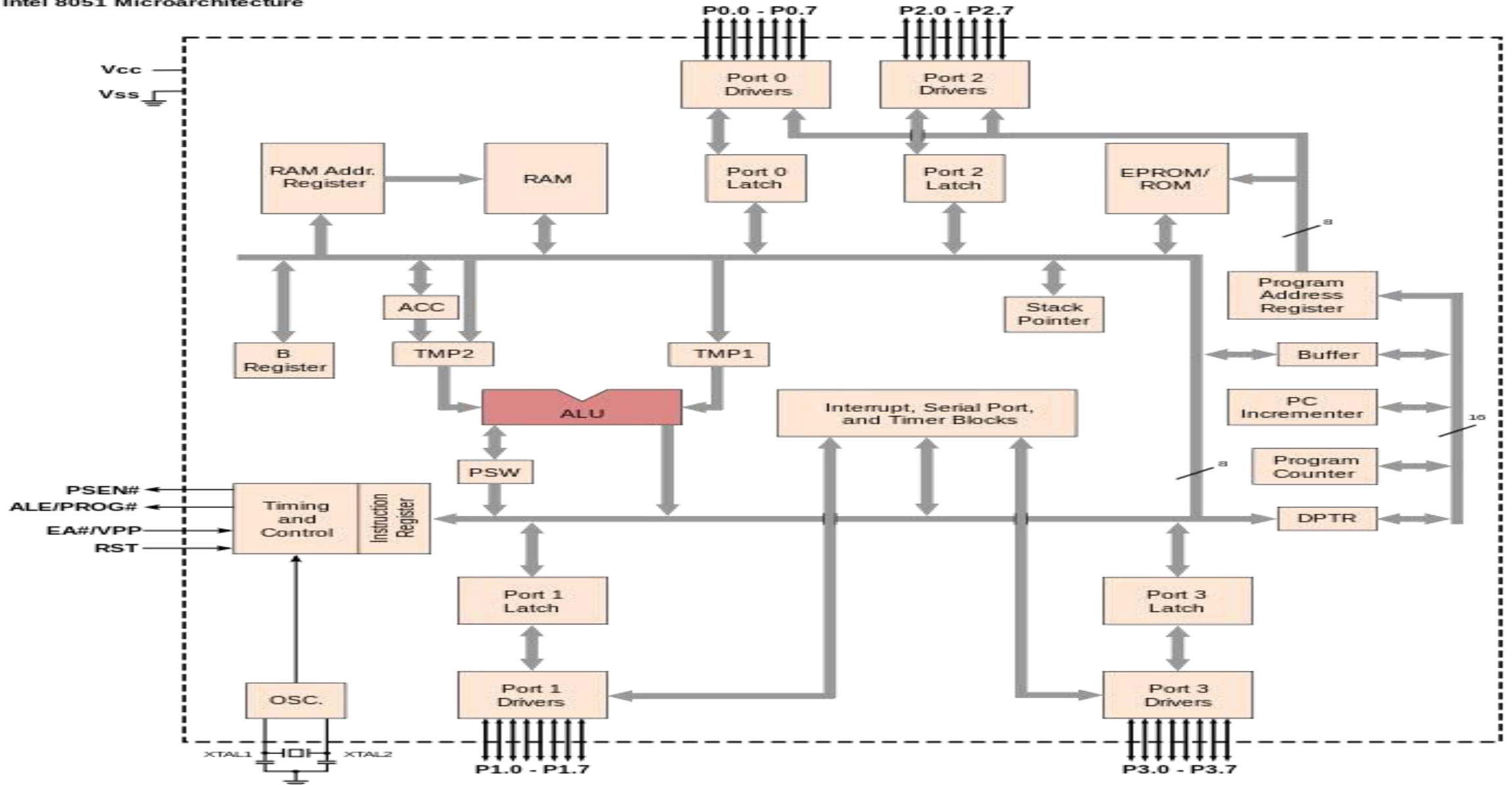
<b>Intel 4004</b>	<b>4 bit (2300 PMOS trans, 108 kHz)</b>	<b>1971</b>
<b>Intel 8048</b>	8 bit	<b>1976</b>
<b>Intel 8031</b>	8 bit (ROM-less)	.
<b>Intel 8051</b>	8 bit (Mask ROM)	<b>1980</b>
<b>Microchip PIC16C64</b>	8 bit	<b>1985</b>
<b>Motorola 68HC11</b>	8 bit (on chip ADC)	.
<b>Intel 80C196</b>	16 bit	<b>1982</b>
<b>Atmel AT89C51</b>	8 bit (Flash memory)	.
<b>Microchip PIC 16F877</b>	<b>8 bit (Flash memory + ADC)</b>	.

# Various features of 8051 microcontroller

- 8-bit CPU
- 16-bit Program Counter
- 8-bit Processor Status Word (PSW)
- 8-bit Stack Pointer
- Internal RAM of 128bytes
- Special Function Registers (SFRs) of 128 bytes
- 32 I/O pins arranged as four 8-bit ports (P0 - P3)
- Two 16-bit timer/counters : T0 and T1
- Two external and three internal vectored interrupts
- One full duplex serial I/O

# ARCHITECTURE OF 8051 MICROCONTROLLER:

Intel 8051 Microarchitecture



## **ALU:**

- It is 8 bit unit
- It performs arithmetic operation as addition, subtraction, multiplication, division, increment and decrement.
- It performs logical operations like AND, OR and EX-OR. It manipulates 8 bit and 16 bit data
- It calculates address of jump locations in relative branch instruction.
- It performs compare, rotate and compliment operations
- 8051 micro controller contains 34 general purpose registers or working registers

## **Accumulator(A-reg):**

It is 8 bit register

It is bit and byte accessible

Result of arithmetic & logic operations performed by ALU is accumulated by this register.

Therefore it is called accumulator register

It is used to store 8 bit data and to hold one of operand of ALU units during arithmetical and logical operations

Most of the instructions are carried out on accumulator data.

### **B-register**

It is special 8 bit math register. It is bit and byte accessible.

It is used in conjunction with A register as I/P operand for ALU. It is used as general purpose register to store 8 bit data.

### **PSW**

It is 8 bit register.

Its address is D0H and It is bit and byte accessible.

It has 4 conditional flags or math flags which sets or resets according to condition of result.

It has 3 control flags, by setting or resetting bit required operation or function can be achieved.

### **FLAG:**

**Carry Flag(CY):** During addition and subtraction any carry or borrow is generated then carry flag is set otherwise carry flag resets.

It is used in arithmetic, logical, jump, rotate and Boolean operations.

**Auxiliary carry flag(AC):** If during addition and subtraction any carry or borrow is generated from lower 4 bit to higher 4 bit then AC sets else it resets.

It is used in BCD arithmetic operations.

**Overflow flag(OV):** If in signed arithmetic operations result exceeds more than 7 bit then OV flag sets else resets.

It is used in signed arithmetic operations only.



**Parity flag(P):** If in result, even no. Of ones "1" are present than it is called even parity and parity flag sets.

In result odd no. Of ones "1" are present than it is called odd parity and parity flag resets.

### **CONTROL FLAGS:**

**FO:** It is user defined flag. The user defines the function of this flag. The user can set test and clear this flag through software.

**RS1 and RS0:** These flags are used to select bank of register by resetting those flags

### **Program counter (PC):**

The Program Counter (PC) is a 2-byte address which tells the 8051 where the next instruction to execute is found in memory.

It is used to hold 16 bit address of internal RAM, external RAM or external ROM locations.

It is important to note that PC isn't always incremented by one and never decremented.

**Data pointer register (DTPR):** It is a 16-bit register used to hold address of external or internal RAM where data is stored or result is to be stored.

It is used to store 16 bit data.

Each register can be used as general purpose register to store 8 bit data and can also be used as memory location.

DPTR does not have single internal address.

It functions as Base register in base relative addressing mode and in-direct jump.

**Stack pointer (SP):**

It is 8-bit register. It is byte addressable.

Its address is 81H.

It is used to hold the internal RAM memory location addresses which are used as stack memory.

When the data is to be placed on stack by push instruction, the content of stack pointer is incremented by 1, and when data is retrieved from stack, content of stack of stack pointer is decremented by 1.

### **Special function Registers(SFR):**

The 8051 microcontroller has 11 SFR divided in 4 groups:

#### **Timer/Counter register**

8051 microcontroller has 2-16 bit Timer/counter registers called Timer-reg-T0 And Timer/counter Reg-T1.

Each register is 16 bit register divide into lower and higher byte register

These register are used to hold initial no. of count.

All of the 4 register are byte addressable.

#### **Timer control register**

8051 microcontroller has two 8-bit timer control register i.e. TMOD and TCON register.

## **TMOD Register:**

It is 8-bit register.

Its address is 89H.

It is byte addressable.

It used to select mode and control operation of time by writing control word.

## **TCON register:**

It is 8-bit register. Its address is 88H.

It is byte addressable.

Its MSB 4-bit are used to control operation of timer/ counter and LSB 4-bit are used for external interrupt control.

## **Serial data register:**

8051 micro controller has 2 serial data register viz. SBUF and SCON.

### **Serial buffer register (SBUF):**

It is 8-bit register.

It is byte addressable .

Its address is 99H.

It is used to hold data which is to be transferred serially.

### **Serial control register (SCON):**

It is 8-bit register.

It is bit/byte addressable.

Its address is 98H.

The 8-bit loaded into this register controls the operation of serial communication.

**Interrupt register:** 8051  $\mu$ C has 2 8-bit interrupt register.

**Interrupt enable register (IE):** It is 8-bit register.

It is bit/byte addressable.

Its address is A8H.

It is used to enable and disable function of interrupt.

### **Interrupt priority register (IP):**

It is 8-bit register.

It is bit/byte addressable.

Its address is B8H.

It is used to select low or high level priority of each individual interrupts.

### **Power control register (PCON):**

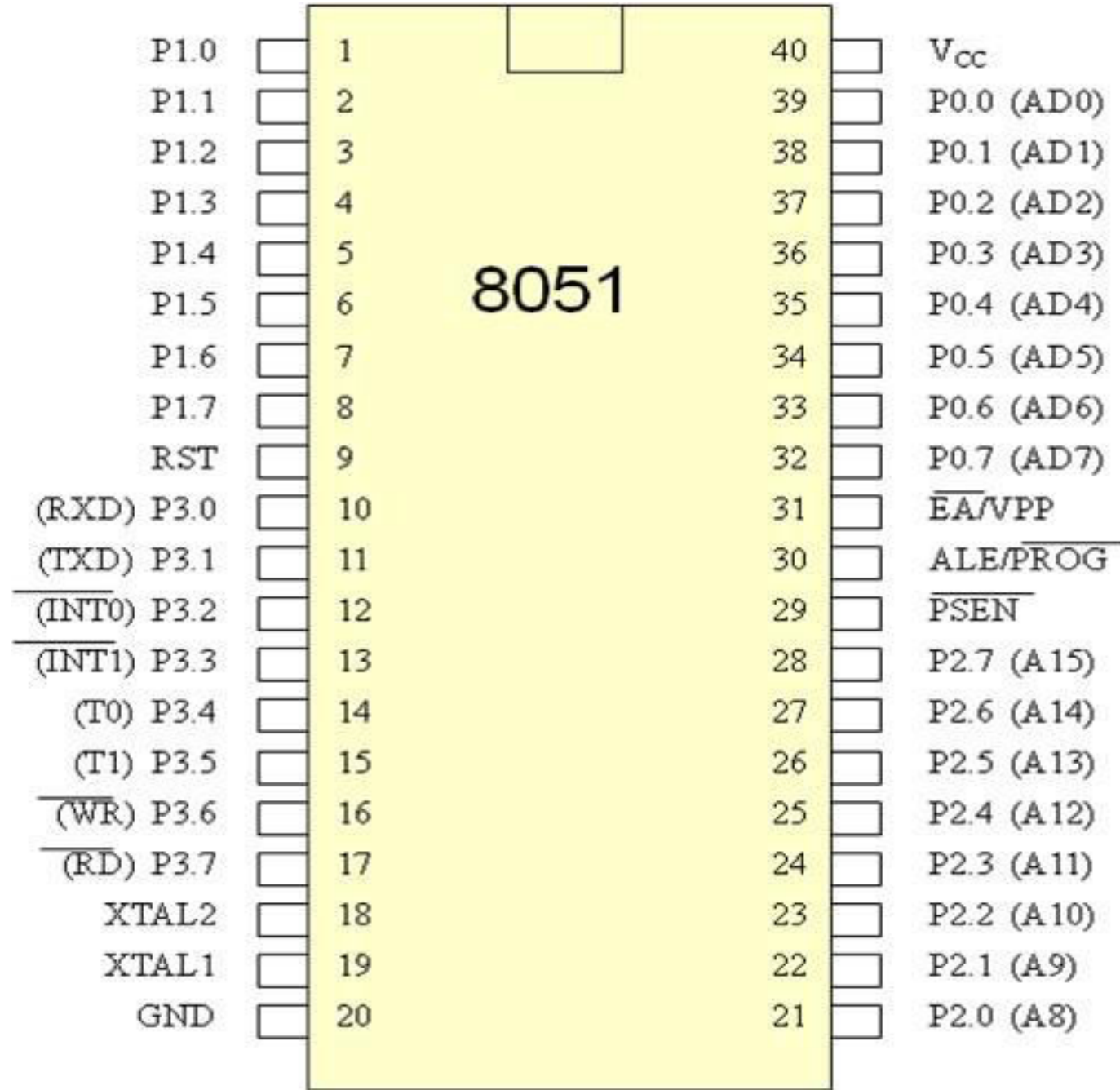
It is 8-bit register.

It is byte addressable .

Its address is 87H.

Its bits are used to control mode of power saving circuit, either idle or power down mode and also one bit is used to modify baud rate of serial communication.

# PIN DIAGRAM



## ***Pinout Description***

<b>Pins 1-8</b>	<b>PORT 1.</b> Each of these pins can be configured as an input or an output.
<b>Pin 9</b>	<b>RESET.</b> A logic one on this pin disables the microcontroller and clears the contents of most registers. In other words, the positive voltage on this pin resets the microcontroller. By applying logic zero to this pin, the program starts execution from the beginning.
<b>Pins10-17</b>	<b>PORT 3.</b> Similar to port 1, each of these pins can serve as general input or output. Besides, all of them have alternative functions
<b>Pin 10</b>	<b>RXD.</b> Serial asynchronous communication input or Serial synchronous communication output.
<b>Pin 11</b>	<b>TXD.</b> Serial asynchronous communication output or Serial synchronous communication clock output.
<b>Pin 12</b>	<b>INT0.</b> External Interrupt 0 input
<b>Pin 13</b>	<b>INT1.</b> External Interrupt 1 input



<b>Pin 14</b>	<b>T0. Counter 0 clock input</b>
<b>Pin 15</b>	<b>T1. Counter 1 clock input</b>
<b>Pin 16</b>	<b>WR. Write to external (additional) RAM</b>
<b>Pin 17</b>	<b>RD. Read from external RAM</b>
<b>Pin 18, 19</b>	<b>XTAL2, XTAL1. Internal oscillator input and output. A quartz crystal which specifies operating frequency is usually connected to these pins.</b>
<b>Pin 20</b>	<b>GND. Ground.</b>
<b>Pin 21-28</b>	<b>Port 2. If there is no intention to use external memory then these port pins are configured as general inputs/outputs. In case external memory is used, the higher address byte, i.e. addresses A8-A15 will appear on this port. Even though memory with capacity of 64Kb is not used, which means that not all eight port bits are used for its addressing, the rest of them are not available as inputs/outputs.</b>
<b>Pin 29</b>	<b>PSEN. If external ROM is used for storing program then a logic zero (0) appears on it every time the microcontroller reads a byte from memory.</b>

**Pin 31** EA. By applying logic zero to this pin, P2 and P3 are used for data and address transmission with no regard to whether there is internal memory or not. It means that even there is a program written to the microcontroller, it will not be executed. Instead, the program written to external ROM will be executed. By applying logic one to the EA pin, the microcontroller will use both memories, first internal then external (if exists).

**Pin 32-39** PORT 0. Similar to P2, if external memory is not used, these pins can be used as general inputs/outputs. Otherwise, P0 is configured as address output (A0-A7) when the ALE pin is driven high (1) or as data output (Data Bus) when the ALE pin is driven low (0).

**Pin 40** VCC. +5V power supply.

## **IO Port Usage in 8051**

The four 8-bit I/O ports P0, P1, P2 and P3 each uses 8 pins.

All the ports upon RESET are configured as input, ready to be used as input ports.

When the first 0 is written to a port, it becomes an output port.

To reconfigure it as an input, 1 must be sent to the port.

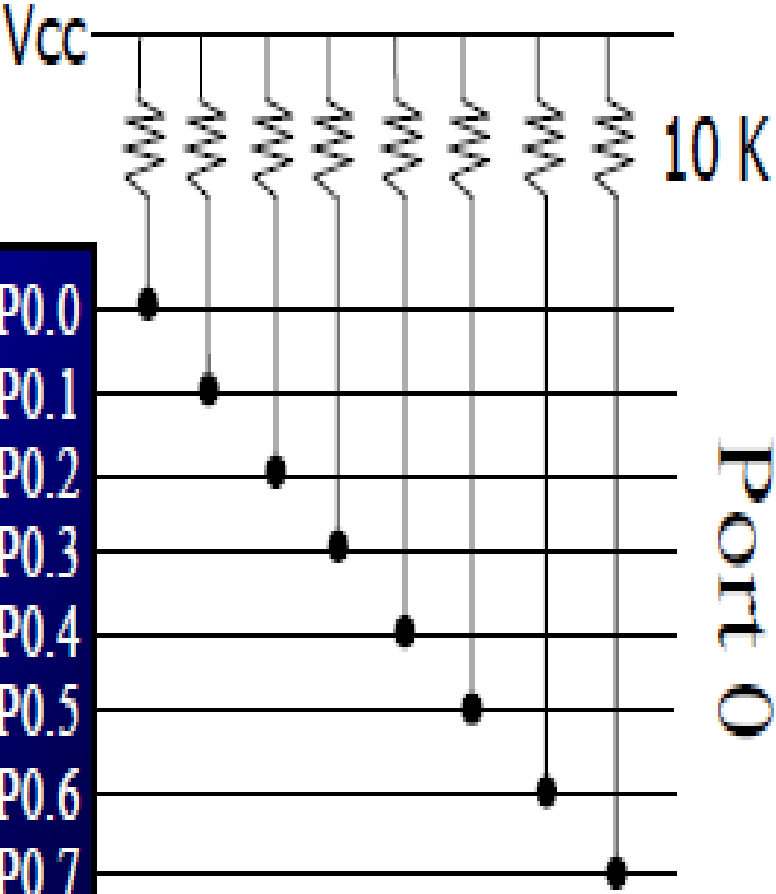
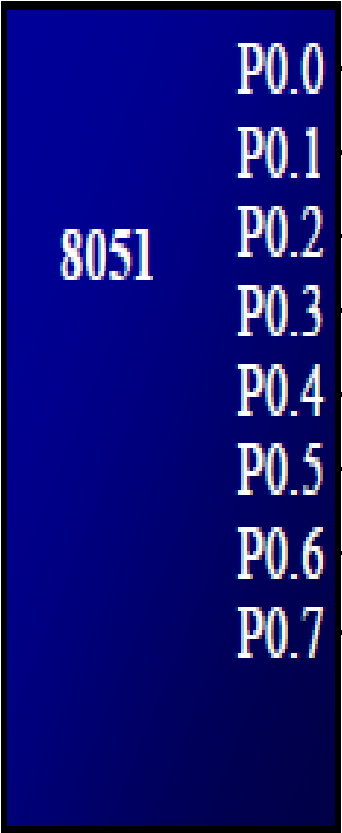
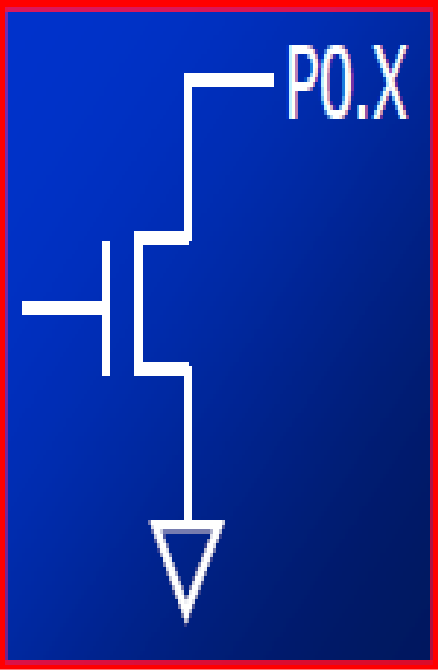
To use any of these ports as an input port, it must be programmed.

It can be used for input or output, each pin must be connected externally to a 10K ohm pull-up resistor.

This is due to the fact that P0 is an open drain, unlike P1, P2, and P3.

Open drain is a term used for MOS chips in the same way that open collector is used for TTL chips.

# Port 0 with Pull up registers



The following code will continuously send out to port 0 the alternating value 55H and AAH

```
BACK: MOV A,#55H
```

- MOV P0,A
- ACALL DELAY
- MOV A,#0AAH
- MOV P0,A
- ACALL DELAY
- SJMP BACK
- **Port 0 as input**
- In order to make port 0 an input, the port must be programmed by writing 1 to all the bits.

Port 0 is configured first as an input port by writing 1s to it, and then data is received from that port and sent to P1

- MOV A,#0FFH ;A=FF hex
- MOV P0,A ;make P0 an i/p port ;by writing it all 1s
- BACK: MOV A,P0 ;get data from P0
- MOV P1,A ;send it to port 1 SJMP BACK

### **Dual role of Port 0**

Port 0 is also designated as AD0-AD7, allowing it to be used for both address and data. When connecting an 8051/31 to an external memory, port 0 provides both address and data.

### **Port 1 can be used as input or output**

In contrast to port 0, this port does not need any pull-up resistors since it already has pull-up resistors internally.

Upon reset, port 1 is configured as an input port.

The following code will continuously send out to port 0 the alternating value 55H and AAH

```
MOV A,#55H
```

- BACK: MOV P1,A
- ACALL DELAY CPL A
- SJMP BACK

To make port 1 an input port, it must be programmed as such by writing 1 to all its bits.

Port 1 is configured first as an input port by writing 1s to it, then data is received from that port and saved in R7 and R5

- MOV A,#0FFH ;A=FF hex
- MOV P1,A ;make P1 an input port ;by writing it all 1s
- MOV A,P1 ;get data from P1
- MOV R7,A ;save it to in reg R7

- `ACALL DELAY` ;wait
- `MOV A,P1` ;another data from P1
- `MOV R5,A` ;save it to in reg R5

## **Port 2 can be used as input or output**

- Just like P1, port 2 does not need any pullup resistors since it already has pull-up resistors internally.
- Upon reset, port 2 is configured as an input port.
- To make port 2 an input port, it must be programmed as such by writing 1 to all its bits.
- In many 8051-based systems, P2 is used as simple I/O.
- In 8031-based systems, port 2 must be used along with P0 to provide the 16-bit address for the external memory.
- Port 2 is also designated as A8 – A15, indicating its dual function.
- Port 0 provides the lower 8 bits via A0 – A7.



## Port 3 can be used as input or output

Port 3 does not need any pull-up resistors. Port 3 is configured as an input port upon reset.

Port 3 has the additional function of providing some extremely important signals

P3 Bit	Function	Pin
P3.0	RxD	10
P3.1	TxD	11
P3.2	$\overline{\text{INT0}}$	12
P3.3	$\overline{\text{INT1}}$	13
P3.4	T0	14
P3.5	T1	15
P3.6	$\overline{\text{WR}}$	16
P3.7	$\overline{\text{RD}}$	17

**Serial communications**

**External interrupts**

**Timers**

**Read/Write signals of external memories**

In systems based on 8751, 89C51 or DS89C4x0, pins 3.6 and 3.7 are used for I/O while the rest of the pins in port 3 are normally used in the alternate function role

Port 3 alternate functions

## **INSTRUCTION SET.**

### **Instruction Timings**

T-state, Machine cycle and Instruction cycle are terms used in instruction timings.

**T-state** is defined as one subdivision of the operation performed in one clock period. The terms 'T- state' and 'clock period' are often used synonymously

**Machine cycle** is defined as 12 oscillator periods. A machine cycle consists of six states and each state lasts for two oscillator periods. An instruction takes one to four machine cycles to execute an instruction.

**Instruction cycle** is defined as the time required for completing the execution of an instruction. The 8051 instruction cycle consists of one to four machine cycles

# 8051 Instructions

The instructions of 8051 can be broadly classified under the following headings.

- Data transfer instructions
- Arithmetic instructions
- Logical instructions
- Branch instructions
- Subroutine instructions
- Bit manipulation instructions

## Data transfer instructions.

In this group, the instructions perform data transfer operations of the following types.

Move the contents of a register Rn to A

- MOV A,R2
- MOV A,R7

Move the contents of a register A to Rn

- MOV R4,A
- MOV R1,A

Move an immediate 8 bit data to register A or to Rn or to a memory location(direct or indirect)

```
MOV A, #45H
```

```
MOV R6, #51H
```

Move the contents of a memory location to A or A to a memory location using direct and indirect addressing

```
MOV A, 65H
```

```
MOV 45H, A
```

Move the contents of a memory location to Rn or Rn to a memory location using direct addressing

```
MOV R3, 65H
```

```
MOV 45H, R2
```

Move the contents of memory location to another memory location using direct and indirect addressing

```
MOV 47H, 65H
```

## **Push and Pop instructions**

MOV R6, #25H

MOV R1, #12H

PUSH 6      [SP]=08 [08]=[06]=25H //CONTENT OF 08 IS 25H

PUSH 1      [SP]=09      [09]=[01]=12H //CONTENT OF 09 IS 12H

PUSH 4      [SP]=0A      [0A]=[04]=F3H //CONTENT OF 0A IS F3H

## **Exchange instructions**

The content of source ie., register, direct memory or indirect memory will be exchanged with the contents of destination ie., accumulator.

XCH A,R3

XCH A,@R1

XCH A,54h

## **Arithmetic instructions.**

The 8051 can perform addition, subtraction. Multiplication and division operations on 8 bit numbers.

### **Addition**

In this group, we have instructions to

Add the contents of A with immediate data with or without carry.

```
ADD A, #45H
```

Add the contents of A with register Rn with or without carry.

```
ADD A, R5
```

```
ADDC A, R2
```

Add the contents of A with contents of memory with or without carry using direct and indirect addressing

```
ADD A, 51H
```

```
ADDC A, 75H
```

# Subtraction

In this group, we have instructions to

Subtract the contents of A with immediate data with or without carry.

SUBB A, #45H

Subtract the contents of A with register Rn with or without carry.

SUBB A, R5

SUBB A, R2

Subtract the contents of A with contents of memory with or without carry using direct and indirect addressing

SUBB A, 51H

SUBB A, 75H

## Multiplication

**MUL AB.** This instruction multiplies two 8 bit unsigned numbers which are stored in A and B register. After multiplication the lower byte of the result will be stored in accumulator and higher byte of result will be stored in B register.

Eg.    MOV A,#45H ; [A]=45H  
      MOV B,#0F5H        :[B]=F5H  
      MUL AB               :[A] x [B] = 45 x F5 = 4209  
                              :[A]=09H, [B]=42H

**DIV AB.** This instruction divides the 8 bit unsigned number which is stored in A by the 8 bit unsigned number which is stored in B register. After division the result will be stored in accumulator and remainder will be stored in B register.

- Eg. MOV A,#45H        :[A]=0E8H
- MOV B,#0F5H        :[B]=1BH
- DIV AB               :[A] / [B] = E8 /1B = 08 H with remainder 10H  
                              :[A] = 08H, [B]=10H



## **DA A (Decimal Adjust After Addition).**

When two BCD numbers are added, the answer is a non-BCD number. To get the result in BCD, we use DA A instruction after the addition

Eg 1: MOV A,#23H

MOV R1,#55H

ADD A,R1 // [A]=78

DA A // [A]=78     **no changes in the accumulator after da a**

Eg 2: MOV A,#53H

MOV R1,#58H

ADD A,R1 // [A]=ABh

DA A // [A]=11, C=1 . ANSWER IS 111. **Accumulator data is changed after DA A**

## **Increment**

Increments the operand by one.

INC increments the value of source by 1

If the initial value of register is FFh, incrementing the value will cause it to reset to 0.

## **Decrement**

Decrements the operand by one.

DEC decrements the value of source by 1. If the initial value of is 0, decrementing the value will cause it to reset to FFh.

## **Logical Instructions**

### **Logical AND**

**ANL** destination, source:

ANL does a bitwise "AND" operation between source and destination, leaving the resulting value in destination

The value in source is not affected

"AND" instruction logically AND the bits of source and destination.

**ANL A,#DATA**

**ANL A, Rn**

**Logical OR**

**ORL destination, source**

ORL does a bitwise "OR" operation between source and destination, leaving the resulting value in destination.

The value in source is not affected. " OR " instruction logically OR the bits of source and destination.

**ORL A,#DATA**

**ORL A, Rn**

**Logical Ex-OR**

**XRL** destination, source

XRL does a bitwise "EX-OR" operation between source and destination, leaving the resulting value in destination.

The value in source is not affected. " XRL " instruction logically EX-OR the bits of source and destination.

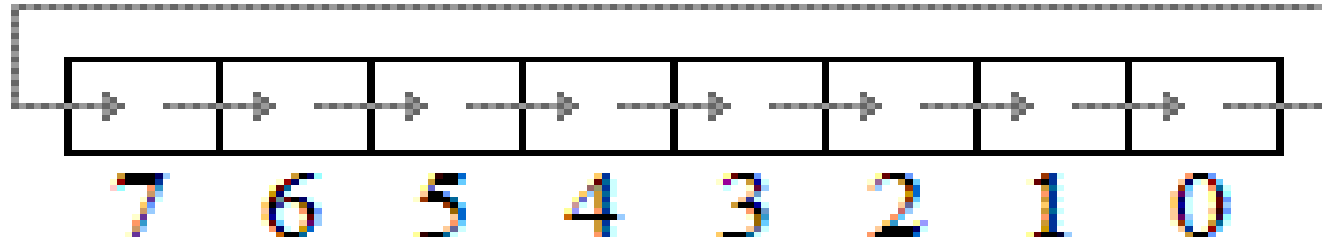
**XRL A,#DATA**

**XRL A,Rn**

## Rotate Instructions RR A

This instruction is rotate right the accumulator.

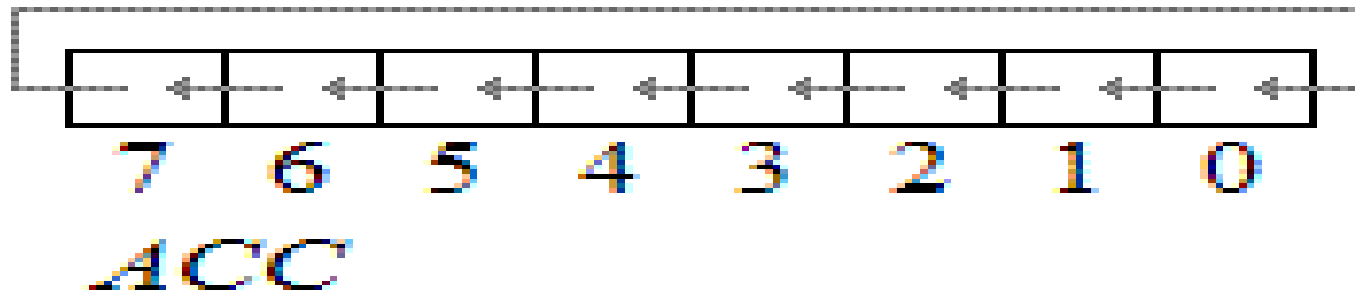
Each bit is shifted one location to the right, with bit 0 going to bit 7.



## RLA

*ACC*

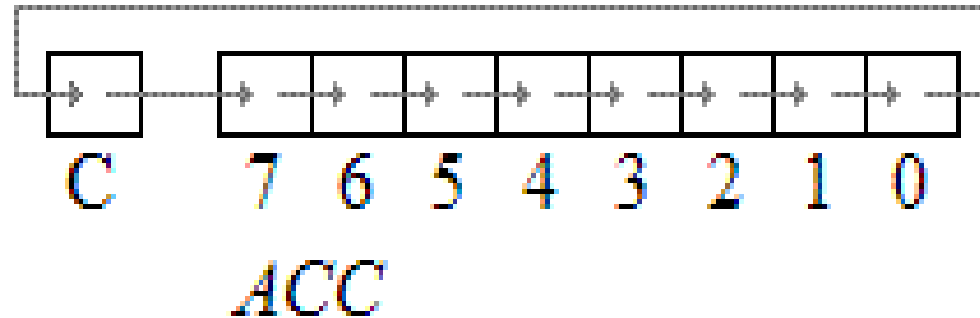
Rotate left the accumulator. Each bit is shifted one location to the left, with bit 7 going to bit 0



## RRC A

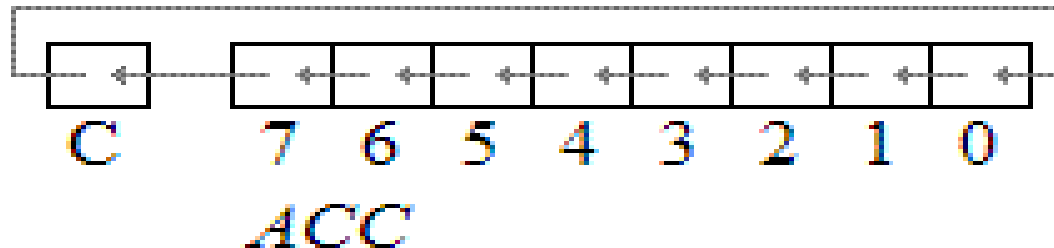
Rotate right through the carry.

Each bit is shifted one location to the right, with bit 0 going into the carry bit in the PSW, while the carry was at goes into bit 7



## RLC A

Rotate left through the carry. Each bit is shifted one location to the left, with bit 7 going into the carry bit in the PSW, while the carry goes into bit 0.



# **Branch (JUMP) Instructions**

## **Jump and Call Program Range**

There are 3 types of jump instructions. They are:-

- Relative Jump
- Short Absolute Jump
- Long Absolute Jump

## **Relative Jump**

Jump that replaces the PC (program counter) content with a new address that is greater than or less than is called a relative jump.

## **The advantages of the relative jump are**

Only 1 byte of jump address needs to be specified in the 2's complement form

- Specifying only one byte reduces the size of the instruction and speeds up program execution.
- The program with relative jumps can be relocated without reassembling to generate absolute jump addresses.

Instructions that use Relative Jump

SJMP <relative address>; this is unconditional jump

The remaining relative jumps are conditional jumps

- JC <relative address>
- JNC <relative address>
- JB bit, <relative address>
- JNB bit, <relative address>
- JBC bit, <relative address>



- CJNE <destination byte>, <source byte>, <relative address>
- DJNZ <byte>, <relative address>
- JZ <relative address>
- JNZ <relative address>

### **Short Absolute Jump**

In this case only 11bits of the absolute jump address are needed.

In 8051, 64 Kbyte of program memory space is divided into 32 pages of 2 kbyte each. The instruction length becomes 2 bytes.

### **Long Absolute Jump/Call**

The entire program memory from 0000H to FFFFH use long absolute jump.

The absolute address has to be specified in the op-code, the instruction length is 3 bytes

Another classification of jump instructions is

- Unconditional Jump
- Conditional Jump

**The unconditional jump** is a jump in which control is transferred unconditionally to the target location.

**LJMP** (long jump). This is a 3-byte instruction.

First byte is the op-code and second and third bytes represent the 16-bit target Address which is any memory location from 0000 to FFFFH.

**AJMP**(Absolute jump)this causes unconditional branch to the indicated address, by loading the 11 bit address to 0 -10 bits of the program counter.

**SJMP** (short jump). This is a 2-byte instruction. First byte is the op-code and second byte is the relative target address, 00 to FFH

To calculate the target address of a short jump, the second byte is added to the PC value

### **Conditional Jump instructions.**

JBC Jump if bit = 1 and clear bit JNB Jump if bit = 0

JB Jump if bit = 1

JNC Jump if CY = 0

JC Jump if CY = 1

CJNE reg,#data Jump if byte  $\neq$  #data

CJNE A,byte Jump if A  $\neq$  byte

DJNZ Decrement and Jump if A  $\neq$  0

JNZ Jump if A  $\neq$  0

JZ Jump if A = 0

All conditional jumps are short jumps

## **Bit level jump instructions:**

Bit level JUMP instructions will check the conditions of the bit and if condition is true, it jumps to the address specified in the instruction. All the bit jumps are relative jumps.

## **Subroutine CALL And RETURN Instructions**

Subroutines are handled by CALL and RET instructions

There are two types of CALL instructions

### **LCALL address(16 bit)**

This is long call instruction which unconditionally calls the subroutine located at the indicated 16 bit address.

This is a 3 byte instruction.

## **ACALL address(11 bit)**

This is absolute call instruction which unconditionally calls the subroutine located at the indicated 11 bit address. This is a 2 byte instruction.

## **RET instruction**

RET instruction pops top two contents from the stack and load it to PC.

## **Bit manipulation instructions.**

8051 has 128 bit addressable memory.

Bit addressable SFRs and bit addressable PORT pins.

It is possible to perform following bit wise operations for these bit addressable locations.

## **LOGICAL AND**

ANL C,BIT(BIT ADDRESS)

ANL C, /BIT;

## LOGICAL OR

- ORL C,BIT(BIT ADDRESS)
- ORL C, /BIT

## CLR bit

- CLR bit
- CLR C

## CPL bit

- CPL bit
- CPL C

# ADDRESSING MODES

Various methods of accessing the data are called addressing modes. 8051 addressing modes are classified as follows.

Immediate addressing

Register addressing.

Direct addressing.

Indirect addressing.

Indexed addressing

Relative addressing.

Absolute addressing.

Long addressing.

Bit inherent addressing.

Bit direct addressing.

## **Immediate addressing.**

In this addressing mode the data is provided as a part of instruction itself. In other words data immediately follows the instruction.

Eg.   MOV A,#30H

      ADD A, #83

## **Register addressing.**

In this addressing mode the register will hold the data. One of the eight general registers (R0 to R7) can be used and specified as the operand.

MOV A,R0

ADD A,R6



## **Direct addressing**

There are two ways to access the internal memory. Using direct address and indirect address.

Using direct addressing mode we can not only address the internal memory but SFRs also.

In direct addressing, an 8 bit internal data memory address is specified as part of the instruction

In this addressing mode, data is obtained directly from the memory.

```
MOV A,60h
```

```
ADD A,30h
```

## **Indirect addressing**

The indirect addressing mode uses a register to hold the actual address that will be used in data movement.

Registers R0 and R1 and DPTR are the only registers that can be used as data pointers.

Indirect addressing cannot be used to refer to SFR registers.

Both R0 and R1 can hold 8 bit address

```
MOV A,R0
```

```
ADD A,R1
```

## **Indexed addressing.**

In indexed addressing, either the program counter (PC), or the data pointer (DTPR)—is used to hold the base address, and the A is used to hold the offset address

Indexed addressing is used with JMP or MOVC instructions

MOVC A, @A+DPTR // copies the contents of memory location pointed by the sum of the accumulator A and the DPTR into accumulator A.

MOVC A, @A+PC // copies the contents of memory location pointed by the sum of the accumulator A and the program counter into accumulator A.

## **Relative Addressing**

Relative addressing is used only with conditional jump instructions.

The relative address is an 8 bit signed number, which is automatically added to the PC to make the address of the next instruction.

The advantage of relative addressing is that the program code is easy to relocate and the address is relative to position in the memory.

SJMP

LOOP1 JC

BACK

### **Absolute addressing**

Absolute addressing is used only by the AJMP (Absolute Jump) and ACALL (Absolute Call) instructions.

These are 2 bytes instructions

The absolute addressing mode specifies the lowest 11 bit of the memory address as part of the instruction.

AJMP LOOP1

ACALL

LOOP2

## **Long Addressing**

The long addressing mode is used with the instructions LJMP and LCALL. These are 3 byte instructions. The address specifies a full 16 bit destination address so that a jump or a call can be made to a location within a 64 Kbyte code memory space.

Eg. LJMP FINISH

LCALL

DELAY

## **Bit Inherent Addressing**

In this addressing, the address of the flag which contains the operand, is implied in the opcode of the instruction.

Eg. CLR C ;

## Bit Direct Addressing

In this addressing mode the direct address of the bit is specified in the instruction.

The RAM space 20H to 2FH and most of the special function registers are bit addressable.

Bit address values are between 00H to 7FH.

Eg.    CLR 07h    ;    Clears the bit 7 of 20h RAM space  
      SETB 07H    ;    Sets the bit 7 of 20H RAM space.

## **Timing diagram**

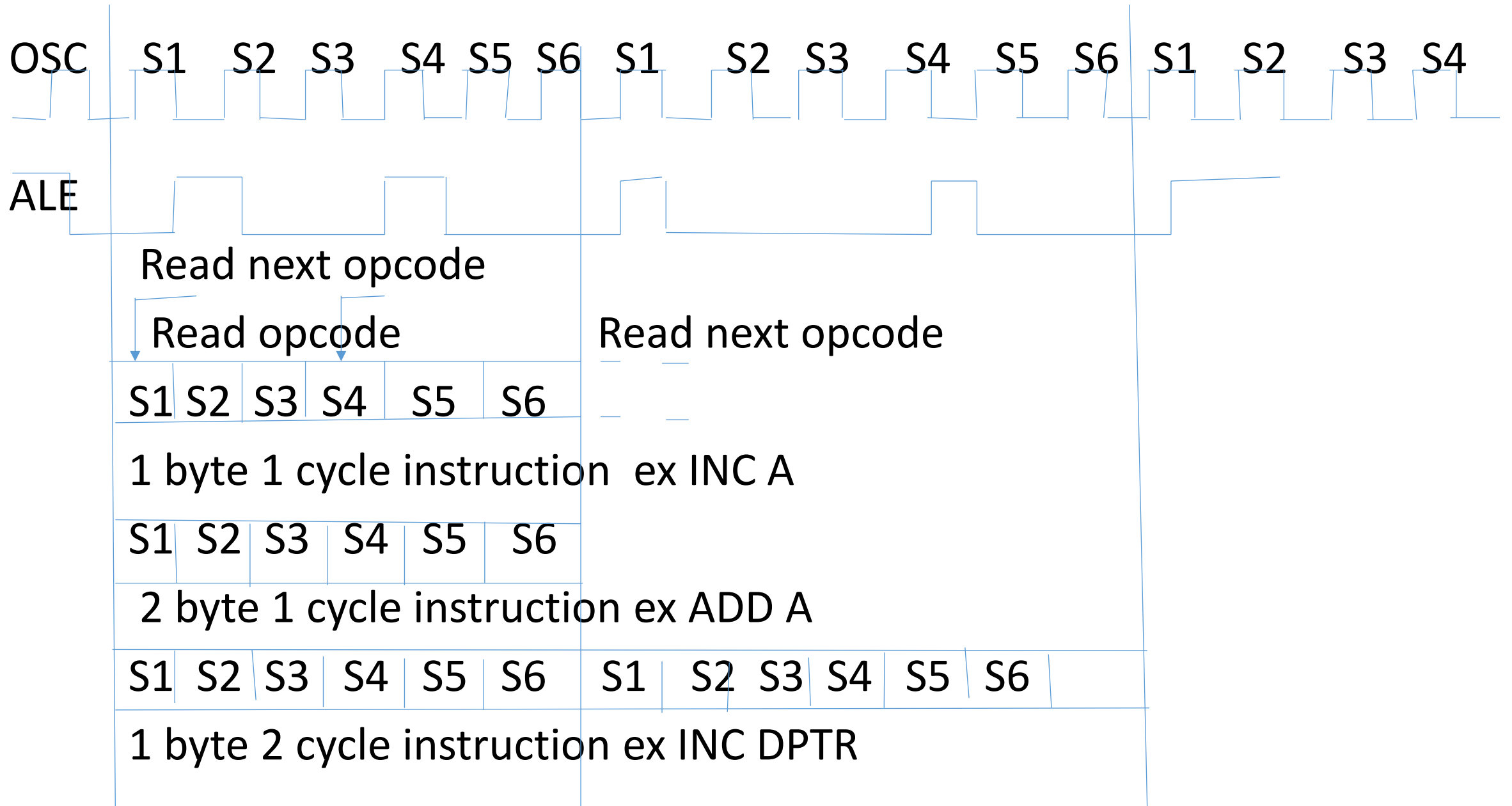
The timing diagram provides the information about the various condition of the signal in which the machine cycle is executed

## **Machine cycle**

It consist of a sequence of 6 states numbered S1 through S6.Each state time lasts for two oscillator period.

The machine cycle take 12 oscillator period or 1microsecond if the oscillator frequency is 12MHz.

Each phase divided into Phase1 half and a Phase 2 half.





Normally two program fetches are generated during each machine cycle, even if the instruction is executed.

If the instruction is executed it does not need more code bytes.

CPU simply the extra fetch and the program counter not incremented

Execution of one cycle instruction begins during State 1 of machine cycle when the opcode is latched into the instruction register

The second fetch occurs during S4 of the same machine cycle

Execution is complete at the end of state 6 of this machine cycle

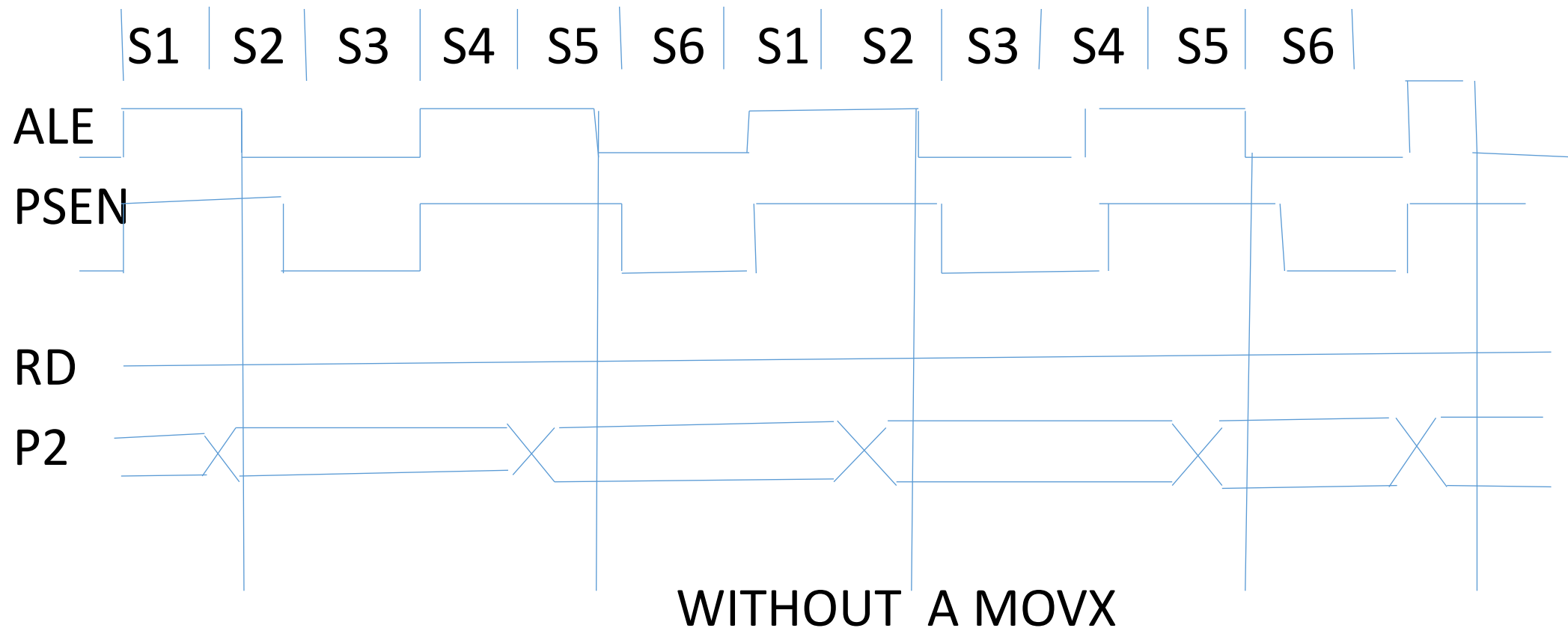
The MOVX instruction takes two machine cycles to execute.

No program fetch is generated during the second cycle of a MOVX instruction

To fetch or execute the sequence of MOVX instruction

To fetch or execute the sequence are same whether the program memory is internal or external to the chip

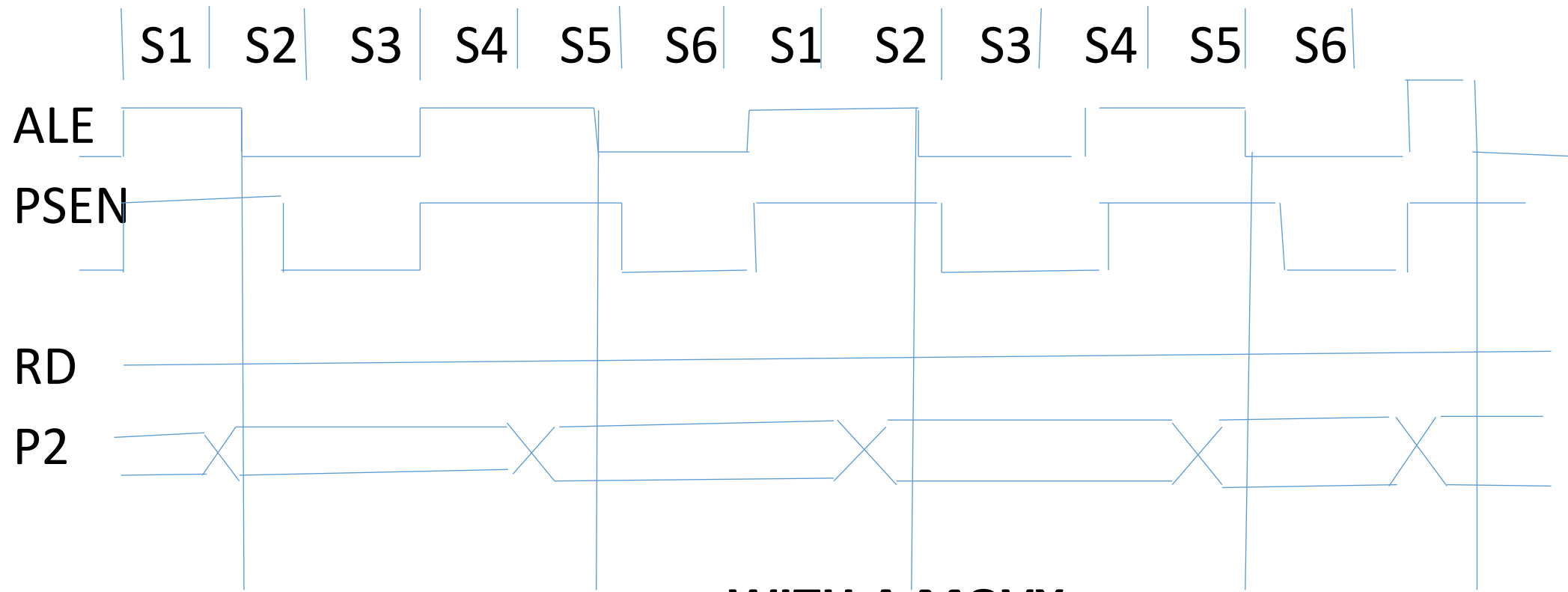
Execution time does not depend on the whether the program memory is internal or external



The signals and timing involved in program fetch when memory program is external

If program memory is external then the program memory read

PSEN is normally activated twice per machine cycle



**WITH A MOVX**

If an access to external data memory two PSEN is skipped because the address and data bus is mainly used for data memory access

Data memory bus cycle takes twice as much time as Program memory bus cycle

ALE is used to latch the low address byte from P0 into the address latch

PSEN is not activated and program address are not emitted

ALE continuous to activate twice per machine cycle and is available as an clock output signal

One ALE skipped during the execution of the MOVX instruction

# **19MZC12&19RAC12 / MICROPROCESSOR AND APPLICATIONS**

**Prepared by**

**Mr.C.RAMKUMAR,**

Assistant Professor,

Department of Electrical and Electronics Engineering,

Muthayammal Engineering College (Autonomous),

Rasipuram – 637 408.

# Unit – III PROGRAMMING AND ADVANCED CONTROLLERS

## Basic programming ALP of 8085

8085 program to sum of two 8 bit number without carry

### Assumption

The starting address of the program is 2000.

Memory address of the first number is 2050.

Memory address of the second number is 2051.

Memory address of the result is 2052.

### Program

Memory address	Mnemonics	Comment
2000	LDA 2050	This instruction will load the number from memory to the accumulator
2003	MOV B,A	This instruction will move the content of accumulator to register B

<b>Memory address</b>	<b>Mnemonics</b>	<b>Comment</b>
2004	LDA 2051	This instruction will load the number from memory to the accumulator
2007	ADD B	This instruction will sum the content of accumulator with the content of register B
2008	STA 2052	This instruction will store the content of accumulator to the memory address 2052
200B	HLT	This instruction will terminate the program

## Example

Input 2050 : 03

2051 : 04

Output 2052 : 07

8085 program to subtract two 8 bit number with or without carry

### **Assumption**

First number is at 2500 memory address

Second number is at 2501 memory address

Store the result into 2502

Borrow into 2503 memory address



<b>Memory address</b>	<b>Mnemonics</b>	<b>Comment</b>
2000	MVI C,00	Move is used to transfer the data from memory to accumulator
2002	LHLD 2500	LHLD is used to load register pair directly using 16-bit address
2005	MOV A,H	MVI is used to move data immediately into any of the register
2006	SUB L	SUB is used to subtract two number where one number is in accumulator
2007	JNC 200B	JNC is used to jump if no borrow
200A	INR C	INR is used to increase register by 1
200B	STA 2502	STA is used to store the content of accumulator into memory
200E	MOV A,C	The instruction will move the content of C register to Accumulator
2010	STA 2503	This instruction will store the content of accumulator to memory address
2013	HLT	HLT is used to end the program

Input data

2501 : 03

2500 : 04

Output data

2503 : 01 Borrow

2502 : 01 Result

**8085 program to multiply two 8 bit number with or without carry**

Two 8 bit number stored at the address 2050 and 2051

Result is stored at the address 3050 and 3051

<b>Memory address</b>	<b>Mnemonics</b>	<b>Comment</b>
2000	LHLD 2050	Load the content of 2051 in H and content of 2050 in L
2003	XCHG	Exchange content of H with D and content of L with E
2004	MOV C,D	Move the content of D to C register
2005	MVI D 00	Move immediate the content of 00 to D
2007	LXI H 0000	Load 0000 to H
200A	DAD D	DAD D add HL and DE and the result store in HL
200B	DCR C	Decrement the content by 1
200C	JNZ 200A	JUMP if nozero
200F	SHLD 3050	Store the value of H at memory location 3051 and L at 3050
2012	HLT	Stop the executing program

Input data

2051 : 07

2050 : 43

Output data

3051 : 01

3050 : D5

### **8085 program to divide two 8 bit number**

Two 8 bit number stored at the address 2050 and 2051

Result is stored at the address 3050 and 3051

<b>Memory address</b>	<b>Mnemonics</b>	<b>Comment</b>
2000	LHI 2050	Load the HL pair register with address 2050 memory location
2003	MOV B,M	Move the content of memory to B register
2004	MOV C,00	Move the content of 00 to C register
2006	INX H	Increment the register pair HL
2007	MOV A,M	Move the content of memory to A register
2008	CMP B	Compare the content of accumulator and register B
2009	JC 2011	Jump if address 2011 carry flag is set
200C	SUB B	Subtract the content of accumulator with B register
200D	INR C	Increment the register C
200E	JMP 2008	Stop the executing program
20011	STA 3050	Store the remainder at memory location 3050
2014	MOV A,C	Move the content of C register into accumulator
2015	STA 3051	Store the remainder at memory location 3051
2018	HLT	Stop the program

Input data

FF 2051

FF 2050

Output data

01 3051

FE 3050

## **Introduction to 8051 Programming in Assembly language**

Assembly language is a low-level programming language.

It is used to write the program code in terms of mnemonics

Even though there are many high level language that are currently in demand

Assembly Programming language is popularly used in many applications.

It can be used for direct hardware manipulations

In 8051 programming code is a less number of clock cycle by consuming less memory when compared to other high-level languages.

## **8051 Programming in assembly language**

The assembly language is a fully hardware related programming languages

The assembly language is developed by mnemonics therefore users cannot understand it easily to modify the program

Microcontroller can understand only binary language in the form of 0's and 1's

An assembler convert the assembly language to binary language and then it stores the microcontroller memory

## **Rules of Assembly language**

The assembly code must be written in upper case letter

The label must be followed by colon

All symbols and labels must begin with a letter

All comments are typed in lower case

The last line of the program must be the end directive

The assembly language mnemonics are in the form of opcode such as MOV,ADD,JMP which is used to perform operation

Example MOV a,b

Where MOV is opcode

And a,b is operand

## **OPCODE**

The opcode is a single instruction that can be executed by CPU. Here the opcode is a MOV instruction



# **OPERAND**

The operand are single piece of data that can be operated by the op-code

Example : multiplication operation is performed by the operand

MUL a,b

## **Elements of Assembly Language program**

Assembler Directives

Instruction set

Addressing modes

### **Assembler Directives**

The assembling directive gives the direction to the CPU.

The 8051 microcontroller consist of various kinds of assembly directive to give the direction to control unit

The most useful directives are 8051 programming such as

ORG

DB

EQU

END

ORG (origin)

The directive indicates the Start of the program. This is used to set the register address during assembly

Syntax : ORG0000h

ORG directive

0000h memory address

## **DB Define Byte**

Define Byte is used to allow a strings of byte.

Example : DB"EDGEFX"

DB is directive

EDGEFX is string

## **EQU Equivalent**

Equivalent directive is used to Equate the address of the variable

Ex reg,equ,0h

## **END**

End directive is used to indicate the end of the program

Ex reg,equ,0h

END

## **Addressing mode**

Direct addressing

Register addressing

Indirect addressing

Indexed addressing

## **Instruction set**

Data Transfer Instruction

Arithmetic Instruction

Logical Instruction

Branching Instruction

# **LOOP STRUCTURE WITH COUNTING AND INDEXING**

The program is an implementation of certain logic by executing group of instructions

To implement program logic we need to help some of programming techniques like looping, counting, indexing and code conversion

## **Looping, counting and indexing**

### **Looping**

In this technique program is instructed to execute certain set of instruction repeatedly to execute a particular task number of times.

For example To add ten number the result will be stored in consecutive memory location we have to perform addition ten times

## **Counting**

This technique allow the programmer to count how many times the instruction /set of instruction is executed.

## **Indexing**

This technique allow the programmer to point or refer the data stored in sequential memory location one by one.

Let us see the program loop to understand looping, counting and indexing.

The program loop is the basic structure which forces the processor to repeat a sequence of instruction

Loop have four sections

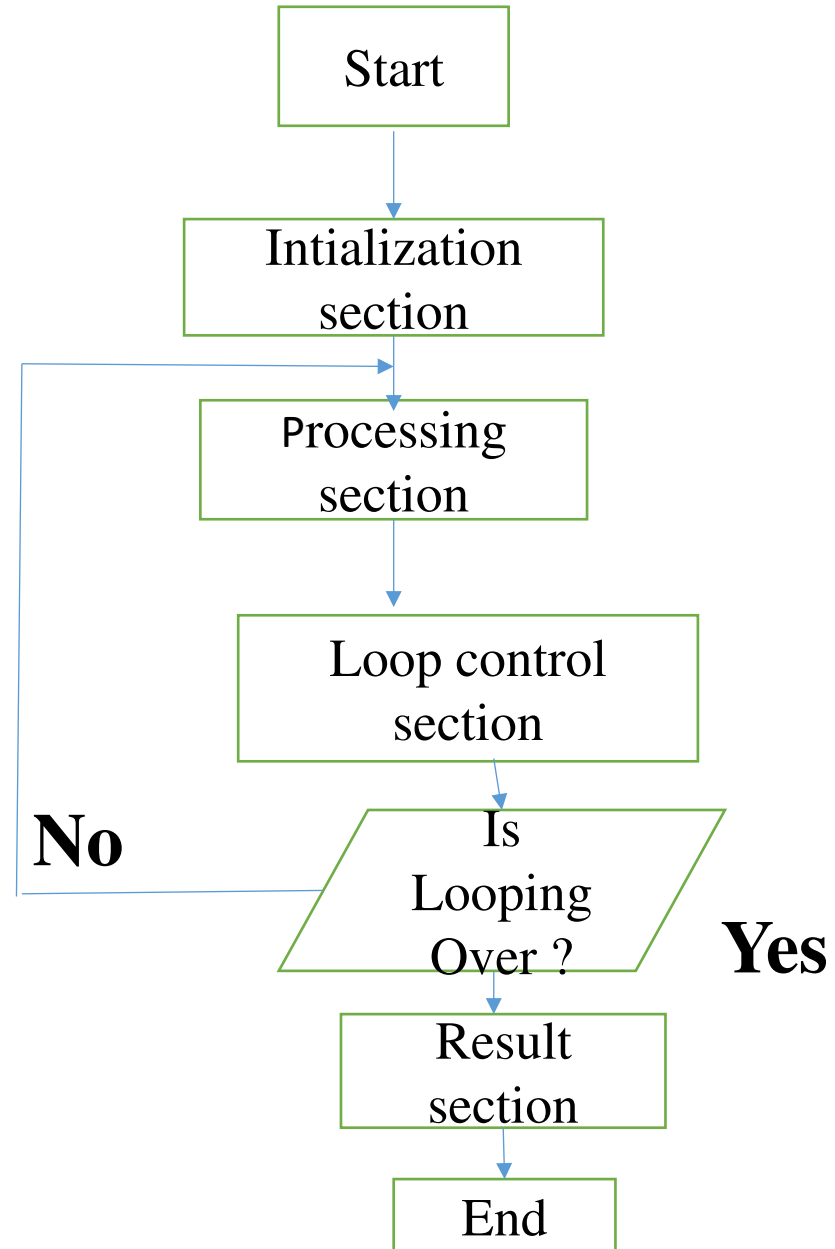
1.Initialization section

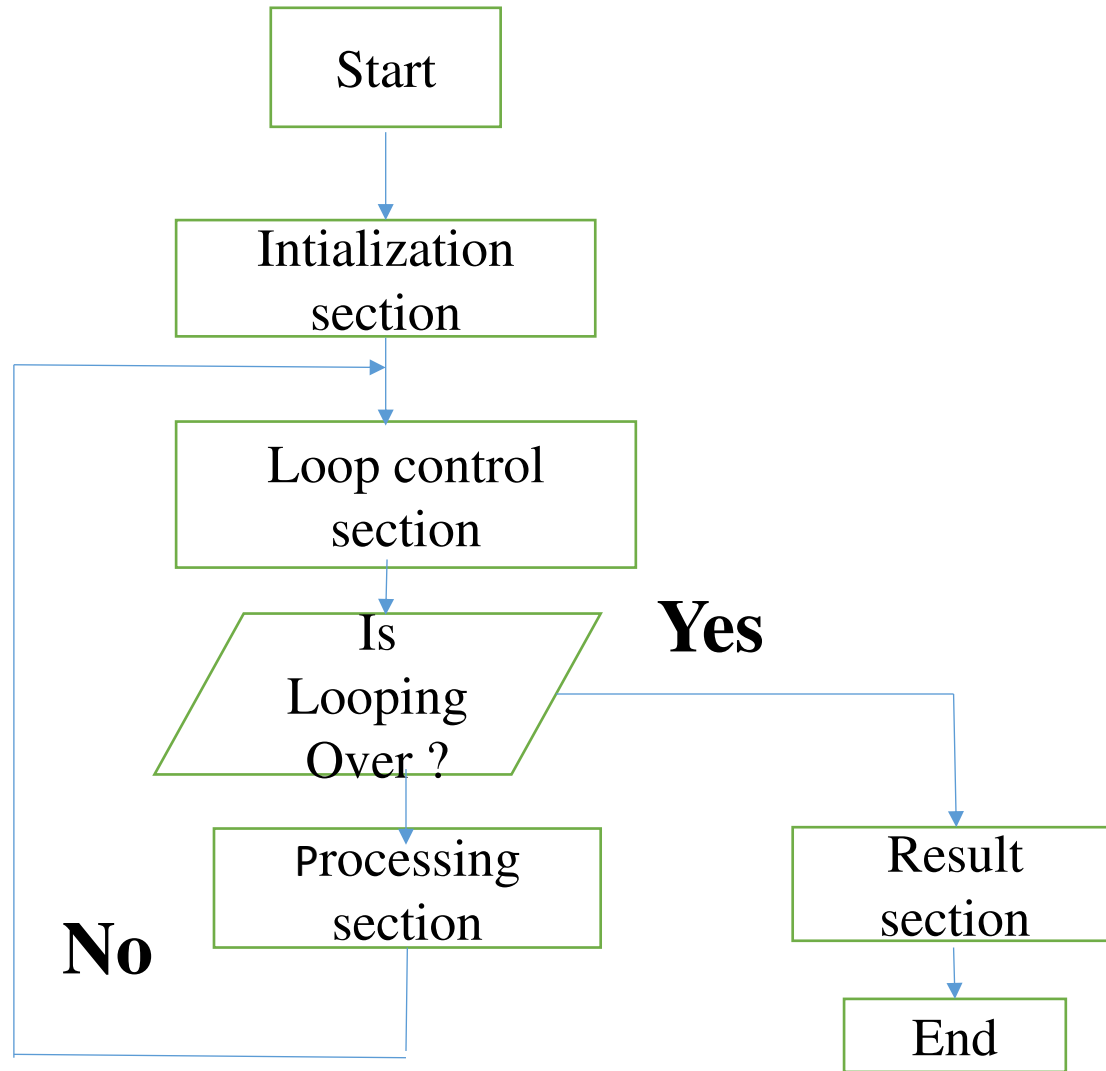
2.Processing section

3. Loop control section

4. Result section

## Flow chart







The initialization section establishes the starting values of  
Loop counter for counting how many loops is executed  
Address registers for indexing which gives pointer to memory location and  
other variables

The actual data manipulation occurs in the processing section

The loop control structure update counters and pointers for next instruction

The result section can analyze and store the result

The processor can executes initialization section and result section only once  
while it may executes processing section and loop control section many times.

The execution time of loop will be mainly dependent on the execution time of  
processing section and loop control section

## **Example program for looping**

### **Calculate sum of series of numbers**

The length of the series is in memory location 2200H and the series itself begins from memory location 2201H

1. Assume the sum to be 8 bit number. Store the sum at memory location 2300H

2. Assume the sum to be 16bit number. Store the sum at memory location 2300H and 2301H

Sample problem

2200H =04H

2201H=20H

2202H=15H

2203H=13H

2204H=22H

Result = 20+15+13+22=60H

2300H=60H

## **Program**

LDA 2200H

MOV C,A - Initialize counter

SUB A -Sum =0

LXI 2201H -Initialize pointer

back ADD M - sum = sum+data

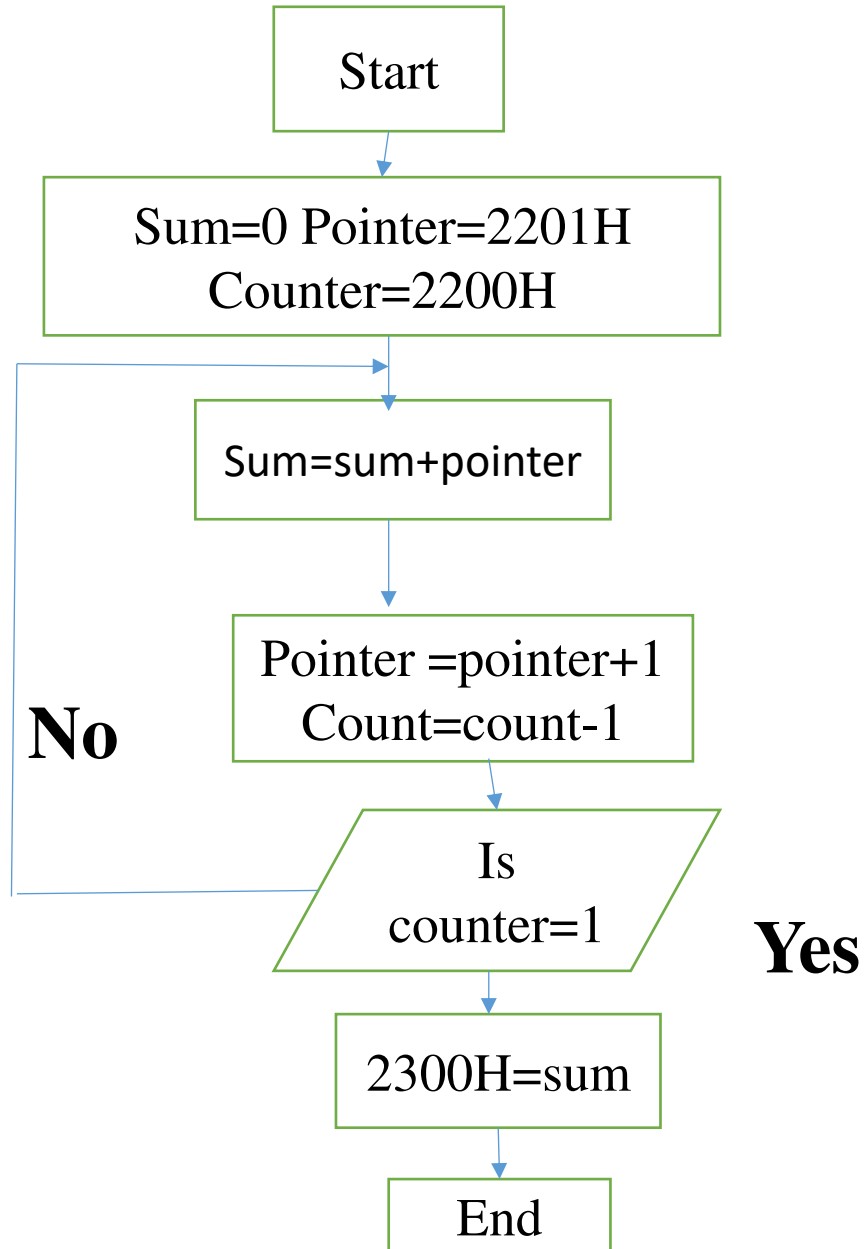
INX H - Increment pointer

DCR C - Decrement counter

JNZ BACK – If counter = 0

STA 2300H – store sum

HLT – End of the program

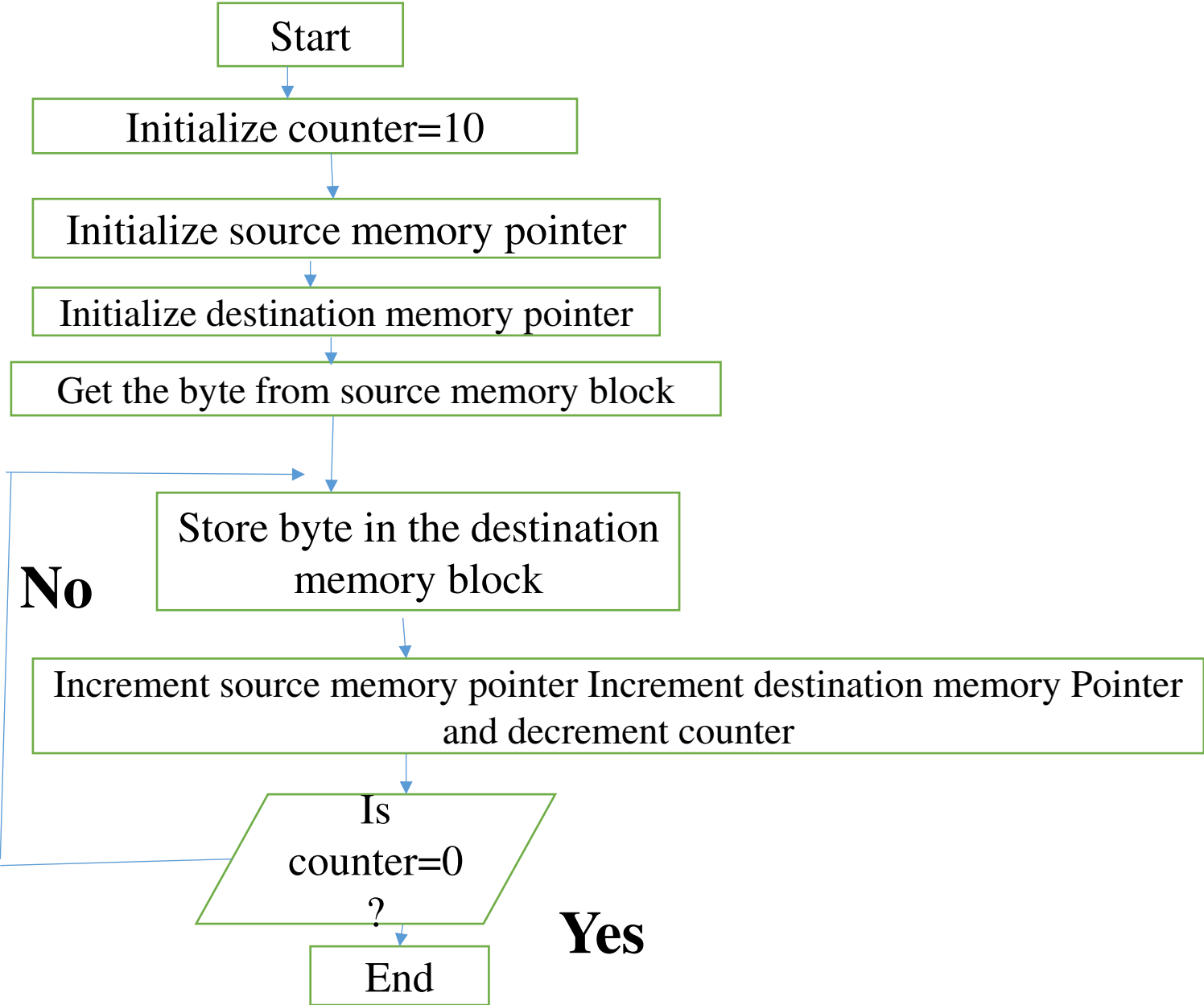


## **Indexing**

Indexing means referring the data in sequential manner. Datas are stored in the consecutive memory location

Example : Data transfer from Memory block B1 to Memory Block B2

Transfer ten bytes of data from memory to another memory block .Source memory blocks starts from memory location 2200H whereas destination memory block starts from memory location 2300H



## Program

MOVI C,0AH - Initialize counter

LXI 2200H -Initialize source memory pointer

LXI 2300H -Initialize destination memory pointer

back MOV A,M -Get byte from source memory block

STAX D - Store byte in destination memory block

INX H - Increment pointer

INX D - Increment destination memory pointer

DCR C - Decrement counter

JNZ BACK – if counter  $\neq 0$  repeat

HLT – End of the program

Example for looping and counting

Find the largest of given number

The length of the block in memory location 2200H and block itself start from memory location 2201H. Store the maximum number in memory location 2300H.

Consider adding array of 8-bit number

```
LXI H,9000H
```

```
MOV C,M
```

```
INX H
```

```
XRA A
```

```
MOV B,A
```

```
LOOP ADD M
```



JNC INCR

INR B

INCR INX H

DCR C

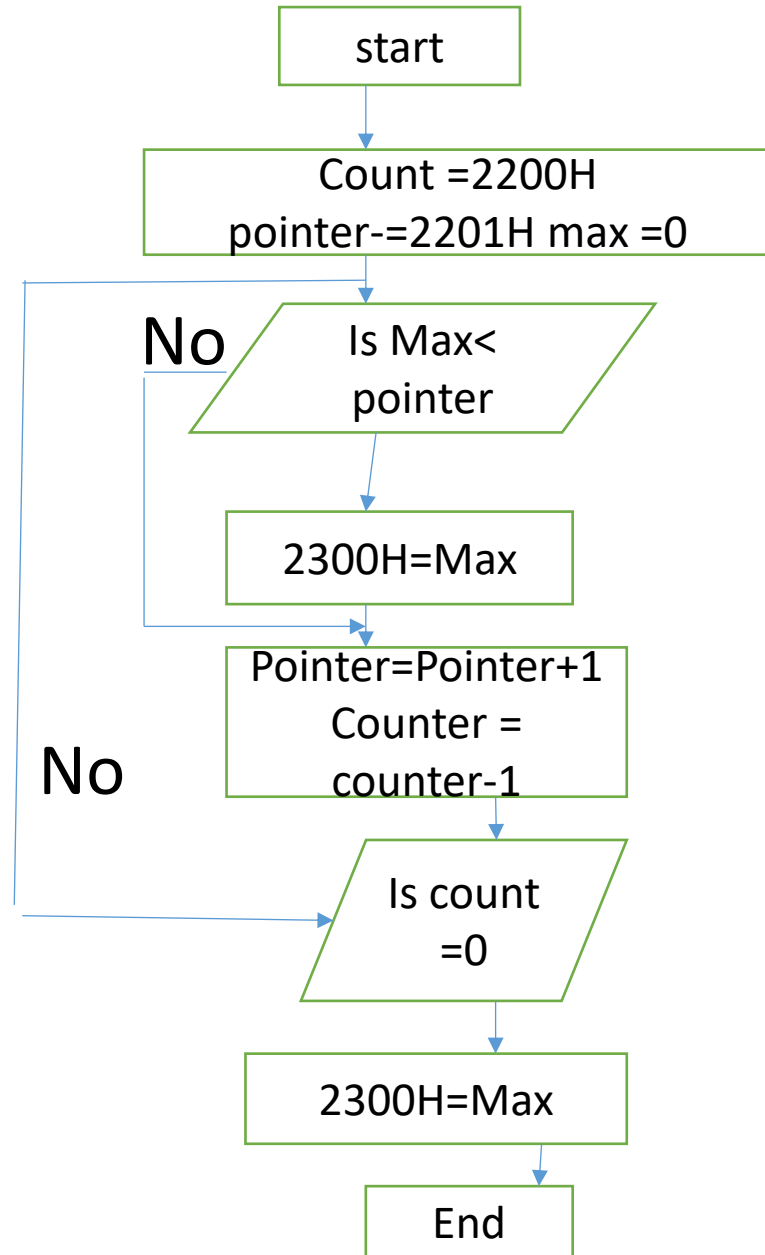
JNZ LOOP

MOV M,B

INX H

MOV M,A

HLT



## **Problem**

2200H =04H

2201H=34H

2202H=A9H

2203H=78H

2204H=56H

Result = 2202H =A9H

## **Program**

LDA ,2200H

MOV C,A –Initialize counter

XRA A Maximum=Maximum possible value0

BACK CMP M Is number > maximum

JNC SKIP

MOV A,M Yes replace

SKIP INX H

DCR C

JNZ BACK

STA 2300H store maximum number

HLT end of the program

# **SUBROUTINE**

The group of instruction are written to perform these operation and the group of instruction are known as Subroutine

This also called as main program

When a main program calls a subroutine the program execution is transferred to the subroutine and after the completion of the subroutine the program execution return to the main program

## **Subroutine Instruction**

The subroutine mainly implemented into two instruction namely CALL and RET.

CALL is used to call a subroutine and the call instruction is written into a main program

RET is used to return a subroutine and the return Instruction is written in the subroutine to return to the main program

When subroutine called the content of program counter are stored on stack the program execution transferred to the subroutine address

When return instruction is executed at the end of the subroutine the memory address stored in the stack is retrieved and the sequence of execution is resumed in the main program

## **CALL**

Call subroutine in conditional located at the memory address specified in 16-bit operation

This instruction place the address of next instruction on the stack and transfer the program execution to the subroutine address

## **RET**

Return unconditionally from the subroutine

This instruction locates the return address on the top of the stack and transfers the program execution back to the calling program

The general characteristics of CALL and RET instruction

1. The CALL instructions are 3-byte instruction .The second byte specifies the low order byte , and the third byte specifies the high order byte of the subroutine address

The Return instruction are 1-byte instruction

A CALL instruction must be used in conjunction with a RETURN instruction in the subroutine

**Type of subroutine are**

1. Multiple call subroutine
2. Nested subroutine
3. Multiple ending subroutine



## **Multiple Call Subroutine**

This is a subroutine called from many location in the main program.

Example : DELAY routine is a multiple call subroutine

These type of routine are easy to trace and need minimal stack space

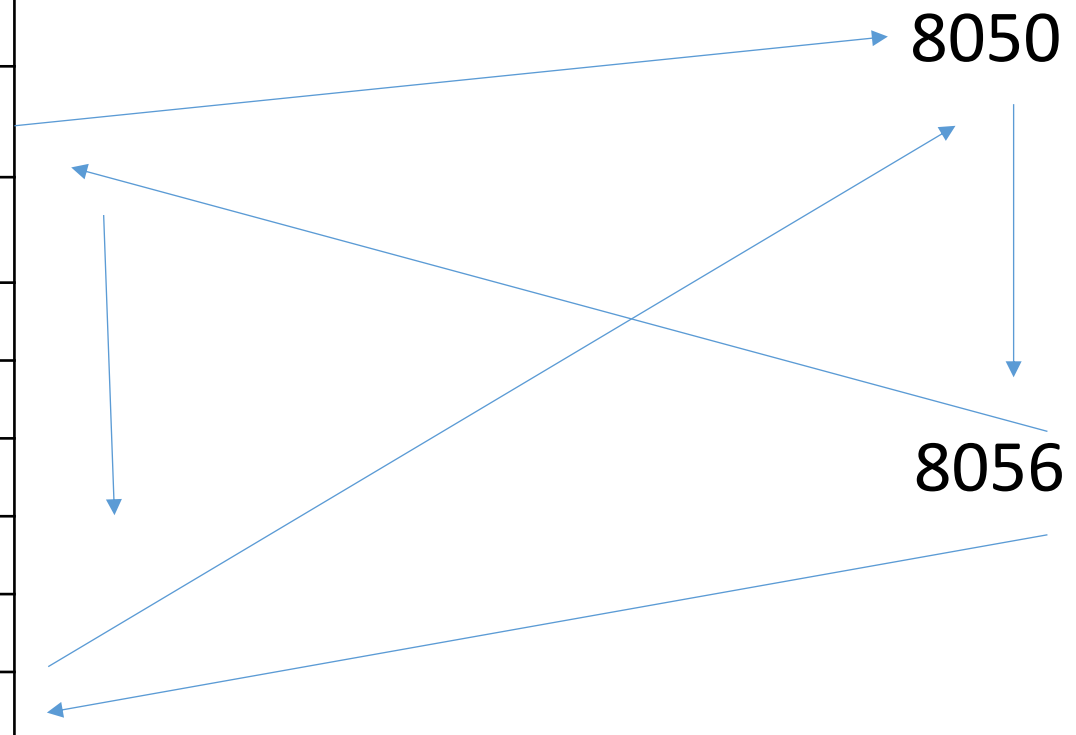
When CALL instruction starts to execute the subroutine is called as 8050 memory location

The return address is stored on the stack and the stack pointer is decremented by two location

8000	
8001	CALL
8002	80
8003	50
8004	
	CALL
	80
	50

sUBROUTINE

RET

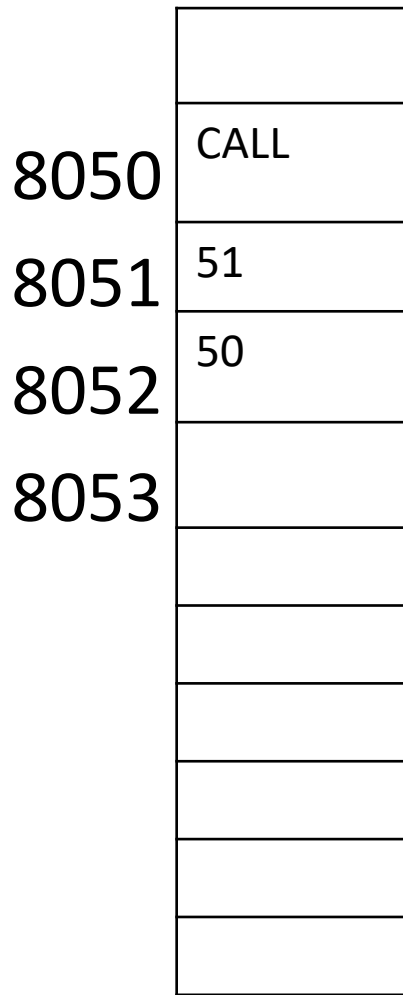


## **Nested Subroutine**

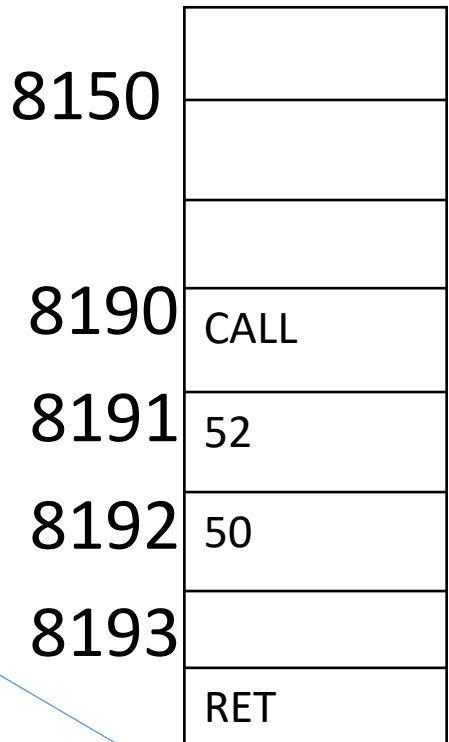
When subroutine called by another subroutine it is called nested subroutine.

When a subroutine calls another subroutine all return address are stored on stack.

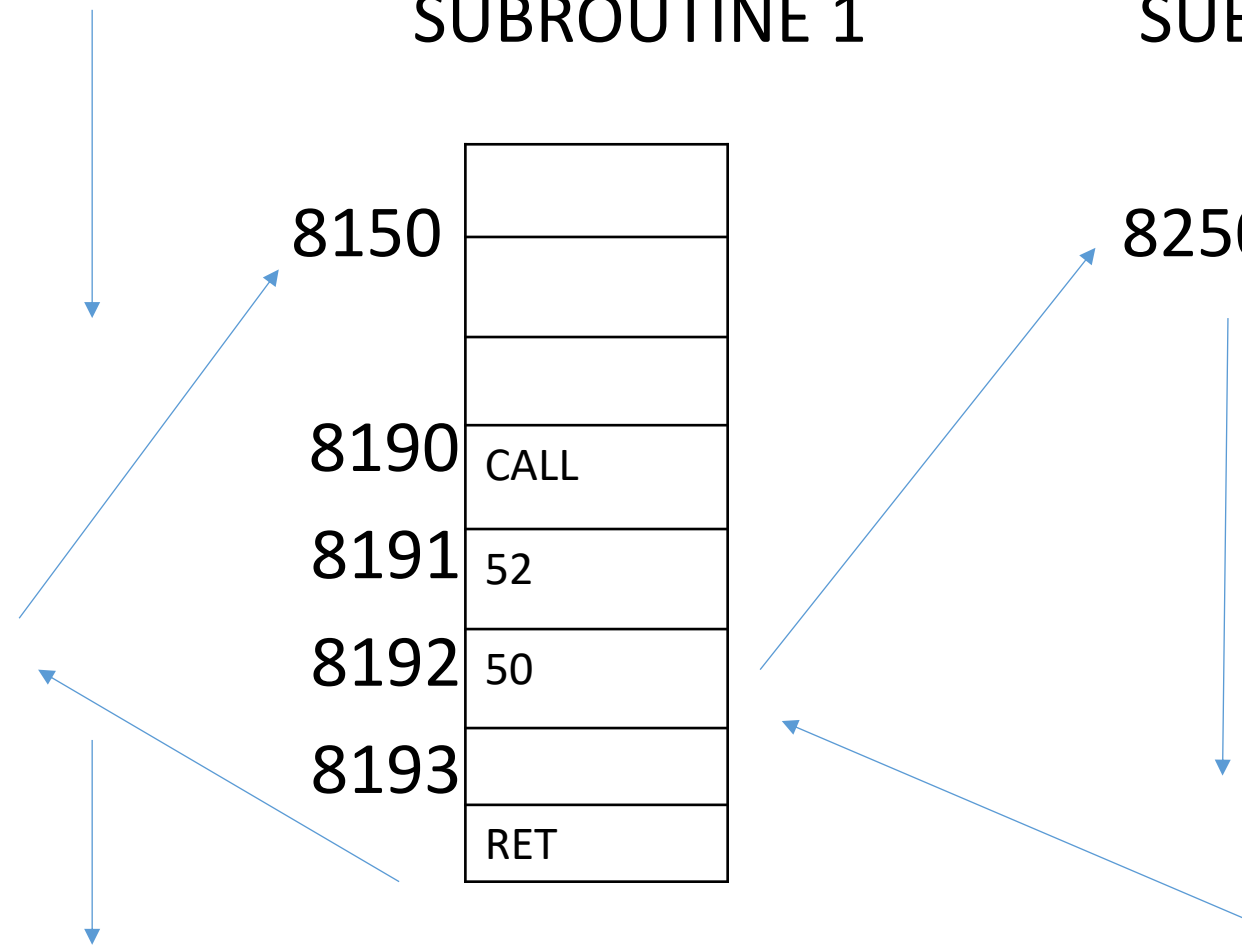
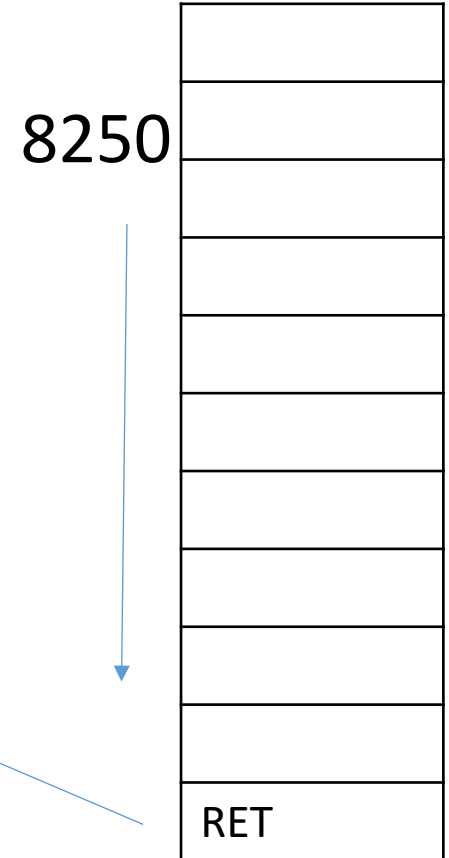
Therefore the number of available stack location limits the extending of nesting



### SUBROUTINE 1

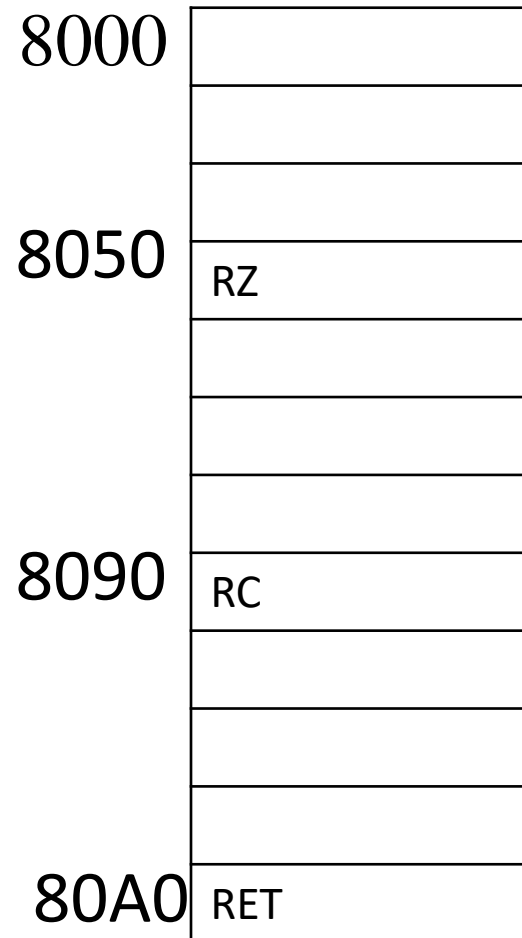


### SUBROUTINE 2



## Multiple Ending Subroutine

When a subroutine can be terminated at more than one place it is called a multiple ending subroutine



## **Subroutine parameters passing**

Various information is passed between a calling program and a subroutine a procedure called parameter passing

Therefore it is important to document a subroutine clearly and carefully

The documentation include

- 1.Function of subroutine
- 2.Input/Output parameters
- 3.Registers used or modified
- 4.List of other subroutine called by this subroutine

Example program for subroutine

Find the factorial of a number

Write a program to calculate the factorial of a number between 0 to 8

Program

LXI SP,27FFH        Initialize stack pointer

LDA 2200H            Get the number

CPI02H              Check if the number is quarter than 1

OC LAST

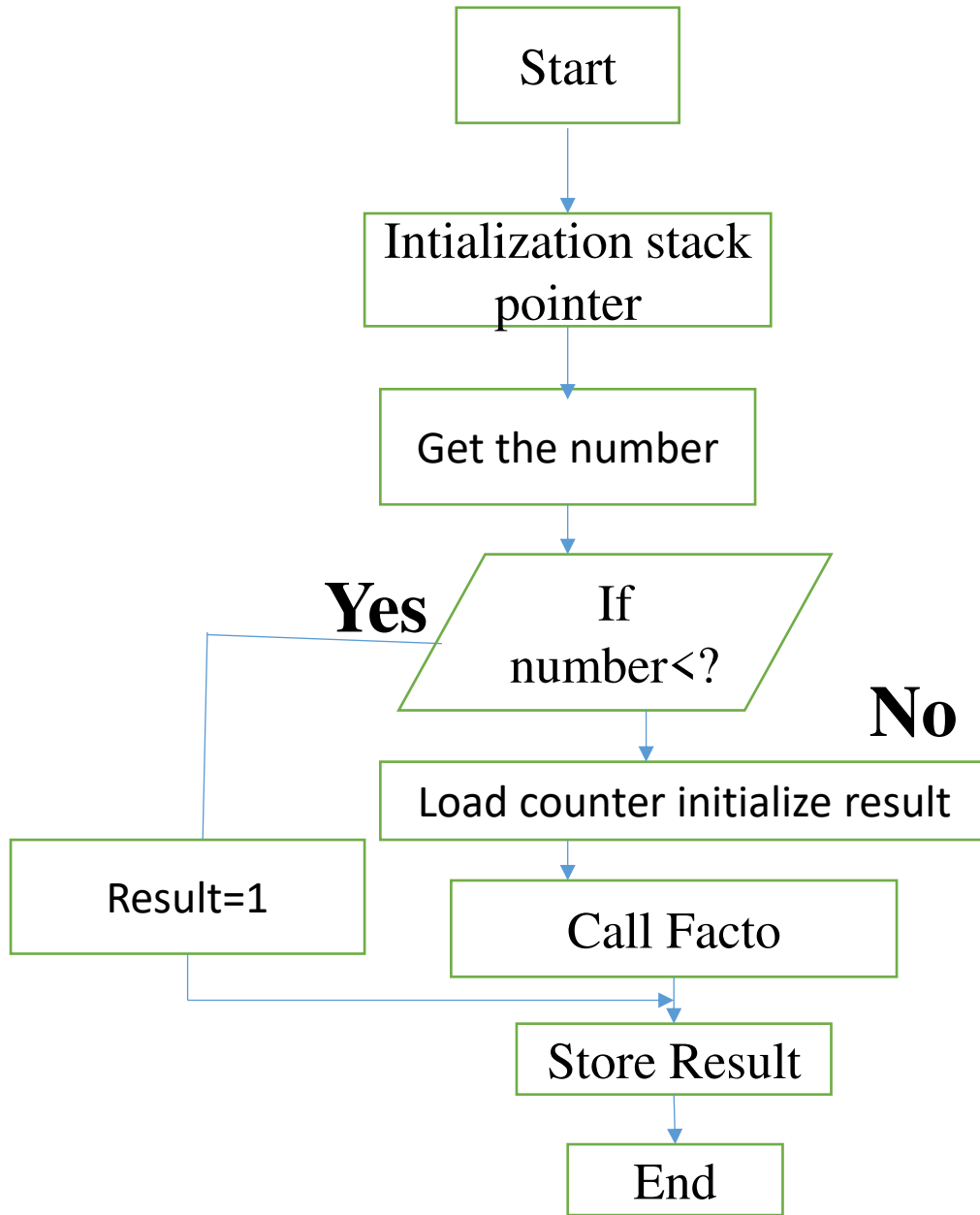
MVI D,00H           Load number as a result

MOV E,A

DCR A

MOV C,A	Load counter one less than number
CALL FACTO	Call subroutine factor
XCHG	Get the result in HL
SHLD 2201H	Store the result in memory
JMP END	
LAST LXIH,0001H	
SHLD 2201H	Store result =01
END HLT	





# **PIC MICROCONTROLLER AND ITS ARCHITECTURE**

Peripheral Interface Controller (PIC) is microcontroller developed by Microchip, PIC microcontroller is fast and easy to implement program when we compare other microcontrollers like 8051.

It is easy to interfacing with other peripherals PIC became successful microcontroller.

We know that microcontroller is an integrated chip which consists of RAM, ROM, CPU, TIMERS, COUNTERS etc.

PIC is a microcontroller which also consists of ram, rom, CPU, timers, counter, ADC (analog to digital converters), DAC (digital to analog converter).

PIC also supports the protocols like CAN, SPI, UART for interfacing with other peripherals.

PIC mainly used modified Harvard architecture and also supports RISC (Reduced Instruction Set Computer)

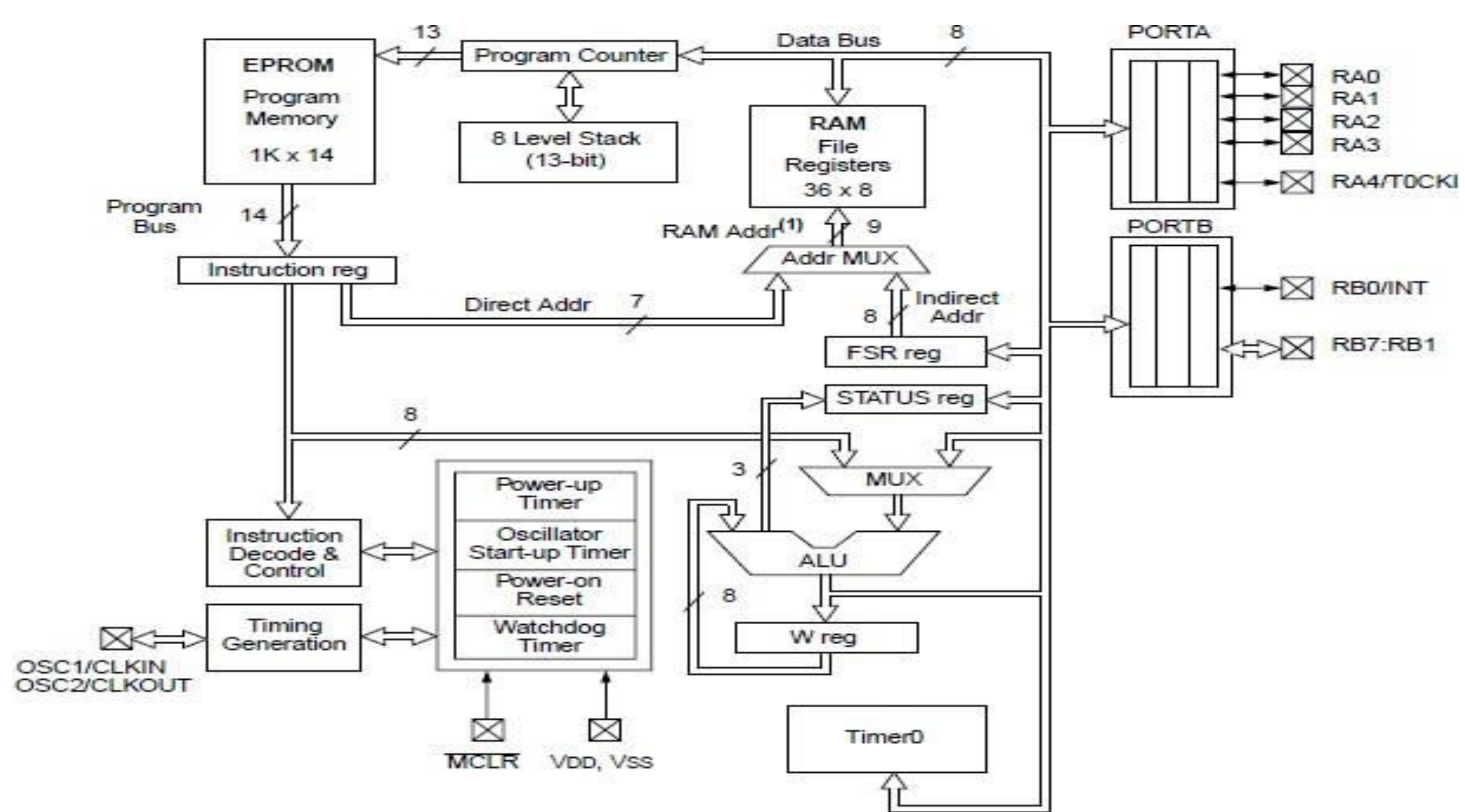
## **ARCHITECTURE OF PIC16C6X/7X**

PIC 16CXX are 8 bit microcontrollers.

The arithmetic and logic unit of these devices is 8-bit wide.

It can perform operation on temporary working register and any register file.

When the instruction is executed the ALU may affect the various flag namely Carry(C), Digital Carry(DC), Zero(Z)



## **CPU (Central Processing Unit):**

CPU is not different from other microcontrollers

PIC microcontroller's CPU consists of

Arithmetic logic unit (ALU)

Memory unit (MU)

Control unit (CU)

Accumulator

ALU mainly used for arithmetic operations and taking the logical decisions.

memory used for storing the instruction which is to be processed and also storing the instructions after processing

Control unit is used for controlling all the peripherals which are connected to the CPU both internal peripherals and external peripherals.

Accumulator is used for storing the results and used for further processing.

PIC micro controller supports the RISC architecture that is reduced instruction set computer.

- RISC has very few instructions (approx. ~ 35) which are used in the program.
- Length of the instruction is small and fixed and takes same amount of time for processing.
- As the instruction is small it will take less time to process another words CPU will be fast.
- Compiler need not be complex and debugging will be very easy in the programmer point of view.

## **W register**

W means working register .

It consist of 8-bit wide

It contain one of the source operand during the execution of instruction and may serve as the destination for the result of operation

It can be used only for ALU operation

## **STATUS REG**

0	0	RP0	TO	PD	Z	DC	C
---	---	-----	----	----	---	----	---

RP0 – Register bank select

TO-Reset status bit(Time out bit)

PD – Reset status bit(Power down bit)

Z-Zero bit

DC-Digital carry / Borrow bit

C-Carry / Borrow bit

**Carry bit**

When 8 bit operand are added a 9 bit result can occur

During subtraction if borrow occurs the carry bit is cleared otherwise carry is set

A borrow will not occur if the result of subtraction greater than zero

**Digital carry**

The bit carry from the lower 4<sup>th</sup> bit during 8 bit addition.

Example subtraction of BCD number there is borrow from 3<sup>rd</sup> or 4<sup>th</sup> bit position DC is cleared otherwise DC is set



# **DIFFERENT BLOCKS OF PIC MICROCONTROLLER**

- Power ON reset and brown out reset
- Watch dog timer
- Input/output ports
- Analog to digital conversion ADC
- Interrupt control
- USART
- EEPROM/OTP/ROM
- Program memory
- STATUS register
- Program counter stack
- FSR register

## **POWER ON RESET AND BROWN OUT RESET:**

- Power on and brown out reset make sure that the chip operate only when the supply within limit.
- In brown out reset mode, when the power supply goes below a specified voltage it reset the PIC.

## **WATCH DOG TIMER:**

- A watch dog timer reset the processor, if any malfunctions in the software program and program deviates from normal operation.

## **INPUT/OUTPUT PORTS:**

- Each port has a bidirectional capability, input and output ports are multiplexed with alternate functions for peripheral devices on the microcontroller.

## **ANALOG TO DIGITAL CONVERSION ADC:**

An ADC converts an analog signal into an equivalent digital values.

## **INTERRUPT CONTROL :**

➤ PIC microcontroller has one vectored interrupt location ( i.e. 0004H) but has 12 independent interrupt source can control, when the central processing unit will deal with each source.

There is no interrupt priority, only one interrupt is served at a time.

However interrupts can be masked.

## **USART:**

➤ A universal synchronous asynchronous receiver transmission when the serial data is transmitted asynchronously, data stream is generated with the transmitter clock.

➤ The receiver must synchronize the incoming data stream to the receiver clock.

## **Memory:**

Memory module in the PIC consists of RAM, ROM and STACK

RAM (Random Access Memory) which is a volatile memory used for storing the data temporarily in its registers.

The RAM registers is divided into 2 types.

They are General purpose registers (GPR) and Special purpose registers (SPR).

**GPR:** general purpose registers as the name implies for general usage. For example if we want to multiply any two numbers using PIC we generally take two registers for storing the numbers and multiply the two numbers and store the result in other registers.

So general purpose registers will not have any special function or any special permission, CPU can easily access the data in the registers.

## **SPR**

Special function registers are having the specific functions, when we use this register they will act according to the functions assigned to them.

They cannot be used like normal registers.

For example you cannot use STATUS register for storing the data, STATUS registers are used for showing the status of the program or operation.

User cannot change the function of the Special function register

## **ROM**

ROM (Read Only memory) is a non volatile memory used for storing the data permanently.

In microcontroller ROM will store the complete instructions or program, according the program microcontroller will act.

Rom is also called program memory in this memory user will write the program for microcontroller and save it permanently and get executed by the CPU.

According to the instruction executed by the CPU the PIC microcontroller will perform the task.

In ROM there are different types which are used in different PIC microcontrollers.

## **EEPROM**

In the normal ROM we can write the program for only one time we cannot reuse the Microcontroller for another time where as in the EEPROM (Electrically Erasable Programmable Read Only Memory) we can program the ROM for number of times.

## **Flash Memory**

Flash memory is also PROM in which we can read write and erase the program more than 10,000 times. Mostly PIC microcontroller uses this type of ROM.

## **Bus**

Bus is mainly used for transferring and receiving the data from one peripheral to another.

There are two types of buses.

### **Data Bus**

It is used to transfer/receive only the data.

### **Address Bus**

It is used to transmit the memory address from peripherals to CPU.

I/O pins are used for interfacing the external peripherals, UART and USART is serial communication protocol which is used for interfacing serial devices like GPS, GSM, IR, Bluetooth etc.



## **Advantages of PIC Microcontroller:**

- They are reliable and malfunctioning of PIC percentage is very less.
- The performance of the PIC is very fast because of using RISC architecture.
- Power consumption is also very less when compared to other micro controllers.
- The programmer point of view interfacing is very easy, also we can connect analog devices directly without any extra circuitry and use them.
- Programming is also very easy when compared to other microcontrollers.

## **Disadvantages of PIC Microcontroller:**

- The length of the program will be big because of using RISC (35 instructions).
- Program memory is not accessible and only one single accumulator is present.

# **19MZC12&19RAC12 / MICROPROCESSOR AND APPLICATIONS**

**Prepared by**

Mr.C.RAMKUMAR,

Assistant Professor,

Department of Electrical and Electronics Engineering,

Muthayammal Engineering College (Autonomous),

Rasipuram – 637 408.

# **UNIT – IV PROGRAMING AND INTERFACING OF 8085 AND 8051**

## **Programmable peripheral interface (8255)**

### **Architecture of 8255**

The parallel input-output port chip 8255 is also called as programmable peripheral input- output port.

The Intel's 8255 is designed for use with Intel's 8-bit, 16-bit and higher capability microprocessors.

It has 24 input/output lines which may be individually programmed in two groups of twelve lines each, or three groups of eight lines.

The two groups of I/O pins are named as Group A and Group B.

Each of these two groups contains a subgroup of eight I/O lines called as 8-bit port and another subgroup of four lines or a 4-bit port.

Thus Group A contains an 8-bit port A along with a 4-bit port. C upper.

The port A lines are identified by symbols PA0-PA7 while the port C lines are identified as PC4-PC7.

Similarly, Group B contains an 8-bit port B, containing lines PB0-PB7 and a 4-bit port C with lower bits PC0- PC3.

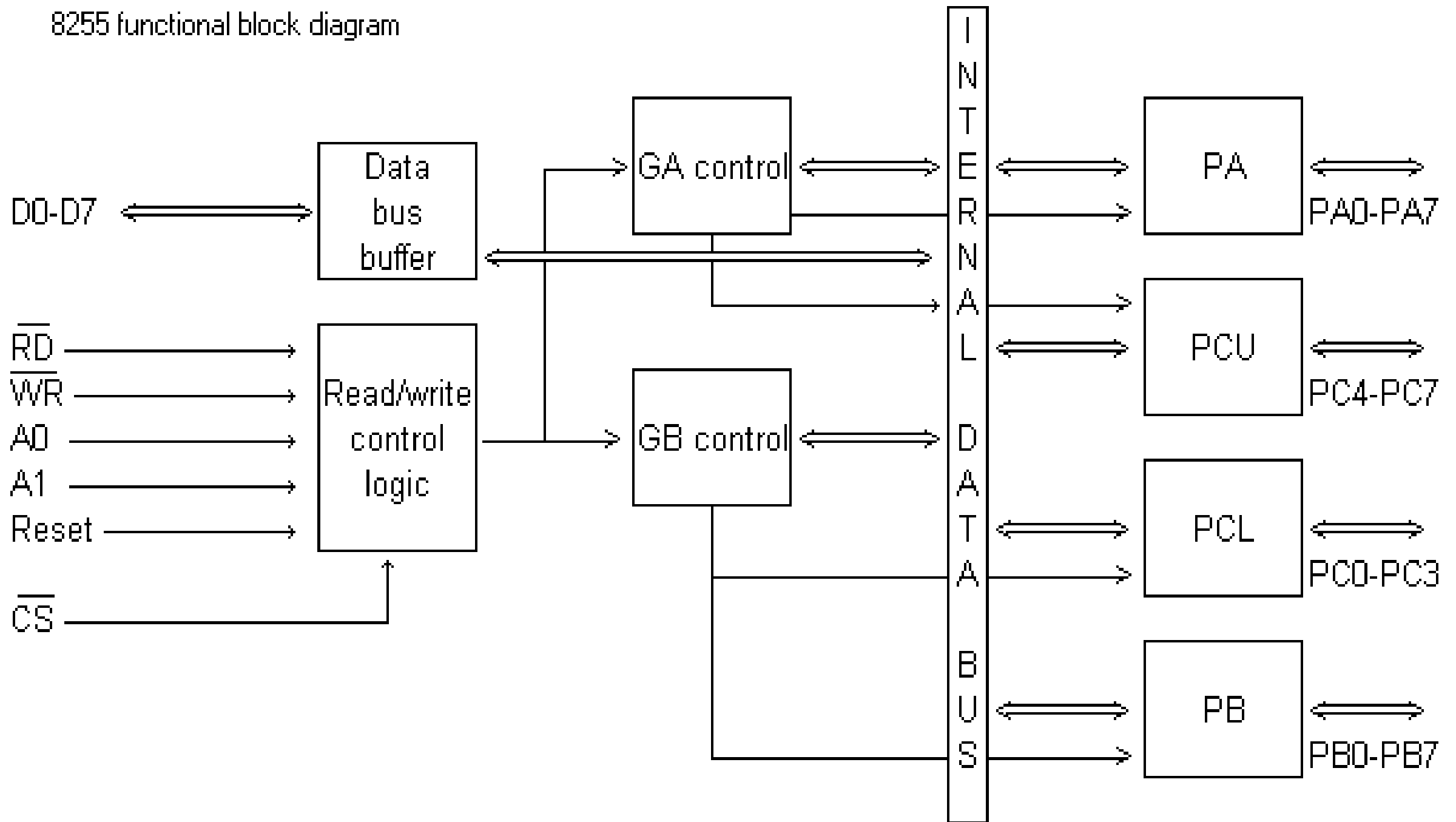
The port C upper and port C lower can be used in combination as an 8-bit port C.

Both the port C are assigned the same address.

Thus one may have either three 8-bit I/O ports or two 8-bit and two 4-bit ports from 8255.

All of these ports can function independently either as input or as output ports.

8255 functional block diagram

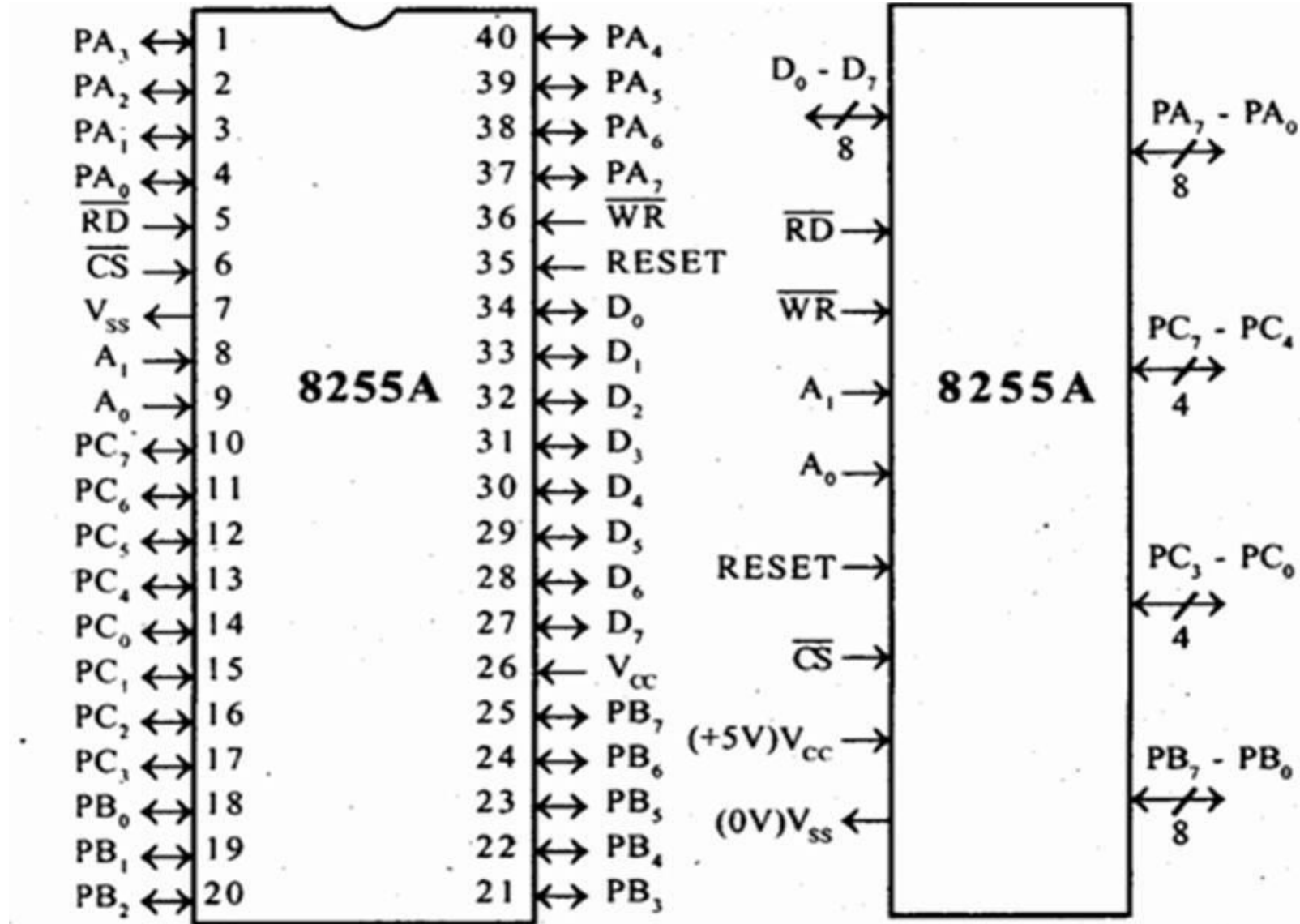


This can be achieved by programming the bits of an internal register of 8255 called as control word register (CWR).

This buffer receives or transmits data upon the execution of input or output instructions by the microprocessor.

The control words or status information is also transferred through the buffer.

# Pin Diagram of 8255



The signal description of 8255 are briefly presented as follows:

### **PA7-PA0**

These are eight port A lines that acts as either latched output or buffered input lines depending upon the control word loaded into the control word register.

### **PC7-PC4**

Upper nibble of port C lines. They may act as either output latches or input buffers lines. This port also can be used for generation of handshake lines in mode 1 or mode 2.

### **PC3-PC0**

These are the lower port C lines, other details are the same as PC7-PC4 lines.

### **PB0-PB7**

These are the eight port B lines which are used as latched output lines or



buffered input lines in the same way as port A.

## **RD**

This is the input line driven by the microprocessor and should be low to indicate read operation to 8255.

## **WR**

This is an input line driven by the microprocessor. A low on this line indicates write operation.

## **CS**

This is a chip select line. If this line goes low, it enables the 8255 to respond to RD and WR signals, otherwise RD and WR signal are neglected.

## **A1-A0**

These are the address input lines and are driven by the microprocessor.

These lines A1-A0 with RD, WR and CS from the following operations for 8255.

These address lines are used for addressing any one of the four registers, i.e. three ports and a control word register

In case of 8086 systems, if the 8255 is to be interfaced with lower order data bus, the A0 and A1 pins of 8255 are connected with A1 and A2 respectively.

## **D0-D7**

These are the data bus lines those carry data or control word to/from the microprocessor.

## **RESET**

A logic high on this line clears the control word register of 8255. All ports are set as input ports by default after reset.

## **Operational Modes of 8255**

There are two main operational modes of 8255:

Input/output mode

Bit set/reset mode

### **Input / Output Mode**

There are three types of the input/output mode. They are as follows:

#### **Mode 0**

In this mode, the ports can be used for simple input/output operations without handshaking

If both port A and B are initialized in mode 0, the two halves of port C can be either used together as an additional 8-bit port, or they can be used as individual 4-bit ports.

Since the two halves of port C are independent, they may be used such that one-half is initialized as an input port while the other half is initialized as an output port.

The input output features in mode 0 are as follows:

O/p are latched.

I/p are buffered not latched.

Port do not have handshake or interrupt capability.

## Mode 1

When we wish to use port A or port B for handshake (strobed) input or output operation, we initialize that port in mode 1 (port A and port B can be initialized to operate in different modes, ie, for eg., port A can operate in mode 0 and port B in mode 1).

Some of the pins of port C function as handshake lines.

For port B in this mode (irrespective of whether is acting as an input port or output port), PC0, PC1 and PC2 pins function as handshake lines.

If port A is initialized as mode 1 input port, then, PC3, PC4 and PC5 function as handshake signals.

Pins PC6 and PC7 are available for use as input/output lines.

The mode 1 which supports handshaking

Two ports i.e. port A and B can be used as 8-bit I/O port.

Each port uses three lines of port c as handshake signal and remaining two signals can be function as I/O port.

Interrupt logic is supported.

Input and Output data are latched.

## **Mode 2**

Only group A can be initialized in this mode. Port A can be used for bidirectional handshake data transfer.

This means that data can be input or output on the same eight lines (PA0 – PA7).

Pins PC3 - PC7 are used as handshake lines for port A.

The remaining pins of port C (PC0 - PC2) can be used as input/output lines if group B is initialized in mode 0.

In this mode, the 8255 may be used to extend the system bus to a slave microprocessor or to transfer data bytes to and from a floppy disk controller.

### **Bit Set/Reset (BSR) mode**

In this mode only port b can be used (as an output port).

Each line of port C (PC0 - PC7) can be set/reset by suitably loading the command word register no effect occurs in input-output mode.

The individual bits of port c can be set or reset by sending the signal OUT instruction to the control register.

# **Programmable Interrupt Controller (8259)**

## **Features**

8 levels of interrupts.

Can be cascaded in master-slave configuration to handle 64 levels of interrupts.

Internal priority resolver.

Fixed priority mode and rotating priority mode.

Individually maskable interrupts.

Modes and masks can be changed dynamically.

Accepts IRQ, determines priority, checks whether incoming priority  $>$  current level being serviced, issues interrupt signal.



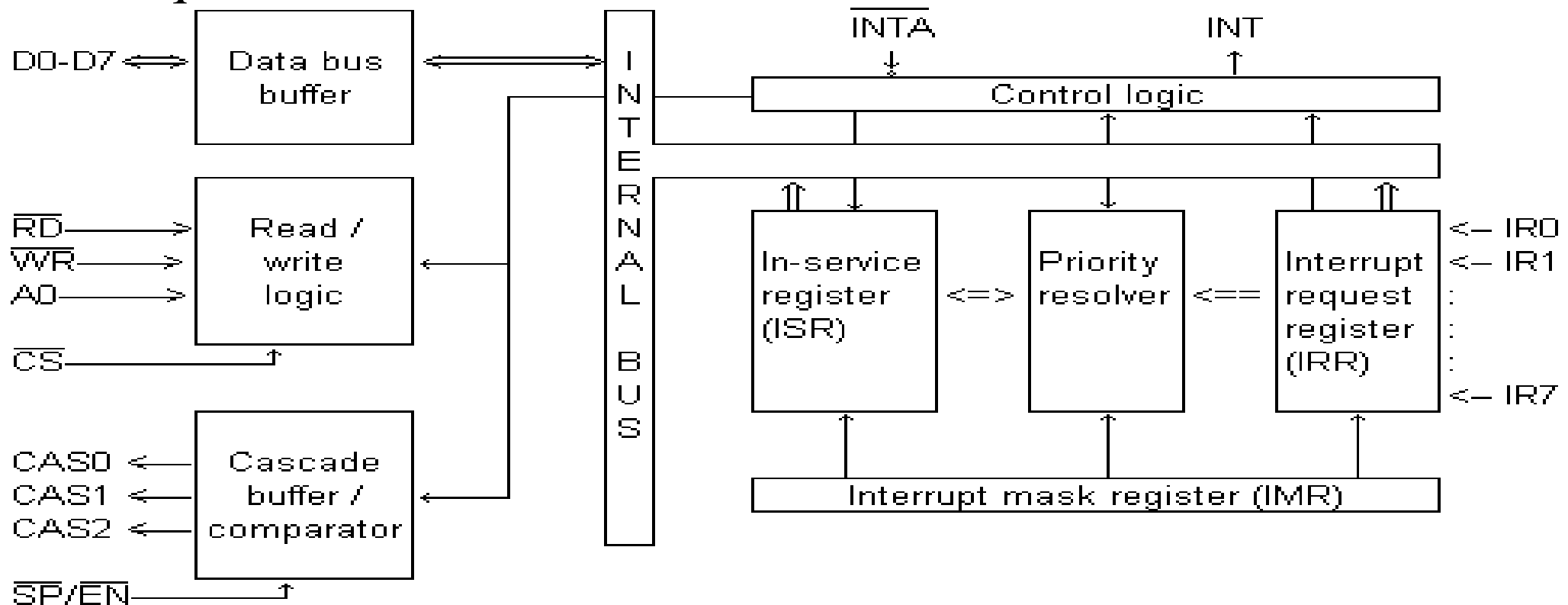
In 8085 mode, provides 3 byte CALL instruction.

In 8086 mode, provides 8 bit vector number.

Polled and vectored mode.

Starting address of ISR or vector number is programmable.

No clock required.



# Pinout

$\overline{CS}$	1	8259 PIC	28	Vcc
$\overline{WR}$	2		27	A0
$\overline{RD}$	3		26	$\overline{INTA}$
D7	4		25	IR7
D6	5		24	IR6
D5	6		23	IR5
D4	7		22	IR4
D3	8		21	IR3
D2	9		20	IR2
D1	10		19	IR1
D0	11		18	IR0
CAS0	12		17	INT
CAS1	13		16	$\overline{SP/EN}$
gnd	14		15	CAS2

## **IRR**

The interrupt at IRQ input lines are handled by Interrupt request internally IRR stores all the interrupt request it is in order to serve them one by one on the priority basis

## **IR0-IR7**

This pin act as input to accept interrupt request to the CPU. In edge triggered mode an interrupt service it is requested by raising and IR pin as Low to a high state

## **ISR**

This stores all the interrupt request those are being served ISR keep a track of the requests being served

## **Priority Resolver**

This unit determines the priorities of the interrupt requests appearing simultaneously.

The highest is selected and stored into selected and stored into the corresponding bit of ISR

## **IMR**

This register store the bit required to mask the interrupt inputs

## **Interrupt Control Logic**

This block manages the interrupt and interrupt acknowledgement signal to be sent to the CPU for serving one of the eight interrupt request

## **INTA-bar**

This pin is an input used to strobe in 8259 interrupt vector on to the data bus

## **Data bus buffer**

This tristate bidirectional buffer interface internal 8259 bus to the microprocessor system data bus.

## **D0-D7**

This pin from a bidirectional data bus that carries 8 bit data either to control word or from status word register

## **Read/Write control register**

This circuit accepts and decodes command from CPU

## **CS**

This is an active low chip select signal for enabling RD and WR operation

## **WR**

This pin is an active low write enable input to 8259

## **RD**

This pin is an active low read enable input to 8259

## **Cascade buffer / Comparator**

This blocks store and compare the ID of all the 8259 used in the microprocessor system.

The three i/o pins CASO-2 are output when the 8259 is used as master

## **CAS0-CAS2**

A signal 8259 provides eight vectored inputs.

If more interrupts are required the 8259 is used in cascaded mode

## **PS/EN**

This pin is a dual purpose pin .When chip is used in buffered mode it can be used as buffered enable to control buffer

## **INT**

This pin goes high whenever a valid interrupt request is asserted.This is used to interrupt the CPU and is connected to the interrupt input to the CPU

## **Interrupt sequence in an 8085 system**

One or more IR lines are raised high that set corresponding IRR bits

8259A resolves priority and sends an INT signal to CPU

The CPU acknowledgement with INTA pulse

Upon receiving INTA signal from the CPU the highest priority ISR bit is set and the corresponding IRR bit is reset.

8259A does not drive data during this period

## **Command word of 8259**

The command word of 8259 classified into two groups

Initialization command word(ICW)

Operation Command word(OCW)

Initialization command word(ICW)

Before it starts functioning the 8259 must be initialized by writing two or four command word into respective command word register

These are called as Initialization command word

The initialization command word can be defined into two types of

Initialization command word 1

Initialization command word 2

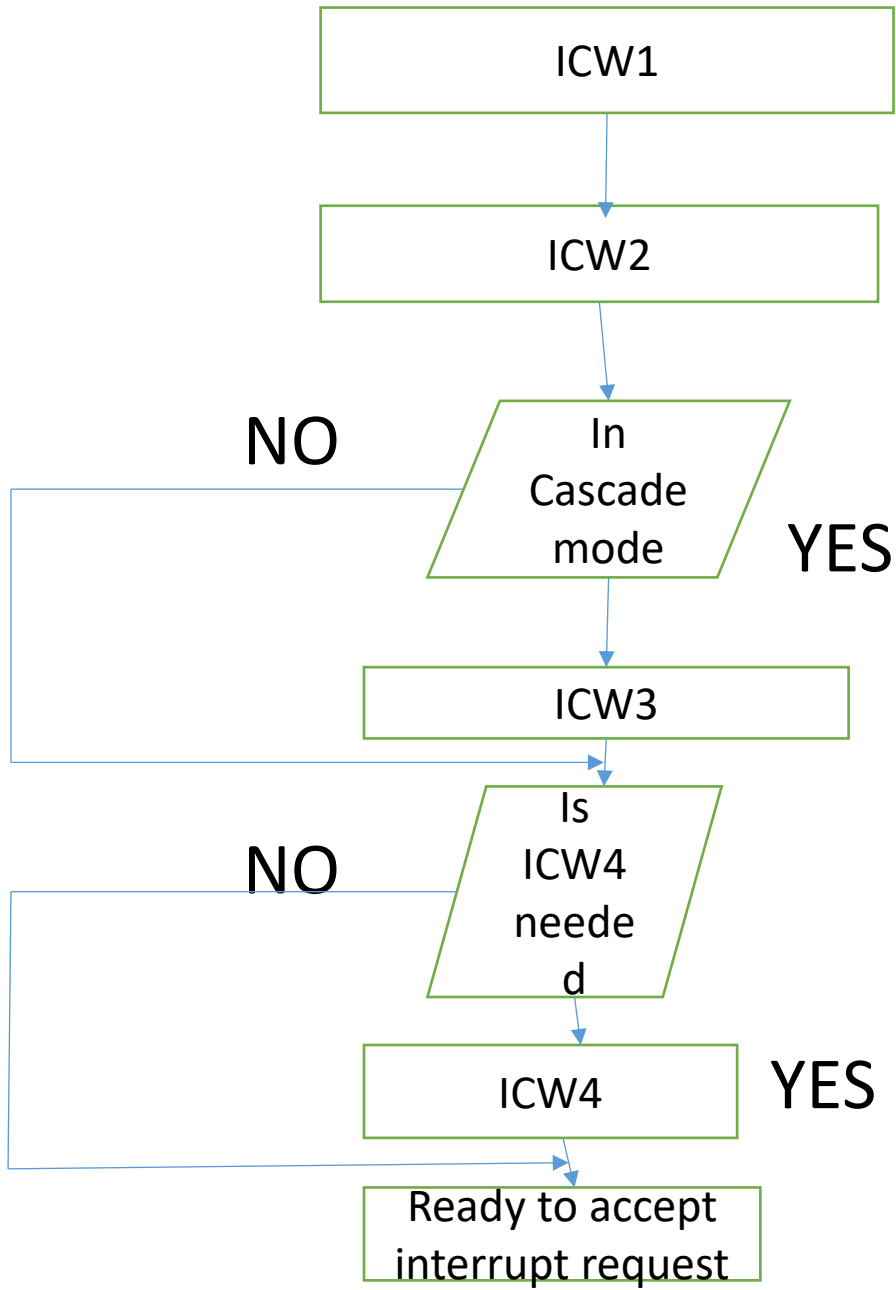


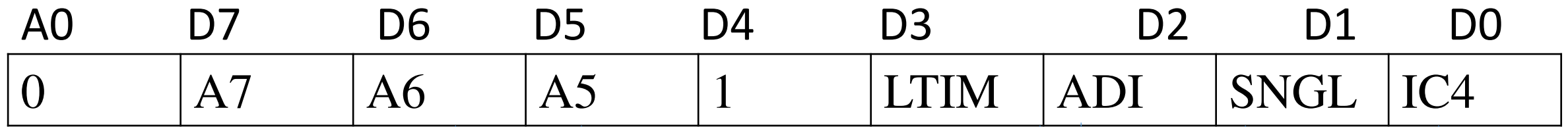
## **Initialization command word 1(ICW1)**

If A0=0 and D4=1 the control word is recognized as ICW1. It contains control bits for edge/level triggered mode

## **Initialization command word 2(ICW2)**

If A0=1 the control word is recognized as ICW2. The ICW2 stores the details regarding interrupt vector address





ICW1

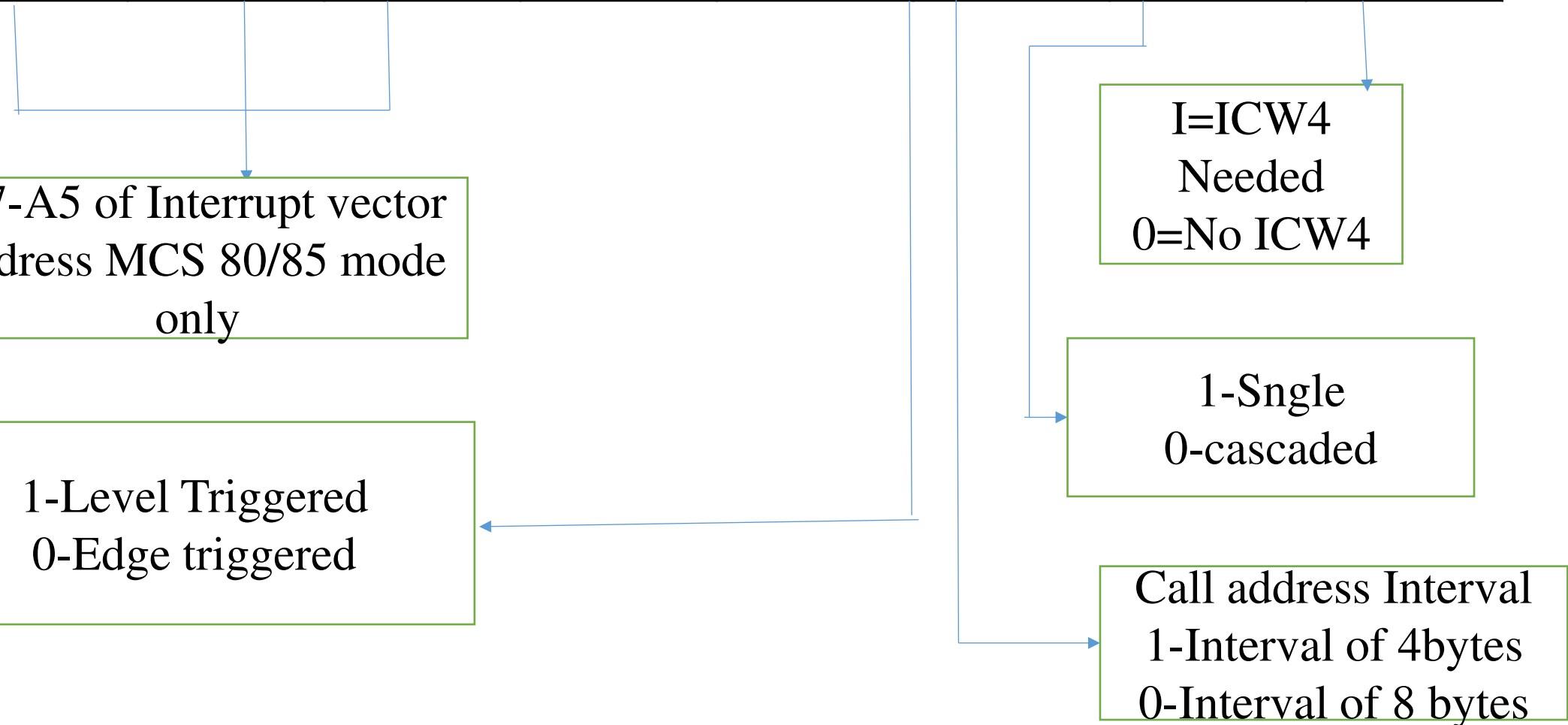
A7-A5 of Interrupt vector address MCS 80/85 mode only

1-Level Triggered  
0-Edge triggered

I=ICW4  
Needed  
0=No ICW4

1-Sngle  
0-cascaded

Call address Interval  
1-Interval of 4bytes  
0-Interval of 8 bytes



## ICW2

A0            D7            D6            D5            D4            D3            D2            D1            D0

1	T7	T6	T5	T4	T3	A10	A9	A8
---	----	----	----	----	----	-----	----	----

## ICW3

A0            D7            D6            D5            D4            D3            D2            D1            D0

1	S7	S6	S5	S4	S3	S2	S1	S0
---	----	----	----	----	----	----	----	----

## ICW4

A0            D7            D6            D5            D4            D3            D2            D1            D0

1	0	0	0	SFNM	BUF	M/S	AEOI	PM
---	---	---	---	------	-----	-----	------	----

## **Operating modes of 8259**

The different modes of operation can be programmed by setting or resetting the appropriate bits of ICW or OCW

The different modes of operation are

1. Fully nested mode
2. End of Interrupt
3. Automatic Rotation
4. Automatic EOI Mode
5. Specific Rotation
6. Special Mask Mode
7. Edge and Level Triggered mode
8. Reading 8259 Status

## **End of Interrupt**

The ISR bit can be reset either with AEOI bit of ICW1 or by EOI command issued before returning from the interrupt service routine

There are two types of EOI command

1. Specific command

2. Non-specific command

When 8259 can be operated in the fully nested mode it can determine the ISR bit

is to be reset on EOI

When non-specific EOI command is issued to 8259 it will automatically reset the highest ISR bit

## **Automatic Rotation**

This used in the application of all the interrupt device are of equal probability

In this mode an Interrupt request level receives the priority after it served at the time next device to be served gets the highest priority sequence

## **Automatic EOI Mode**

Till AEOI = 1 in ICW4 the 8259 operates in AEOI mode.

This mode only used in nested multi level interrupt structure.

## **Specific Rotation**

In this mode at bottom priority level can be selected using L2,L1 and L0.

The selected bottom priority can fixes other priority

## **Special Mask mode**

In special mask mode when a mask bit is set in OCW1 it inhibit further interrupt at the level and enable the interrupt from other level

## **Edge and Level triggered**

This mode decides whether the interrupt should be edge triggered or level triggered. If the bit of ICW1=0 they are edge triggered otherwise the interrupt are level triggered

## **Reading of 8259 Status**

The status of internal register of 8259 can be read using this mode..OCW3 can read IRR and ISR while OCW1 is used to read IMR



9.Poll Command

10.Specially Fully Nested Mode

11.Buffered mode

12.Cascaded mode

### **Fully Nested Mode**

This is the default mode of operation of 8259.

IR0 has the higher priority and IR7 has the lowest one.

When the interrupt priority like the highest priority request is determined and the vector is placed on the data bus

If the ISR bit is set all the priority interrupts are inhibited but higher level priority will generate an interrupt then the interrupt can be acknowledge only if the microprocessor interrupt enable the flag IF is set

# **8251 UNIVERSAL SYNCHRONOUS ASYNCHRONOUS RECEIVER TRANSMITTER (USART)**

The 8251 is a USART (Universal Synchronous Asynchronous Receiver Transmitter) for serial data communication.

As a peripheral device of a microcomputer system, the 8251 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.

## **Need for PCI**

The three modes of data transmission are

Simplex

Duplex

Half duplex

**Simplex**

In this mode data can be transmitted by in one direction over a single communication channel

**Duplex**

In this mode data can be transferred between two transceivers in both directions simultaneously

**Half duplex**

In this mode data transmission take place in either direction but at a time data may be transmitted only in one direction

## **Features**

It is a universal synchronous and asynchronous Communication Controller

It support the standard protocols like

5 to 8 bit character format

Odd even or no parity generation

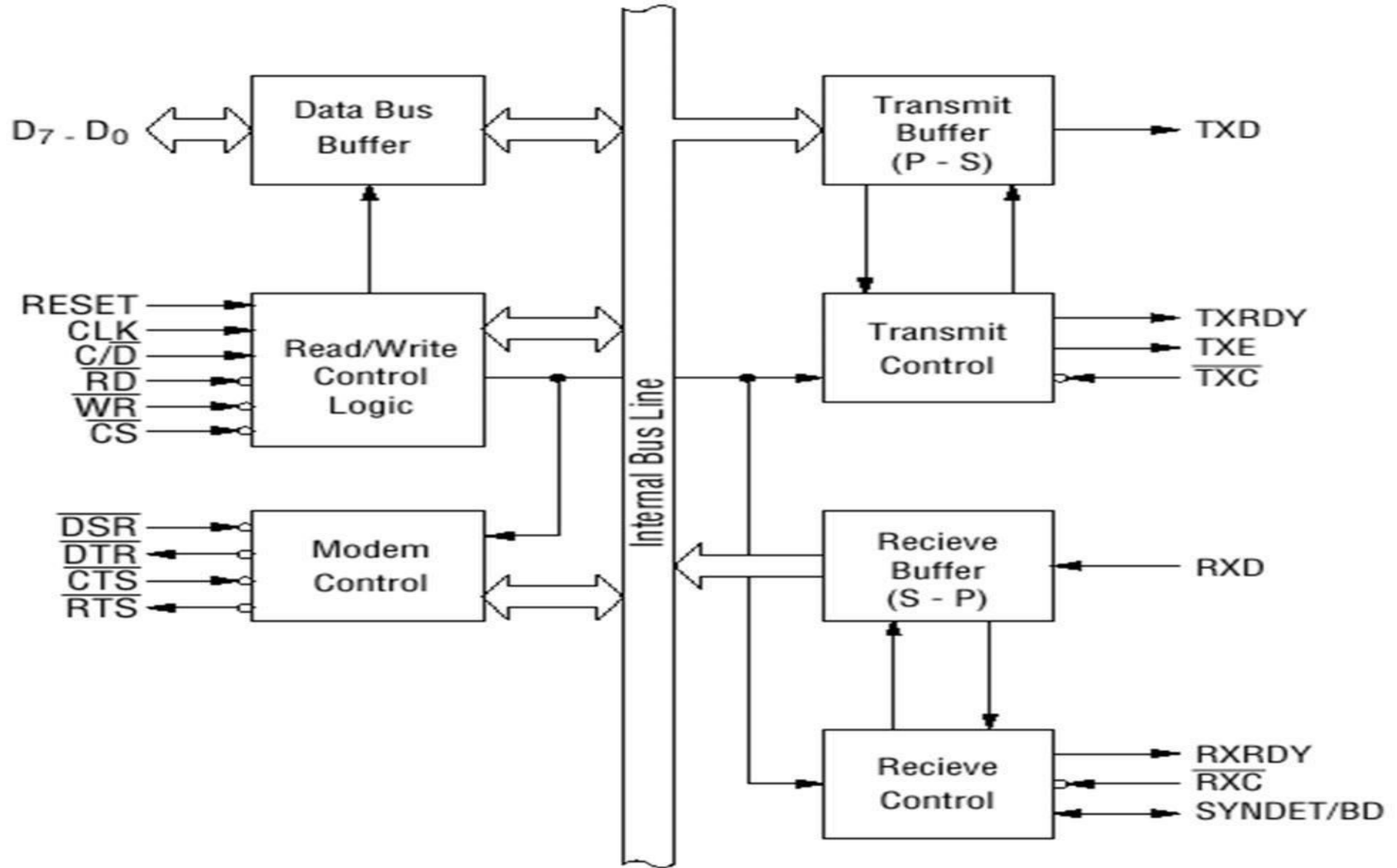
Baud rate from DC to 19.2k baud

False start bit detection

Break character generation

It allow full duplex transition

# Block diagram



## **Data bus buffer**

It interfaces the internal bus of 8251 with the system bus

### **D0-D7**

This is 8 bit data bus used to read or write status command word or data from or to the 8251

## **Read/Write Control logic**

It controls the operation of the peripheral depending upon the operation initiated by CPU

### **C/D control word data**

This pin together with RD and WR inputs inform 8251 that the word on the data bus is either a data or control word / status information

## **RD**

This is active low signal and it inform 8251 that the CPU is reading either data or status information from internal register

## **WR**

This is active low signal and its inform 8251 that the CPU is reading either data or control word to 8251

## **CS**

It is an active low select input

If it is high no read or write operation can be carried out on 8251

## **CLK**

It is used to generate internal device timing and is connected to the clock generator output

## **RESET**

A high on this input forces 8251 into an idle state and remains idle till it is low again and a new control word

## **Modem Control**

It handles the modem handshake signal to coordinate the communication between the modem and USART

## **DSR**

It is used to check if the data set is ready when communicating with a modem

## **DTR**

It is used to indicate that the device is ready to accept data when 8251 is communicating with a modem



## **RTS**

It indicates the modem that the receiver is ready to receive data byte from the modem

## **CTS**

If this low the 8251 is enabled to transmit the serial data

## **Transmit control**

It transmits the data byte received by the data buffer from the CPU for further serial communication

## **TXC**

It controls the rate at which the character is to be transmitted

## **TXRDY**

It indicates CPU that the transmit buffer is ready to accept a new character for the transmission from the CPU

## **TXE**

When high it indicates that transmit buffer does not have any character to transmit

## **Transmit buffer**

It receives parallel data and converter is it to serial data

## **TXD**

This carries serial data bits along with other information like start, stop bit and parity bit

## **Receiver buffer**

It converts serial data to parallel data gives to the microprocessor

## **RXD**

It receives the composite stream of data to be received by 8251

## **Receiver control**

This decided the receiver frequency as controlled by RXC input frequency

### **RXRDY**

It indicates that 8251 contains a character to be ready

### **RXC**

This controls the rate at which the character to be received

## **8251 Operating modes**

8251 can be programmed to operate in its various mode using the mode control

A set of control word may be written in the internal register

Once 8251 is programmed as required the TXRDY is raised high to signal of CPU

8251 is ready to receive data byte from it to be converted serial format and transmitted

In receiver mode 8251 receives serial data byte from modem or I/O device

After receiving the data byte the RXRDY signal is raised high to inform the CPU

The RXRDY signal is automatically reset after the CPU read the received Byte

The control word of 8251 are divided into

Mode Instruction control word

Command word instruction control word

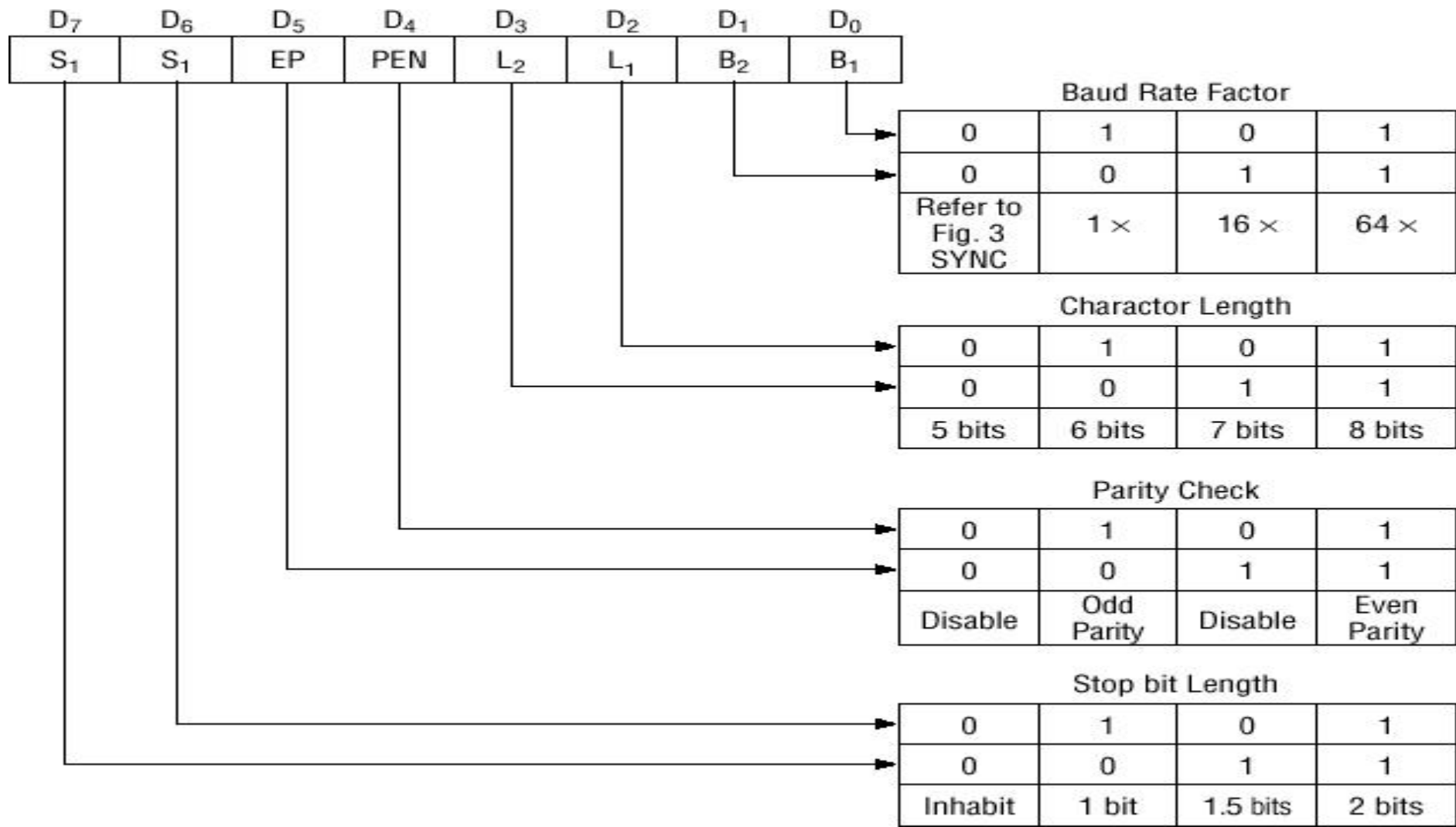
## **Mode Instruction control word**

Mode Instruction control word define the general operation characteristics of the 8251

After the internal or external reset must be written to configure the 8251

Once written to the configure 8251 SYNCH command instruction can be Programmed

To change the mode of operation from synchronous to asynchronous



## **Asynchronous mode(Transmission)**

When a data character is sent to 8251 by the CPU it starts the bits prior to the serial data bit followed by optional parity bit and stop bit using the asynchronous mode instruction control

This sequence is transmitted by using TXD output pin

When no data character are sent by the CPU to 8251

## **Asynchronous mode(Receive)**

An RXD input line marks a start bit.

A baud rate of 16X and 64X this start bit is again checked at the center of the start bit pulse and if detect the low it is a valid start bit and the bit counter starts counting

The bit counters can locate at the data bit , parity bit and stop bit.

If a low level can be detected in the stop bit the framing error flag is set

RXRDY pin is then realized high to indicate to the CPU that the character is ready

### **Synchronous mode(Transmission)**

The TXD output is high until the CPU send a character to 8251 .

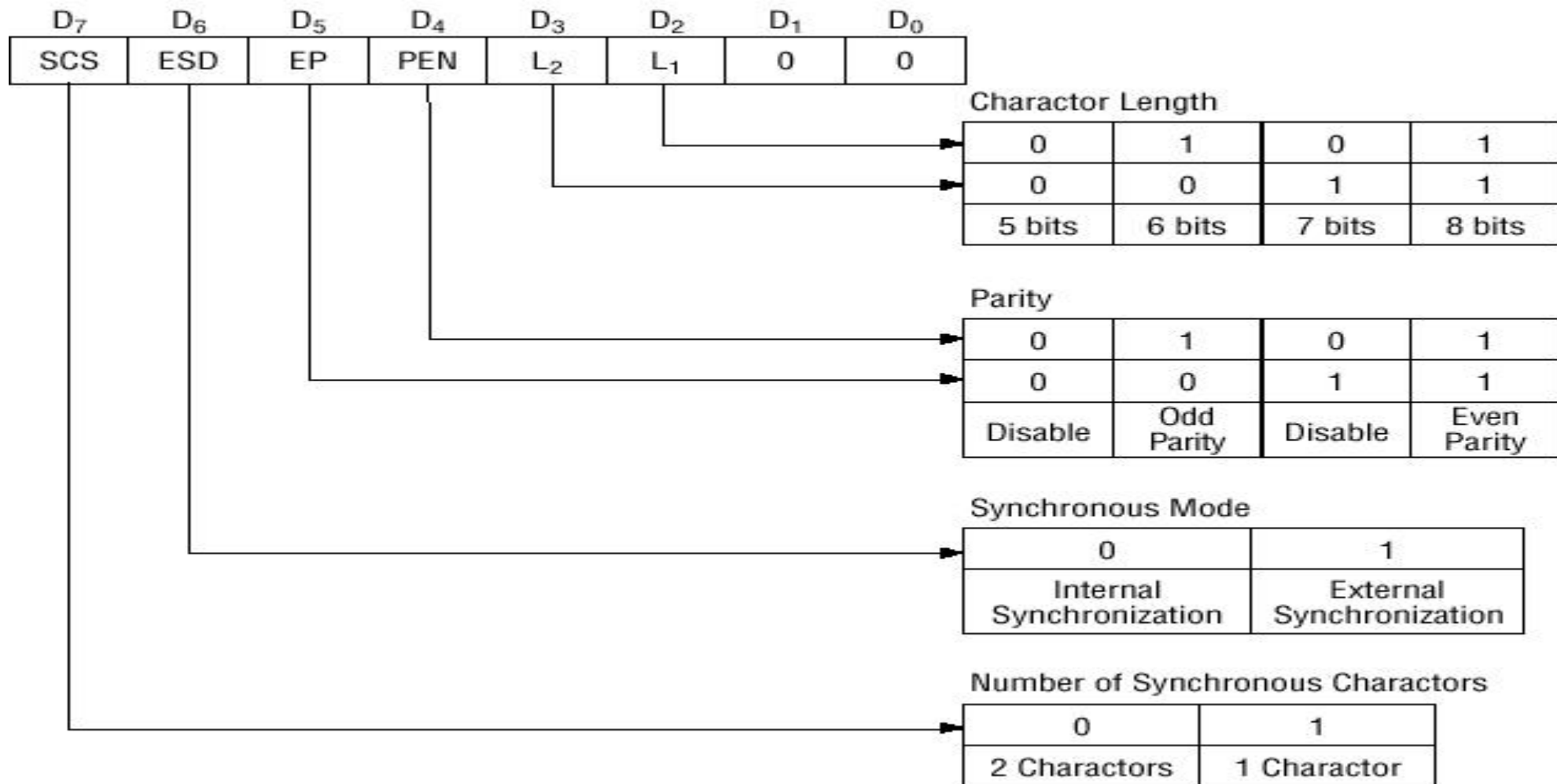
When CTS line goes low the first character is serially transmitted out

### **Synchronous mode(Receiver)**

In this mode character synchronization can be achieved internally or externally

If this mode is programmed then ENTER HUNT command should be included in the first command instruction word written into CPU





# Command instruction Format



**EN** If 1 HUNT for SYNC Character

**IR** Internal reset high forces 8251 to mode instruction format

**RTS** Request to send

If 1 RTS =0

**ER** Reset error flag PE , OE , FE if ER =1

**SBRK** Send Brake character

1-TxD force 0

0-Normal operation

**RXE** Receive Enable

1-Enable 0-Disable

**DTR** Data Terminal Ready

If 1 DTR =0

**TXEN** 1-Transmit enable

0-Disable

### **Status Read Instruction format**

<b>DSR</b>	<b>SYNDET</b>	<b>FE</b>	<b>OE</b>	<b>PE</b>	<b>TxEMPTY</b>	<b>ExRDY</b>	<b>TxRDY</b>
------------	---------------	-----------	-----------	-----------	----------------	--------------	--------------

It is used by CPU to read the status of the active 8251 to conform if any error condition or other condition like the requirement of processor service has been deleted during the operation

**DSR** This indicates that DSR is at zero level

**FE Framing error** Framing error flag is set when a valid stop bit is not detected at the end of every character and reset by ER command instruction

**OE Over run Error**

This flag is set when the CPU does not read a character before the next one becomes available and is reset by command instruction

**PE Parity error**

This is set if error detected . Reset by ER bit of command instruction

**TxRDY** This pin indicates that USART is ready to accept a data character or command

## **8254 PROGRAMMABLE INTERVAL TIMER**

The 8254 is a programmable interval timer/counter specially designed for use in real time applications for timing and counting function such as binary counting , generation of accurate time delay

### **FEATURES**

Compatible with all Intel Microprocessor

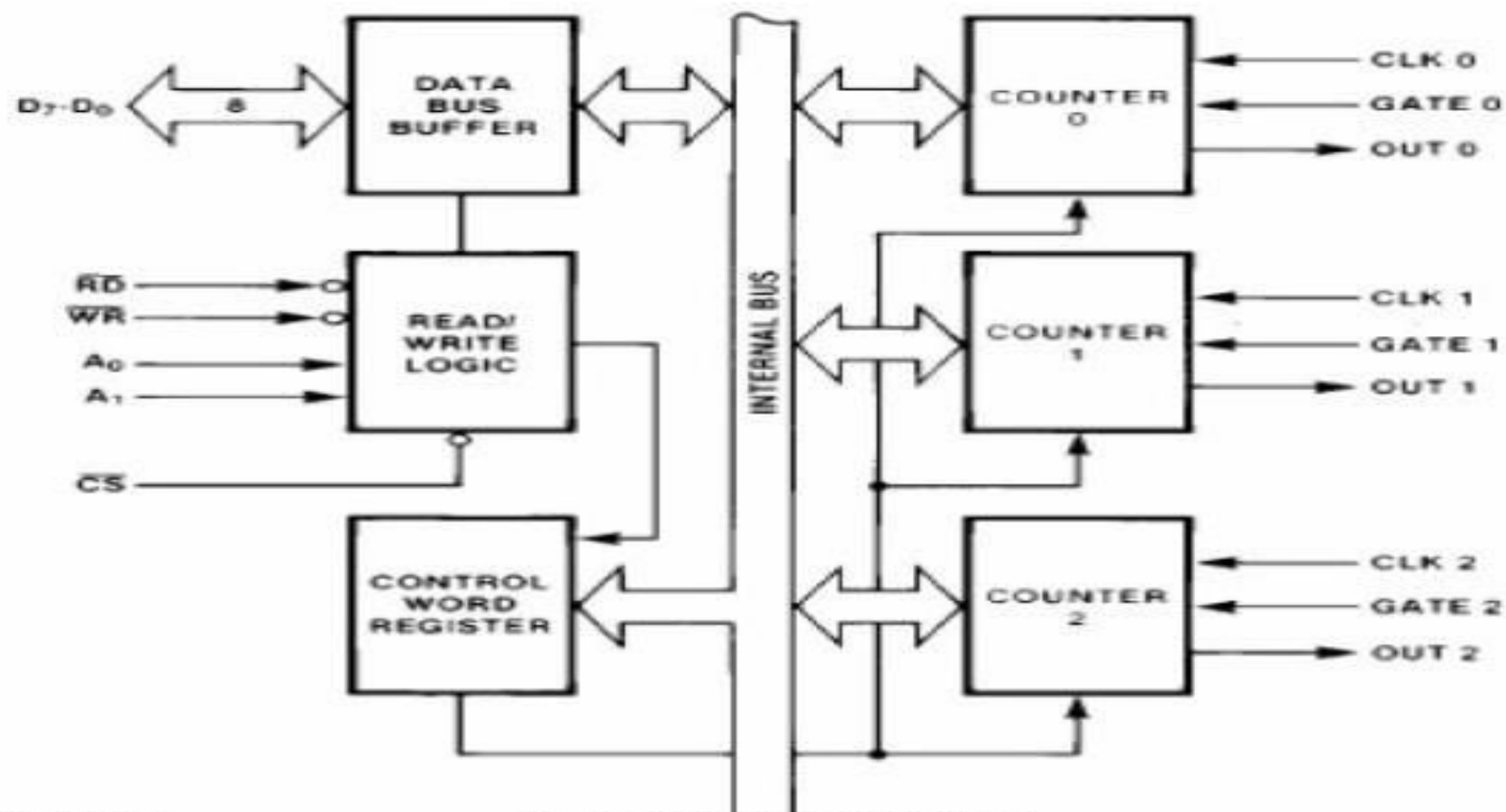
Handles input from DC to 10 MHz

Clock frequency is 10MHz

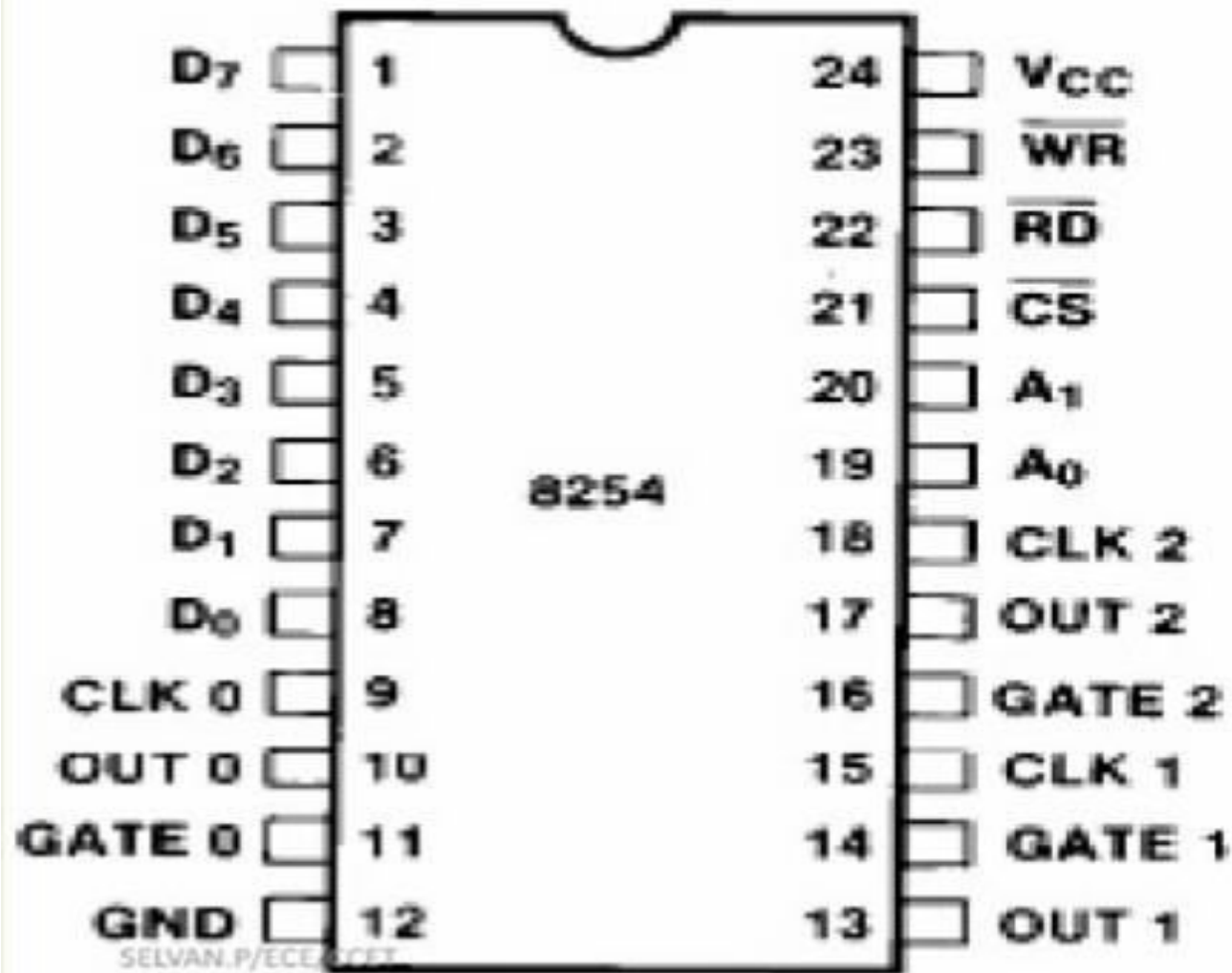
Status Read Back command

Three Independent 16-Bit counter

## 8254 Block diagram



# PIN DIAGRAM



The 8254 is a programmable interval timer / counter designed for use with Intel microcomputer system

In general multi-timing element that can be treated as an array of I/O ports in the system software

It consist of three identical counting circuits each of which has CLK and GATE inputs and OUT outputs

It solves the most common problem in any microcomputer system the generation of accurate time delay under software control

8254 to match the requirement and program one of the counter for the desired Delay

After the desired delay 8254 will interrupt the CPU



Some of other counter/timer function common to microcomputer which can be implemented with the 8254 are

Real time clock

Event-counter

Programmable rate generator

Square wave generator

Complex waveform generator

## **DATA BUS BUFFER**

This is 3-state bidirectional 8-bit buffer is used to interface the 8254 to the system bus

## **READ/WRITE LOGIC**

The read/write logic accepts the input from the system bus and generate the

Control signal for the other functional block of the 8254

A1 and A0 select one of the three counter or the control word register to be read /write

A low on the RD input tells the 8254 that the CPU is reading one of the counter

A low on the WR input tells the 8254 that the CPU is writing either a control word or an initial count

## **CONTROL WORD REGISTER**

The control word is selected by the read/write logic when A1,A0 =11

CPU does a write operation to the 8254 the data is stored in the control word register

Control word is used to define the operation of the counters

Control word register is used only for written cannot be used for read

## **COUNTER 0,COUNTER 1,COUNTER 2**

These three functional blocks are identical in operations so only single counter are described

The counters are fully independent .

Each counter may operate in a different mode

Each counter has 3 logical line CLOCK,GATE and OUT

CLOCK and GATE are input signal and OUT are output signal

The function of these lines changes depending on how the device is initialized

Gate signal act as a start pulse depending on the mode of operation of the counter

Clock act as clock input to the counter

The counter can easily read by the CPU

Read/Write control logic signal

Counter	RD	WR	A0	A1	Function
counter0	1	0	0	0	Load counter 0
	0	1	0	0	Read counter 0
Counter 1	1	0	0	1	Load counter 1
	0	1	0	1	Read counter 1
Counter 2	1	0	1	0	Load counter 2
	0	1	1	0	Read counter 2
control word	1	0	1	1	Write control word register

Read/Write control supports five control signal RD,WR,A0,A1 and CS

Address line A0 and A1 selects counter0 or counter1 or counter2 or the control word register

Based on the RD or WR signals are enabled at the particular time determined the selected counter or control word register to perform read or write operation

### **Control word format**

<b>D7</b>	<b>D6</b>	<b>D5</b>	<b>D4</b>	<b>D3</b>	<b>D2</b>	<b>D1</b>	<b>D0</b>
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

# SC-Select Counter

SC0 SC1

0	0	Select counter 0
0	1	Select counter 1
1	0	Select counter 2
1	1	Read back command

## BCD

0	Hexadecimal count
1	Binary coded decimal

M2 M1 M0

0	0	0	Mode 0
0	0	0	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	0	Mode 5

RL0 RL1

0	0	Counter latch
0	1	Read/Write least significant bytes only
1	0	Read/Write most significant bytes only
1	1	Read/Write least significant bytes only first then most significant byte

## **Read Back Command**

This command allows the user to check the count value

## **Operating modes of 8254**

Each of the three counters of 8254 can be operated in one of the following six modes of operation

Mode 0 : Interrupt of Terminal Count TC

Mode 1 : Programmable One shot

Mode 2 : Rate generator

Mode 3 : Square wave generator

Mode 4 : Software Triggered Strobe

Mode 5 : Hardware Triggered Strobe



## **Mode 0**

In this mode output is initially low after the mode is set

The OUT remains low even after the count value is loaded in the counter

Counter starts decrementing the count value after the falling edge of the clock if the GATE input is high

The counting process continues till the terminal count is reached

When terminal count TC is reached OUT goes high

The high OUT signal can be used to interrupt the processor

If the GATE signal is high at the time the counting is high

If the GATE signal is low counting is terminated

## Mode 1

This mode of operation is called programmable one shot

8254 can be used as a monostable multivibrator

The monostable multivibrator is decided by count loaded into count register

GATE input is used as trigger input

OUT remains high till the count is loaded in the count register

OUT remains low till the count becomes zero

## Mode 2

This mode is called rate generator or divide by N counter

If N is loaded as count value the OUT becomes low only for one clock cycle

The OUT signal is initially high

The GATE signal is low the gate signal force the OUT signal as high

The counter can generates the low pulses

The count down can starts at the time count becomes zero

The low pulses are equal to one clock cycle

Mode 3

In this mode 8254 can act as a square wave generator

When the count  $N$  loaded is even then for half of the count  $N/2$  pulses the OUT is high and for remaining  $N/2$  pulses is low

If the count  $N$  loaded is odd then for  $N+1/2$  pulses the OUT is high for remaining pulses OUT remains low

## Mode 4

This mode can be termed as software triggered strobe

If the mode is set the OUT is high

When a count is loaded counting down starts

In terminal the OUT goes low for one clock cycle

If the low pulses can be used as strobe peripheral are interfaced with microprocessor

If the new count is loaded into count register the previous counting is progress

If the counting can proceeds according to the new count

## Mode 5

This mode of operation is called hardware triggered strobe

In this mode signal is generated at the rising edge

This mode can be used to generate the strobe

If the OUT goes high the count can be loaded

The counter starts counting the rising edge

The OUT low when TC is reached

## **8237**

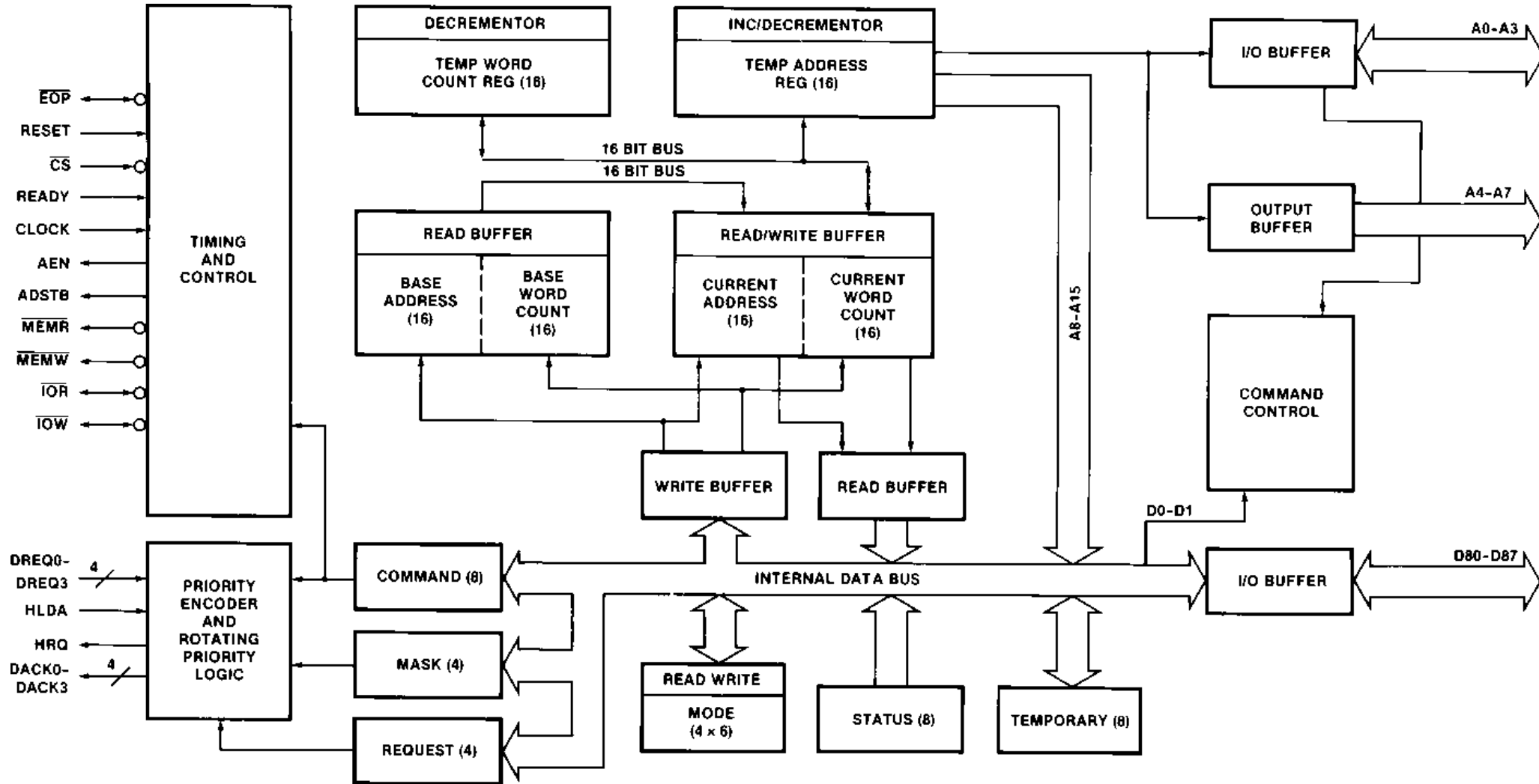
It is also called as DMA controller

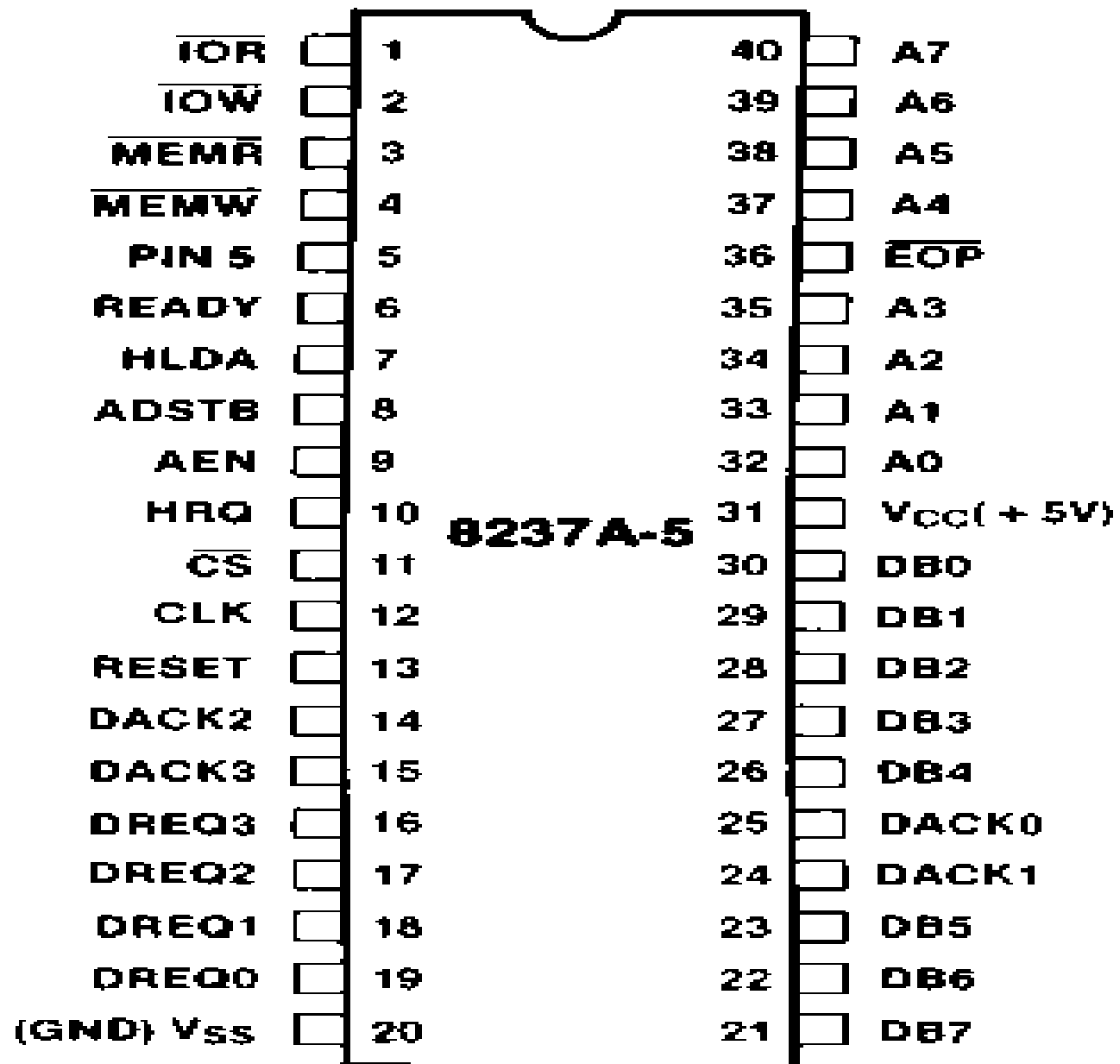
DMA mode of transfer is the fastest among all the mode of data transfer

In this mode the device may transfer data directly / memory without any interface from CPU

DMA is initiated only after receiving HLDA signal from CPU

DMA mode of data transfer between several device DMA controller is used





The chip supports four DMA channel i.e four peripheral device can independently request for DMA data transfer through these channel It has an 8-bit internal data buffer read / write unit control unit priority resolving unit along with set of register

### **I/O buffer**

The 8-bit bidirectional data buffer is interfaced with internal bus of 8237 and also with external data bus

### **Timing and control unit**

The control logic unit control the sequence of DMA operations of the following control signals are AEN, MEMR, MEMW



## **EOP**

### **END OF PROCESS**

End of Process is an active low bidirectional signal.

The 8237A allows an external signal to terminate an active DMA service

The 8237A also generates a pulse when the terminal count (TC) for any channel is reached.

The reception of EOP, either internal or external, will cause the 8237A

The mask bit and TC bit in the status word will be set

## **RESET**

Reset is an active high input which clears the Command, Status, Request and Temporary registers.

## **CS**

Chip Select is an active low input used to select the 8237A as an I/O device during the Idle cycle.

This allows CPU communication on the data bus.

## **READY**

Ready is an input used to extend the memory read and write pulses from the 8237A to accommodate slow memories or I/O peripheral devices.

## **CLOCK INPUT**

Clock Input controls the internal operations of the 8237A and its rate of data transfers.

The input may be driven at up to 5 MHz

## **AEN**

### **ADDRESS ENABLE**

Address Enable enables the 8-bit latch containing the upper 8 address bits onto the system address bus.

AEN can also be used to disable other system bus drivers during DMA transfers.

## **ADSTB**

### **ADDRESS STROBE**

The active high, Address Strobe is used to strobe the upper address byte into an external latch.

## **MEMR**

### **MEMORY READ:**

The Memory Read signal is an active low three-state output used to access data from the selected memory location during a DMA Read or a memory-to-memory transfer.

## **MEMW**

### **MEMORY WRITE:**

The Memory Write is an active low three-state output used to write data to the selected memory location during a DMA Write or a memory-to-memory transfer.

## **IOR**

### **I/O READ:**

I/O Read is a bidirectional active low three-state line.

In the Idle cycle, it is an input control signal used by the CPU to read the control registers.

In the Active cycle, it is an output control signal used by the 8237A to access data from a peripheral during a DMA Write transfer.

## **IOW**

### **I/O WRITE:**

I/O Write is a bidirectional active low three-state line. In the Idle cycle, it is an input control signal used by the CPU to load information into the 8237A

In the Active cycle, it is an output control signal used by the 8237A to load data to the peripheral during a DMA Read transfer

## **DREQ0 – DREQ3**

### **DMA REQUEST**

The DMA Request lines are individual asynchronous channel request inputs used by peripheral circuits to obtain DMA service.

In fixed Priority, DREQ0 has the highest priority and DREQ3 has the lowest priority

## **HLDA**

### **HOLD ACKNOWLEDGE:**

The active high Hold Acknowledge from the CPU indicates that it has relinquished control of the system busses.

# **HRQ**

## **HOLD REQUEST:**

This is the Hold Request to the CPU and is used to request control of the system bus.

If the corresponding mask bit is clear, the presence of any valid DREQ causes 8237A to issue the HRQ.

## **DAK0 – DAK3**

### **DMA ACKNOWLEDGE:**

DMA Acknowledge is used to notify the individual peripherals when one has been granted a DMA cycle.

The sense of these lines is programmable.

Reset initializes them to active low.

## **DB0 – DB7**

### **DATA BUS:**

The Data Bus lines are bidirectional three-state signals connected to the system data bus.

The outputs are enabled in the Program condition during the I/O Read to output the contents of an Address register, a Status register, the Temporary register or a Word Count register to the CPU.

The outputs are disabled and the inputs are read during an I/O Write cycle when the CPU is programming the 8237A control registers

In memory-to-memory operations, data from the memory comes into the 8237A on the data bus during the read-from- memory transfer.

In the write-to-memory transfer, the data bus outputs place the data into the new memory location.



## **A0 – A3**

### **ADDRESS:**

The four least significant address lines are bidirectional three-state signals. In the Idle cycle they are inputs and are used by the CPU to address the register to be loaded or read.

In the Active cycle they are outputs and provide the lower 4 bits of the output address.

## **A4–A7**

### **ADDRESS:**

The four most significant address lines are three-state outputs and provide 4 bits of address. These lines are enabled only during the DMA service.

## **PIN5:**

This pin should always be at a logic HIGH level.

An internal pull-up resistor will establish a logic high when the pin is left floating.

It is recommended however, that PIN5 be connected to VCC.

## **VCC**

POWER: a5V supply.

## **VSS**

GROUND: Ground.

## **Register of 8237**

### **Current Address register**

It consist of four DMA channel and also it consist of 16 bit address register

It is used to hold the memory address can be accessed the DMA

The address is automatically incremented or decremented after the data transfer

The current address register can be initialized after the EOP

### **Current Word count register**

It consist of 16 bit memory address

It is used to hold the number of bytes to be transferred

The word count is decremented after each data transfer

When count become zero the EOP will be generated

Current word count register can initialize after the EOP

## **Base address register**

It consist of 16 bit address register

It hold the initialize address of the current data transfer

## **Base Word count Register**

It consist of 16-bit address register

It hold the initialize word count of the current data transfer

## **Command register**

It consist of 8-bit register i.e D0-D7

It controls the complete operation of 8237

D0 - If the input is 1 memory to memory can be enable

- D1 – If the input is 1 channel 0 address hold enable
- D2 – If the input is 1 controller enable is selected
- D3 – If the input is 1 compressed timing can be selected
- D4 – If the input is 1 rotating priority is selected
- D5 – If the input is 1 extended write operation
- D6 – If the input is 1 DREQ can sensed
- D7 – DACK and Acknowledge high

## **Mode set register**

It consist of 8 bit register

It is written by the CPU in program mode



Demand mode select 0 0

Single mode select 0 1

Block mode select 1 0

Cascaded mode select 1 1

1-Auto initialize enable

1-Address decrement start

00 – Verify transfer 00 – Control 0 select

01 – Write transfer 01 – Control 1 select

10-Read transfer 10-Control 2 select

11 – illegal 11-Control 3 select

## **Single mode**

In this mode only one bit is transferred per request

## **Burst mode**

In this mode operation of microprocessor is temporarily suspended at the time the external device can access the memory

When a block is ready for data transfer the DMA can send the HOLD signal to microprocessor

The DMA controller take control of buses and transfer data directly between I/O device and memory

This mode is called as visible mode

When there is a significant time delay between the transfer of two successive data bytes

In this mode data transfer only one or two bytes can be transferred at the time  
During the data transfer operation the microprocessor performing instruction  
decoding

When an I/O device is ready to transfer the data using DMA

It can request the MP for DMA cycle

Once the request is granted the I/O device data transfer one or two bytes of  
data to memory

After some times when the device is again ready to data transfer it repeat the  
process again and again

### **Block Mode**

Block of data can be transferred until a TC is reached zero



## **Demand mode**

This mode is similar of block mode

Block of data can be transferred until a TC is reached zero or an EOP is detected

## **Cascaded mode**

More than one 8237 can be connected together to provide more than four DMA channel

## **Memory-Memory transfer mode**

Data cannot be transferred from one set of memory address to the other set of memory for this it uses one temporary register in between the destination and source

# Request register

Each channel has request associated



Set request bit – 1

Channel select

Reset request bit – 0

D3-D7 do not care

D1-D0

00 – Channel 0 has requested the register

01- Channel 1 has requested

10-Channel 2 has requested

11-Channel 3 has requested

# Mask register



Clear mask bit – 0

00 – Channel 0 has selected

01- Channel 1 has selected

10-Channel 2 has selected

11-Channel 3 has selected

## Status register

**D7** **D6** **D5** **D4** **D3** **D2** **D1** **D0**



D0-If 1 ch0 reached TC

D1-If 1 ch1 reached TC

D2-If 1 ch2 reached TC

D3-If 1 ch3 reached TC

D4 -If 1 ch3 has requested

D5-If 1 ch2 has requested

D6-If 1 ch1 has requested

D7-If 1 ch0 has requested

## **Priorities of DMA**

Fixed priority

Rotating Priority scheme

### **Fixed priority**

In fixed priority each device connected to a channel then the channel is assigned a fixed priority

In this scheme DREQ is the lowest priority and the next DrQ0 is the highest Priority

### **Rotating Priority**

In Rotating Priority mode the priority is assigned to the channel are not fixed .

In this scheme DRQ0 is the highest priority and DRQ3 is the lowest priority

Then after the device at channel 0 get a priority then the priority goes down and the channel 0 become a lowest priority channel

Channel 1 is now become the highest priority

## **DMA operation**

The 8237 is able to perform three types of operation such a verify DMA operation i.e read operation and write operation

The complete DMA operation of 8237 in single byte data transfer can be Described

## **DMA Execution**

The process of data transfer from the peripheral to the system memory under the DMA controller can be classified into two modes

1.Slave mode

2.Master mode

### **1.Slave mode**

In slave mode the DMA controller is treated as a peripheral

The microprocessor select the DMA controller through the chip select

The microprocessor write the control word in channel register and command/status register by using control signal IOW and IOR

In this mode the output signal of 8237 such as A7-A4 ,MEMW and MEMR

### **2.Master mode**

In 8237 master mode keep checking for a DMA request and the steps in data transfer are as

When the peripheral is ready for data transfer it send a high signal to DRQ

When DRQ has been received at the time channel has been enabled the control logic set HRQ high

In the next the HLDA signal send to 8237

After receiving HLDA signal the DMA output are 16-bit address on the bus

The DMA can send DACK to the peripheral

The DMA controller continues the data transfer

At the end of data transfer the EOP signal can low and the peripheral device can complete the data transfer

## **DMA Transfer and operation**

The 8237 is able to accomplish three types of operation



1. Verify DMA
2. Write operation
3. Read operation

A single byte transfer using 8237 may be requested by an i/o device using any one of the 8237 DRQ inputs

8237 sense the HRQ signal to the CPU and its HLD input

If HLDA signal is received by DMA controller it indicates the bus is available for transfer

The DMA controller generate the read and write command to transfer the bytes from i/o device

If more then one channel request service simultaneously transfer will occur in burst transfer

The 8237 has a READY input to interface it with low speed devices

The 8237 can be interfaced as a memory or I/o devices

## **8237 Command and Programming**

### **Clear First / Last**

It can exists in internal flip-flop of 8237 is called First/Last Flip flop

This flip flop output decides whether the lower byte or upper byte

It can also selected the 16-bit register will be read or write

Thus the clearing this command of CPU will address the higher or lower bytes in appropriate sequence

### **Clear Mask register**

A mask set register is set the DMA channel is disable so that the DMA request not entered

If the mask register is cleared individually or collectively the DMA channel are Enabled

## **Master Clear Command**

Using this command all the internal register of 8237 are cleared when the mask register are set

After executing the command the DMA controller disable all the DMA channel can enter into the idle cycle

## **8279 KEYBOARD / DISPLAY CONTROLLER**

Intel 8279 is a general purpose keyboard display controller that simultaneously drives the display of a system and interface a keyboard with the CPU

The keyboard display interface scan the keyboard to identify the if any key has been pressed and send the code of the pressed key to the CPU

It also transmits the data received from the CPU to the display device

The keyboard is interfaced either in the interrupt or polled method

In the interrupt mode the processor is requested service only if any is key

Pressed otherwise the CPU can proceed

In the polled mode the CPU periodically read an internal flag of 8279 to check for a key pressure

The keyboard section can interface an array of maximum 64 keys with the CPU

The keyboard entries are debounced and stored in an 8 bit FIFO RAM that is further accessed by the CPU

If more than eight characters are entered into the FIFO the overrun status is set

If a FIFO contains a valid key entry the CPU is interrupted or the CPU checks the status to read the entry

Once the CPU reads the key entry the FIFO is updated

The 8279 normally provides a maximum of 16 7 segment display interface with CPU

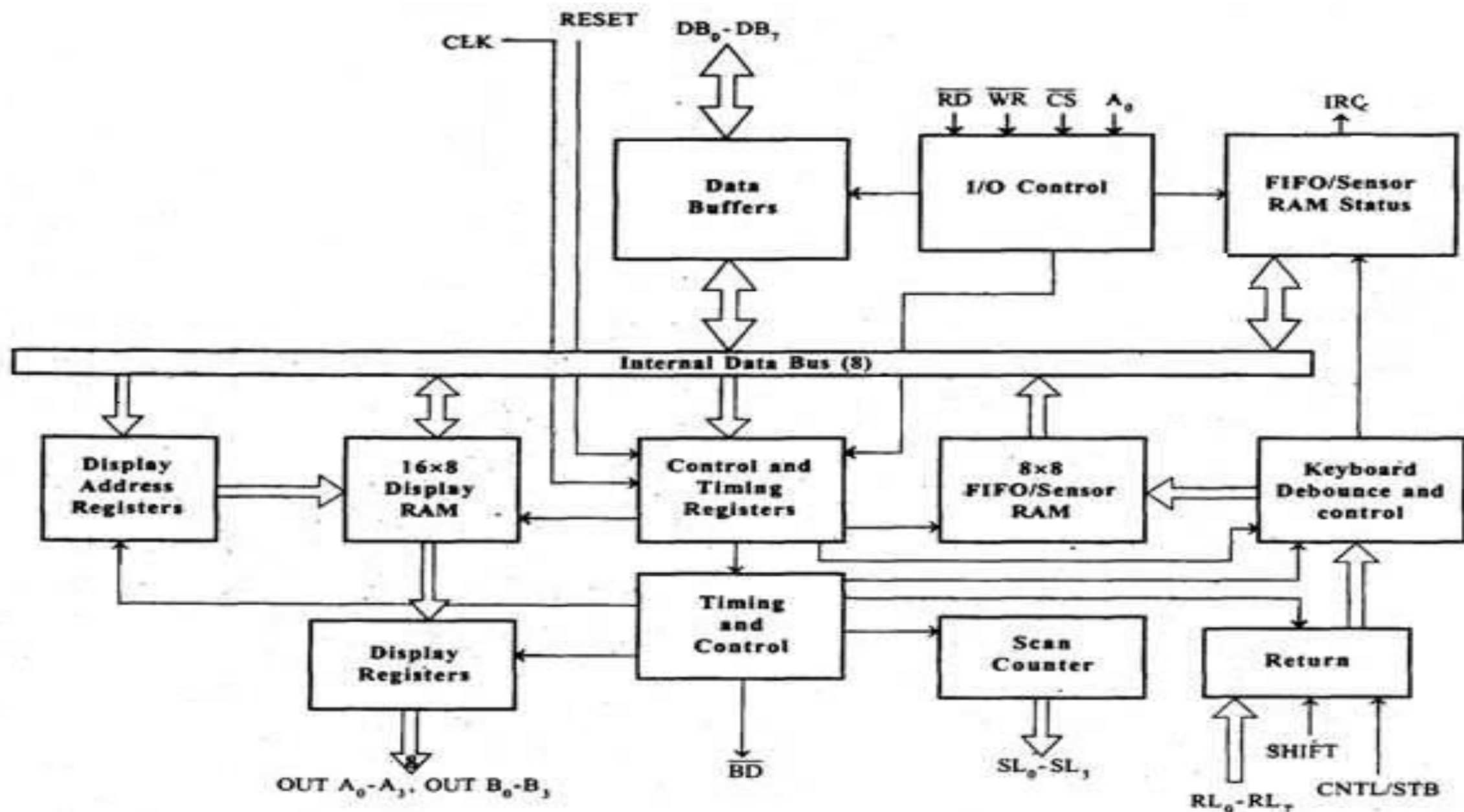
It contains 16 byte display RAM that can be used either as an integrated block of the 16x8 bits or two 16x4 bit blocks of RAM

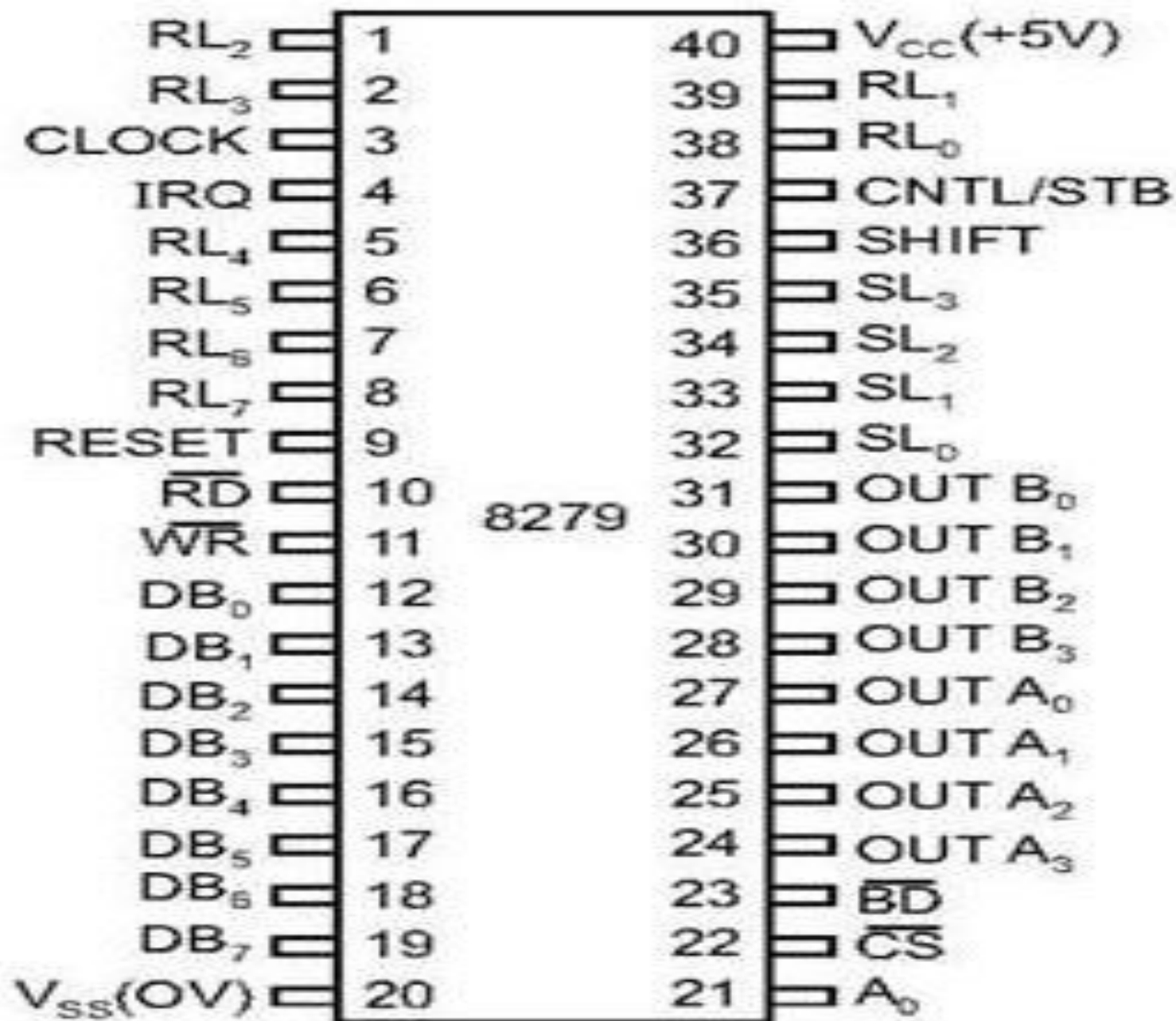
The data entry to RAM block is controlled by CPU using the command word of 8279

The keyboard display controller chip 8279 provides

A set of four scan lines or eight return lines for interfacing keyboards

A set of eight output lines for interfacing the display







## **I/O control and Data bus**

The I/O control section controls the flow of data to the 8279

The data bus buffer interface the external bus of the system with internal bus of 8279

The I/O is enabled only if CS is low

The pin A0, RD and WR select the command status or read / write operation carried out by the CPU with 8279

## **DB0-DB7**

These are bi directional data bus lines . The data and command word to the CPU are transferred

## **CS**

Chip select A low on this line enables 8279 for normal read or write operation

Otherwise the pin is high

## **A0**

A high on this line indicates the transfer of a command or status information

A low on this line indicates the transfer of data

This is used to select one of the internal register of 8279

## **RD,WR**

Input /output , Read / Write

The input pins enable the data buffer to receive or send data over the data bus

## **Control and Timing Register**

The register store the keyboard and display mode and other operating conditions performed by CPU

The registers are  $A0=1$  and  $WR=0$

## **CLK**

This is a clock input used to generate internal timing required by 8279

## **RESET**

This pin is used to reset 8279.

A high on this line reset 8279

## **BD**

The output pin is used to blank the display during digital switching

## **Scan counter**

There are two ways in which scan line can be interfaced to the keyboard

The scan counter has two modes to scan key matrix and refresh the display

Encoded mode

Decoded mode

In encoded mode the counter provides binary count is to be externally decoded to provide scan line for keyboard and display

In decoded scan mode the counter internally decodes the least significant bit

### **SL0-SL3**

These line is used to scan the keyboard matrix and display digits

These line can be programmed as encoded or decoded using the mode control register

### **Return buffer and keyboard debounce and control**

This section scan for a key closure row wise.If a key closer is detected the keyboard debounce unit debounce the key entry

After the debounce if the key continues to detect the code of pressed key

The four scan line are interfaced to the rows of keyboard

RL0-RL7 Return lines

These are input lines which is connected to one terminal of key and the other terminal of key are connected to decoded scan lines

## **SHIFT**

The shift input lines is stored along with each key code in FIFO in scanned keyboard

It is pulled up internally to keep high and pulled low with a key closure

## **CNTL/STB-CONTROL/STROBED I/P MODE**

In keyboard mode this line is used as a control input and stored in FIFO on a key closure

## **FIFO sensor RAM and status logic**

In keyboard or stored input this block acts a 8 byte FIFO RAM.

Each key code of the pressed key is entered in the order of the entry and in mean time read by the CPU till the RAM become empty

## **IRQ**

This interrupt output line goes high where the data in the FIFO sensor RAM

The interrupt line goes low with each FIFO RAM read operation but if the FIFO RAM further contain any key code entry to read CPU this pin again goes high to generate an interrupt to the CPU

## **Display Address Register and Display RAM**

The display address register hold the address of the word currently being written or read by the CPU to or from the display RAM

## **OUT A0 – OUT A3 and OUT B0 – OUT B3**

These are output port for two 16\*4 and 16\*8 internal display refresh register

The two 4 bit port may also as one 8 bit port

### **Modes of operation**

There are two types of mode

Input mode

Output mode

Keyboard mode

Display mode

### **Input mode**

Scanned keyboard mode

Scanned sensor matrix

Strobed input

## **Output mode**

Display scan

Display entry

## **Keyboard mode**

Scanned keyboard mode with 2 key lockout

Scanned keyboard with N-key rollover

## **Display mode**

Left entry mode

Right entry mode