

①

Unit - I

8085 processor

Introduction:-

The features of 8085 includes,

- * 8-bit microprocessor
- * Operates on single +5V power supply.
- * It has 16 ~~bit~~ address lines [$2^{16} = 64\text{Kbytes of memory}$]
- and 8 data lines.
- * It can operate with a 3MHz clock frequency. The 8085A-2 version can operate at the maximum frequency of 2MHz.
- * The low 8-bit address bus (A_0-A_7) & data bus (D_0-D_7) are multiplexed to reduce number of external pins.

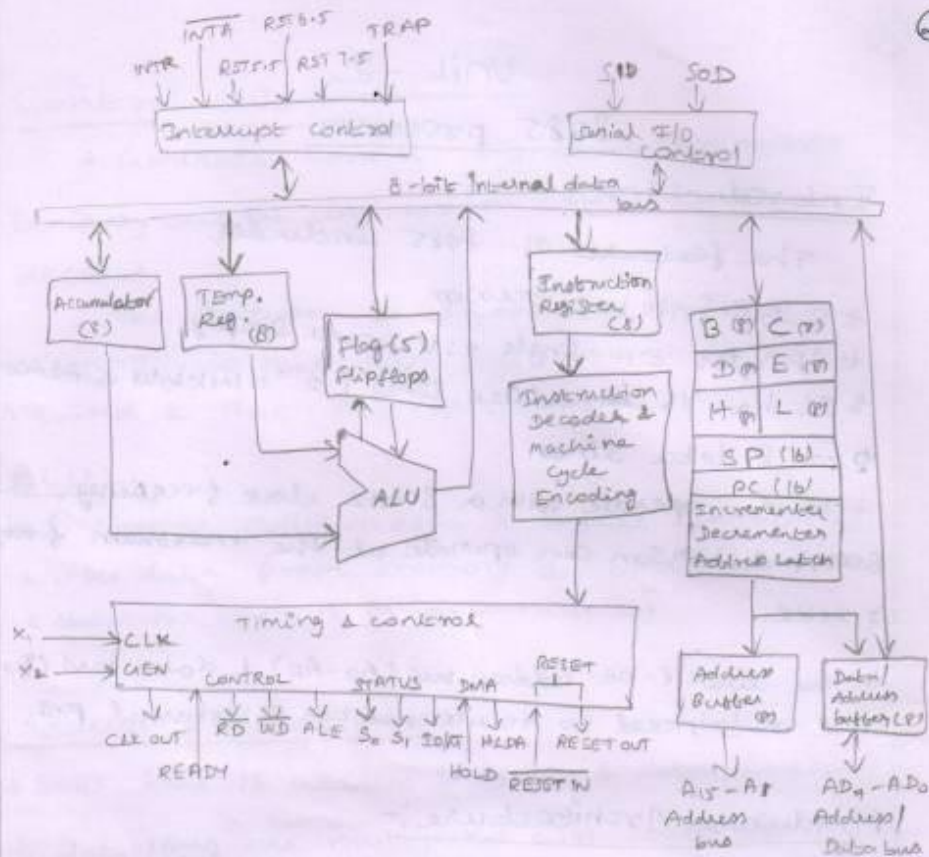
Hardware Architecture:-

It consists of various functional blocks below,

- * Registers
- * ALU
- * Instruction Decoder & Machine cycle encoder
- * Address bus
- * Address/Data bus
- * Incrementer/Decrementer address latch
- * Interrupt control
- * Serial I/O control
- * Timing & Control circuitry

The main units of 8085 are

- * Control unit
- * ALU
- * Registers
- * Interrupts
- * Internal data bus



Functional block diagram of 8085 microprocessor.

Registers:-

1. General Purpose registers:-

- * B, C, D, E, H & L are 8-bit registers
- * BC, DE, HL pairs are 16-bit registers
- For BC register pair, B - higher order byte
C - Lower order byte.
- * HL pair acts as data pointer/memory pointer.

2. Temporary Registers:

- * ALU has 2 inputs.
- * One input is accumulator & other is temporary data register.

(3)

* W & Z are temporary registers used to hold 8-bit data during execution of some instructions.

3. Special Purpose Registers:-

a, Accumulator:-

* It is a 8-bit register, used in arithmetic & logical operations.

b, Flag Register:-

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
S	Z	X	AC	X	P	X	CY

S - Sign flag → D₇ = 1, then negative number
D₇ = 0, then positive number.

Z - Zero flag → D₆/Z = 1, if result of operation is zero.

AC - Auxiliary carry flag, → This flag is set if there is a carry from lower nibble to higher nibble. (D₃ to D₄)

P - Parity flag → number of ones in accumulator.
even parity means flag set.
odd parity means flag reset.

CY - carry flag → This flag is set if there is a overflow.

c, Instruction Register:-

* processor fetches opcode of instruction from memory.

* CPU stores the opcode in a register called instruction register.

Program Counter (PC) - register to hold the address of the next instruction

Stack Pointer (SP) - It is a 16-bit register to point into memory. stack is an area of memory used to hold data that will be retrieved soon.

(4)

Control Unit:-

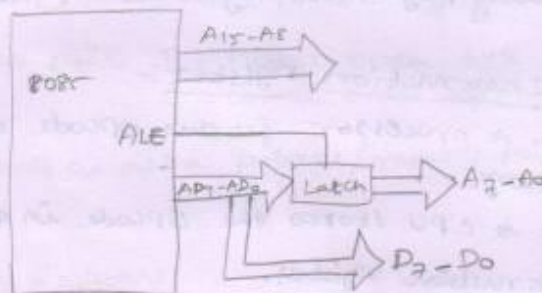
- + Generate Control Signals within processor to carry out the instruction which has been decoded.
- + The processor is connected to I/O or memory, so that data goes where it is required & that ALU operations occur.

ALU:-

- + Performs arithmetic & logical operations.
- + Uses data from memory for operations
- + Store the result in accumulator.

Address & data buses:-

- + 8085 has 16 address lines & 8 data lines.
- + data lines are multiplexed with lower address lines hence $AD_0 - AD_7$ are bidirectional while $A_8 - A_{15}$ are unidirectional.
- + It is necessary to know that bit pattern in low order lines are address/data.



(5)

Interrupts:-

8085 has 5 interrupts (pins). They are

- * TRAP
- * RST 5.5
- * RST 6.5
- * RST 7.5
- * INTR

* RST 5.5, RST 6.5, RST 7.5 are all automatically vectored & maskable.

* TRAP is also automatically vectored & non-maskable i.e., it cannot be disabled.

* RST 5.5, RST 6.5, RST 7.5 can be enabled/disabled selectively.

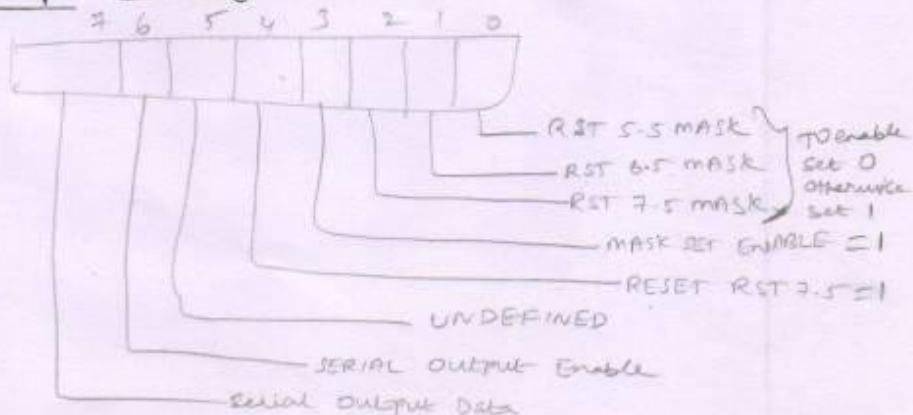
* The interrupt request on the line is acknowledged by 8085 on \overline{INTA} output line.

* There are 2 software instructions - EI & DI

EI - Enable Interrupt

DI - Disable Interrupts except TRAP

* On receiving the acknowledgement, the interrupting device places the address of ISR on data bus. The microprocessor executes the ISR (interrupt service routine) to service the interrupts.

Interrupt masking:

⑥

Serial Input - Output

SID (Serial Input Data) - SID line used by output device to send bit serial data to processor

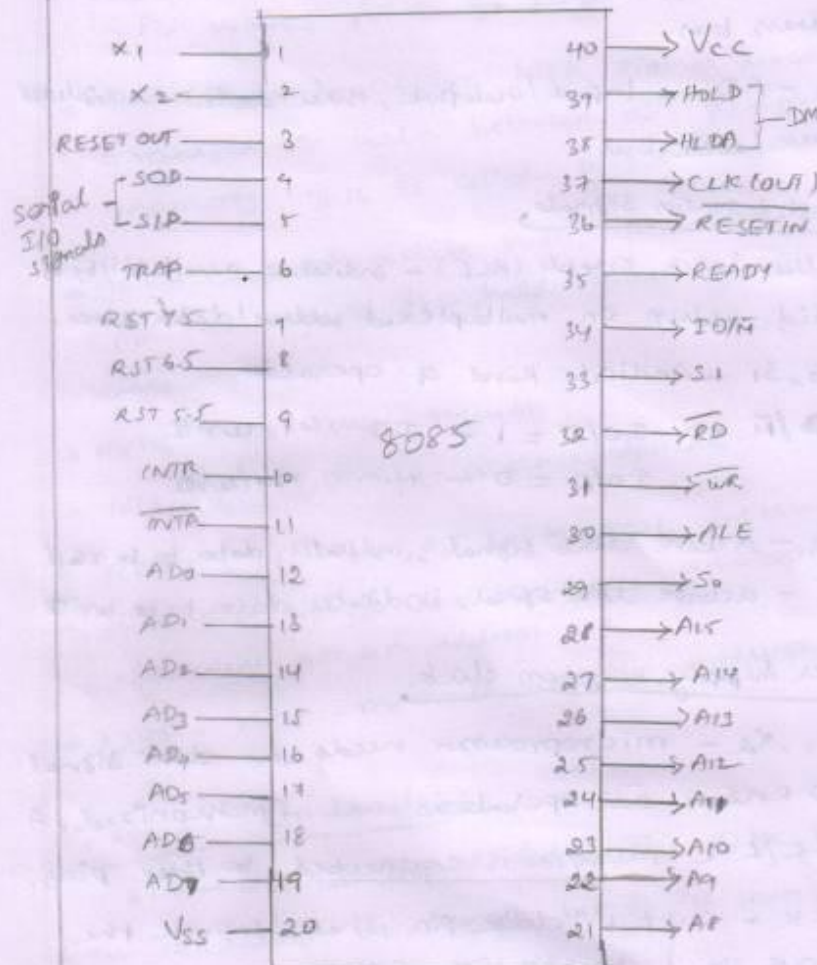
SOD (Serial Output Data) - SOD line used by processor to output bit serial data to input devices.

S _i	S _o	Operation
0	0	Halt
0	1	memory or I/O write
1	0	memory or I/O read
1	1	Instruction Fetch

RESET IN - when goes low, reset processor.

RESET OUT - Used to reset other associated circuits.

PIN DESCRIPTION OF 8085



* 8085 is 40 pin dual in line package.

operate at 3MHz clock frequency.

Signals classified into six groups

i) Address bus ii) Data bus.

iii) Control & status bus iv) Power supply & system clock.

v) Externally initiated signals

vi) serial I/O signals

⑧

i) Address & Data bus..

* $A_{16} - A_{15}$ - output, these state higher order address bus.

* $A_{14} - A_7$ - Input/output, these state multiplexed address/data bus.

ii) Control & Status Signals,

* Address Latch Enable (ALE) - indicates availability of valid address on multiplexed address/data lines.

* S_0, S_1 - indicate kind of operation

* $I/O/\overline{M} \rightarrow I/O/\overline{M} = 1$ - I/O read/write
 $I/O/\overline{M} = 0$ - Memory read/write

* \overline{RD} - active low signal, indicates data to be read

* \overline{WR} - active low signal, indicates data to be write

iii) Power supply & system clock,

* X_1, X_2 - microprocessor needs a clock signal to ensure all operations are synchronized. A R-C/L-C network is connected to these pins.

* CLK - output clock pin that provide the clock signal to rest of system

* power supply $V_{CC} - +5V$, V_{SS} - ground.

iv) Externally Initiated signals:-

* $\overline{RESET IN}$ - Set PC to zero & resets interrupt enable & HLDA flip flops. Resets the processor.

* $\overline{RESET OUT}$ - indicates CPU is being reset. Reset signal for peripheral devices.

(9)

- * **Ready (input)** - When high, indicates memory/peripheral is ready to send or receive data. When low, CPU waits for it to go high.
- * **Hold (input)** - active high signal used in direct transfer of data between a peripheral device & memory. This is called direct memory access.
- * **HLDA (output)** - active high signal indicates that CPU has received the hold request & that it releases the buses.
- * **INTR** - Interrupt request.
- * **INTA** - Interrupt acknowledgement.
- * **RST 7.5, RST 6.5, RST 5.5 (restart interrupts)** - Hardware interrupt signal used to make processor execute subroutine at an address.
- * **Trap** - non-maskable restart interrupt, highest priority.

V) Serial I/O Signals:

SID - Serial input data - The data bit on this line is loaded on seventh bit of accumulator when RIM instruction is executed.

SOD - serial output data - It is set/reset as specified by SIM instruction.

(15)

Data Transfer Mechanisms:

* Data can be transferred in many ways. Mechanisms differs based on characteristics such as addressing of device, amount of data transferred, method of data transfer, interaction among devices.

Based on the addressing of the device

i) I/O mapped I/O access

* I/O device / memory are handled separately.
* Separate control signals are used for memory access & for I/O device read/write operation.

ii) memory mapped I/O.

* Each I/O device is treated like a memory.
* Same control signal are used for I/O device read/write operation & for memory access.
* Each I/O device is identified by a unique address in the memory address range.

Based on the program & hardware involved.

i) Programmed data transfer

* The instructions for programmed data transfer are written & controlled by programmer & executed by processor.

* Programmed I/O data transfer are identical to read & write options for memories & device registers.

(11)

* programmed data transfer can take place at a time determined by programmer.

a, polled mode of data transfer:

* data is read from inputs when CPU is ready.
* processor then executes the data transfer instructions.

* Data is written into output device by processor when it executes write instruction corresponding to that output device.

b, Interrupt driven data transfer:

* when device is ready, it gives an interrupt signal to the processor indicating device is ready.

* In interrupt service routine (ISR), a program is executed to read data from the corresponding input device. Similarly output device give interrupt to processor when it can accept data.

* The programmer have to write an ISR for data transfer to corresponding output device.

ii, Direct Memory Access (DMA)

DMA is a technique to transfer data between the peripheral I/O devices and the memory without intervention of processor.

a, Burst / Block transfer mode

* a complete block of data is transferred in a single DMA cycle.

* The system bus is released by peripheral or DMA controller only after the required bytes are transferred.

(12)

b) Cycle Stealing / interleaved mode

* a block of data is transferred over many DMA cycles.

* The system bus is released to the processor after a byte / set of bytes are transferred in one DMA cycle.

* It takes several DMA cycles to complete the transfer of one block of data.

Based on the method of data transfer & access.

i) Parallel data transfer

* all bits in a word are simultaneously transmitted.

a) Synchronous mode :

* data is read/written between the I/O device & processor irrespective of its status.

* Input device is synchronism with processor & is ready with data whenever processor reads the data.

b) Handshake I/O mode :

* processor check for status of I/O device before data transfer.

* An input device give signal to processor indicating that it is ready ~~to~~ with the data. The processor checks continuously for the reception of this signal & upon reception can read the data.

* An output device give signal to processor indicating that it is ready to accept the data.

(13)

The processor before writing to output device check for this signal. If the signal indicate ready processor write data to output device.

ii) Serial data Transfer:

* only one bit is transferred over a data transfer line.

* All bits can be transmitted using shift register.

a) Synchronous Data transfer.

* The device that sends the data & the device that receive the data are synchronized with common clock.

* Data transfer takes place with fixed & known time frame.

b) Asynchronous Data transfer.

* Data words are transmitted with a random time frame between them.

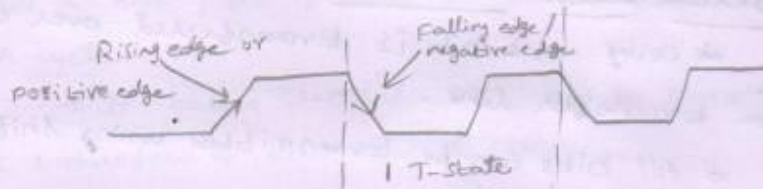
* processors use interrupts & other software techniques to synchronize random timing between data words, so as to receive data completely.

Timing Diagrams:-

8085 microprocessor is designed to fetch instruction pointed to the program counter & then decode & execute the instruction within the processor.

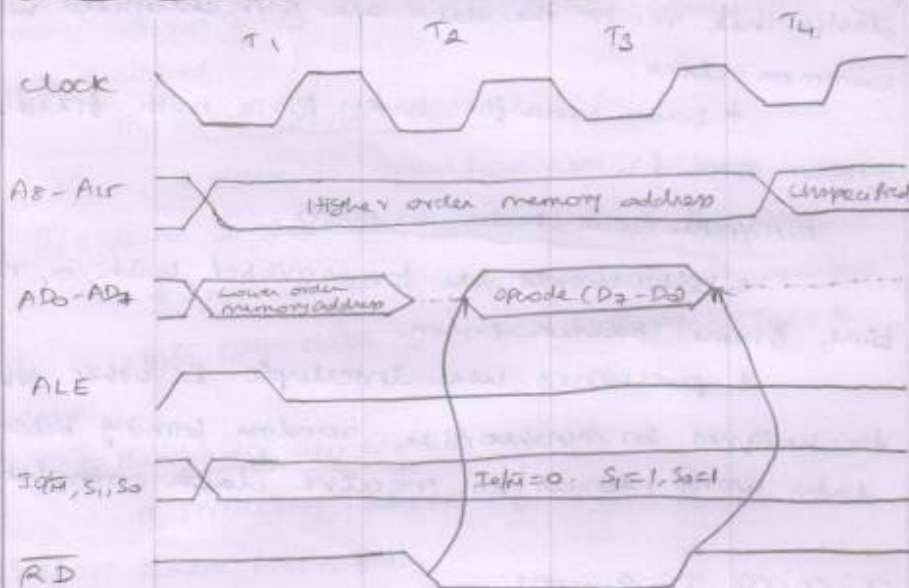
(14)

To execute a program, the 8085 performs various operations such as opcode fetch, operand fetch, memory read/write or I/O read/write.



Time period $T = 1/f$, f - clock frequency

Opcode Fetch machine cycle

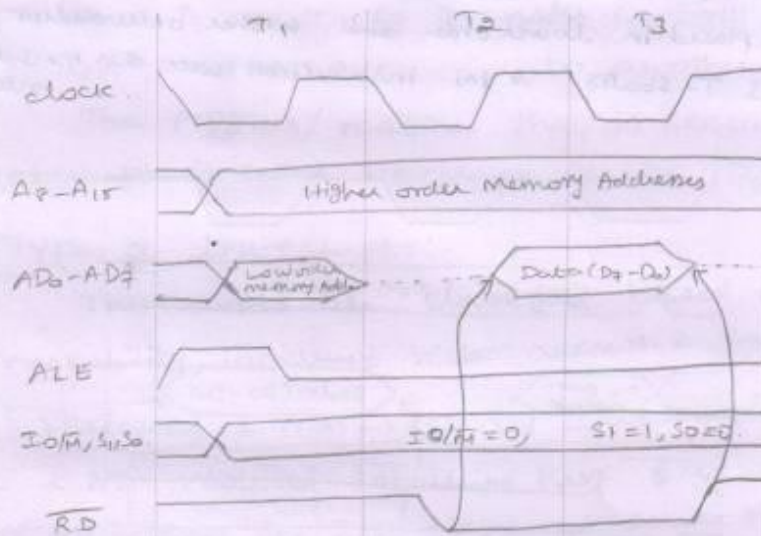


T_1 clock cycle	T_2 clock cycle	T_3	T_4
$IO/M = 0$ $ALE = 1$	$\overline{RD} = 0$	$\overline{RD} = 1$	+ opcode decode + instruction execution

(15)

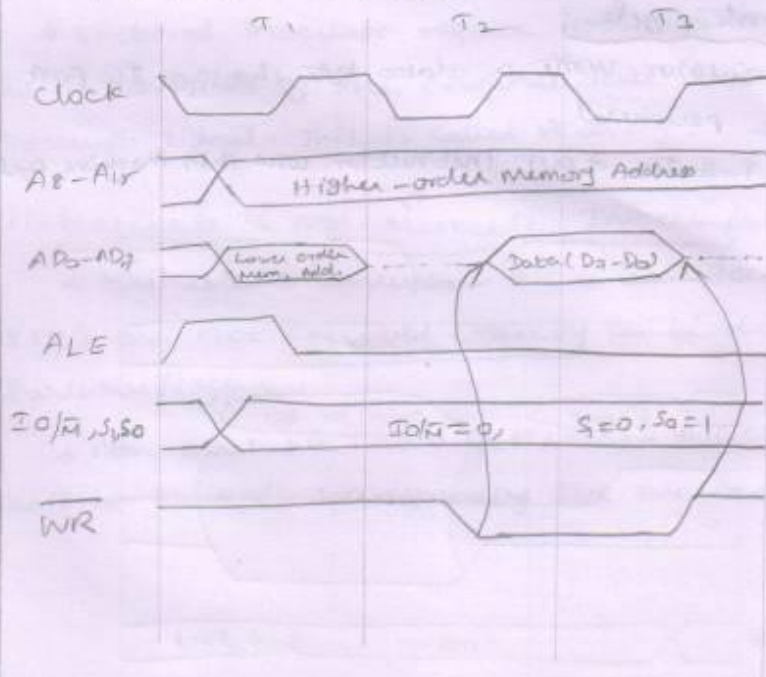
Memory Read machine cycle

* 3 T-states * $S_0 = 0$



Memory Write machine cycle:-

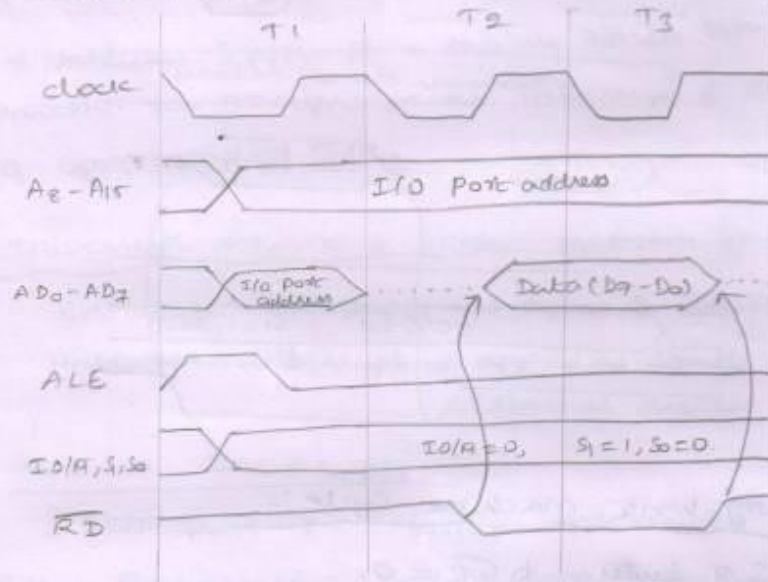
* 3 T-states * $\overline{WR} = 0$.



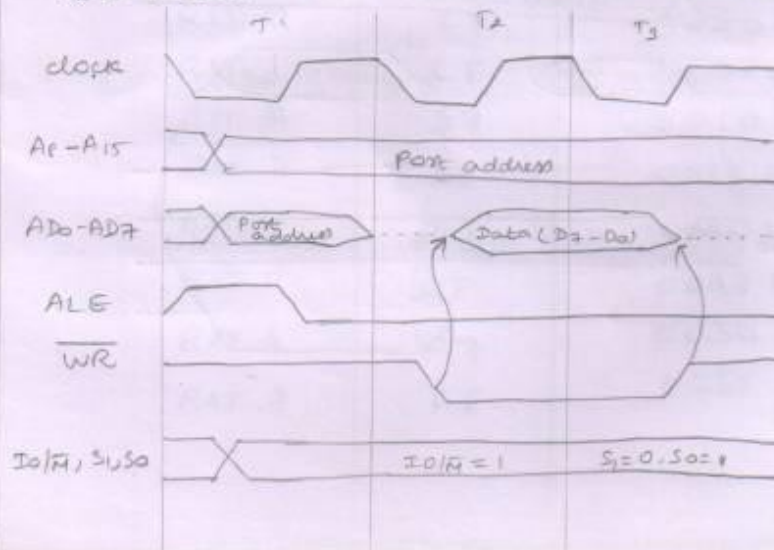
(16)

I/O Read cycle:

* processor read a data byte from I/O Port
 to from peripheral, 8085 uses 8-bit port address
 which is placed in lower order and higher order address bus
 * 3 T-states * IN instruction uses this machine cycle

I/O write cycle:

* processor write a data byte to an I/O port
 or to a peripheral.
 * 3 T-states * OUT instruction uses this machine cycle.



(17)

Interrupts:

Interrupt is a mechanism by which the processor is made to transfer control from its current program execution to another program.

The program/routine that is executed upon interrupts is called interrupt service routine (ISR).

Types of interrupts:-

Interrupts are classified based on their maskability, interrupt vector address & source.

i) Vectored & Non-Vectored Interrupts:-

* Non vectored interrupt have fixed interrupt vector address for ISR (Interrupt Service Routine) for different interrupt signals. Useful for small systems, where there are few interrupt sources & software structure is not complicated.

* Vectored Interrupt require interrupt vector address to be supplied by the external device that gives the interrupt signal. This is called vectoring.

ii) Maskable & Non-Maskable Interrupts.

* Maskable Interrupts - this can be blocked, corresponding ISR's are not executed. Masking can be done by hardware/software.

* Non-maskable Interrupts (NMI) - this can be recognized, can't be blocked, corresponding ISR are executed.

2. Software & Hardware Interrupts:

* Software Interrupts - after execution transfer the control to predefined ISR. Instructions are included in program by programmer.

* Hardware Interrupt - this are signals given to processor for recognition as interrupt & execution of corresponding ISR.

Interrupt sources & vector addresses in 8085.

Software Interrupts → Form of instructions

Hardware interrupts → applied as signals from external devices.

Software Interrupts:

* 8085 have 8 software interrupts instructions called Restart (RST) instructions. These are one byte instructions.

Instruction	Hex code	Interrupt vector address
RST 0	C7	0000 H
RST 1	CF	0008 H
RST 2	D7	0010 H
RST 3	DF	0018 H
RST 4	E7	0020 H
RST 5	EF	0028 H
RST 6	F7	0030 H
RST 7	FF	0038 H

Hardware Interrupts:

* 8085 have 5 hardware interrupts - INTR, RST 5.5, RST 6.5, RST 7.5 & Trap.

* To apply an interrupt, logic 1 signal should be applied at these pins.

* If signal on any of these pin is logic 1 & corresponding ~~mask~~ interrupt is not masked, & corresponding ISR is executed.

Interrupt	Interrupt vector address	Maskable/Non maskable	Edge/Level triggered	priority
Trap	0024H	nonmaskable	Level	1
RST 7.5	003CH	maskable	Rising edge	2
RST 6.5	0034H	maskable	Level	3
RST 5.5	002CH	maskable	Level	4
INTR	Decide by hardware	maskable	Level	5

Masking of Interrupts:

* Masking can be done for 4 hardware interrupts - RST 7.5, RST 6.5, RST 5.5 & INTR.

* EI (enable Interrupt) & DI (disable interrupt) are 2 software instructions.

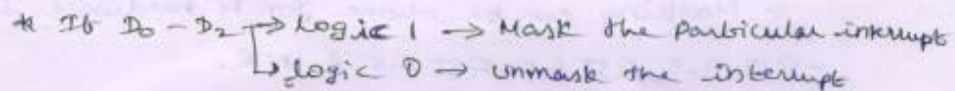
* EI enabled the interrupts

* DI disables the interrupts

* Various Interrupts can be enabled/disabled by masking.

SIM (Set Interrupt Mask) Instruction

* sm instruction when executed read the contents of accumulator & masks / ~~un~~masks the interrupts.



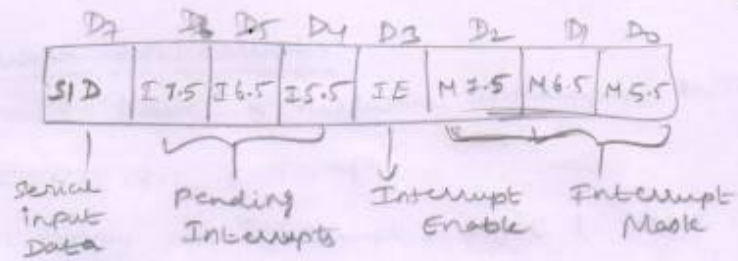
$$+D_2 = 1$$

RIM (Read Interrupt Mask) Instruction

* When RIM instruction executed, accumulator is loaded with current status of interrupt masks & pending interrupts.

(21)

Accumulator bit pattern after RIM instruction execution



- * D0, D1, D2 - interrupts masks set
- * D3 - interrupt enable/disabled
- * D4, D5, D6 - pending Interrupts

6

I/O and Memory Interface

6.1 Introduction

The important components of any computer system are, processor, memory, and I/O devices (peripherals). The processor fetches instructions (opcodes and operands/data) from memory, processes them and stores results in memory. The other components of the computer system (I/O devices) may be loosely called the Input/Output system. The main function of I/O system is to transfer information between processor or memory and the outside world.

6.1.1 Input / Output System

The Input/Output system has two major functions :

- Interface to the processor and memory via the system bus,
- Interface to one or more I/O devices by tailored data links

For clear understanding refer Fig. 6.1. Links to peripheral devices are used to exchange control, status and data between the I/O system and the external devices.

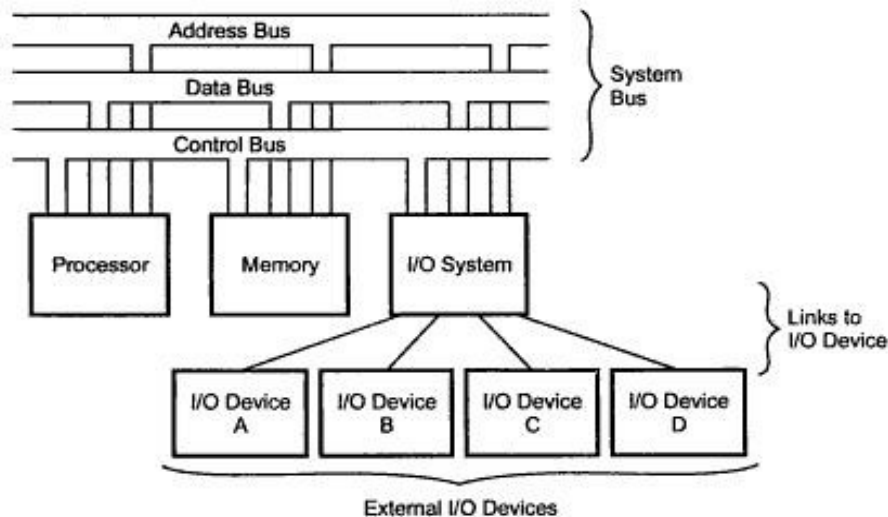


Fig. 6.1 Generic model of computer system
(6 - 1)

Copyrighted material

An important point that is to be noted here is, an I/O system is not simply mechanical connectors for connecting different devices required in the system to the system bus. It contains some logic for performing function of communication between the peripheral (I/O device) and the bus. The I/O system must have an interface internal to the computer (to the processor and memory) and an interface external to the computer (to the external I/O devices).

6.1.2 Requirements of I/O System

The I/O system is nothing but the hardware required to connect an I/O device to the bus. It is also called **I/O interface**. The major requirements of an I/O interface are :

1. Control and timing
2. Processor communication
3. Device communication
4. Data buffering
5. Error detection

All these requirements are explained here. The I/O interface includes a control and timing requirements to co-ordinate the flow of traffic between internal resources (memory, system bus) and external devices. Processor communication involves different types of signal transfers such as

- Processor sends commands to the I/O system which are generally the control signals on the control bus.
- Exchange of data between the processor and the I/O interface over the data bus.
- The data transfer rate of peripherals is often much slower than that of the processor. So it is necessary to check whether peripheral is ready or not for data transfer. If not, processor must wait. So it is important to know the status of I/O interface. The status signals such as BUSY, READY can be used for this purpose.
- A number of peripheral devices may be connected to the I/O interface. The I/O interface controls the communication of each peripheral with processor. So it must recognize one unique address for each peripheral connected to it.

The I/O interface must be able to perform device communication which involves commands, status information and data.

Data buffering is also an essential task of an I/O interface. Data transfer rates of peripheral devices are quite high than that of processor and memory. The data coming from memory or processor are sent to an I/O interface, buffered and then sent to the peripheral device at its data rate. Also, data are buffered in I/O interface so as not to tie up the memory in a slow transfer operation. Thus the I/O interface must be able to operate at both peripheral and memory speeds.

Copyrighted material

I/O interface is also responsible for error detection and for reporting errors to the processor. The different types of errors are mechanical, electrical malfunctions reported by the device such as bad disk track, unintentional changes to the bit pattern, transmission errors etc. To fulfil all these requirements the important blocks necessary in any I/O interface are shown in Fig. 6.2.

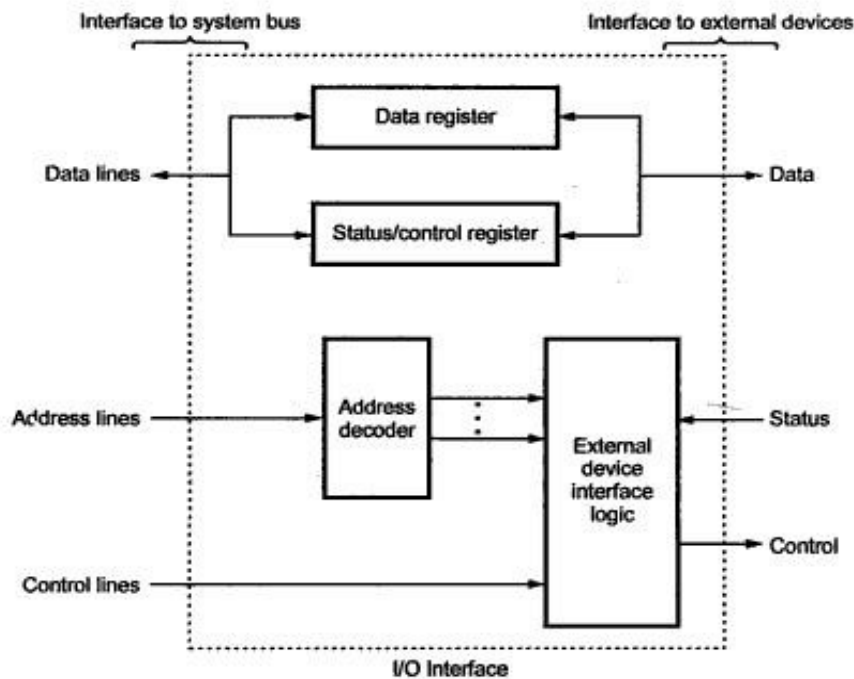


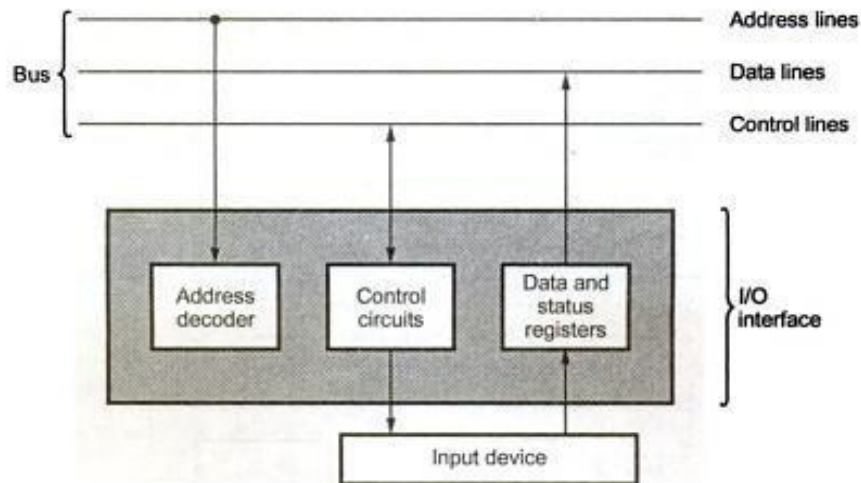
Fig. 6.2 Block diagram of I/O interface

As shown in the Fig. 6.2, I/O interface consists of data register, status/control register, address decoder and external device interface logic. The data register holds the data being transferred to or from the processor. The status/control register contains information relevant to the operation of the I/O device. Both data and status/control registers are connected to the data bus. Address lines drive the address decoder. The address decoder enables the device to recognize its address when address appears on the address lines. The external device interface logic accepts inputs from address decoder, processor control lines and status signal from the I/O device and generates control signals to control the direction and speed of data transfer between processor and I/O devices.

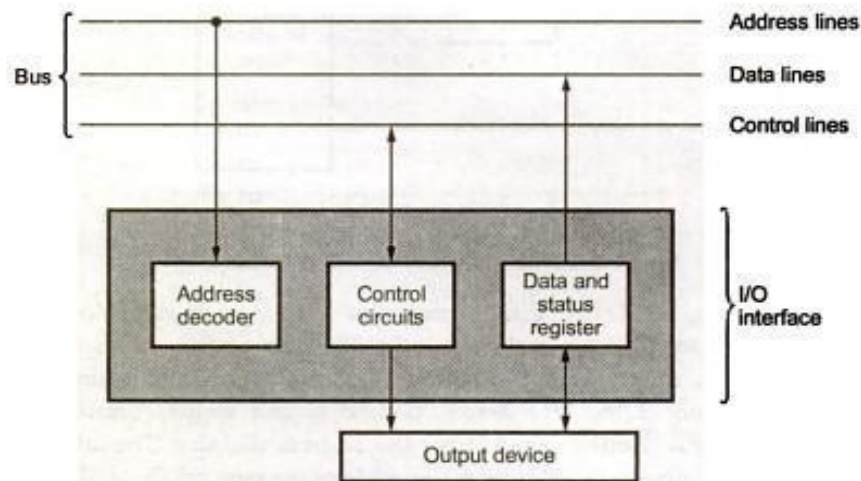
The Fig. 6.3 shows the I/O interface for input device and output device. Here, for simplicity block schematic of I/O interface is shown instead of detail connections. The address decoder enables the device when its address appears on the address lines. The data register holds the data being transferred to or from the processor. The status register

Copyrighted material

contains information relevant to the operation of the I/O device. Both the data and status registers are assigned with unique addresses and they are connected to the data bus.



(a) I/O interface for input device



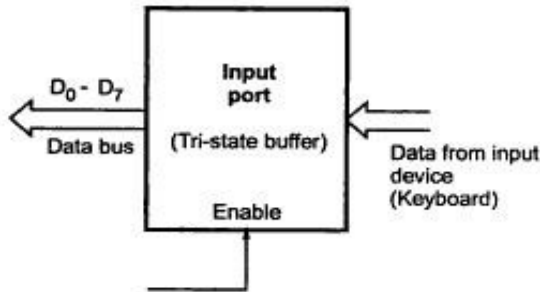
(b) I/O interface for output device

Fig. 6.3

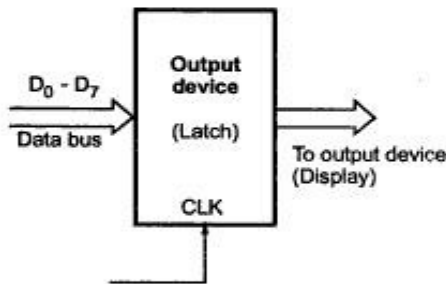
6.1.3 I/O Ports

The simplest form of I/O interface is an I/O port. The data transfer between microprocessor and input device is done with the help of input port. The data transfer between microprocessor and output device is done with the help of output port.

Copyrighted material

Input port :**Fig. 6.4**

the buffer. Once the buffer is enabled, data from the input device is available on the data bus. Microprocessor reads this data by initiating read command.

Output port :**Fig. 6.5**

It is used to read data from the input device such as keyboard. The simplest form of input port is a buffer. The input device is connected to the microprocessor through buffer as shown in the Fig. 6.4. This buffer is a tri-state buffer and its output is available only when enable signal is active. When microprocessor wants to read data from the input device (keyboard), the control signals from the microprocessor activates the buffer by asserting enable input of

It is used to send data to the output device such as display from the microprocessor. The simplest form of output port is a latch. The output device is connected to the microprocessor through latch, as shown in the Fig. 6.5. When microprocessor wants to send data to the output device, it puts the data on the data bus and activates the clock signal of the latch, latching the data from the data bus at the output of latch. It is then available at the output of latch for the output device.

6.2 I/O Data Transfer Techniques

In I/O data transfer, the system requires the transfer of data between I/O devices and microprocessor using I/O interface. It uses various techniques to perform I/O operations. These are :

- Program Controlled I/O or Programmed I/O or Polled I/O.
- Interrupt Driven I/O.
- Hardware Controlled I/O or DMA.

Copyrighted material

Instruction Set

Instruction set of 8085 can be classified in following groups:

- ▢ Data Transfer Instructions
 - ▢ These instructions can perform data transfer operations between
 - ▢ Registers of 8085 e.g. MOV
 - ▢ 8085 registers and main memory e.g. LDA, STA, MOV, LDAX, STAX, MVI, LXI etc.
 - ▢ Accumulator register and I/O devices e.g. IN, OUT
 - ▢ Data transfer instructions never affect the flag bits

Instruction Set

Contd..

- ▢ Arithmetic Instructions
 - ▢ 8085 can perform only 8-bit addition, subtraction and compare operations. These operations are always performed with accumulator as one of the operands. The status of the result can be verified by the contents of flag register.
 - ▢ Op-codes for arithmetic instructions include ADD, ADI, ADC, ACI, SUB, SUI, SBI, DAI, DAD, CPI.
- ▢ Logical Instructions
 - ▢ 8085 can perform 8-bit basic logical operations: AND, OR, XOR, NOT with some special operations such as rotate and shift operations
 - ▢ Logical instructions also modify the flag bits.
 - ▢ Op-codes for logical instructions include ANA, ANI, ORA, ORI, XRI, XRI, CMA, RAL, RLC, RAR, RRC etc.

Instruction Set

Contd..

Program Control Instructions

- ▢ These instructions are used to transfer the program control:
 - ▢ to jump from one memory location to any other memory location within a program
 - ▢ from one program to another program called as a subroutine
- ▢ 8085 Instruction set consists of following program control instructions:
 - ▢ Jump Instructions
 - ▢ Call & Return Instructions
 - ▢ Restart Instructions

Instruction Set

Contd..

Program control instructions

- ▢ Unconditional or Conditional
 - ▢ Unconditional program control instructions perform branching operation unconditionally
 - ▢ Conditional program control instructions perform branching operation with reference to the condition of flag bits.

Instruction Set

Contd..

- ▢ Unconditional Program control instructions are
 - ▢ JMP
 - ▢ Call & RET
 - ▢ RST n (n=0-7)
- ▢ Conditional Program control instructions are
 - ▢ JNC, JC, JNZ, JZ, JP, JM, JPE, JPO
 - ▢ CNC, CC, CNZ, CZ, CP, CM, CPE, CPO
 - ▢ RNC, RC, RNZ, RZ, RP, RM, RPE, RPO

Instruction Set

Contd..

Machine control Instructions

- ▢ These instructions include special instructions such as
 - ▢ HLT - To halt the CPU
 - ▢ NOP - To perform no operation
 - ▢ SIM - To set the masking of hardware interrupts and serial output data
 - ▢ RIM - To read the status of interrupt mask and serial input data
 - ▢ EI - Enable Interrupt
 - ▢ DI - Disable Interrupt

Addressing Modes

- 8085 instructions can be classified in following addressing modes
 - Register Addressing mode
 - Instructions which have their operands in registers only e.g. MOV, ADD, SUB, ANA, ORA, XRA etc.
 - Immediate Addressing mode
 - Instructions in which operand immediately follows the op-code e.g. MVI, LXI, ADI, SUI, ANI, ORI etc.
 - Direct Addressing mode
 - Instructions have their operands in memory and the 16-bit memory address is specified in the instruction e.g. LDA, STA, LHLD, SHLD etc.

Addressing Modes

Contd..

- Register Indirect Addressing mode
 - Instructions have their operand in memory and the 16-bit memory address is specified in a register pair e.g. LDAX, STAX, PUSH, POP etc.
- Implicit Addressing mode
 - These instructions have their operand implied in the op-code itself e.g. CMA, CMC, STC etc.

Instruction size

An instruction is assembled in the memory of a microcomputer system in binary form. The size of an instruction signifies how much memory space is required to load an instruction in the memory. 8085 instructions are of following sizes:

- One-byte instructions
 - e.g. MOV, ADD, ANA, SUB, ORA etc.
- Two-byte instructions
 - e.g. MVI, ADI, ANI, ORI, XRI etc.
- Three-byte instructions
 - e.g. LXI, LDA, STA, LHLD, SHLD etc.

Instruction Format

An **instruction** is a command to the microprocessor to perform a given task on a specified data. Each instruction has two parts: one is task to be performed, called the **operation code** (opcode), and the second is the data to be operated on, called the **operand**. The operand (or data) can be specified in various ways. It may include 8-bit (or 16-bit) data, an internal register, a memory location, or 8-bit (or 16-bit) address. In some instructions, the operand is implicit.

Instruction word size

The 8085 instruction set is classified into the following three groups according to word size:

1. One-word or 1-byte instructions
2. Two-word or 2-byte instructions
3. Three-word or 3-byte instructions

In the 8085, "byte" and "word" are synonymous because it is an 8-bit microprocessor.

However, instructions are commonly referred to in terms of bytes rather than words.

One-Byte Instructions

A 1-byte instruction includes the opcode and operand in the same byte. Operand(s) are internal register and are coded into the instruction.

Example: MOV A,B

Two-Byte Instructions

In a two-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. Source operand is a data byte immediately following the opcode. Example: MVI A, 32H

Three-Byte Instructions

In a three-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit address. Note that the second byte is the low-order address and the third byte is the high-order address.

Three byte instructions - opcode + data byte + data byte Example: LXI 21H, 0520H

8085 INSTRUCTION SET

INSTRUCTION DETAILS

DATA TRANSFER INSTRUCTIONS

Opcode	Operand	Description
Copy from source to destination		
MOV	Rd, Rs M, Rs Rd, M	This instruction copies the contents of the source register into the destination register; the contents of the source register are not altered. If one of the operands is a memory location, its location is specified by the contents of the HL registers. Example: MOV B, C or MOV B, M
Move immediate 8-bit		
MVI	Rd, data M, data	The 8-bit data is stored in the destination register or memory. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: MVI B, 57H or MVI M, 57H
Load accumulator		
LDA	16-bit address	The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered. Example: LDA 2034H
Load accumulator indirect		
LDAX	B/D Reg. pair	The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered. Example: LDAX B
Load register pair immediate		
LXI	Reg. pair, 16-bit data	The instruction loads 16-bit data in the register pair designated in the operand. Example: LXI H, 2034H or LXI H, XYZ
Load H and L registers direct LHLD 16-bit address		
		The instruction copies the contents of the memory location pointed out by the 16-bit address into register L and copies the contents of the next memory location into register H. The contents of source memory locations are not altered. Example: LHLD 2040H

Store accumulator direct
STA 16-bit address

The contents of the accumulator are copied into the memory location specified by the operand. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. Example: STA 4350H

Store accumulator indirect
STAX Reg. pair

The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered.
Example: STAX B

Store H and L registers direct SHLD 16-bit address

The contents of register L are stored into the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand. The contents of registers HL are not altered. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address.
Example: SHLD 2470H

Exchange H and L with D and E XCHG none

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.
Example: XCHG

Copy H and L registers to the stack pointer

none The instruction loads the contents of the H and L registers into the stack pointer register, the contents of the H register provide the high-order address and the contents of the L register provide the low-order address. The contents of the H and L registers are not altered.

Example: SPHL

Exchange H and L with top of stack XTHL none

The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register. The contents of the H register are exchanged with the next stack location (SP+1); however, the contents of the stack pointer register are not altered.
Example: XTHL

Push register pair onto stack PUSH Reg.
pair

The contents of the register pair designated in the operand are copied onto the stack in the following sequence. The stack pointer register is decremented and the contents of the high-order register (B, D, H, A) are copied into that location. The stack pointer register is decremented again and the contents of the low-order register (C, E, L, flags) are copied to that location.

Example: PUSH B or PUSH A

Pop off stack to register pair
POP Reg. pair

The contents of the memory location pointed out by the stack pointer register are copied to the low-order register (C, E, L, status flags) of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand. The stack pointer register is again incremented by 1. Example: POP H or POP A

Output data from accumulator to a port with 8-bit address
OUT 8-bit port address

The contents of the accumulator are copied into the I/O port specified by the operand.
Example: OUT F8H

Input data to accumulator from a port with 8-bit address
8-bit port address

The contents of the input port designated in the operand are read and loaded into the accumulator.
Example: IN 8CH

ARITHMETIC INSTRUCTIONS

Opcode	Operand	Description
Add register or memory to accumulator		
ADD	R M	The contents of the operand (register or memory) are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. Example: ADD B or ADD M
Add register to accumulator with carry		
ADC	R M	The contents of the operand (register or memory) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. Example: ADC B or ADC M
Add immediate to accumulator		
ADI	8-bit data	The 8-bit data (operand) is added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ADI 45H
Add immediate to accumulator with carry		
ACI	8-bit data	The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ACI 45H
Add register pair to H and L registers		
DAD	Reg. pair	The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register. The contents of the source register pair are not altered. If the result is larger than 16 bits, the CY flag is set. No other flags are affected. Example: DAD H

Subtract register or memory from accumulator

SUB	R	The contents of the operand (register or memory) are subtracted from the contents of the accumulator, and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction. Example: SUB B or SUB M
	M	

Subtract source and borrow from accumulator

SBB	R	The contents of the operand (register or memory) and the Borrow flag are subtracted from the contents of the accumulator and the result is placed in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction. Example: SBB B or SBB M
	M	

Subtract immediate from accumulator

8-bit data The 8-bit data (operand) is subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction.
Example: SUI 45H

Subtract immediate from accumulator with borrow

8-bit data The 8-bit data (operand) and the Borrow flag are subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction.

Example: SBI 45H

Increment register or memory by 1

INR	R	The contents of the designated register or memory) are incremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: INR B or INR M
	M	

Increment register pair by 1

INX	R	The contents of the designated register pair are incremented by 1 and the result is stored in the same place. Example: INX H
-----	---	---

Decrement register or memory by 1

DCR R
 M

The contents of the designated register or memory are decremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers.

Example: DCR B or DCR M

Decrement register pair by 1

DCX R

The contents of the designated register pair are decremented by 1 and the result is stored in the same place.

Example: DCX H

Decimal adjust accumulator

DAA none

The contents of the accumulator are changed from a binary value to two 4-bit binary coded decimal (BCD) digits. This is the only instruction that uses the auxiliary flag to perform the binary to BCD conversion, and the conversion procedure is described below. S, Z, AC, P, CY flags are altered to reflect the results of the operation.

If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits.

If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits.

Example: DAA

BRANCHING INSTRUCTIONS

Opcode	Operand	Description
--------	---------	-------------

Jump unconditionally

JMP 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand.
Example: JMP 2034H or JMP XYZ

Jump conditionally

Operand: 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below. Example: JZ 2034H or JZ XYZ

Opcode	Description	Flag Status
JC	Jump on Carry	CY = 1
JNC	Jump on no Carry	CY = 0
JP	Jump on positive	S = 0
JM	Jump on minus	S = 1
JZ	Jump on zero	Z = 1
JNZ	Jump on no zero	Z = 0
JPE	Jump on parity even	P = 1
JPO	Jump on parity odd	P = 0

Unconditional subroutine call
CALL 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. Before the transfer, the address of the next instruction after CALL (the contents of the program counter) is pushed onto the stack. Example: CALL 2034H or CALL XYZ

Call conditionally

Operand: 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below. Before the transfer, the address of the next instruction after the call (the contents of the program counter) is pushed onto the stack. Example: CZ 2034H or CZ XYZ

Opcode	Description	Flag Status
CC	Call on Carry	CY = 1
CNC	Call on no Carry	CY = 0
CP	Call on positive	S = 0
CM	Call on minus	S = 1
CZ	Call on zero	Z = 1
CNZ	Call on no zero	Z = 0
CPE	Call on parity even	P = 1
CPO	Call on parity odd	P = 0

Return from subroutine unconditionally

none The program sequence is transferred from the subroutine to the calling program. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.

Example: RET

Return from subroutine conditionally

Operand: none

The program sequence is transferred from the subroutine to the calling program based on the specified flag of the PSW as described below. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.

Example: RZ

Opcode	Description	Flag Status
RC	Return on Carry	CY = 1
RNC	Return on no Carry	CY = 0
RP	Return on positive	S = 0
RM	Return on minus	S = 1
RZ	Return on zero	Z = 1
RNZ	Return on no zero	Z = 0
RPE	Return on parity even	P = 1
RPO	Return on parity odd	P = 0

Load program counter with HL contents

none The contents of registers H and L are copied into the program counter. The contents of H are placed as the high-order byte and the contents of L as the low-order byte.

Example: PCHL

Restart

0-7 The RST instruction is equivalent to a 1-byte call instruction to one of eight memory locations depending upon the number. The instructions are generally used in conjunction with interrupts and inserted using external hardware. However these can be used as software instructions in a program to transfer program execution to one of the eight locations. The addresses are:

Instruction	Restart Address
RST 0	0000H
RST 1	0008H
RST 2	0010H
RST 3	0018H
RST 4	0020H
RST 5	0028H
RST 6	0030H
RST 7	0038H

The 8085 has four additional interrupts and these interrupts generate RST instructions internally and thus do not require any external hardware. These instructions and their Restart addresses are:

Interrupt	Restart Address
TRAP	0024H
RST 5.5	002CH
RST 6.5	0034H
RST 7.5	003CH

LOGICAL INSTRUCTIONS

Opcode	Operand	Description
Compare register or memory with accumulator		
CMP	R M	<p>The contents of the operand (register or memory) are compared with the contents of the accumulator. Both contents are preserved. The result of the comparison is shown by setting the flags of the PSW as follows: if (A) < (reg/mem): carry flag is set if (A) = (reg/mem): zero flag is set if (A) > (reg/mem): carry and zero flags are reset Example: CMP B or CMP M</p>
Compare immediate with accumulator		
	8-bit data	<p>The second byte (8-bit data) is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison is shown by setting the flags of the PSW as follows:</p> <p>if (A) < data: carry flag is set if (A) = data: zero flag is set if (A) > data: carry and zero flags are reset Example: CPI 89H</p>
Logical AND register or memory with accumulator		
ANA	R M	<p>The contents of the accumulator are logically ANDed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set. Example: ANA B or ANA M</p>
Logical AND immediate with accumulator		
	8-bit data	<p>The contents of the accumulator are logically ANDed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set.</p> <p>Example: ANI 86H</p>

Exclusive OR register or memory with accumulator

XRA R
 M

The contents of the accumulator are Exclusive ORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: XRA B or XRA M

Exclusive OR immediate with accumulator

8-bit data The contents of the accumulator are Exclusive ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.

Example: XRI 86H

Logical OR register or memory with accumulator

ORA R
 M

The contents of the accumulator are logically ORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: ORA B or ORA M

Logical OR immediate with accumulator

8-bit data The contents of the accumulator are logically ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.

Example: ORI 86H

Rotate accumulator left

RLC none Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D7. S, Z, P, AC are not affected.
Example: RLC

Rotate accumulator right

RRC none Each binary bit of the accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0. S, Z, P, AC are not affected.
Example: RRC

Rotate accumulator left through carry

RAL none Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0. CY is modified according to bit D7. S, Z, P, AC are not affected.
Example: RAL

Rotate accumulator right through carry

RAR none Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7. CY is modified according to bit D0. S, Z, P, AC are not affected.
Example: RAR

Complement accumulator

none The contents of the accumulator are complemented. No flags are affected.

Example: CMA

Complement carry

none The Carry flag is complemented. No other flags are affected. Example: CMC

Set Carry

STC none The Carry flag is set to 1. No other flags are affected.
Example: STC

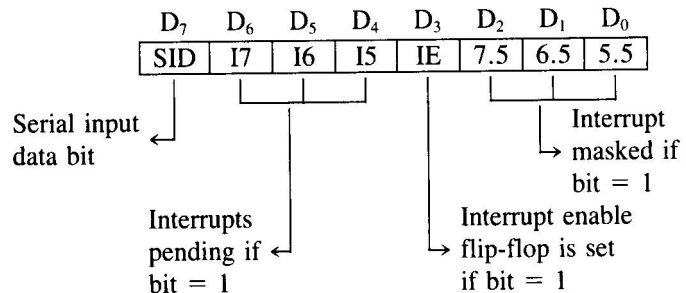
CONTROL INSTRUCTIONS

Opcode	Operand	Description
No operation	none	No operation is performed. The instruction is fetched and decoded. However no operation is executed. Example: NOP
Halt and enter wait state	none	The CPU finishes executing the current instruction and halts any further execution. An interrupt or reset is necessary to exit from the halt state. Example: HLT
Disable interrupts	none	The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected. Example: DI
Enable interrupts	none	The interrupt enable flip-flop is set and all interrupts are enabled. No flags are affected. After a system reset or the acknowledgement of an interrupt, the interrupt enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to reenable the interrupts (except TRAP). Example: EI

Read interrupt mask

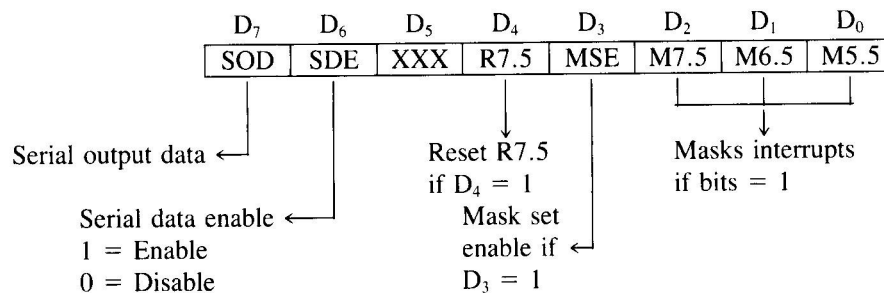
none This is a multipurpose instruction used to read the status of interrupts 7.5, 6.5, 5.5 and read serial data input bit. The instruction loads eight bits in the accumulator with the following interpretations.

Example: RIM

**Set interrupt mask**

none This is a multipurpose instruction and used to implement the 8085 interrupts 7.5, 6.5, 5.5, and serial data output. The instruction interprets the accumulator contents as follows.

Example: SIM



- ☐ **SOD**—Serial Output Data: Bit D₇ of the accumulator is latched into the SOD output line and made available to a serial peripheral if bit D₆ = 1.
- ☐ **SDE**—Serial Data Enable: If this bit = 1, it enables the serial output. To implement serial output, this bit needs to be enabled.
- ☐ **XXX**—Don't Care
- ☐ **R7.5**—Reset RST 7.5: If this bit = 1, RST 7.5 flip-flop is reset. This is an additional control to reset RST 7.5.
- ☐ **MSE**—Mask Set Enable: If this bit is high, it enables the functions of bits D₂, D₁, D₀. This is a master control over all the interrupt masking bits. If this bit is low, bits D₂, D₁, and D₀ do not have any effect on the masks.
- ☐ **M7.5**—D₂ = 0, RST 7.5 is enabled.
= 1, RST 7.5 is masked or disabled.
- ☐ **M6.5**—D₁ = 0, RST 6.5 is enabled.
= 1, RST 6.5 is masked or disabled.
- ☐ **M5.5**—D₀ = 0, RST 5.5 is enabled.
= 1, RST 5.5 is masked or disabled.

3.7 Assembly Language Programming

A program is a set of instructions arranged in the specific sequence to do the specific task. It tells the microprocessor what it has to do. The process of writing the set of instructions which tells the microprocessor what to do is called "**Programming**". In other words, we can say that programming is the process of telling the processor exactly how to solve a problem. To do this, the programmer must "**speak**" to the processor in a language which processor can understand.

3.7.1 Steps Involved in Programming

- **Specifying the problem :**
The first step in the programming is to find out which task is to be performed. This is called specifying the problem. If the programmer does not understand what is to be done, the programming process cannot begin.
- **Designing the problem-solution :**
During this process, the exact step by step process that is to be followed (program logic) is developed and written down.
- **Coding :**
Once the program is specified and designed, it can be implemented. Implementation begins with the process of coding the program. Coding the program means to tell the processor the exact step by step process in its language. Each processor has a set of instructions. Programmer has to choose appropriate instructions from the instruction set to build the program.
- **Debugging :**
Once the program or a part of program is coded, the next step is debugging the code. Debugging is the process of testing the code to see if it does the given task. If program is not working properly, debugging process helps in finding and correcting errors.

To write a program, programmer should know :

- How to develop program logic?
- How to tell the program to the processor?
- How to code the program?
- How to test the program?

3.7.2 Flowchart

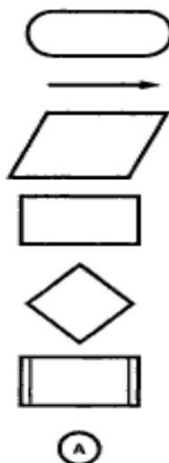


Fig. 3.7 Graphic symbols used in flowchart

To develop the programming logic programmer has to write down various actions which are to be performed in proper sequence. The flow chart is a graphical tool that allows programmer to represent various actions which are to be performed. The graphical representation is very useful for clear understanding of the programming logic.

The Fig. 3.7 shows the graphic symbols used in the flow chart.

Oval : It indicates start or stop operation.

Arrow : It indicates flow with direction.

Parallelogram : It indicates input/output operation.

Rectangle : It indicates process operation.

Diamond : It indicates decision making operation.

Copyrighted material

Double sided rectangle : It indicates execution of pre-defined process (subroutine).

Circle with alphabet : It indicates continuation.

A: Any alphabet

The Fig. 3.8 shows sample flowchart.

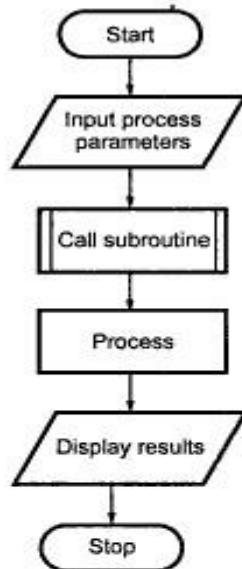


Fig. 3.8 Sample flowchart

3.7.3 Assembly Language Program

Let us define a program statement as 'write an assembly language program to add two numbers'. The three tasks are involved in this program :

- Load two hex numbers
- Add numbers and
- Store the result in the memory

These tasks can be symbolically presented as flowchart, as shown in the Fig. 3.9.

Next job is to find the suitable 8085 assembly language instruction/s for each task. These instructions are as follows :

Task 1 instructions :

```

MVI A, 20H    ; Load 20H as a first
               ; number in register A
MVI B, 40H    ; Load 40H as a second number
               ; in register B
  
```

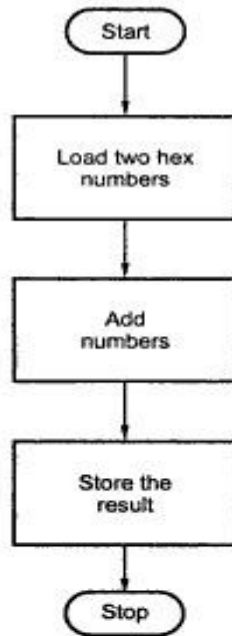


Fig. 3.9 Flowchart for addition of two numbers

Task 2 Instruction :

```

ADD B      ; Add two numbers and save
           ; result in register A
  
```

Task 3 Instruction :

```

STA 2200H  ; Store the result in memory
           ; location 2000H
HLT        ; Stop the program execution
  
```

We want to execute three tasks in a sequence, thus writing corresponding instructions in the same sequence constitutes an assembly language program.

3.7.4 Assembly Language Program to Machine Language Program

Once the assembly language program is ready, it is necessary to convert it in the machine language program. It is possible to do this by referring the proper hex code for each assembly instruction from the 8085 instruction set manual. This process is known as **hand assembly** and the resulted machine language program is also known as **hex code**. Let us see the hex code for our program.

Mnemonics	Hex code
MVI A, 20H	3EH ← Opcode 20H ← Operand

MVI B, 40H	06H ← Opcode
	40H ← Operand
ADD B	80H ← Opcode
STA 2200H	32H ← Opcode
	00H ← Operand (lower byte of address)
	22H ← Operand (higher byte of address)
HLT	76H ← Opcode

3.7.5 Storing Hex Code in the Memory

Once the hex code is ready, it has to be loaded in the memory of specially designed microprocessor system (Microprocessor training kit) for execution. To perform this task we should know the address range of read/write memory in the system. Let us assume that the read/write memory ranges from 2000H to 22FFH. The microprocessor training kit has keypad to enter the hex code in the memory. It provides a special routines (monitor program) to enter a hex code byte by byte and execute the program. Typical steps for storing hex code in the memory from address from address 2000H are as follows :

1. Reset the microprocessor system by pressing the RESET key.
2. Enter into store mode by pressing SET key.
3. Enter the address of the memory 2000H, where the first hex code (starting address of the program) is to be stored using hex keys.
4. Enter the hex code using hex keys.
5. Increment the memory address by 1 using INC key.
6. Repeat steps 4 and 5 until the last hex code.

3.7.6 Executing the Program

The microprocessor training kit provides a procedure to execute the program. To activate the procedure we have to enter the starting address of the program (2000H in our example). To enter this address we have to go into execute mode by pressing GO key and enter the starting address using hex keys. Once the starting address is entered, the program can be executed by pressing EXECUTE key. The EXECUTE key procedure loads the starting address of our program, 2000H into the program counter and program control is transferred from monitor program to our program.

After this microprocessor reads one hex code at a time, and when it fetches the complete instruction, it executes that instruction. Then it fetches the next instruction and this process continues until the last instruction in the program is executed.

4.1 Looping, Counting and Indexing

Before going to implement these techniques, we will get conversant with these techniques and understand their use.

Looping : In this technique, the program is instructed to execute certain set of instructions repeatedly to execute a particular task number of times. For example, to add ten numbers stored in the consecutive memory locations we have to perform addition ten times.

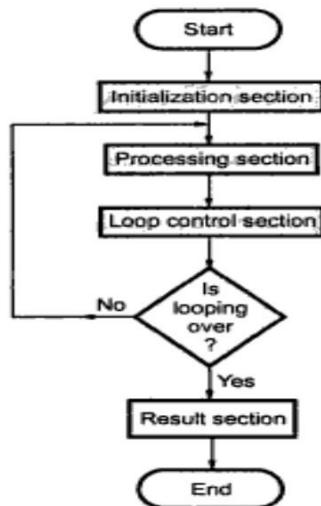
Counting : This technique allows programmer to count how many times the instruction/set of instructions are executed.

Indexing : This technique allows programmer to point or refer the data stored in sequential memory locations one by one. Let us see the program loop to understand looping, counting and indexing.

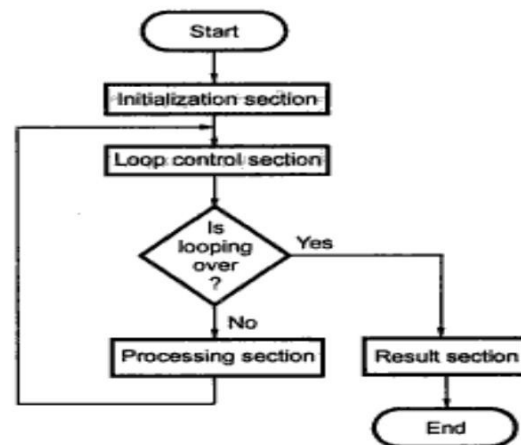
The program loop is the basic structure which forces the processor to repeat a sequence of instructions. Loops have four sections.

1. Initialization section.
2. Processing section.
3. Loop control section
4. Result section.

Flowchart



Flowchart 1



Flowchart 2

1. The initialization section establishes the starting values of
 - loop counters for counting how many times loop is executed,
 - address registers for indexing which give pointers to memory locations and
 - other variables
2. The actual data manipulation occurs in the processing section. This is the section which does the work.
3. The loop control section updates counters, indices (pointers) for the next iteration.
4. The result section analyzes and stores the results.

Note : The processor executes initialization section and result section only once, while it may execute processing section and loop control section many times. Thus, the execution time of the loop will be mainly dependent on the execution time of the processing section and loop control section. The flowchart 1 shows typical program loop. The processing section in this flowchart is always executed at least once. If you interchange the position of the processing and loop control section then it is possible that the processing section may not be executed at all, if necessary. Refer flowchart 2.

5

Stacks and Subroutines

The stack is a part of read/write memory that is used for temporary storage of binary information during the execution of a program. The binary information is basically the intermediate results and the return address in case of subroutine calls.

1. For the application programs, the internal memory of the microprocessor (registers) is not sufficient to store the intermediate results. These intermediate results can be stored temporarily on the stack and can be referred back when required.
2. A subroutine is a group of instructions, performs a particular subtask which is executed number of times. It is written separately. The microprocessor executes this subroutine by transferring program control to the subroutine program. After completion of subroutine program execution, the program control is returned back to the main program. The use of subroutines is a very important technique in designing software for microprocessor systems because it eliminates the need to write a subtasks repeatedly; thus it uses memory more efficiently. For implementation of subroutine technique, it is necessary to define stack. In the stack, the address of the instruction in the main program which follows the subroutine call is stored.

5.1 Concept of Stack and Subroutines

This section explains the concept of stack and subroutines in detail.

5.1.1 Stack

The stack is a portion of read/write memory set aside by the user for the purpose of storing information temporarily. When the information is written on the stack, the operation is called PUSH. When the information is read from stack, the operation is called POP.

The microprocessor stores the information, much like stacking plates. Using this analogy of stacking plates it is easy to illustrate the stack operation.

Fig. 5.1 shows the stacked plates. Here, we realize that if it is desired to take out the first stacked plate we will have to remove all plates above the first plate in the reverse



Fig. 5.1 Stacked plates

order. This means that to remove first plate we will have to remove the third plate, then the second plate and finally the first plate. This means that, the first information pushed on to the stack is the last information popped off from the stack. This type of operation is known as a first in, last out (FILO). This stack is implemented with the help of special memory pointer register. The special pointer register is called the **stack pointer**. During PUSH and POP operation, stack

pointer register gives the address of memory where the information is to be stored or to be read. The stack pointer's contents are automatically manipulated to point to stack top. The memory location currently pointed by stack pointer is called **top of stack**.

The stack pointer SP, is a 16-bit register in the 8085A which is manipulated by the microprocessor's control section, during stack related instructions.

5.1.1.1 Stack Related Instructions

The 8085A supports following stack related instructions :

LXI SP, data (16) :

It initializes stack pointer with 16-bit address.

SPHL :

It copies the contents of HL register pair into the stack pointer.

PUSH rp :

It is used to write 16-bit data in the stack.

POP rp :

It is used to read 16-bit data from the stack.

CALL addr :

It transfers the program control to the subroutine program after storing the return address in the stack.

RET :

It reads the return address from the stack and transfers the program control back to the instruction following the CALL.

Copyrighted material

INX SP :

It increments the contents of stack pointer by one.

DAD SP :

It adds the contents of stack pointer into the contents of HL register pair and stores the result in the HL register pair.

XTHL :

It exchanges the contents of memory location pointed by the stack pointer with the contents of L register and the contents of the next memory location with the contents of H register.

Now we will see the detail operation and the use of these instructions.

5.1.1.2 Detail Operation and the use of Stack Related Instructions**LXI SP, Data and SPHL : Initializes Stack Pointer.**

Before execution of any stack related instruction, stack pointer must be initialized with a valid memory address. The stack pointer can be initialized by two ways.

1. **Direct way :** LXI SP, data (16-bit) ; Loads 16-bit data into SP
2. **Indirect way :** LXI H, data (16-bit) ; Loads 16-bit data into HL
SPHL ; Loads the contents of HL into SP

Normally, the stack pointer is initialized by the direct way. When a programmer wishes to set the stack pointer to a value that has been computed by the program, indirect way is used. The computed value is placed in H and L and the contents of HL register pair then moved into the stack pointer.

Note :

1. The stack pointer can be initialized anywhere in the read/write memory map. However, as a general practice, the stack pointer is initialized at the highest Read/Write memory location so that it will be less likely to interfere with a program.
2. Since the 8085A's stack pointer is decremented before data is written to the stack, the stack pointer can actually be initialized to a value one higher than the highest read/write memory location available.

PUSH and POP : Temporarily stores the contents of register pair and program internal status word, and retrieves when required.

When programmer realizes the shortage of the registers, he stores the present contents of the registers in the stack with the help of PUSH instruction and then uses the registers for other function. After completion of other function programmer loads the previous contents of the register from the stack with the help of POP instruction.

Copyrighted material

PUSH Operation : In PUSH operation, 16-bit data is stored on the stack. This 16-bit data is stored in two operations. In the first operation, stack pointer is decremented by one and then the higher byte of the 16-bit data is stored at the memory location pointed by stack pointer. In the second operation, stack pointer is again decremented by one and then the lower byte of the 16-bit data is stored at the memory location pointed by stack pointer. The Fig. 5.2 shows steps involved in the PUSH operation.

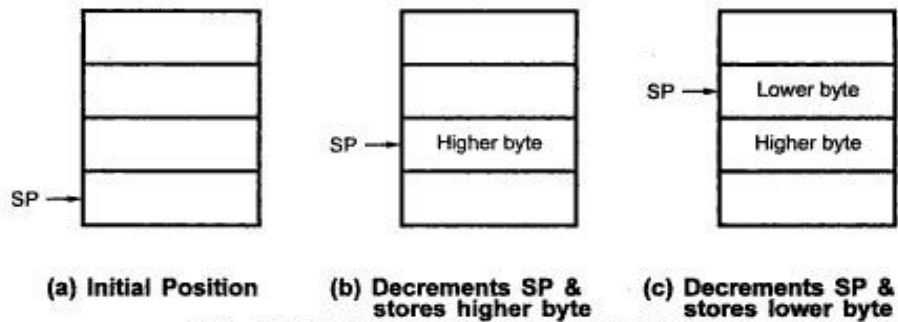


Fig. 5.2 Steps Involved in PUSH Operation

Example 1 : The contents of BC register pair are 1020H and contents of stack pointer are 27FFH. Then after execution of PUSH B instruction the stack contents are as shown in the Fig. 5.3.

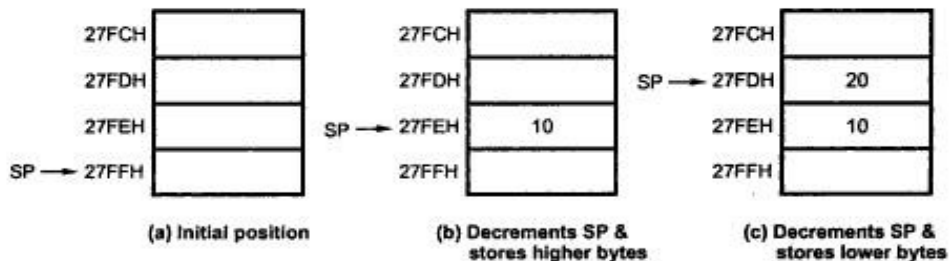


Fig. 5.3 Steps involved in PUSH Operation with example

POP Operation : In POP operation, 16-bit data is read from the stack. This 16-bit data is read in two operations. In the first operation, the contents from the memory location pointed by stack pointer are loaded into lower byte of register pair and then the stack pointer is incremented by one. In the second operation, the contents from the

Copyrighted material

memory location pointed by stack pointer are loaded into higher byte of register pair and then the stack pointer is incremented by one. Fig. 5.4 shows steps involved in the POP operation.

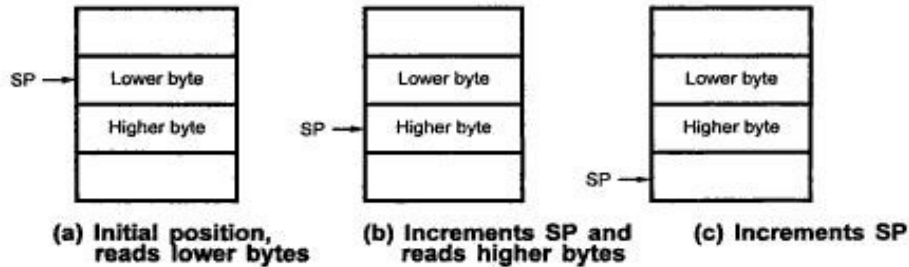


Fig. 5.4 Steps Involved in POP Operation

Example 2 : If the initial contents of stack and contents of stack pointer are as shown in Fig. 5.5 (a) then after execution POP B contents of BC register will be 3040 (B = 30H and C = 40H).

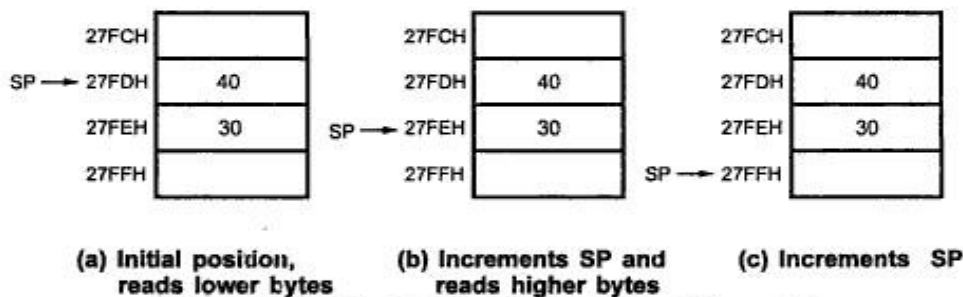


Fig. 5.5 Steps Involved in POP operation with example

Example 3 : Let us consider the following program

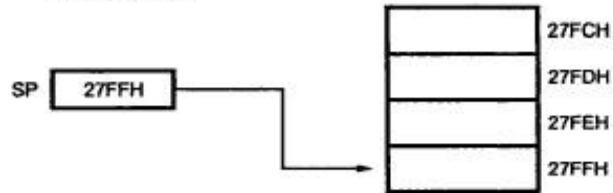
```
LXI SP, 27FFH
LXI B, 2030H
LXI D, 4045H
PUSH D
PUSH B
MOV A, C
ADD E
MOV D, A
POP B
POP D
```

We will see the contents of stack and stack pointer after execution of each instruction in the above program.

Copyrighted material

Instruction 1 :

LXI SP, 27FFH

**Instruction 2 :**

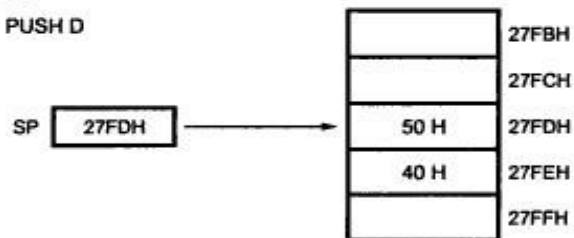
LXI B, 2030H

**Instruction 3 :**

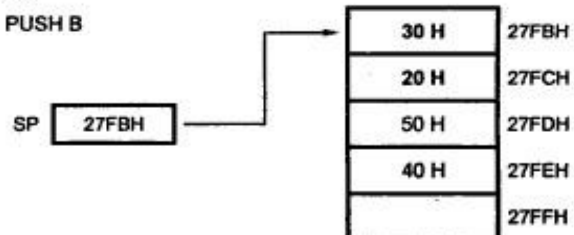
LXI D, 4050H

**Instruction 4 :**

PUSH D

**Instruction 5 :**

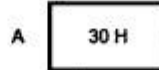
PUSH B



Copyrighted material

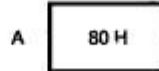
Instruction 6 :

MOV A, C



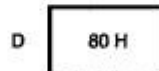
Instruction 7 :

ADD E



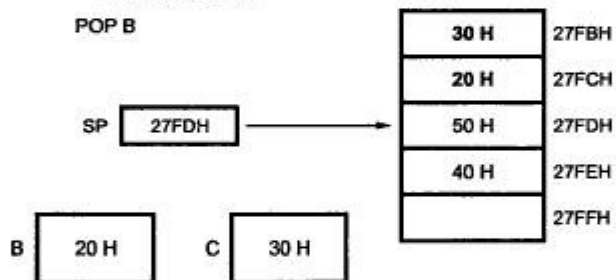
Instruction 8 :

MOV D, A



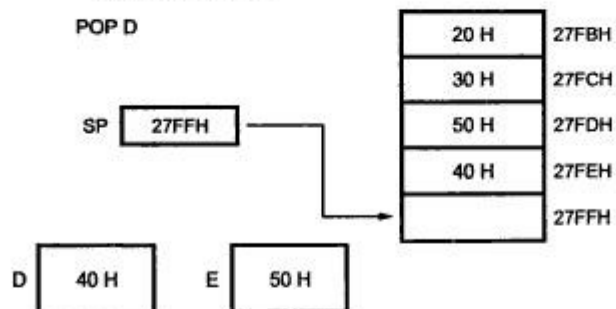
Instruction 9 :

POP B



Instruction 10 :

POP D



This program shows us how registers can be used for more than one purpose. This program initializes the stack pointer at 27FFH and stores the original contents of BC and DE register pairs in the stack. Now registers B, C, D and E are free to use for intermediate

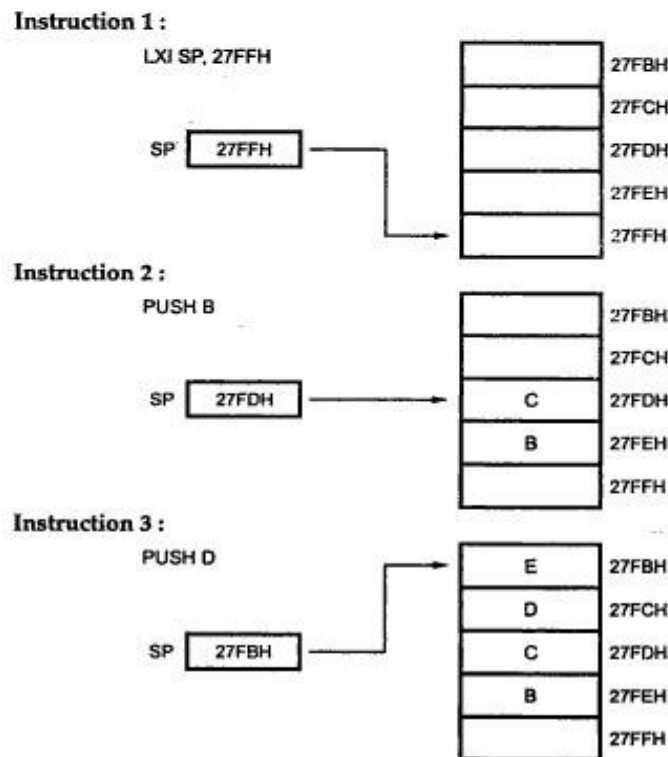
Copyrighted material

calculations. Once these calculations are over, we can get back the original contents of B, C, D and E registers from stack by POP B and POP D instructions. We know that stack works in FILO fashion. Therefore, the sequence of getting the contents back from the stack should be exactly in the reverse order that of the sequence of storing the contents in the stack. In this example the contents of BC register pair are stored after storing the contents of DE register pair in the stack. But while getting the contents of register pairs from the stack, first BC register pair contents are retrieved and then DE register pair contents are retrieved.

➡ **Example 5.1 :** Exchange the contents of BC and DE registers without using any other MPU general purpose register.

Solution : LXI SP, 27FFH
 PUSH B
 PUSH D
 POP B
 POP D

We will see the contents of stack and stack pointer after execution of each instruction in the above program.

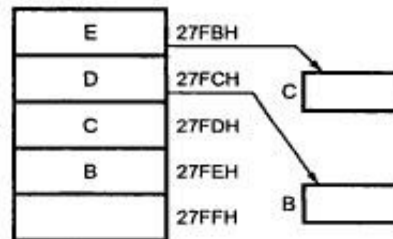


Copyrighted material

Instruction 4 :

POP B

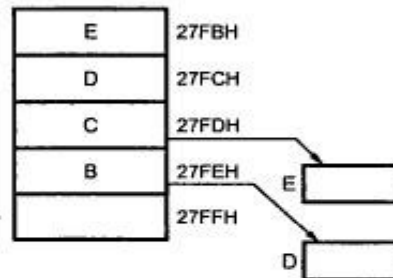
SP 27FDH



Instruction 5 :

POP D

SP 27FFH



This program shows us that if you reverse the order of retrieving the contents from the stack we will not get the original contents of the register pairs. Here, the contents of DE and BC register pairs will be exchanged after execution of the program.

➡ **Example 5.2 :** Write a program to load the flag register contents in C register.

Solution : We know that 8085 does not provide any instruction to transfer the contents of flag register to any general purpose register. Therefore, to load flag register contents into any register, it is necessary to use stack. Following program explains how to use stack to load flag register contents in the C register.

Program :

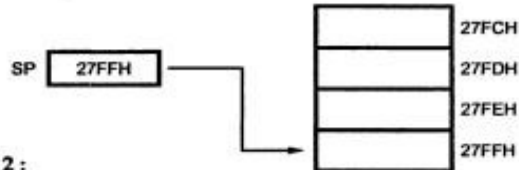
```
LXI SP, 27FFH
PUSH PSW
POP B
```

We will see the contents of stack and stack pointer after execution of each instruction in the program.

Copyrighted material

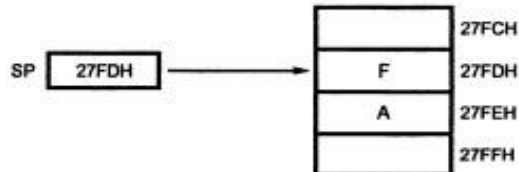
Instruction 1 :

LXI SP, 27FFH



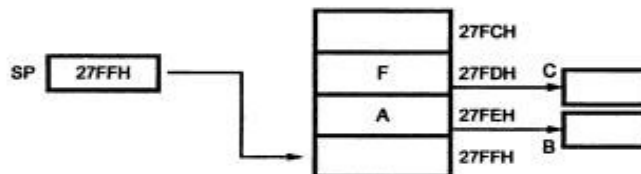
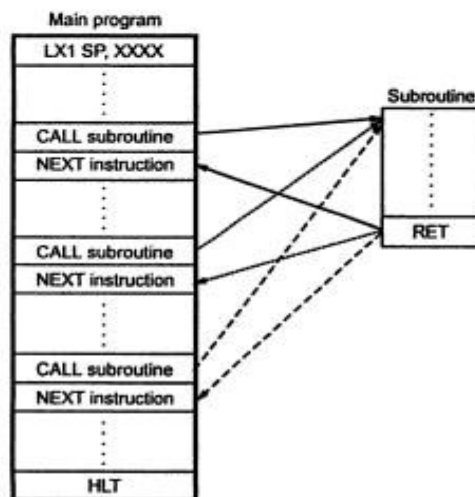
Instruction 2 :

PUSH PSW



Instruction 3 :

POP B

**CALL Address and RET : Implements Subroutines****Fig. 5.6 Program flow**

Whenever we need to use a group of instructions several times throughout a program there are two ways to avoid rewriting of the group of instructions. One way is to write the group of instructions as a separate subroutine. We can then just CALL the subroutine whenever we need to execute that group of instructions. For calling the subroutine we have to store the return address on the stack. This process takes some time. If the group of instructions is bit enough then this overhead time and execution time are comparable. In such cases, it is not desirable to write subroutines. For these cases, we can use macros. Macro is also a group of

Copyrighted material

instructions. Each time we "CALL" a macro in our program, the assembler will insert the defined group of instructions in place of the "CALL". An important point here is that the assembler generates machine codes for the group of instructions each time macro is called. So there is not overhead time involved in calling and returning from a subroutine. The disadvantage of macro is that it generates in line code each time when the macro is called which takes more memory.

5.1.2 Subroutines

From the previous discussions, we know that the subroutine is a group of instructions stored as a separate program in the memory and it is called from the main program whenever required.

The 8085A microprocessor has two instructions to implement subroutines: **CALL** and **RET**. **CALL** instruction is used to call a subroutine in the main program and **RET** instruction is the last instruction in the subroutine to return it back to the main program. The **CALL** instruction saves the address of the instruction following it and then transfers the program control to the first instruction in the subroutine. When subroutine execution is completed the **RET** instruction reads the return address from the stack and transfers control back to the instruction following the **CALL**.

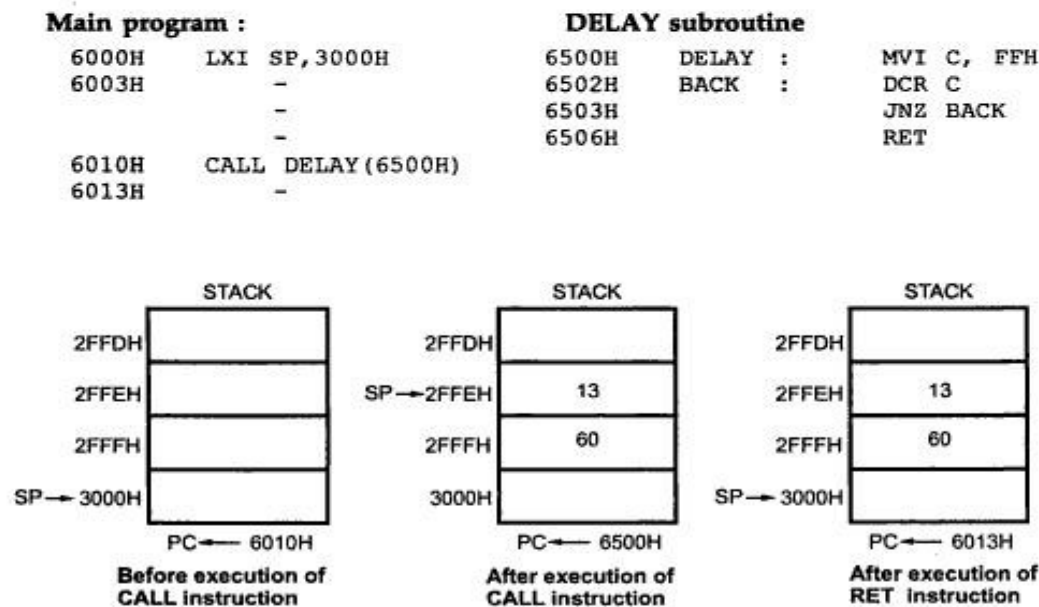


Fig. 5.7 Details in the execution of CALL and RETURN instructions

Copyrighted material

Here, the main program initializes stack pointer at 3000H memory location and executes instructions in sequence till the execution of CALL instruction. After execution of CALL instruction program control is transferred to the delay subroutine stored at memory address 6500H. Before transfer of control to the subroutine the address of the instruction (6013H) which is after the CALL instruction, is stored in the stack. At the end of delay subroutine RET instruction is executed, which reads the return address (6013H) from the stack and transfers the program control to the instruction which is after CALL instruction.

Conditional Call and Return Instructions :

In addition to the unconditional CALL and RET instructions, the 8085A instruction set includes eight conditional CALL instructions and eight conditional RET instructions. These conditions are checked by reading the status of respective flags. If the condition associated with the conditional CALL is not met, the instruction following the CALL is executed. If the condition is met, the program counter contents are saved on the stack, and the address contained in the CALL instruction is loaded into program counter. The number of machine cycles and T-states required by a conditional CALL depends on whether or not the condition is satisfied. When the condition is not satisfied, two machine cycles with a total of nine T-state are required to fetch, decode and execute the instruction. When the condition is satisfied, five machine cycles with 18 T-states are required.

Conditional CALL Instructions

Instruction code	Description	Condition for CALL
CC	Call on carry	CY = 1
CNC	Call on not carry	CY = 0
CP	Call on positive	S = 0
CM	Call on minus	S = 1
CPE	Call on parity even	P = 1
CPO	Call on parity odd	P = 0
CZ	Call on zero	Z = 1
CNZ	Call on not zero	Z = 0

Table 5.1 Conditional calls

Copyrighted material

In case of a conditional return instruction, the sequence returns to the main program if the condition is met otherwise, the sequence in the routine is continued.

Instruction code	Description	Condition for RET
RC	Return on carry	CY = 1
RNC	Return on not carry	CY = 0
RP	Return on positive	S = 0
RM	Return on minus	S = 1
RPE	Return on parity even	P = 1
RPO	Return on parity odd	P = 0
RZ	Return on zero	Z = 1
RNZ	Return on not zero	Z = 0

Table 5.2 Conditions for return

5.2 Parameters Passing Techniques

We often want a subroutine to process some data or address variable from the main program. For processing it is necessary to pass these address variables or data, usually referred to as passing parameters to the subroutine. There are four ways to pass parameters to and from the subroutine :

1. Using registers
2. Using general memory
3. Using pointers
4. Using stack

5.2.1 Passing Parameters using Registers

The data, to be passed is stored in the registers and these registers are accessed in the subroutine to process the data. In this technique, the main program loads internal registers with appropriate values before calling the subroutine and subroutine then obtains these values by referring pre-defined registers, as shown in the following example.

Example :

Passing Parameters Using Registers

```
; Main program
.
.
MVI C, 08H      ; Data to be passed is loaded in the
                ; register
CALL SUB1
MOV A, D        ; Main program accesses result from
                ; register.
.
.
```

Copyrighted material

```

; subroutine
  SUB1:  MOV B, C      ; Subroutine accesses the data from C
                ; register
        .
        MOV D, A      ; Stores result in D register.
        RET

```

5.2.2 Passing Parameters using Memory

For the cases where we have to pass few parameters to and from a subroutine, registers are a convenient way to do it. However in cases where we need to pass a large number of parameters to subroutine we use memory. This memory may be a dedicated section of general memory or a part of stack. In this technique, the main program loads the pre-defined memory locations with appropriate values before calling the subroutine and subroutine then obtains these values by referring these predefined memory locations, as shown in the following example.

Example :

Passing Parameters Using General Memory

```

; Main program
      LXI H, 2200H ; Initialize memory pointer
      MVI M, 50H  ; Load data into memory
      .
      CALL SUB1
      LDA 2300H   ; Main program accesses result from
                ; memory location 2300H.
      .
; Subroutine
  SUB1:  LDA 2200H ; Subroutine accesses the data from
                ; memory location.
      .
      STA 2300H   ; Stores result in memory location
                ; 2300H
      RET

```

The subroutine program stores the generated results in the memory locations before execution of RET. The main program accesses results from these memory locations for further processing.

5.2.3 Passing Parameters using Pointers

In this technique, the main program stores the parameters to be passed in the memory, usually in the consecutive memory locations. Then it loads the internal register pair or pre-defined memory locations with the starting address of the parameter list. The subroutine obtains parameter list by accessing it in sequence from the given address.

Example :

Passing Parameters Using Pointers

```

; Main program

```

Copyrighted material

```

    LXI H, 2200H      ; Initialize memory
    MOV M, A
    INX H
    MOV M, B
    :
    :
    INX H
    MVI M, 30H
    :
    SHLD 3000H        ; Store memory pointer
    CALL SUB1
    :
; Subroutine
SUB1: LHL 3000H        ; Subroutine access pointer (address) to data
      MOV A, M
      :
      :
      RET

```

5.2.4 Passing Parameters using Stack

The stack can be used to pass parameters. To pass parameters to the subroutine using stack, it is necessary to push them on the stack before the call for the subroutine in the main program. The instructions in the subroutine read these parameters from the stack. Whenever the stack is used to pass parameters, it is very important to keep track of what is pushed on the stack and where the stack pointer points all the time in the program.

Example :

Passing Parameters Using Stack

```

; Main program
LXI B, 1020H ; Load BC register pair with 1020H
PUSH B       ; Store BC register pair on stack.
CALL SUB1
:
; Subroutine
SUB1
POP H        ; Store the return address into HL register
             ; pair.
POP B        ; The subroutine accesses data from stack using
             ; BC register pair.
PCHL         ; Here it is not possible to use return
             ; instruction, since stack pointer not pointing
             ; the return address. But the address is available
             ; in HL. Thus the PCHL instruction is used to
             ; transfer program control to the main program.

```

Example : Write a program to exchange the higher and lower nibble of ten 8-bit numbers stored from location 2200H. Make use of subroutine and explain different parameter passing techniques.

Copyrighted material

Solution :**a) Parameter passing using register A****Main program :**

```

        LXI SP, 27FFH    ; Initializes stack pointer
        LXI H, 2200H    ; Initializes memory pointer
        MVI C, 0AH      ; Initializes counter
BACK:   MOV A, M          ; Stores number (passing parameter)
                          ; in A register
        CALL EXCHANGE    ; Calls subroutine exchange
        MOV M, A          ; Stores the result from register A
        INX H            ; Increments memory pointer
        DCR C            ; Decrements counter
        JNZ BACK         ; If not zero, repeat
        HLT              ; Stop

```

Subroutine program :

```

EXCHANGE: RLC
          RLC
          RLC
          RLC              ; Rotate 4 times left to exchange
                          ; the nibbles
          RET              ; Return to the main program

```

b) Passing parameters using memory location**Main program :**

```

        LXI SP, 27FFH    ; Initializes stack pointer
        LXI H, 2200H    ; Initializes memory pointer
        MVI C, 0AH      ; Initializes counter
BACK :   MOV A, M          ; Gets the number
        STA 2300H        ; Stores the number (parameter) at
                          ; 2300H
        CALL EXCHANGE    ; Calls subroutine exchange
        LDA 2300H        ; Gets the result
        MOV M, A          ; Stores the result
        INX H            ; Increments memory pointer
        DCR C            ; Decrements counter
        JNZ BACK         ; If not zero, repeat
        HLT              ; Stop

```

Subroutine program :

```

EXCHANGE: LDA 2300H      ; Gets the parameter
          RLC
          RLC
          RLC
          RLC              ; Rotate 4 times left to exchange
                          ; the nibbles
          STA 2300H      ; Store the result
          RET              ; Return to main program

```

c) Passing parameters using pointer

Copyrighted material


```

        LXI SP, 27FFH    ; Initializes stack pointer
        LXI H, 2200H    ; Initializes memory pointer
        MVI C, 0AH      ; Initializes counter
BACK:   SHLD 2300H       ; Store pointer to passing
        ; parameter
        CALL EXCHANGE   ; Calls subroutine exchange
        INX H           ; Increments memory pointer
        DCR C           ; Decrements counter
        JNZ BACK        ; If not zero, repeat
        HLT            ; Stop

```

Subroutine program :

```

EXCHANGE: LHL 2300H      ; Gets pointer to parameter
          MOV A, M       ; Gets number
          RLC
          RLC
          RLC
          RLC            ; Rotate 4 times left to exchange
                        ; the nibbles
          MOV M, A       ; Stores the result
          RET            ; Return to main program.

```

d) Passing parameters using stack.

```

        LXI SP, 27FFH    ; Initializes stack pointer
        LXI H, 2200H    ; Initializes memory pointer
        MVI C, 0AH      ; Initializes counter
BACK :   MOV A, M        ; Gets the number
        PUSH PSW        ; Stores number in stack as a
        ; parameter
        CALL EXCHANGE   ; Calls subroutine exchange
        MOV M, A        ; Stores the result
        INX H           ; Increments memory pointer
        DCR C           ; Decrements counter
        JNZ BACK        ; If not zero repeat
        HLT            ; Stop

```

Subroutine program :

```

EXCHANGE: POP H          ; Stores return address in HL
          POP PSW        ; Gets the number in accumulator
          RLC
          RLC
          RLC
          RLC            ; Rotate 4 times left to exchange
                        ; the nibbles
          PCHL           ; Return to main program.

```

5.3 Subroutine Documentation

Subroutine program must provide enough information so that other users can utilize the subroutine without having to examine its internal structure. So along with subroutine program it is necessary to give the following guidelines.

Copyrighted material

1. Description of the purpose of the subroutine
2. A list of passing parameters
3. Return value
4. Registers and memory and memory locations used
5. Sample code

If these guidelines are followed while writing the subroutine then the subroutine can be easily used as a library function in other applications if required.

5.4 Advanced Subroutine Concepts

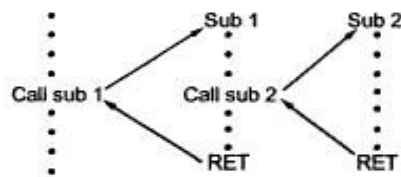


Fig. 5.8 Transfer of control with nested subroutines

When one subroutine calls another subroutine to complete a particular task, the operation is called **nesting**. The second subroutine may in turn call a third subroutine and so on each successive CALL without an intervening return creates an additional level of nesting. These routines are called **nested subroutines**. Fig. 5.8 shows the transfer of control with nested subroutines.

Nested subroutines are commonly classified as :

- Re-entrant Subroutine
- Recursive Subroutine

5.4.1 Re-entrant Subroutine

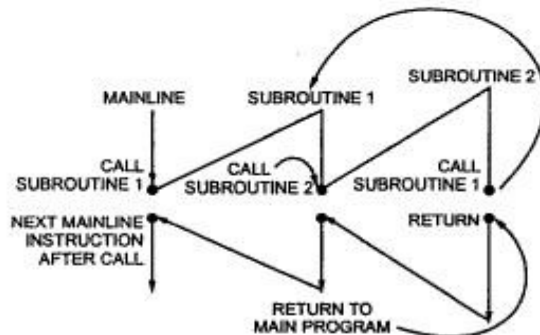


Fig. 5.9 Flow of program execution for re-entrant subroutine

In some situations it may happen that subroutine1 is called from main program, subroutine2 is called from subroutine1 and subroutine1 is again called from subroutine2. In this situation program execution flow re-enters in the subroutine1. This type of subroutines are called **re-entrant subroutines**. The flow of program execution for re-entrant subroutine is shown in Fig. 5.9.

Copyrighted material

5.4.2 Recursive Subroutine

A recursive subroutine is a subroutine which calls itself. Recursive subroutines are used to work with complex data structures called trees. If the subroutine is called with N (recursion depth) = 3, then the N is decremented by one after each subroutine CALL and the subroutine is called until $N = 0$. Fig. 5.10 shows the flow diagram and pseudo code for recursive subroutine.

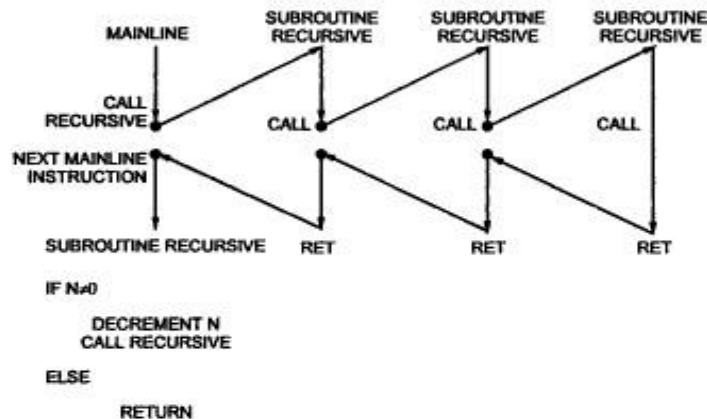


Fig. 5.10 Flow diagram and pseudo code for recursive subroutine

Limitations :

The level of subroutine nesting cannot exceed that supported by the available stack memory in the system. Since the stack pointer is usually initialized to the highest available read/write memory location, and the stack grows toward lower addresses, it is possible that too many PUSH or CALL operations without a sufficient number of intervening POP or RET operations result in the overflow of stack. For example, assume that read/write memory allocated in the system for stack has address range from 2000H to 27FFH. In this case, if stack grows beyond 2000H then it is called as overflow of stack. Care must be taken in memory allocation and program design to prevent this.

Lab Experiment 61 : Solve given equation.

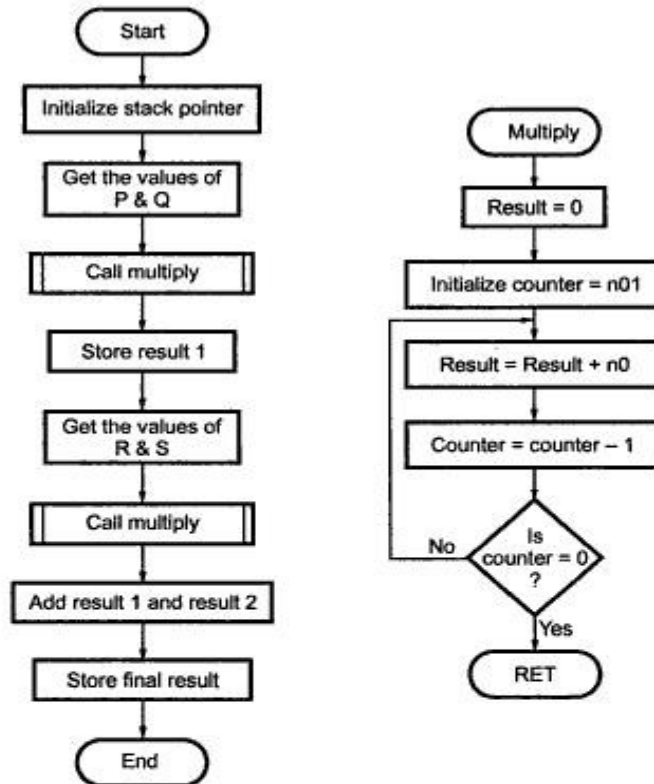
Statement : Write a program to solve following equation

$$X = (P \times Q) + (R \times S)$$

Assume P , Q , R and S numbers are stored at 2200H, 2201H, 2202H and 2203H memory locations respectively. Store the result in memory locations 2204H and 2205H. Make use of subroutine.

Copyrighted material

Flowchart :



Main program :

```

LXI SP, 27FFH ; Initialize stack pointer
LXI H, 2200H ; Initialize memory pointer
CALL MULTIPLY ; Call subroutine multiply
SHLD 2204H ; Store the result 1
LXI H, 2202H ; Set memory pointer for next two numbers
CALL MULTIPLY ; Call subroutine multiply
XCHG ; Save result in DE register pair
LHLD 2204H ; Get the result 1
DAD D ; Add result 1 and result 2
SHLD 2204H ; Store final result
HLT ; Stop
  
```

Multiply subroutine :

```

MULTIPLY: MOV C, M ; Initialize number1 as a counter
          MVI D, 00H ;
  
```

Copyrighted material

```

                INX H          ;
                MOV E, M       ; Get number 2
                LXI H, 0000H    ; Result = 0
BACK :          DAD D          ; Result = Result + number2
                DCR C          ; Decrement counter
                JNZ BACK       ; If not zero, repeat
                RET            ; Return to main program

```

Review Questions

1. What is stack ? Explain the use of the stack, and stack pointer and how they are affected by instructions such as PUSH, POP, CALL and RET.
2. What is subroutine ? How is it useful ? Explain the use of stack in CALL and RETURN instructions.
3. The first four instructions of a typical subroutine are :


```

                PUSH PSW,      PUSH H
                PUSH B,        PUSH D
      
```

 What will be the last five instructions of the subroutine? Explain clearly.
4. If the CALL and RET instructions are not provided in the 8085, could it be possible to write subroutines for this microprocessor ? If so how will you call and return from the subroutine ?
5. Discuss the passing parameters techniques in the subroutine.
6. Explain the necessity of subroutine documentation.
7. What do you mean by nested subroutines ?
8. Explain the re-entrant subroutine.
9. Explain the recursive subroutine with the help of example.

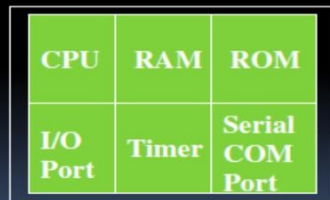
□□□

Copyrighted material

UNIT III-8051 MICROCONTROLLERS

INTRODUCTION

- A smaller computer
- On-chip RAM, ROM, I/O ports...
- Example : Intel's 8051, Motorola's 6811, Zilog's Z8 and PIC 16X



← A single chip

Microcontroller

M.M.Arun Prasath., AP/ECE 26 February 2014

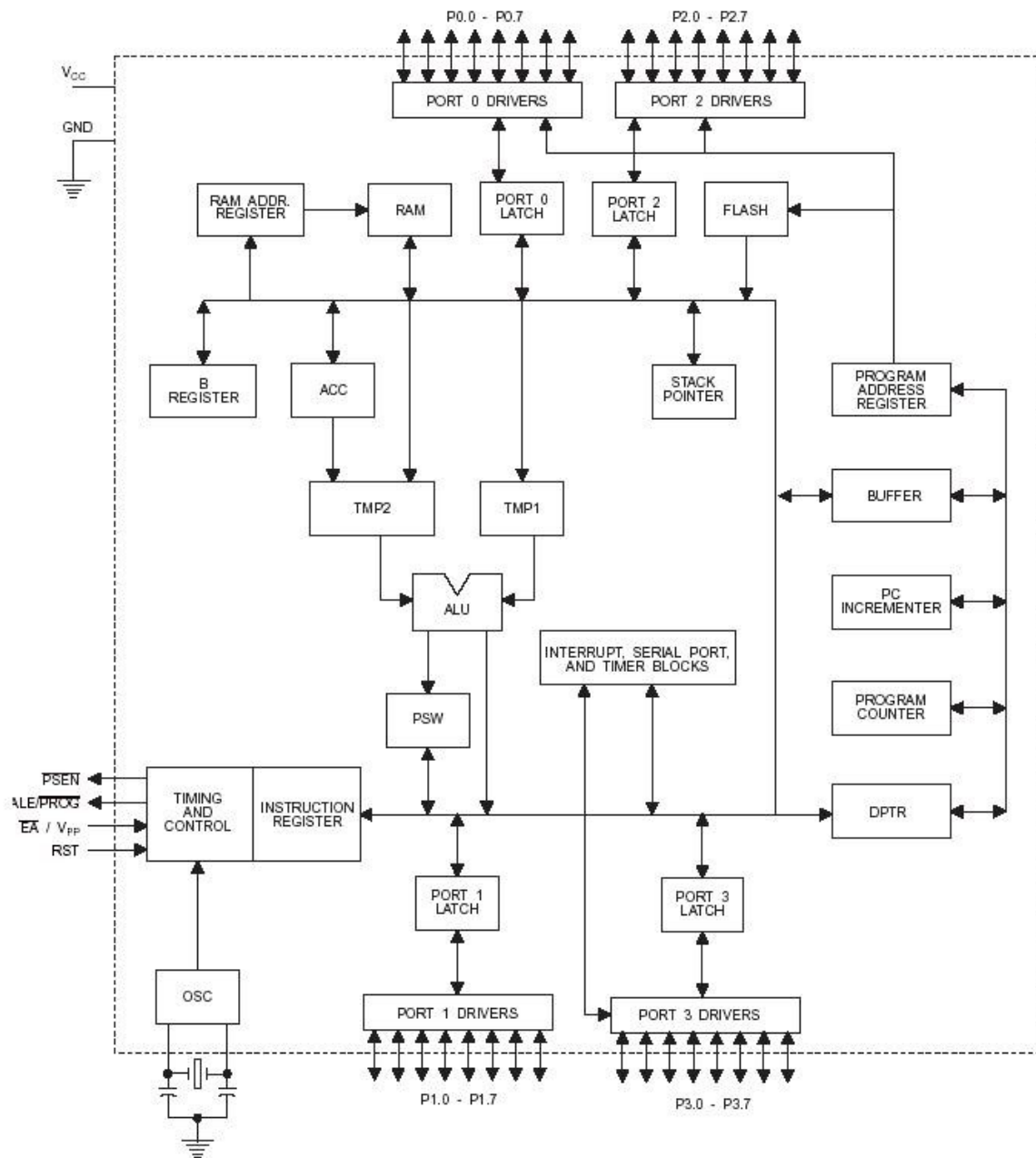
2

ARCHITECTURE 8051

M.M.Arun Prasath., AP/ECE 26 February 2014

8

Block Diagram



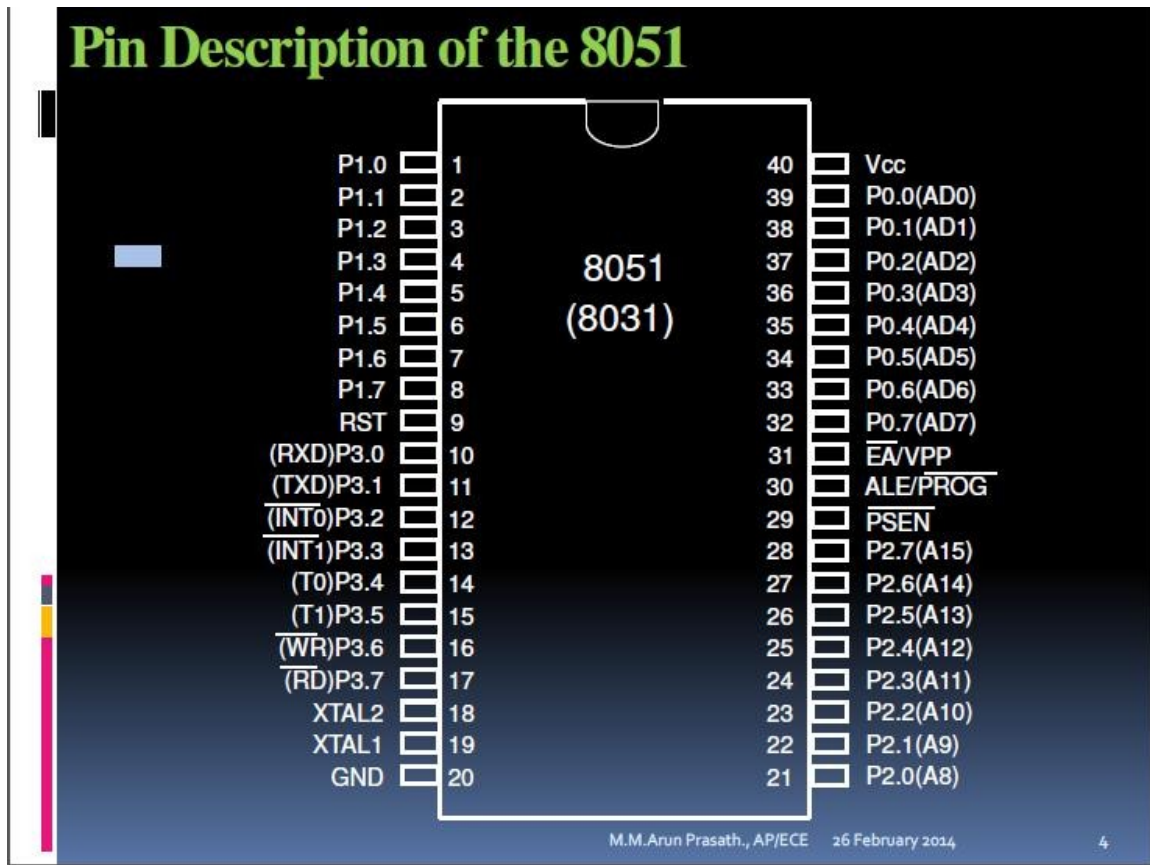
Architecture of 8051 Microcontroller

- **Accumulator:** 8-bit register. Arithmetic operations.
- **B Register:** 8-bit register. General purpose register.
- **Program Status Word:** Set of flags contains the status information. It is one of the SFR.
- **Stack Pointer:** 8-bit register.
- **Data Pointer:** 16-bit register contains a high byte (DPH) and low byte (DPL). It has been allotted two addresses in the SFR bank for its two bytes DPH and DPL.
- **Port 0 to 3 Latches and Drivers:** These 4 latches and driver pairs are allotted to each of the 4 on-chip I/O ports. These latches have been allotted addresses in the SFR bank. Using the allotted address the user can communicate with these ports (P_0, P_1, P_2, P_3)
- **Serial Data Buffer:** It contains 2 independent register one of them is transmit buffer, which is a parallel-in-serial-out register. The other is a receive buffer, which is a serial-in-parallel-out register. It is one of the SFR.
- **Timer Registers:** Two 16-bit registers. TLO, TH0 represent the lower and higher byte of timer register 0, similarly TL1, TH1 represent the lower and higher byte of timer register 1.

M.M.Arun Prasath., AP/ECE 26 February 2014

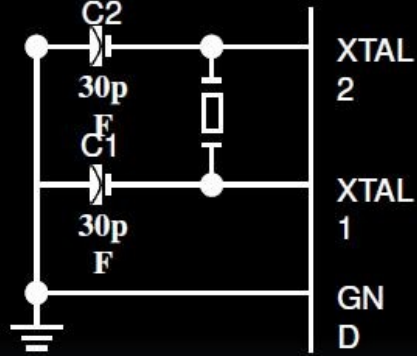
10

- **Control Registers:** It contains control and status information for interrupts
- **Timing and Control Unit:** It derives all the necessary timing at control signals register for internal operation of the circuit.
- **Oscillator:** It generates the basic timing clock signal for the operation of the circuit using crystal oscillator.
- **Instruction Register:** This register decodes the opcode of an instruction to be executed and gives information to the timing and control unit to generate necessary signals on the execution of instruction.
- **EPROM and Program Address Register:** It provides an on chip EPROM and a mechanism to internally address it.
- **RAM and RAM Address Register:** It provide internal 128 bytes of RAM and a mechanism to address it internally.
- **ALU:** It performs 8-bit arithmetic, logical operations over the operands held by temporary register TMP1 and TMP2. users can't access these temp. register.
- **SFR Register Bank:** Special Function register range 80H to FFH.



Pins of 8051

- **VCC (pin 40)** : VCC provides supply voltage to the chip. [+5V]
- **GND (pin 20)** : ground
- **XTAL1 and XTAL2 (pins 19,18)** :
 - These 2 pins provide external clock frequency for the operation.



- **RST (pin 9)** : reset
 - It is a power-on reset.
 - Upon applying a high pulse to RST, the microcontroller will reset and all values in registers will be lost

Pins of 8051

- **EA⁻ (pin 31)** : external access
 - When the pin is high program fetches to address 0000H-0FFFH directed to internal ROM and program fetches to address 1000H-FFFFH are directed to external ROM/EPROM.
 - When the pin is low all addresses (0000H - FFFFH) fetched by program are directed to the external ROM/PROM.
- **PSEN⁻ (pin 29)** : program store enable

It is the active low output control signal used to activate the enable signal of the external ROM/PROM.
- **ALE (pin 30)** : address latch enable
 - AD0-AD7 are multiplexed. To demultiplex these lines and for obtaining lower half of an address, an external latch and ALE signal of 8051 is used.

Pins of 8051

- I/O port pins
 - The four ports P0, P1, P2, and P3.
 - Each port uses 8 pins.
 - All I/O pins are bi-directional.
 - Port 0 (pins 32-39) used as a multiplexed address/data bus.
 - Port 1 (pins 1-8) used only as I/O pins.
 - Port 2 (pins 21-28) used to access external memory when the address is 16bit wide otherwise port2 is used as an I/O port.
 - Port 3 (pins 10-17) multifunctional port pins. It can be programmed to use as I/O or as one of the alternate function. It includes 2 external interrupts, 2 counter inputs, 2 special data lines and 2 timing control.

INTERRUPTS

- Interrupts may be generated by the internal chip operations or provided by the external sources.
- 5 interrupts are available in 8051
- 3 are generated automatically by the internal operations: Timer flag 0, Timer flag 1 & Serial port interrupt (RI or TI)
- 2 are triggered by external signals provided by the circuitry that is connected to pins $INT0$ and $INT1$
- Programmer is able to alter control bits in the IE, IP and TCON.

M.M.Arun Prasath., AP/ECE 26 February 2014

36

INTERRUPT ENABLE (IE)

7	6	5	4	3	2	1	0
EA	--	ET2	ES	ET1	EX1	ET0	EX0

- **EA:** Enable interrupt bit.
- **Bit 6:** not implemented
- **ET2:** future use
- **ES:** Enable serial port interrupt
- **ET1:** Enable Timer 1 overflow interrupt
- **EX1:** Enable external interrupt 1
- **ET0:** Enable Timer 0 overflow interrupt
- **EX0:** Enable external interrupt 0

M.M.Arun Prasath., AP/ECE 26 February 2014

37

INTERRUPT PRIORITY (IP)

7	6	5	4	3	2	1	0
--	--	PT2	PS	PT1	PX1	PT0	PX0

- **Bit 7:** not implemented
- **Bit 6:** not implemented
- **PT2:** future use
- **PS:** Priority of serial port interrupt
- **PT1:** Priority of Timer 1 overflow interrupt
- **PX1:** Priority of external interrupt 1
- **PT0:** Priority of Timer 0 overflow interrupt
- **PX0:** Priority of external interrupt 0

M.M.Arun Prasath., AP/ECE 26 February 2014

38

UNIT IV

8255

PROGRAMMABLE PERIPHERAL INTERFACE - 8255

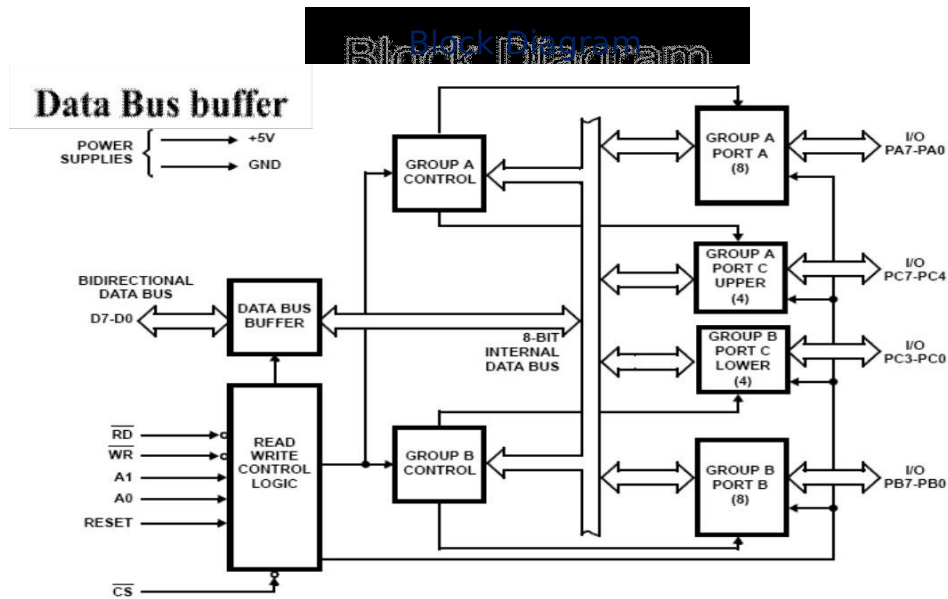
Features

- Parallel Communication Interface port (OR)
- PIO Programmable Input Output Port (OR)
- It is a programmable device. (OR)
- It has 24 I/O programmable pins like PA, PB, PC (8 pins).
- TTL compatible.
- Improved dc driving capability
- Bit set/Reset

8255 PPI

consisting of a number of 8-bit parallel I/O ports (i.e. PORT A, PORT B, PORT C). The ports can be programmed to function either as a input port or as a output port in different operating modes. It requires 4 internal addresses and has one logic LOW chip select pin. Its main functions are to interface peripheral devices to the microprocessor. Basically used for parallel data transfer. operates in mainly two modes.

- (1) Bit Set Reset Mode (BSR Mode).
- (2) I/O Mode.



Data Bus buffer:

- It is a 8-bit bidirectional Data bus.
- Used to interface between 8255 data bus with system bus.

- The internal data bus and Outer pins D₀-D₇ pins are connected internally.
- The direction of data buffer is decided by Read/Control Logic.

Read/Write Control Logic:

- Address signals are A₀, A₁ and CS.
- Control signal are RD and WR.
- Address signals are A₀, A₁ and CS.
- 8255 operation is enabled or disabled by CS.

Group A and Port A and Port C

Group A and Port A and Port C

Signal from CPU & send the command to the individual control blocks.

- Group A send the control signal to port A and Port C (Upper) PC7-PC4.
- Group B send the control signal to port A and Port C (Lower) PC3-PC0.

PORT A:

- This is a 8-bit buffered I/O latch.
- It can be programmed by mode 0, mode 1, mode 2.

PORT B:

- This is a 8-bit buffer I/O latch.
- It can be programmed by mode 0 and mode 1.

PORT C:

- This is a 8-bit Unlatched buffer Input and an Output latch.

- It is splitted into two parts.

- It can be programmed by bit set/reset operation.

Function of Blocks

Data Bus Buffer	It is used to interface the internal data bus of 8255 to the system data bus by reading and writing operations.
Read/write Control logic	It accepts the input from the address bus and issues commands to the individual group blocks. also issues appropriate enabling signals to access the required data/control words/status words.
Port A	It can be programmed in three modes Mode0, Mode1 and Mode2.
Port B	It can be programmed in three modes Mode0 and Mode1.
Port C	It can be programmed for Bit Set/reset operation.

It has a 40 pins of 4 groups.

1. Data bus buffer
2. Read Write control logic
3. Group A and Group B controls
4. Port A, B and C

DATA BUS BUFFER:

.This is a tri state bidirectional buffer used to interface the 8255 to system data bus. Data is transmitted or received by the buffer on execution of input or output instruction by the CPU.

- Control word and status information are also transferred through this unit.

Read/Write control logic

This unit accepts control signals (RD, WR) and also inputs from address bus and issues commands to individual group of control blocks (Group A, Group B).It has the following pins.

- a) CS** – Chipselect : A low on this PIN enables the communication between CPU and 8255.
- b) RD** (Read) – A low on this pin enables the CPU to read the data in the ports or the status word through data bus buffer.
- c) WR** (Write) : A low on this pin, the CPU can write data on to the ports or on to the control register through the data bus buffer.
- d) RESET**: A high on this pin clears the control register and all ports are set to the input mode
- e) A0 and A1** (Address pins): These pins in conjunction with RD and WR pins control the selection of one of the 3 ports.

Group A and Group B controls

These block receive control from the CPU and issues commands to their respective ports.

- Group A - PA and PCU (PC7 –PC4)
- Group B - PCL (PC3 – PC0)
- Control word register can only be written into no read operation of the CW register is allowed.

PORTS

- a) **Port A:** This has an 8 bit latched/buffered O/P and 8 bit input latch. It can be programmed in 3 modes – mode 0, mode 1, mode 2.
- b) **Port B:** This has an 8 bit latched / buffered O/P and 8 bit input latch. It can be programmed in mode 0, mode1.
- c) **Port C :** This has an 8 bit latched input buffer and 8 bit out put latched/buffer. This port can be divided into two 4 bit ports and can be used as control signals for port A and port B. it can be programmed in mode 0.

PIN CONFIGURATION

PA3	1		40	PA4
PA2	2		39	PA5
PA1	3		38	PA6
PA0	4		37	PA7
\overline{RD}	5		36	\overline{WR}
\overline{CS}	6		35	RESET
gnd	7		34	D0
A1	8		33	D1
A0	9		32	D2
PC7	10	8255	31	D3
PC6	11	PPI	30	D4
PC5	12		29	D5
PC4	13		28	D6
PC0	14		27	D7
PC1	15		26	V _{cc}
PC2	16		25	PB7
PC3	17		24	PB6
PB0	18		23	PB5
PB1	19		22	PB4
PB2	20		21	PB3

Function of pins:

- **Data bus(D₀-D₇):** These are 8-bit bi-directional buses, connected to 8085/8086 data bus for transferring data.
- **CS:** This is Active Low signal. When it is low, then data is transfer from 8085/8086.
- **\overline{RD} :** This is Active Low signal, when it is Low read operation will be start.
- **\overline{WR} :** This is Active Low signal, when it is Low Write operation will be start.
- **Address (A0-A1):** This is used to select the ports. like this

A1	A0	Select
0	0	PA
0	1	PB
1	0	PC
1	1	Control reg.

control registers.

- **RESET:** This is used to reset the device. That means clear control registers.
- **the data to peripheral or to send**
- **PA₀-PA₇:** It is the 8-bit bi-directional I/O pins used to send the data to peripheral or to receive the data from peripheral.
- **PB₀-PB₇:** Similar to PA.
- **(Higher groups) PC₀-PC₇:** This is also 8-bit bidirectional I/O pins. These lines are divided into two groups.
 1. PC₀ to PC₃(Lower Groups)
 2. PC₄to PC₇(Higher groups)




These two groups working in separately using 4 data's.

CS	A1	A0	RD	WR	Operation
1	x	x	x	X	Tri state
0	0	0	0	1	Port A to Data bus
0	0	1	0	1	Port B to Data bus
0	1	0	0	1	Port C to Data bus
0	1	1	0	1	Control Word to Data bus
0	0	0	1	0	Data bus to Port A
0	0	1	1	0	Data bus to Port B
0	1	0	1	0	Data bus to Port C
0	1	1	1	0	Data bus to Control Word

Function of Pins

PIN	FUNCTION OF PIN
D0-D7 (Data Bus)	These are bidirectional, tri-state data bus lines are connected to the system data bus. They are used to transfer data and control word from microprocessor (8085) to 8255 or receive data or status word from 8255 to the 8085.
PA0-PA7 (Port A)	These are 8 Bit bidirectional I/O pins used to send data to output device and to receive data from input device. It functions as an 8 Bit data output latch/buffer when used in output mode and as an 8 Bit data input latch/buffer when used in input mode.
PB0-PB7 (Port B)	These are 8 Bit bidirectional I/O pins used to send data to output device and to receive data from input device. It functions as an 8 Bit data output latch/buffer when used in output mode and as an 8 Bit data input latch/buffer when used in input mode.

Function of Pins

PIN	FUNCTION OF PIN
PC0-PC7 (Port C)	These are 8 bit bidirectional I/O pins divided into two groups PCL (PC3-PC) and PCU (PC7-PC4).these groups can individually transfer data in or out when programmed for simple I/O, and used as handshake signals when programmed for handshake or bidirectional modes.
 RD	When this pin is low, the CPU can read data in the ports or the status word through the data bus buffer.
 WR	When this pin is low, the CPU can write data on the ports or in the control register through the data bus buffer.
 CS	This pin can be enabled for data transfer operation between the CPU and 8255.
RESET	This pin is used to reset 8255.i.e control register gets cleared and all the ports are set to the input mode.

Function of Pins

A0-A1

The selection of input port and control word register is done by using A0 and A1 pins in conjunction with RD and WR pins.

0	0	0	1	0	PORT A TO DATA BUS
0	1	0	1	0	PORT B TO DATA BUS
1	0	0	1	0	PORT C TO DATA BUS
0	0	1	0	0	DATA BUS TO PORT A
0	1	1	0	0	DATA BUS TO PORT B
1	0	1	0	0	DATA BUS TO PORT C
1	1	1	0	0	DATA BUS TO CONTROL REGISTER
x	x	x	x	1	DATA BUS TRI STATED
1	1	0	1	0	ILLEGAL CONDITION
x	x	1	1	0	DATA BUS TRI STATED

OPERATING MODES OF 8255

Modes of Operation of 8255

- These are two basic modes of operation of 8255. I/O mode and Bit Set-Reset mode (BSR).
- In I/O mode, the 8255 ports work as programmable I/O ports, while in BSR mode only port C (PC0-PC7) can be used to set or reset its individual port bits.
- Under the I/O mode of operation, further there are three modes of operation of 8255, so as to support different types of applications, mode 0, mode 1 and mode 2.

- **BSR Mode:** In this mode any of the 8-bits of port C can be set or reset depending on D0 of the control word. The bit to be set or reset is selected by bit select flags D3, D2 and D1 of the CWR as given in table.

D ₃	D ₂	D ₁	Selected bits of port C
0	0	0	D ₀
0	0	1	D ₁
0	1	0	D ₂
0	1	1	D ₃
1	0	0	D ₄
1	0	1	D ₅
1	1	0	D ₆
1	1	1	D ₇

Mode 0

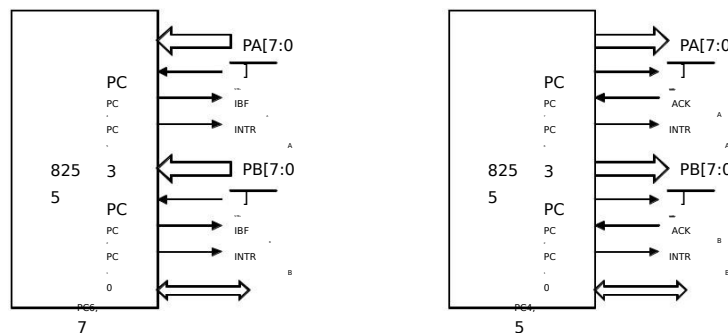
• I/O Modes :

- a) Mode 0 (Basic I/O mode):** This mode is also called as basic input/output mode. This mode provides simple input and output capabilities using each of the three ports. Data can be simply read from and written to the input and output ports respectively, after appropriate initialization.

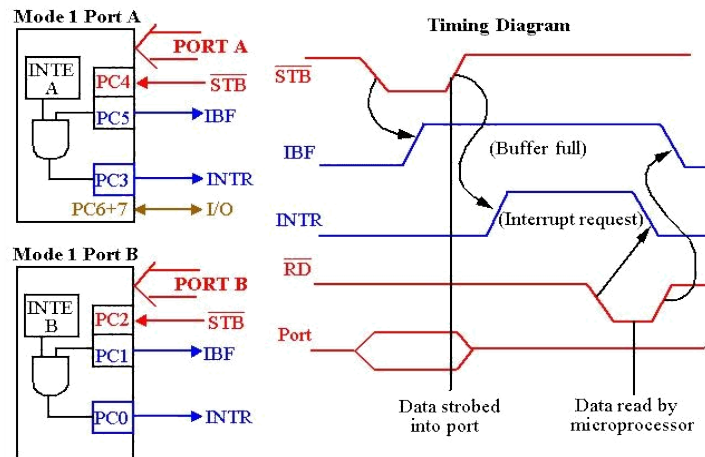
Programming 8255

• Mode 1:

- Ports A and B are programmed as input or output ports
- Port C is used for handshaking



- STB** The strobe input loads data into the port latch on a 0-to-1 transition
- IBF** Input buffer full is an output indicating that the input latch contain information
- INTR** Interrupt request is an output that requests an interrupt
- INTE** The interrupt enable signal is neither an input nor an output; it is an internal bit programmed via the PC4(port A) or PC2(port B) bits.
- PC7,PC6** The port C pins 7 and 6 are general-purpose I/O pins that are available for any purpose.



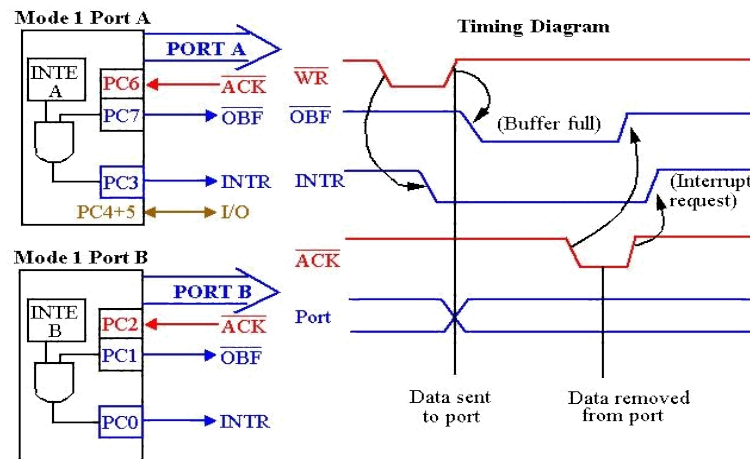
INTR Interrupt request is an output that requests an interrupt.

ACK The acknowledge signal causes the OBF pin to return to 0. This is a response to the external device.

INTR Interrupt request is an output that requests an interrupt.

INTE The interrupt enable signal is neither an input nor an output; it is an internal bit programmed via the PC4(port A) or PC2(port B) bits.

PC7,PC6 The port C pins 7 and 6 are general-purpose I/O pins that are available for any purpose.

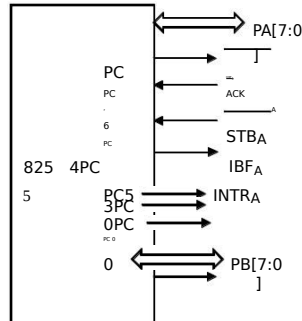


Programming 8255

• Mode 2:

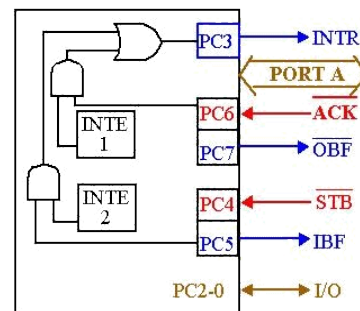
- Port A is programmed to be bi-directional
- Port C is for handshaking
- Port B can be either input or output in mode 0 or

mode 1



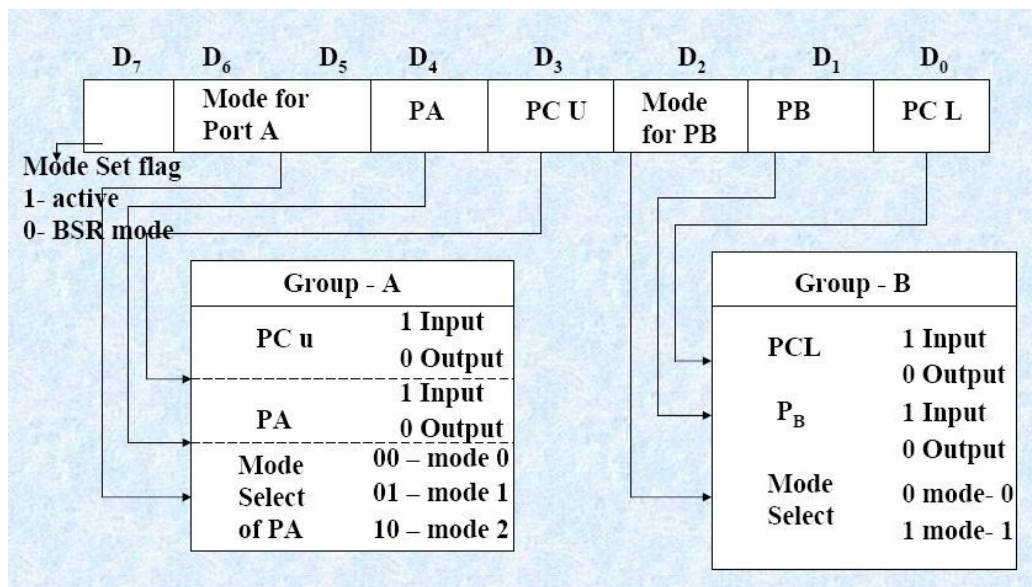
13.7

INTR	Interrupt request is an output that requests an interrupt
OBF	Output buffer full is an output indicating that the output buffer contains data for the bidirectional bus.
ACK	Acknowledge is an input that enables output buffers which are otherwise in their high-impedance state
STB	The strobe input loads data into the port A latch
IBF	Input buffer full is an output indicating that the input buffer contains information for the reversed bi-directional bus
PC[5:0]	Interrupt enable are internal bits that are PC[5:0] and PC[4:0]
PC[2:0]	These port C pins are general-purpose available for any purpose.



14

CONROL WORD



Operation modes:

Mode 0 (Simple Input/Output)

The register can be Set or Reset by sending OUT instruction to the CONTROL

Features:

Mode 0 (Simple Input/Output)

- In this mode, port A, port B and port C is used as individually (Simply).
- Inputs are latched, Inputs are buffered not latched.

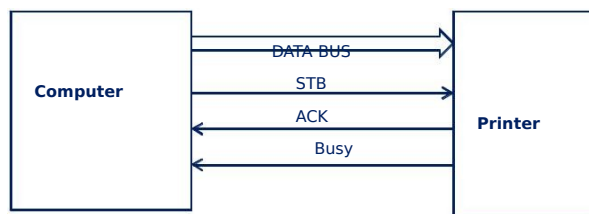
- Outputs are latched, Inputs are buffered not latched.

- Port do not have Handshake or Interrupt capability.

Mode 1 (Input/output with Handshake)

In this mode, input or output is transferred by hand shaking Signals.

- Hand shaking signals is used to transfer data between whose data transfer is not same.



Example:

The computer send the data to the printer large speed compared to the printer. When computer send the data according to the printer speed at the time only, printer can accept.

If printer is not ready to accept the data then after sending the data bus, computer uses another handshaking signal to tell printer that valid data is available on the data bus.

Each port uses three lines from port C as handshake signals

MODE 2: bi-directional I/O data transfer: Transfer over a single 8

- This mode allows bidirectional data transfer over a single 8-bit data bus using handshake signals.
- This feature is possible only Group A
- Port A is working as 8-bit bidirectional.

PC 3-PC 7 is used for handshaking purpose.

The data is sent by CPU through this port, when the peripheral request it.

CONTROL WORD FORMAT:

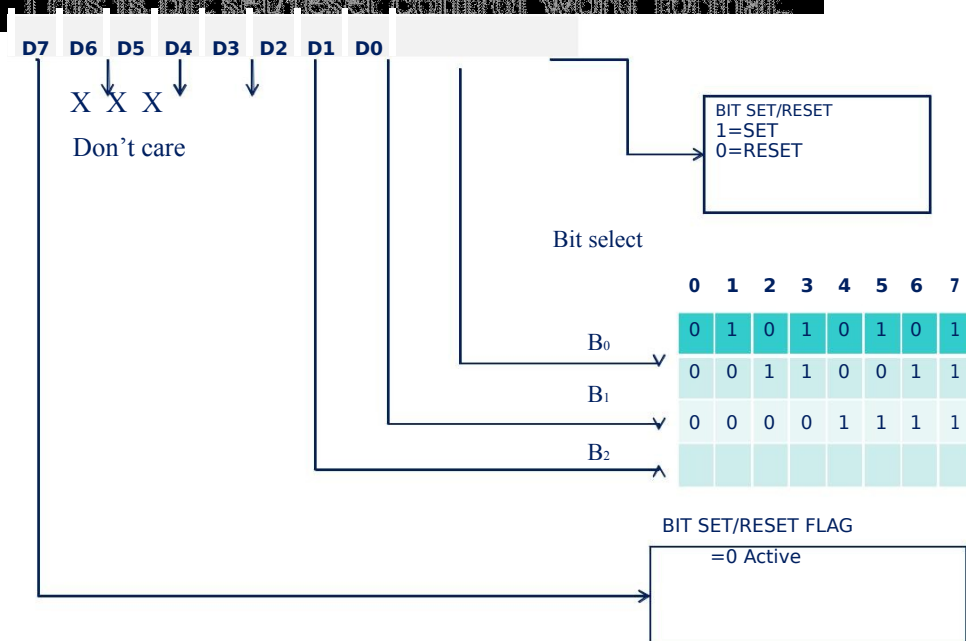
This can be avoid by writing single control word to the control registers, when

in the INP or mode, when RST is high all 14 pins (3 ports) be an input mode.

- i.e all flip flops are cleared and the interrupts are rest.
- This condition is maintained even after RESET goes low.
- Required by writing single control word to the control registers, when required.

8-BIT SET/RESET CONTROL WORD FORMAT:

This is bit set/reset control word format.

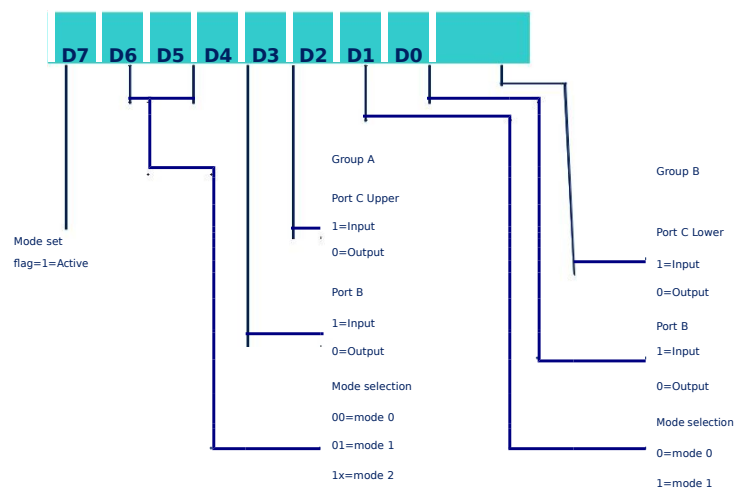


- PC0-PC7 is set or reset as per the status of D0.
- X is a don't care.
- Example.
- PC3 is Set then control register will be 0XXX0111.
- PC4 is Reset then control register will be 0XXX01000.
- X is a don't care.

- The control word for both mode is same.
- Bit D7 is used for specifying whether word loaded in to Bit set/reset mode or Mode definition word.
- D7=1=Mode definition mode.
- D7=0=Bit set/Reset mode.

FOR I/O MODE:

The mode format for I/O as shown in figure



I/O mode

Port A Input Mode 1 (Handshaking operation)

Port C Input Mode 1 (Handshaking operation)

Port C Input Mode 1 (Handshaking operation)

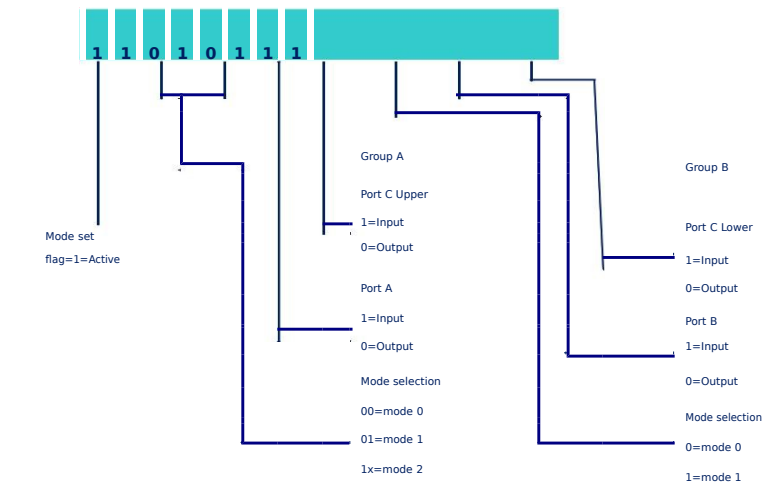
Port B input

Port C input

Mode 1 (Handshaking operation)

• I/O MODE

The mode format for I/O as shown in figure



INTERFACING WITH 8085

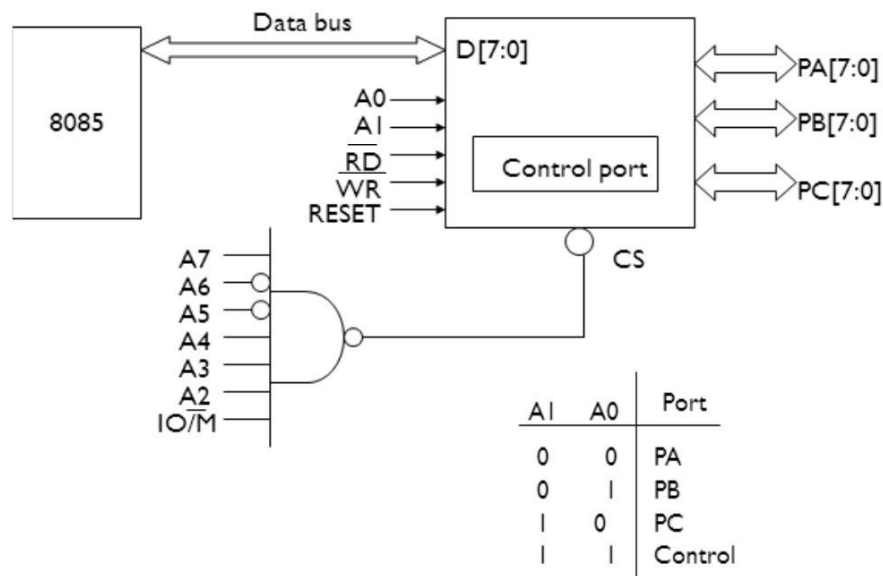
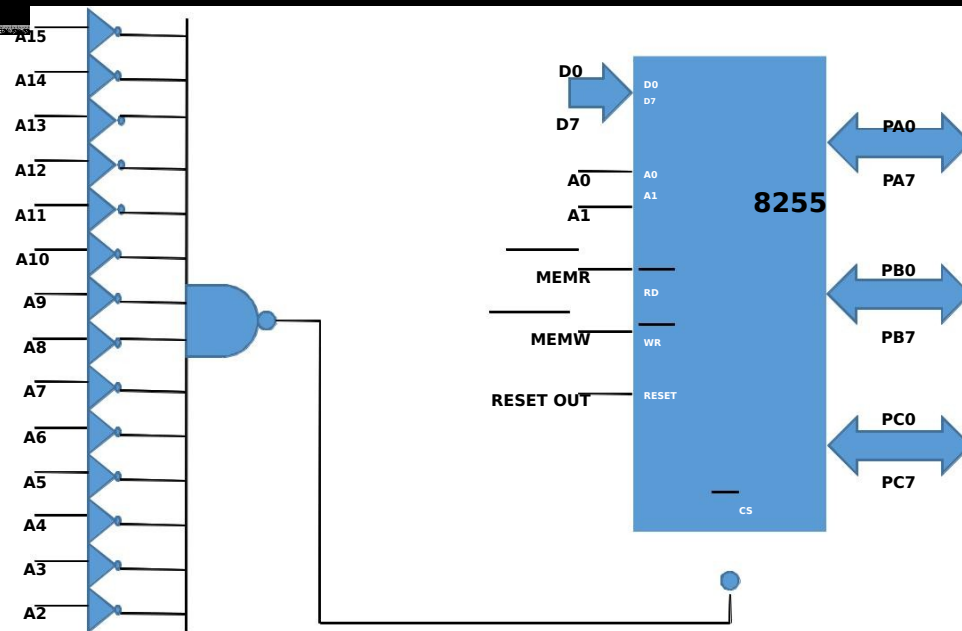
Basic Interfacing Concept

- **Any application of Microprocessor Based system Requires the transfer of data between external circuitry to the Microprocessor and Microprocessor to the External circuitry. User can give information (i.e. input) to the Microprocessor using keyboard and user can see the result or output information from the Microprocessor with the help of display.**
- **Hence interfacing is used to exchange information between two different applications/devices.**

Memory Mapped I/O

- **Device address is of 16 Bit. means A_0 to A_{15} lines are used to generate device address.**
- **MEMR and MEMW control signals are used to control read and write I/O operations.**
- **Data transfer is between Any register and I/O device.**
- **Maximum number of I/O devices are 65536.**
- **Decoding 16 bit address may requires more hardware.**
- **For e.g. MOV R M, ADD M, CMP M etc.**

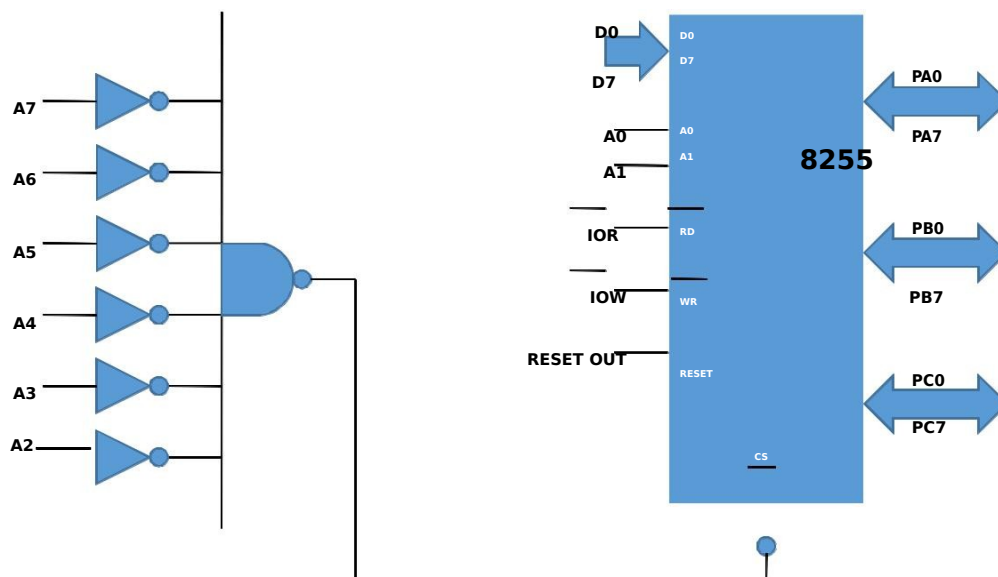
INTERFACING IN MEMORY MAPPED I/O



I/O Mapped I/O

- Device address is of 8 Bit, means A_0 to A_7 or A_8 to A_{15} lines are used to generate device address.
- IOR and IOW control signals are used to control read and write I/O operations.
- Data transfer is between Accumulator and I/O device.
- Maximum number of I/O devices are 256.
- Decoding 16 bit address may requires less hardware.
- For e.g. IN, OUT etc.

INTERFACING IN I/O MAPPED I/O



INTERFACING WITH 8051

13.3 8051 I/O Expansion using 8255

As seen earlier, for interfacing external memory to the 8051, port 0 and port 2 are used as multiplexed address/data bus and a higher order address bus respectively. If the circuit needs the on chip peripherals (e.g. serial I/O and Interrupts) then only 1 port is available for I/O. In such situations, I/O expansion is necessary and it is achieved by using 8255. The Fig. 13.12 shows the expanded I/O ports using 8255. Data bus of 8255 is connected to the port 0. Address lines A_0 and A_1 , after latches are connected to A_0 and A_1 of the 8255.

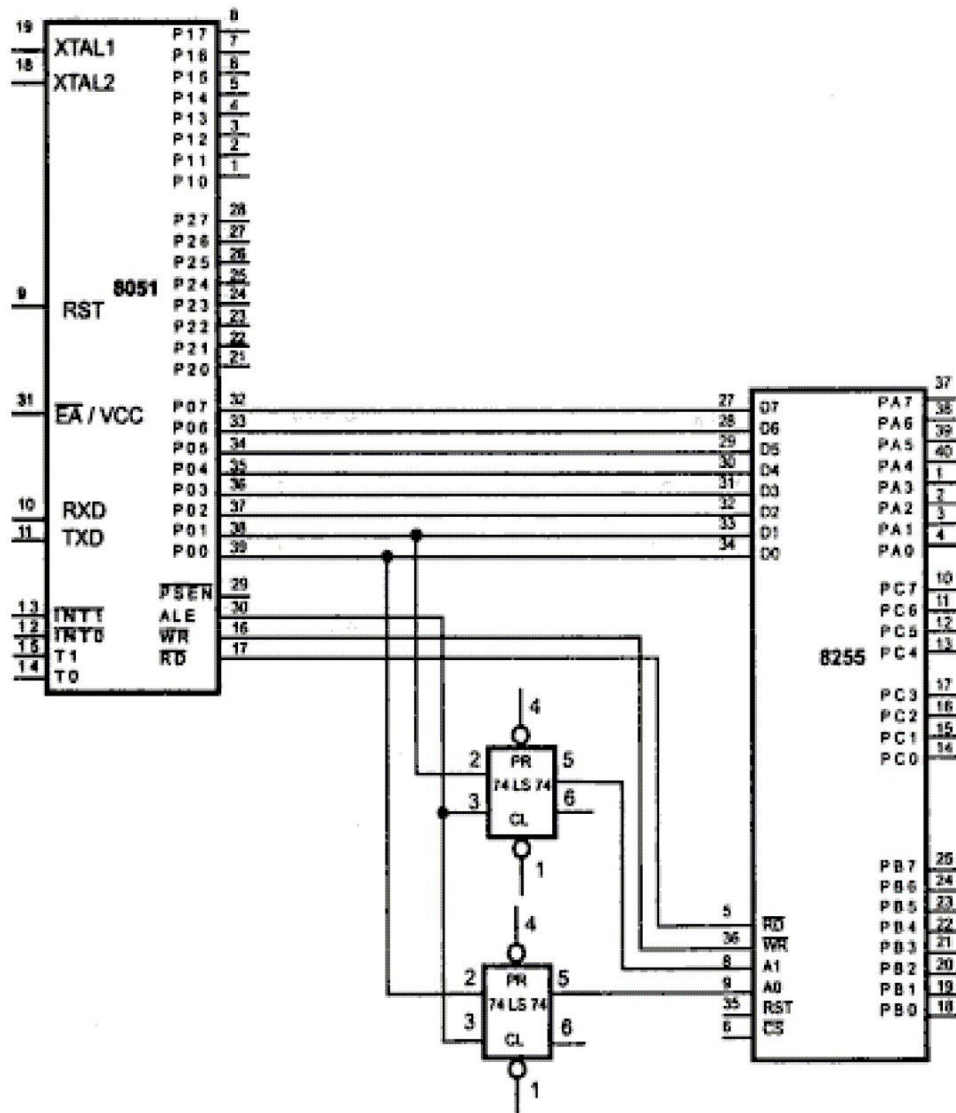


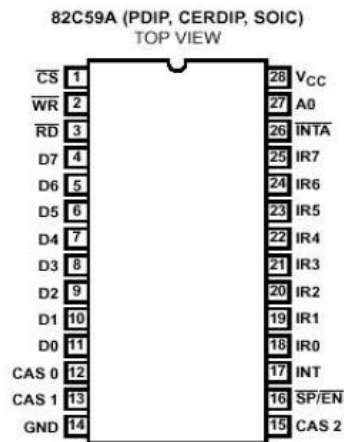
Fig. 13.12 I/O expansion using 8255

8259

ARCHITECTURE

8259 Programmable Interrupt Controller

- The 8259 programmable interrupt controller (PIC) adds eight vectored priority encoded interrupts to the microprocessor.
- It accepts request from the peripheral equipment, determine which of the incoming requests is of the highest importance
- Special features of 8259:
 - Eight level priority controller
 - Expandable to 64 levels
 - Programmable interrupt modes
 - Individual request mask capability



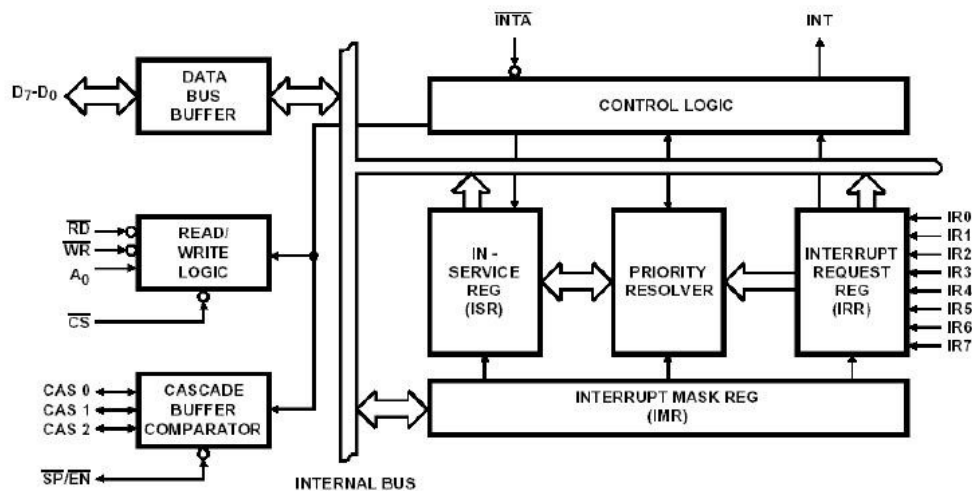
PIN	DESCRIPTION
D7 - D0	Data Bus (Bidirectional)
RD	Read Input
WR	Write Input
A0	Command Select Address
CS	Chip Select
CAS 2 - CAS 0	Cascade Lines
SP/EN	Slave Program Input Enable
INT	Interrupt Output
INTA	Interrupt Acknowledge Input
IR0 - IR7	Interrupt Request Inputs

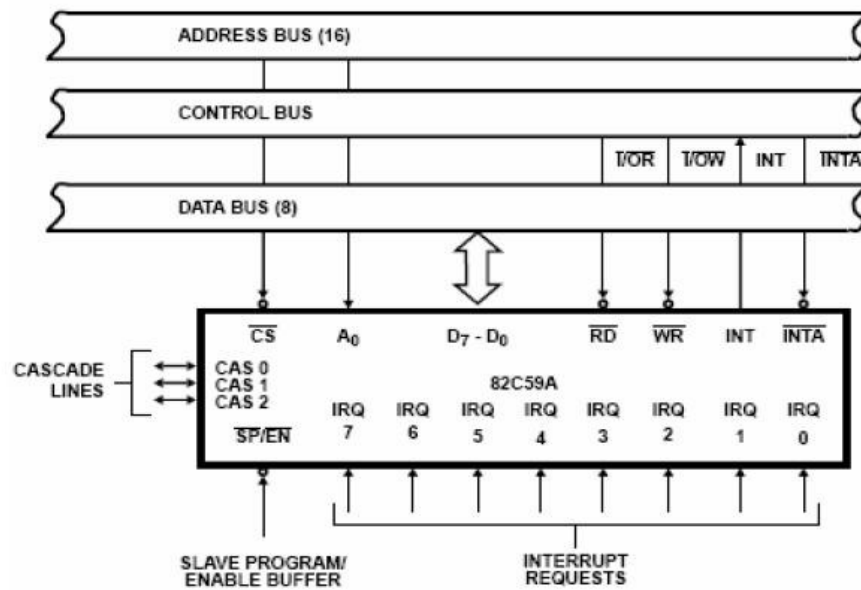
input. It is packaged in a 28-pin DIP, uses NMOS technology and requires a single 5V supply. Circuitry is static, requiring no clock

The 8259A is designed to minimize the software and real time overhead in handling multi-level priority interrupts.

It has several modes, permitting optimization for a variety of system requirements

82C59A Programmable Interrupt Controller





82C59A STANDARD SYSTEM BUS INTERFACE

INTERRUPT REQUEST REGISTER (IRR) AND IN-SERVICE REGISTER (ISR)

The interrupts at the IR input lines are handled by two registers in cascade, the Interrupt Request Register (IRR) and the In-Service (ISR). The IRR is used to store all the interrupt levels which are requesting service; and the ISR is used to store all the interrupt levels which are being serviced.

PRIORITY RESOLVER

This logic block determines the priorities of the bits set in the IRR. The highest priority is selected and stroed into the corresponding bit of the ISR during INTA pulse.

INTERRUPT MASK REGISTER (IMR)

The IMR stores the bits which mask the interrupt lines to be masked. The IMR operates on the IRR. Masking of a higher priority input will not affect the interrupt request lines of lower quality.

INT (INTERRUPT)

This output goes directly to the CPU interrupt input. The V level on this line is designed to be fully compatible with the 8080A, 8085A and 8086 input levels.

INTA (INTERRUPT ACKNOWLEDGE)

INTA pulses will cause the 8259A to release vectoring information onto the data bus. The format of this data depends on the system mode (mPM) of the 8259A. DATA BUS BUFFER

This 3-state, bidirectional 8-bit buffer is used to interface the 8259A to the system Data Bus. Control words and status information are transferred through the Data Bus Buffer. READ/WRITE CONTROL LOGIC

The function of this block is to accept Output commands from the CPU. It contains the Initialization Command Word (ICW) registers and Operation Command Word (OCW) registers which store the various control formats for device operation. This function block also allows the status of the 8259A to be transferred onto the Data Bus. CS (CHIP SELECT)

A LOW on this input enables the 8259A. No reading or writing of the chip will occur unless the device is selected.

WR (WRITE)

A LOW on this input enables the CPU to write control words (ICWs and OCWs) to the

8259A.

RD (READ)

A LOW on this input enables the 8259A to send the status of the Interrupt Request Register (IRR), In Service Register (ISR), the Interrupt Mask Register (IMR), or the Interrupt level onto the Data Bus.

A0

This input signal is used in conjunction with WR and RD signals to write commands into the various command registers, as well as reading the various status registers of the chip. This line can be tied directly to one of the address lines

INTERRUPT SEQUENCE

The events occur as follows in an MCS-80/85 system:

1. One or more of the INTERRUPT REQUEST lines (IR7±0) are raised high, setting the corresponding IRR bit(s).
2. The 8259A evaluates these requests, and sends an INT to the CPU, if appropriate.
3. The CPU acknowledges the INT and responds with an INTA pulse.
4. Upon receiving an INTA from the CPU group, the highest priority ISR bit is set, and the corresponding IRR bit is reset. The 8259A will also release a CALL instruction code (11001101) onto the 8-bit Data Bus through its D7±0 pins.
5. This CALL instruction will initiate two more INTA pulses to be sent to the 8259A from the CPU group.
6. These two INTA pulses allow the 8259A to release its preprogrammed subroutine address onto the Data Bus. The lower 8-bit address is released at the first INTA pulse and the higher 8-bit address is released at the second INTA pulse.
7. This completes the 3-byte CALL instruction released by the 8259A. In the AEOI mode the ISR bit is reset at the end of the third INTA pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt sequence.

PROGRAMMING THE 8259A

The 8259A accepts two types of command words generated by the CPU:

1. Initialization Command Words (ICWs):

Before normal operation can begin, each 8259A in the system must be brought to a starting point - by WR pulses.

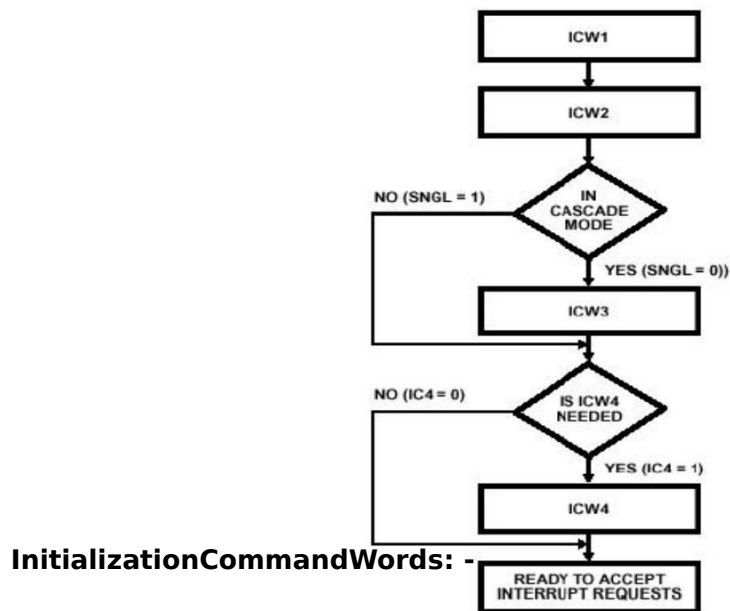
2. Operation Command Words (OCWs): These are the command words which command the 8259A to operate in various interrupt modes.

These modes are:

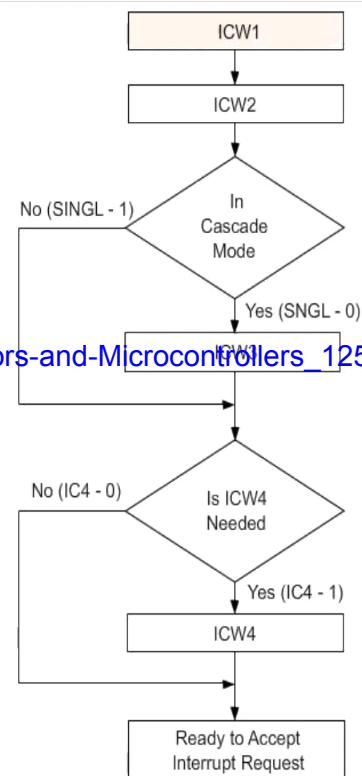
- a. Fully nested mode
- b. Rotating priority mode
- c. Special mask mode
- d. Polled mode

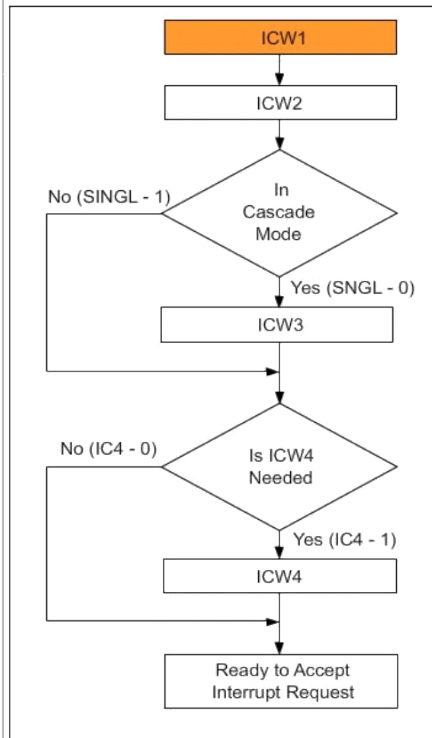
The OCWs can be written into the 8259A anytime after initialization.

Initialization Sequence

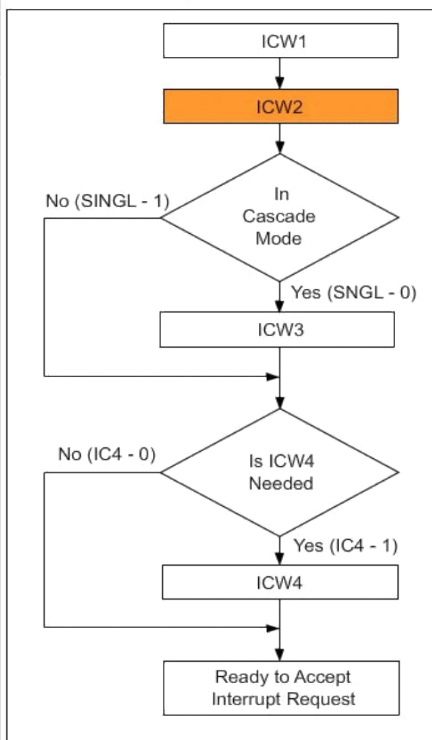


There are four Initialization Command Words for the 8259A that are selected with the help of logic level of A0 pin. When the 8259A is first powered up, it must be sent ICW1, ICW2 and ICW4.



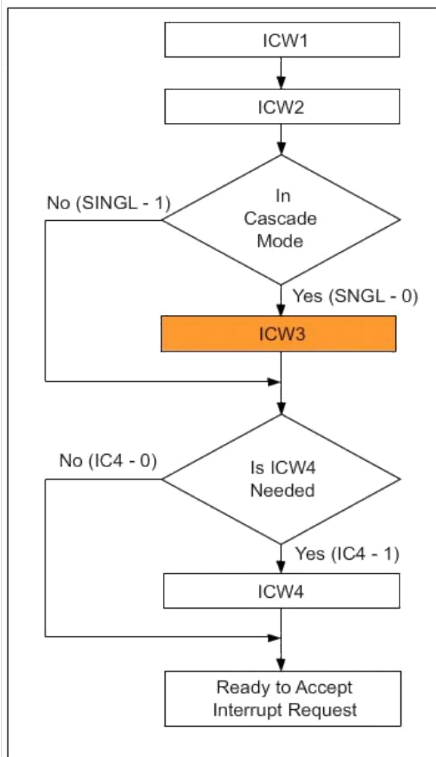
Initialisation Sequence of 8259 A**ICW1 Format**

A7	A6	A5	1	LTIM	ADI	SINGL	IC4
----	----	----	---	------	-----	-------	-----

Initialisation Sequence of 8259 A**ICW2 Format**

A15	A14	A13	A12	A11	A10	A9	A8
-----	-----	-----	-----	-----	-----	----	----

Higher byte of Interrupt Service Routine Address (for 8085)

Initialisation Sequence of 8259 A**ICW3 Format****Master Mode:**

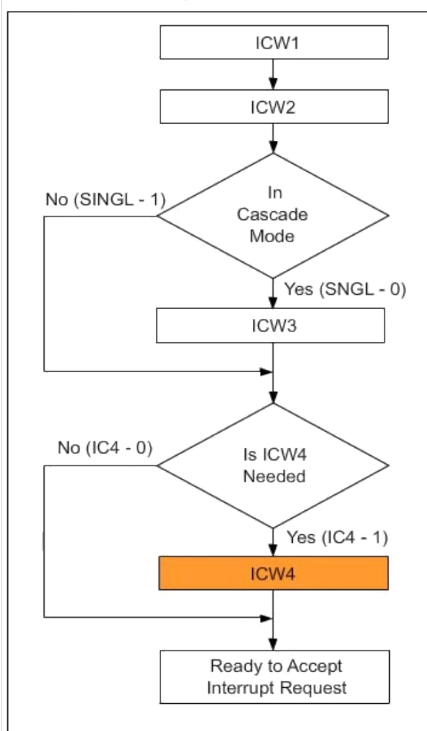
1 = Slave is present on a particular Interrupt Line

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

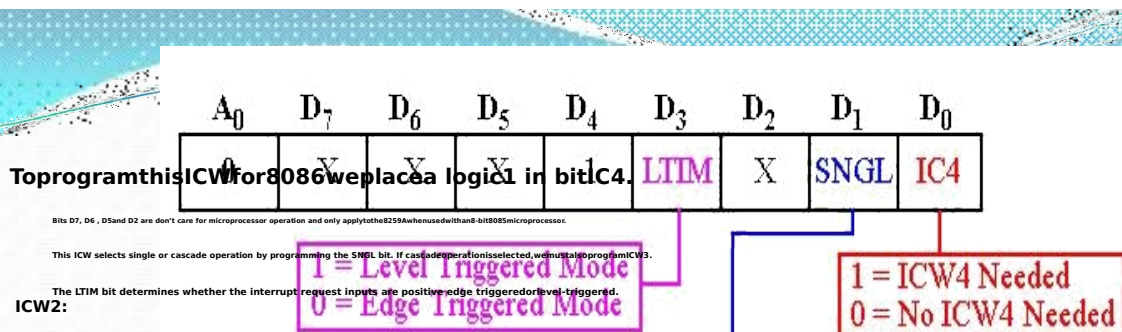
Slave Mode:

ID1, ID2, ID3 is Slave ID Number

0	0	0	0	0	ID3	ID2	ID1
---	---	---	---	---	-----	-----	-----

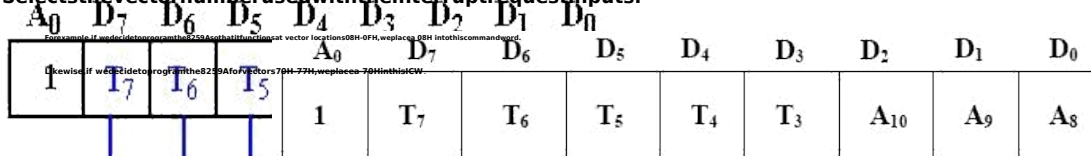
Initialisation Sequence of 8259 A**ICW4 Format**

0	0	0	SFNM	BUF	M/S	AEOI	MODE
---	---	---	------	-----	-----	------	------

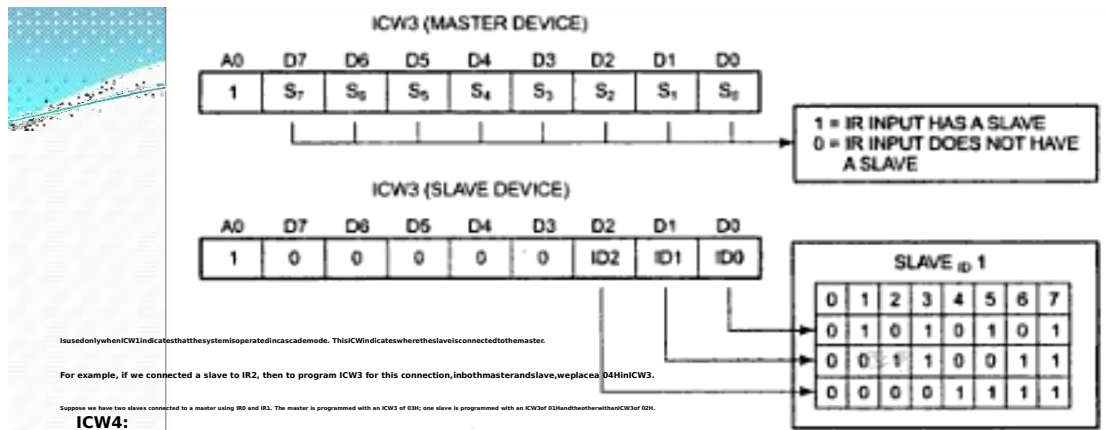
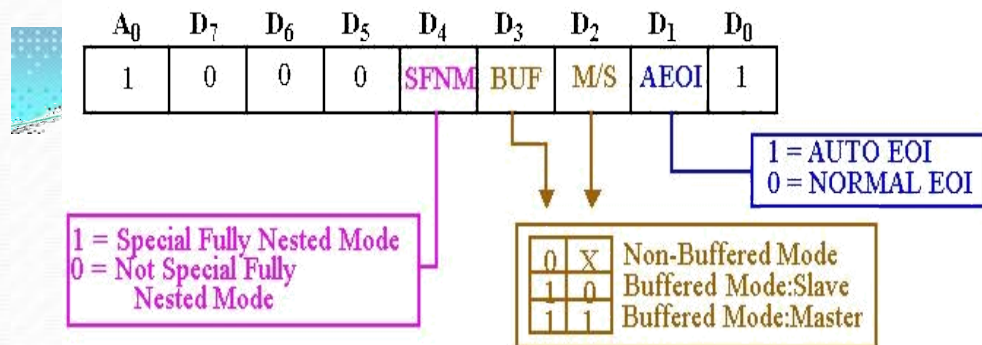
ICW1:

Low order bits are 0 since there are 8 interrupts.

Select the vector number used with the interrupt request inputs.

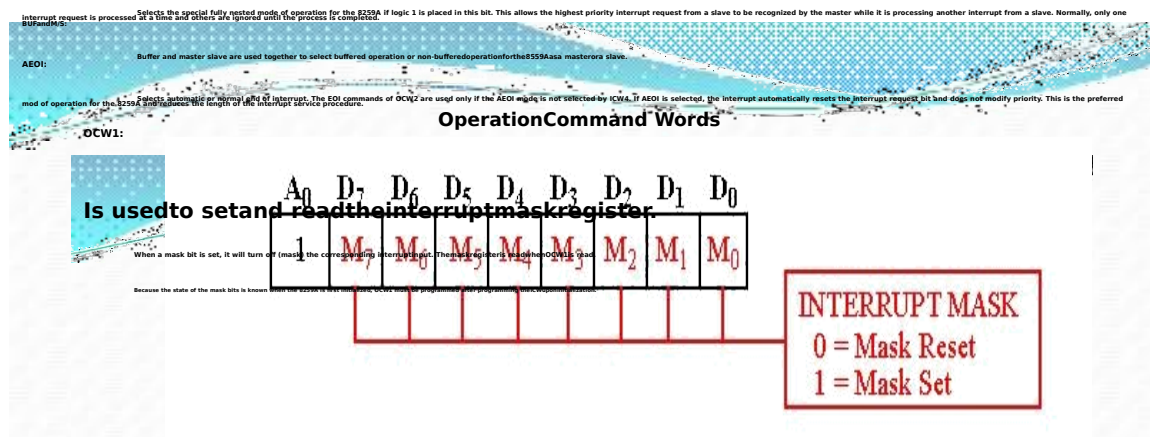


- T₇ – T₃ are A₃ – A₀ of interrupt address
- A₁₀ – A₉, A₈ – Selected according to interrupt request level.
- They are not the address lines of Microprocessor
- A₀ = 1 selects ICW₂

ICW3:**ICW4:**

Is programmed for use with the 8088/8086. This ICW is not programmed in a system that functions with the 8085 microprocessors.

The rightmost bit must be logic 1 to select operation with the 8088 microprocessor and the remaining bits are programmed as follows:



NonspecificEnd-of-Interrupt:

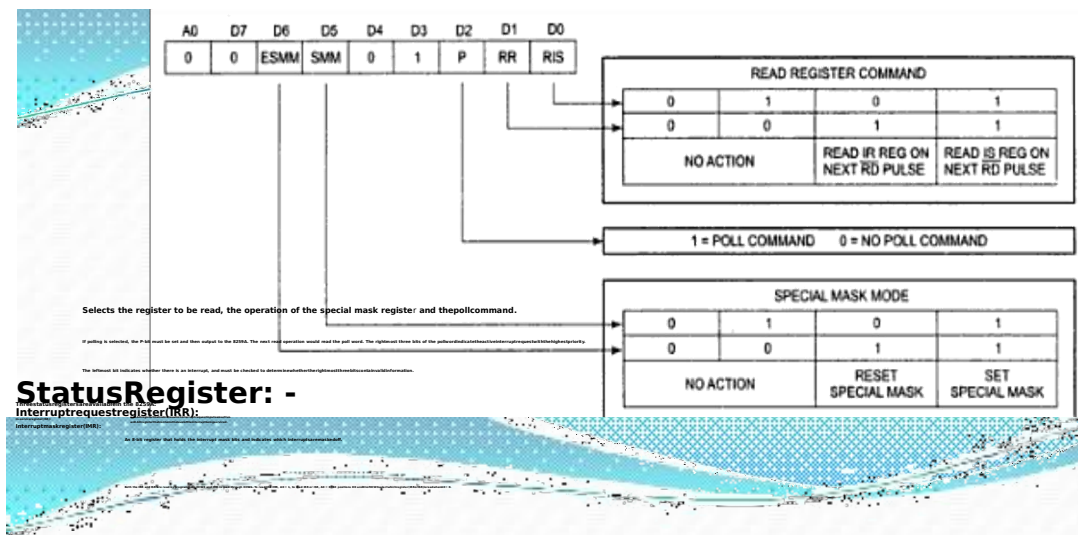
A command sent by the interrupt service procedure to signal the end of the interrupt. The 8086 automatically determines which interrupt flag was active and resets the correct bit of the interrupt status register. Resetting the status bit allows the interrupt to take advantage of nonpriority interrupts.

A command sent by the interrupt service procedure to signal the end of the interrupt.

A command sent by the interrupt service procedure to signal the end of the interrupt. The 8086 automatically determines which interrupt flag was active and resets the correct bit of the interrupt status register. Resetting the status bit allows the interrupt to take advantage of nonpriority interrupts.

A command sent by the interrupt service procedure to signal the end of the interrupt. The 8086 automatically determines which interrupt flag was active and resets the correct bit of the interrupt status register. Resetting the status bit allows the interrupt to take advantage of nonpriority interrupts.

A command sent by the interrupt service procedure to signal the end of the interrupt.

OCW3:

Modes of 8259A PIC

FullyNestedmode

SpecialFullyNestedMode
NonrecursiveInterrupting
SpecialRecursiveInterrupting
SpecialMask
Priority
FixedPriorityMode

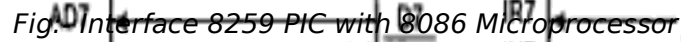
Fullynestedmode:

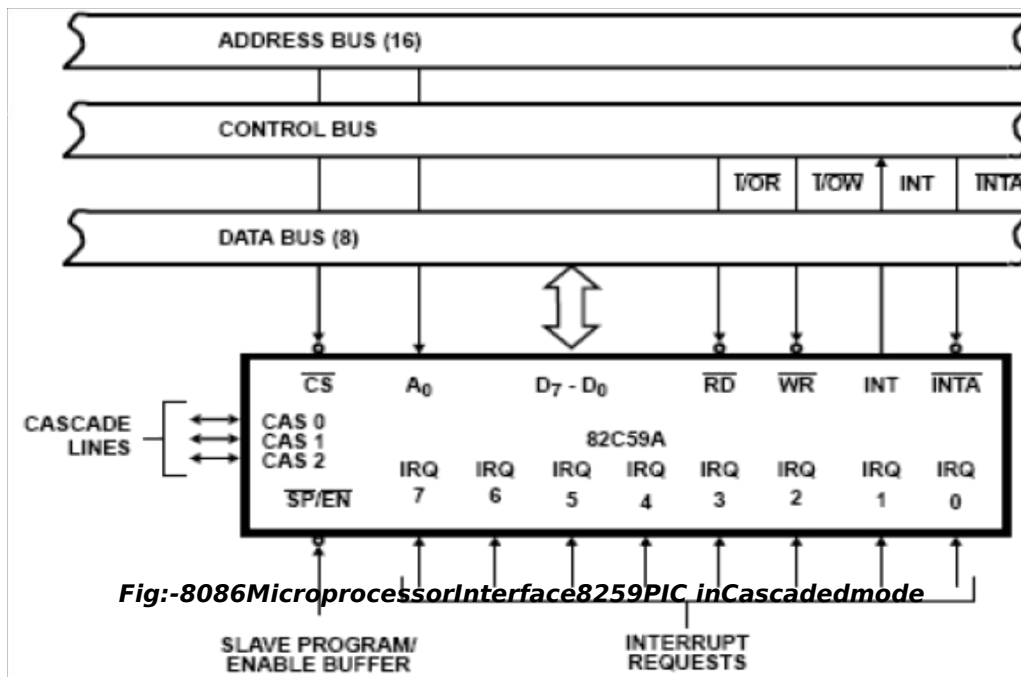
This is a general purpose mode where all IR's are arranged in highest to lowest.

SpecialFullyNestedMode

When an interrupt request from a certain slave is in service, this slave can further send request to the master.

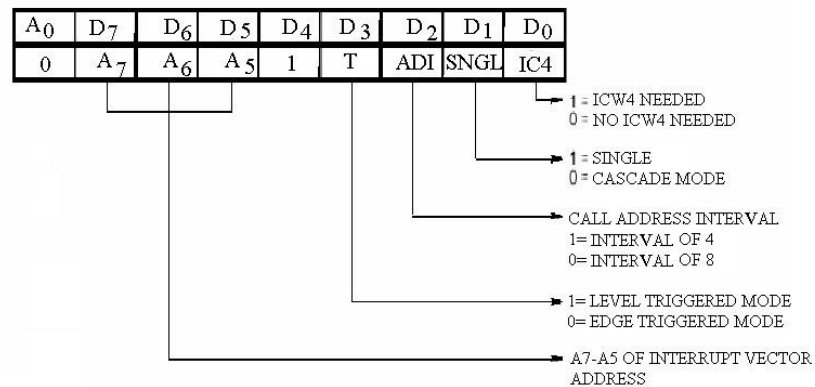
Keywords: *Health care utilization; Mortality; Health care access; Health care quality; Health care costs; Health care equity*





INITIALISATION COMMAND WORDS

ICW1



$$IC4 = 0.$$

A5±A15: Page starting address of service routines .In an MCS 80/85 system, the 8 request levels will generate CALLs to 8 locations equally spaced in memory. These can be programmed to be spaced at intervals of 4 or 8 memory locations, thus the 8 routines will occupy a page of 32 or 64 bytes, respectively. The address format is 2 bytes long (A0±A15). When the routine interval is 4, A0±A4 are automatically inserted by the 8259A, while A5±A15 are programmed externally. When the routine interval is 8, A0±A5 are automatically inserted by the 8259A, while A6±A15 are programmed externally.

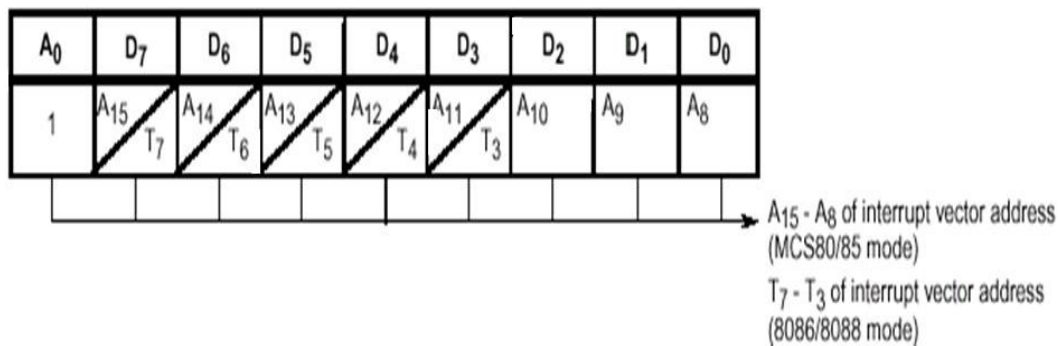
.T: If LTIM e 1, then the 8259A will operate in the level interrupt mode. Edge detect logic on the interrupt inputs will be disabled.

.ADI: CALL address interval. ADI = 1 then interval = 4; ADI e 0 then interval = 8.

.SNGL: Single. Means that this is the only 8259A in the system. If SNGL = 1 no ICW3 will be issued.

.IC4: If this bit is set ICW4 has to be read. If ICW4 is not needed, set

ICW 2



ICW 3

This word is read only when there is more than one 8259A in the system and cascading is used, in which case $SNGL = 0$. It will load the 8-bit slave register.

The functions of this register are:

- In the master mode (either when $SP = 1$, or in buffered mode when $M/S = 1$ in ICW4) a "1" is set for each slave in the system. The master then will release byte 1 of the call sequence (for MCS-80/85 system) and will enable the corresponding slave to release bytes 2 and 3 (for 8086 only byte 2) through the cascade lines.
- In the slave mode (either when $SP = 0$, or if $BUF = 1$ and $M/S = 0$ in ICW4) bits 2:0 identify the slave. The slave compares its cascade input with these bits and, if they are equal, bytes 2 and 3 of the call sequence (or just byte 2 for 8086) are released by it on the Data Bus. Only when there is more than one 8259A in the system and cascading is used, in which

ICW3 (MASTER DEVICE)

A ₀	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
1	S ₇	S ₆	S ₅	S ₄	S ₃	S ₂	S ₁	S ₀

1 = IR input has a slave
0 = IR input does not have a slave

ICW3 (SLAVE DEVICE)

A ₀	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
1	0	0	0	0	0	ID ₂	ID ₁	ID ₀

SLAVE ID (NOTE)

	0	1	2	3	4	5	6	7
0	0	1	0	1	0	1	0	1
1	0	0	1	1	0	0	1	1
2	0	0	0	0	1	1	1	1

ICW4

ICW4

A ₀	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
1	0	0	0	SFNM	BUF	M/S	AEOL	μPM

AEOL mode requires no commands. During the second INTA the ISR bit is reset. The major drawback with this mode is that the ISR doesn't have info on which IR is served. Thus any IR with any priority can *now* Interrupt service routine.

	0	X
1	0	
1	1	

1 = 8086/8088 mode
0 = MCS-80/85 mode

1 = Auto EOI
0 = Normal EOI

- Non buffered mode
- Buffered mode slave
- Buffered mode master

1 = Special fully nested mode
0 = Not special fully nested mode

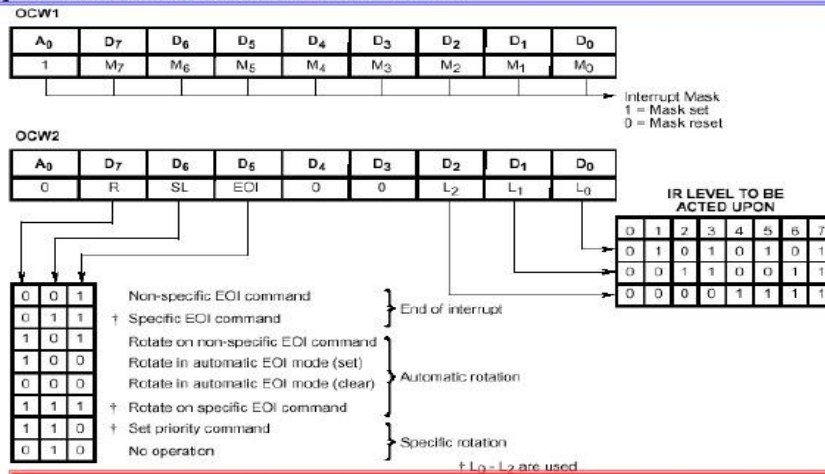
BUF when 1 selects buffer mode. The SP/EN pin becomes an output for the data buffers.

When 0, the SP/EN pin becomes the input for the (MASTER/SLAVE) functionality

M/S is used to set the function of the 8259 when operated in buffered mode. If M/S is set the 8259 will function as the MASTER. If cleared will function as SLAVE. If BUF=0, M/S is to be neglected

OCW1 - OCW2

OCW1 is used to access the contents of the IMR. A READ operation can be performed to the IMR to determine the present setting of the mask. Write operations can be performed to mask or unmask certain bits.



Controller will not confuse OCW2 with ICW1 since D₄ = 1

8254

PROGRAMMABLE INTERVAL TIMER 8254 (8253)

26 February 2014 M.M.Arun Prasath, Asst. Prof./ECE

PROGRAMMABLE INTERVAL TIMER 8254 (8253)

It is a programmable counter/timer chip specifically designed for use in Real Time Applications for Timing and counting function
Such as

- Binary counting
- Generation of accurate time delay
- Generation of Square wave etc

Timer 8253/54 has three timers
Three counters are 16-bit down counters independent of each other.
The counter can count either in binary or BCD.
8254 is an upgraded version of 8253 and they are pin-compatible
All modes of operation are software programmable

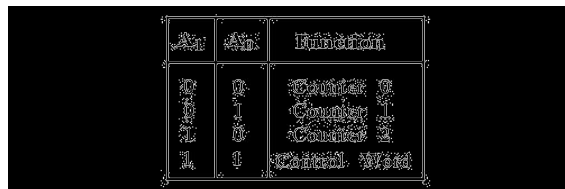
8253 Timer is a 24 pin IC
It consists of 3 Counters : Counter 0, Counter 1, Counter 2

Data Bus Buffer (D0-D7) Used to interface the 8253 to the system data bus D0-D7

Read (RD) It is low, the CPU is inputting data in the counter Write (WR) It is low, the CPU is outputting data in the counter
(or) Loading of counters

Chip select (CS) It is low, the 8253 is enables otherwise No reading or writing operation will be performed.

A0,A1 Used to select one of the counter or Control word



INTRODUCTION

- It is a programmable counter/timer chip designed for use as an Intel microcomputer peripheral
- The 3 counters are 16-bit down counters independent of each other, and can be easily read by the CPU.
- All 3 counters are able to operate either in BCD or in Hexadecimal mode.
- The counter can count either in binary or BCD.
- 8254 is an upgraded version of 8253 and they are pin-compatible
- All modes of operation are software programmable

26 February 2014 M.M.Arun Prasath, Asst. Prof./ECE

ARCHITECTURE

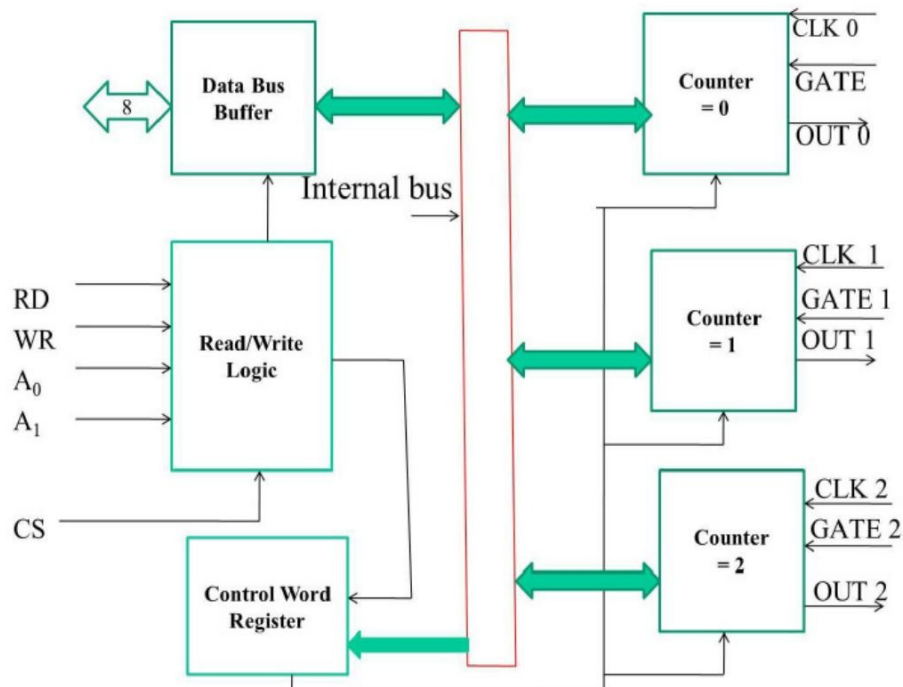
BLOCK DIAGRAM OF 8253/54

- Three counters (0, 1 & 2)
- Each counters has two i/p signals (CLK & GATE) and one o/p signal (OUT)
- Data Bus buffer
- Read/Write control logic
- Control word Register
- Control word registers and counters are selected using A₁ and A₀

A ₁	A ₀	Selection
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Control register

26 February 2014 M.M.Arun Prasath, Asst. Prof./ECE

BLOCK DIAGRAM OF 8253/54



26 February 2014 M.M.Arun Prasath, Asst. Prof./ECE

<p>COUNTERS</p> <ul style="list-style-type: none"> • THREE COUNTERS - C1, C2 & C3 • EACH 16 BIT • IDENTICAL • PRESETTABLE • DOWN COUNTER • OPERATES IN BCD /HEX • CONTROLLED BY LOADING COUNT TO COMMAND WORD REGISTER • "ON THE FLY" READING 	<p>CONTROL BLOCK OF DATA 8254 LOGIC</p> <ul style="list-style-type: none"> • CS' - LOGIC 0 - ENABLES 8254 • RD' - LOGIC 0 - TELLS MP READS COUNT FROM 8254 • WR' - LOGIC 0 - TELLS MP WRITES COUNT/ COMMAND INTO 8254 • A1, A0 - ADDRESS INPUT PINS TO SELECT MODES AND COUNTERS 	<p>DATA 8254 BUFFERS</p> <ul style="list-style-type: none"> • 8 BIT • BIDIRECTIONAL • D0-D7 • CONNECTED TO DATA BUS OF MP • IN READS DATA FROM PERIPHERAL • OUT WRITES DATA TO PERIPHERAL 	<p>CONTROL WORD REGISTER</p> <ul style="list-style-type: none"> • ACCEPTS 8 BIT CONTROL WORD WRITTEN BY MP • CAN ONLY BE WRITTEN (NOT READ) • CONTROL WORD CHOOSES ONE OF THE SIX MODES OF OPERATION
--	---	--	--

DATA BUS BUFFER

The 8-bit bidirectional data buffer interfaces internal circuit of 8254 to microprocessor. Data is transmitted or received by the buffer upon the execution of IN or OUT instruction.

READ/WRITE CONTROL LOGIC

It controls the reading and the writing of the counter registers

The control section has five signals

1. RD_ (Read signal)
2. WR_ (Write signal)
3. CS_ (Chip Select signal)
4. Address line A₁
5. Address line A₀

CONTROL WORD REGISTER

This register is accessed when lines A₁ and A₀ are at logic 1. It is used to write a command word which specifies the counter to be used, its mode is either a Read or Write operation

0	1	0	0	0	WRITE COUNTER 0
0	1	0	0	1	WRITE COUNTER 1
0	1	0	1	0	WRITE COUNTER 2
0	1	0	1	1	WRITE CONTROL WORD
0	0	1	0	0	READ COUNTER 0
0	0	1	0	1	READ COUNTER 1
0	0	1	1	0	READ COUNTER 2
0	0	1	1	1	NO OPERATION (TRISTATED)
0	1	1	X	X	NO OPERATION (TRISTATED)
1	X	X	X	X	8254 NOT SELECTED

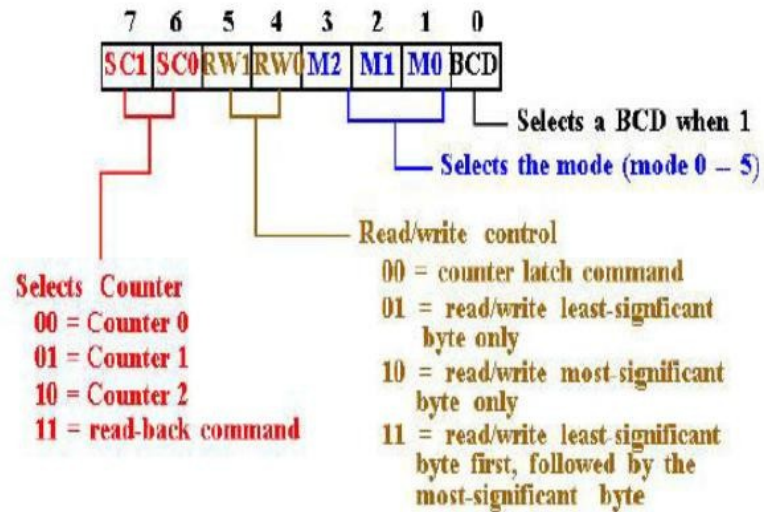
INTERNAL REGISTER OPERATION FOR PROGRAMMING 8253/54

	RD_	RW_	A0	A1	Function
COUNTER 0	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
MODE WORD or CONTROL WORD	1	0	1	1	Write Mode Word
	0	1	1	1	No-operation

26 February 2014 M.M.Arun Prasath, Asst. Prof./ECE

Mode

8254 can operate in 6 different modes, and the gate of a counter is used either to disable or enable counting



26 February 2014 M.M.Arun Prasath., Asst. Prof./ECE

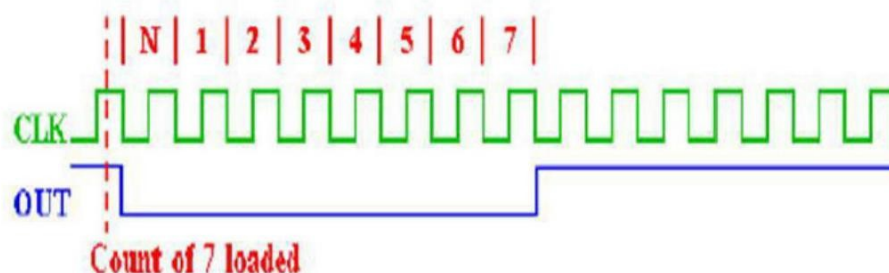
6 MODES OF 8253/54

1. MODE 0: Interrupt on Terminal Count
2. MODE 1: Hardware – Retriggerable One-Shot
3. MODE 2: Rate Generator
4. MODE 3: Square-Wave Generator
5. MODE 4: Software Triggered Strobe
6. MODE 5: Hardware Triggered Strobe

26 February 2014 M.M.Arun Prasath, Asst. Prof./ECE

MODE 0 : Interrupt on Terminal Count

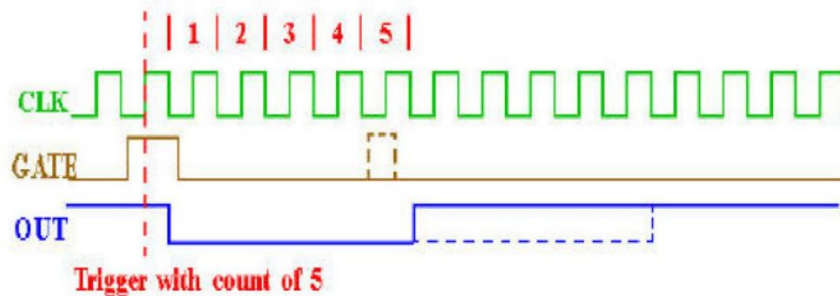
- In this mode, initially the OUT is low.
- Once the count is loaded in the register, the counter is decremented every cycle and when the count reaches zero, the OUT goes high. This can be used as an interrupt
- The OUT remains high until a new count or a command word is loaded.



26 February 2014 M.M.Arun Prasath, Asst. Prof./ECE

MODE 1 : Hardware - Retriggerable one-shot

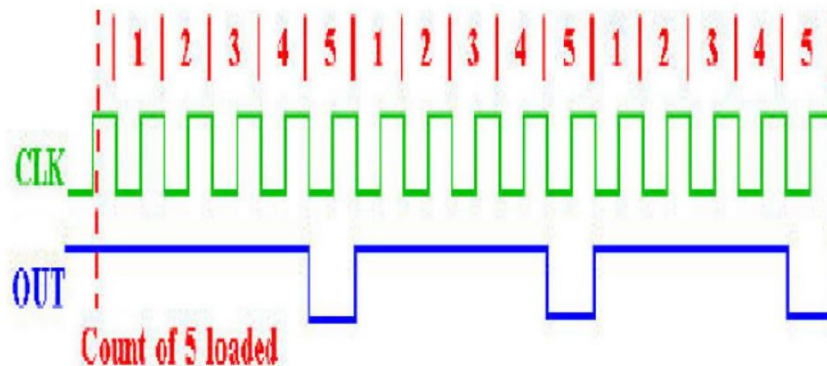
- In this mode, the OUT is initially high
- When the Gate is triggered, the OUT goes low, and at the end of the count, the OUT goes high again, thus generating a one-shot pulse.



26 February 2014 M.M.Arun Prasath., Asst. Prof./ECE

MODE 2 : Rate Generator

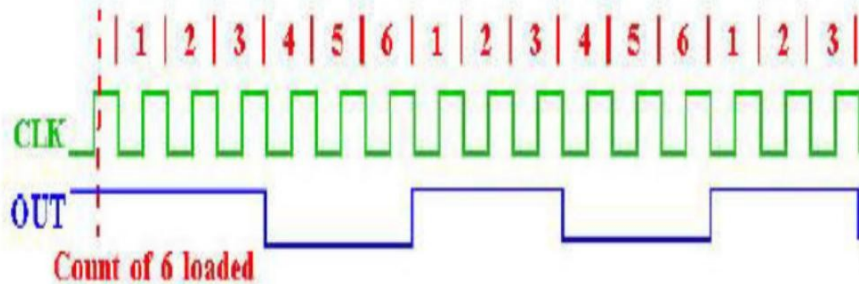
- This mode is used to generate a pulse equal to the clock period at a given interval.
- When a count is loaded, the OUT stays high until the count reaches 1, and then the OUT goes low for one clock period
- The count is reloaded automatically, and the pulse is generated continuously. The count = 1 is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



26 February 2014 M.M.Arun Prasath., Asst. Prof./ECE

MODE 3 : Square wave generator

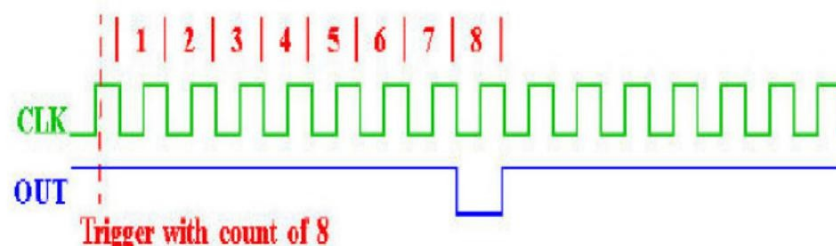
- In this mode, when a count is loaded, the OUT is high.
- The count is decremented by two at every clock cycle, and when it reaches zero, the OUT goes low, and the count is reloaded again.
- This is repeated continuously; thus a continuous square wave with period equal to the period of the count is generated
- The frequency of the square wave is equal to the frequency of the clock divided by the count



26 February 2014 M.M.Arun Prasath, Asst. Prof./ECE

MODE 4 : Software triggered strobe

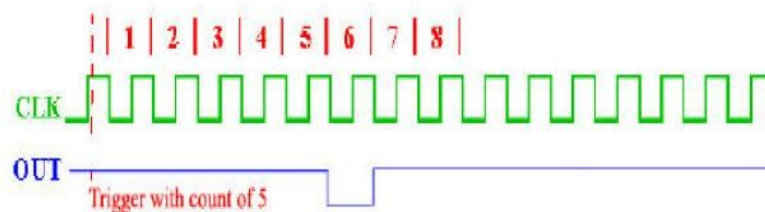
- In this mode, the OUT is initially high; it goes low for one clock period at the end of the count.
- The count must be reloaded for subsequent outputs.



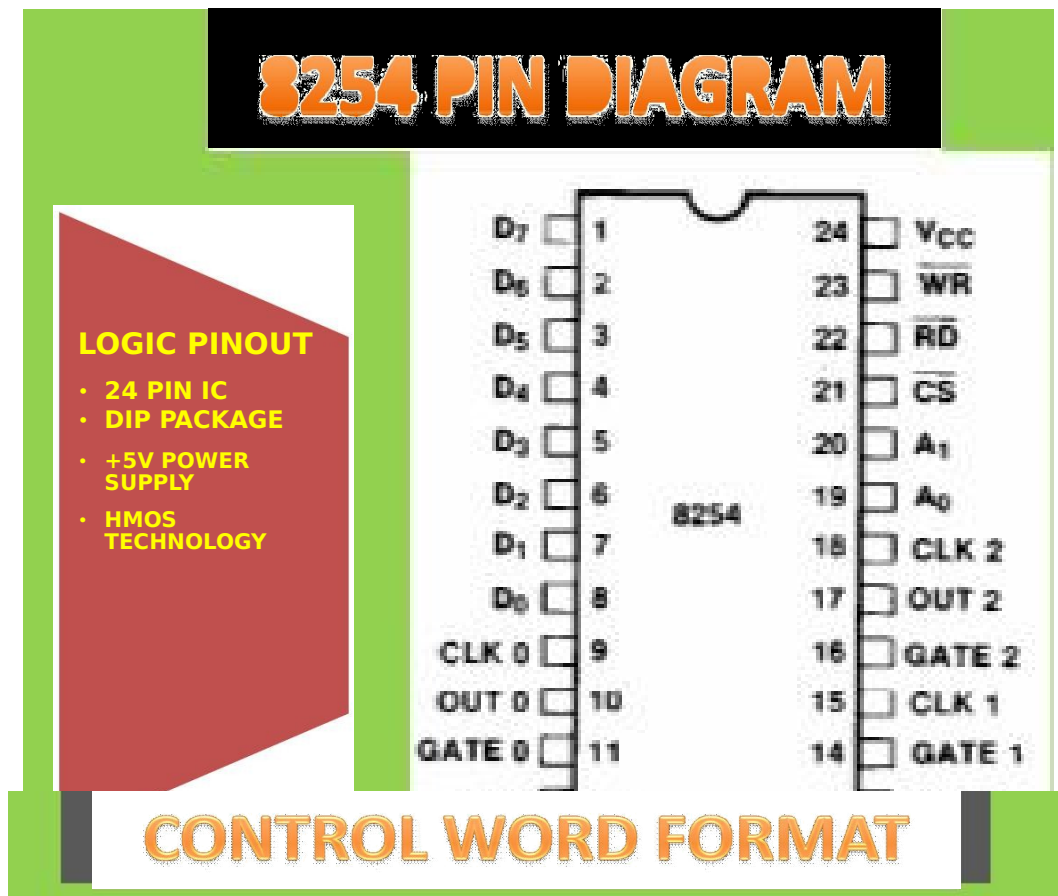
26 February 2014 M.M.Arun Prasath, Asst. Prof./ECE

MODE 5 : Hardware triggered strobe

- This mode is similar to Mode 4, except that it is triggered by the rising pulse at the gate
- Initially, the OUT is low, and when the Gate pulse is triggered from low to high, the count begins.
- At the end of the count, the OUT goes low for one clock period.



20 february 2014 P. L. L. ARUN PRASATH, ASSC. PROF/EECE



D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC—Select Counter

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

RW—Read/Write

RW1	RW0	
0	0	Counter Latch Command (see Read Operations)
0	1	Read/Write least significant byte only
1	0	Read/Write most significant byte only
1	1	Read/Write least significant byte first, then most significant byte

M—Mode

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

BCD

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

Control Word Format $A_1, A_0 = 11$ $CS = 0$ $RD = 1$ $WR = 0$

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC—Select Counter

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

M—Mode

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

RW—Read/Write

RW1	RW0	
0	0	Counter Latch Command (see Read Operations)
0	1	Read/Write least significant byte only
1	0	Read/Write most significant byte only
1	1	Read/Write least significant byte first, then most significant byte

BCD

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

NOTE:

Don't care bits (X) should be 0 to insure compatibility with future Intel products.

Read Operations

There are three possible methods for reading the counters:

- a simple read operation
- the Counter Latch Command
- the Read-Back Command

Simple read operation :

- The Counter which is selected with the A1, A0 inputs, the CLK input of the selected Counter must be inhibited by using either the GATE input or external logic.
- Otherwise, the count may be in the process of changing when it is read, giving an undefined result.

Counter Latch Command:

- SC0, SC1 bits select one of the three Counters
- two other bits, D5 and D4, distinguish this command from a Control Word
- If a Counter is latched and then, some time later, latched again before the count is read, the second Counter Latch Command is ignored.
- The count read will be the count at the time the first Counter Latch Command was issued.

$A_1, A_0 = 11; CS = 0; RD = 1; WR = 0$

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
SC1	SC0	0	0	X	X	X	X

SC1, SC0—specify counter to be latched

SC1	SC0	Counter
0	0	0
0	1	1
1	0	2
1	1	Read-Back Command

D5, D4—00 designates Counter Latch Command

X—don't care

NOTE:

Don't care bits (X) should be 0 to insure compatibility with future Intel products.

Read-back control command:

- The read-back control, word is used, when it is necessary for the contents of more than one counter to be read at a same time.

Count : logic 0, select one of the Counter to be latched

Status : logic 0, Status must be latched to be read status of a counter and is accessed by a read from that counter

$A_0, A_1 = 11$ $\overline{CS} = 0$ $\overline{RD} = 1$ $\overline{WR} = 0$

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
1	1	\overline{COUNT}	\overline{STATUS}	CNT 2	CNT 1	CNT 0	0

D₅: 0 = Latch count of selected counter(s)

D₄: 0 = Latch status of selected counters(s)

D₃: 1 = Select Counter 2

D₂: 1 = Select Counter 1

D₁: 1 = Select Counter 0

D₀: Reserved for future expansion; Must be 0

Status register:

- shows the state of the output pin
- check the counter is in NULL state (0) or not
- how the counter is programmed

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
Output	Null Count	RW1	RW0	M2	M1	M0	BCD
D ₇	1 = OUT Pin is 1 0 = OUT Pin is 0						
D ₆	1 = Null Count 0 = Count available for reading						
D ₅ –D ₀ Counter programmed mode							

PIN DESCRIPTION OF 8253/8254

.The Intel 8253 is a programmable counter / timer chip designed for use as an Intel microcomputer peripheral. It uses N-MOS technology with a single +5V supply and is packaged in a 24-pin plastic DIP.

.It is organized as 3 independent 16-bit counters, each with a counter rate up to 2 MHz

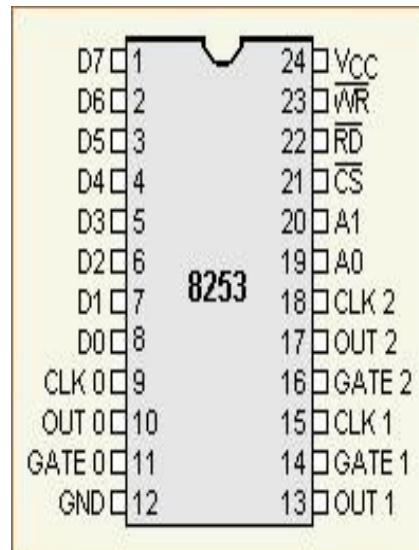
.All modes of operation are software programmable.

.Clock This is the clock input for the counter.

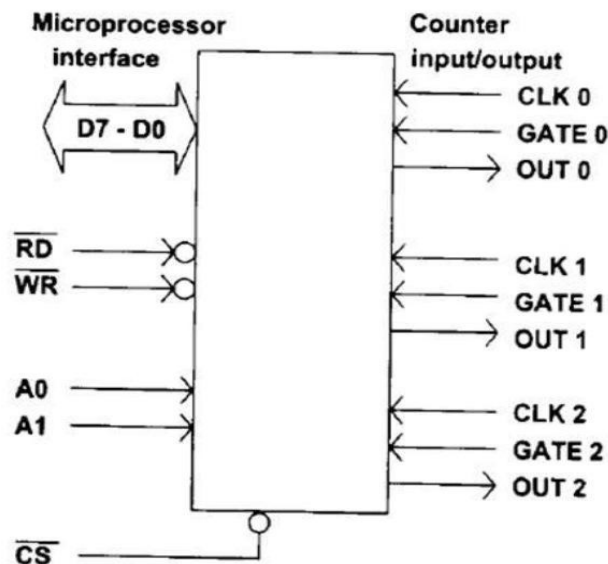
The counter is 16 bits. The maximum clock frequency is 1 / 380 nanoseconds or 2.6 megahertz. The minimum clock frequency is DC or static operation.

.Out This single output line is the signal that is the final programmed output of the device. Actual operation of the out line depends on how the device has been programmed.

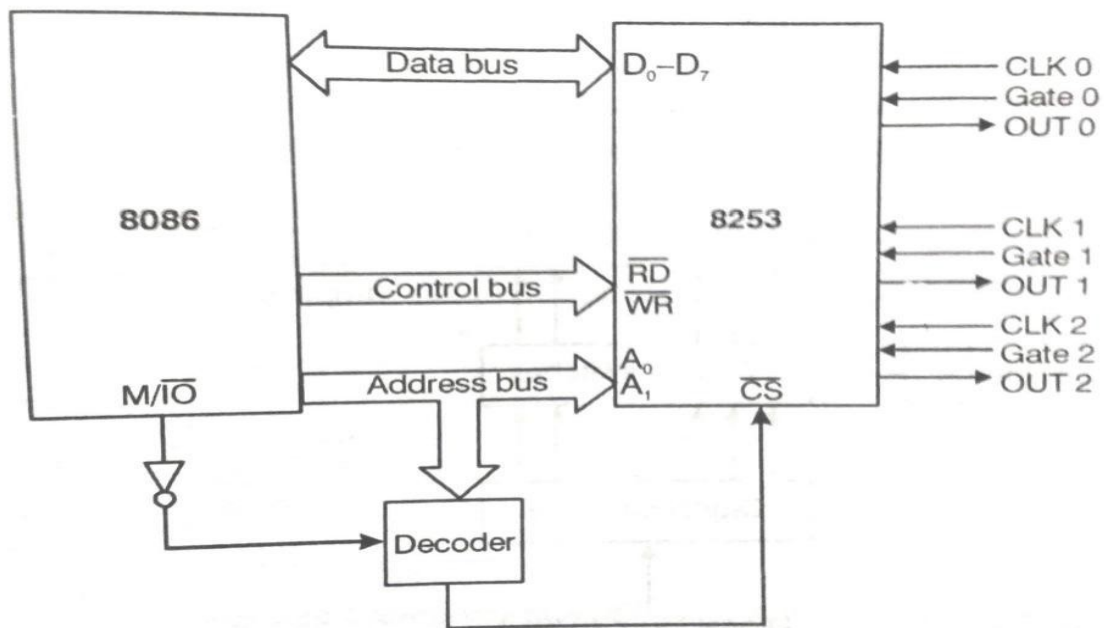
.Gate This input can act as a gate for the clock input line, or it can act as a start pulse, depending on the programmed mode of the counter.



8253/54 Pin Description



INTERFACING 8086 WITH 8253/54



8237 DMA CONTROLLER

Introduction:

- ▢ Direct Memory Access (DMA) is a method of allowing data to be moved from one location to another in a computer without intervention from the central processor (CPU).
- ▢ It is also a fast way of transferring data within (and sometimes between) computer.
- ▢ The DMA I/O technique provides direct access to the memory while the microprocessor is temporarily disabled.
- ▢ The DMA controller temporarily borrows the address bus, data bus and control bus from the microprocessor and transfers the data directly from the external devices to a series of memory locations (and vice versa).

Basic DMA Operation:

- ▢ Two control signals are used to request and acknowledge a direct memory access (DMA) transfer in the microprocessor-based system.
 - ▢ The HOLD signal as an input(to the processor) is used to request a DMA action.
 - ▢ The HLDA signal as an output that acknowledges the DMA action.
- ▢ When the processor recognizes the hold,it stops its execution and enters hold cycles.

Cont.,

- HOLD input has higher priority than INTR or NMI.
- The only microprocessor pin that has a higher priority than a HOLD is the RESET pin.
- HLDA becomes active to indicate that the processor has placed its buses at high-impedance state.

Basic DMA Definitions:

- Direct memory accesses normally occur between an I/O device and memory without the use of the microprocessor.
 - A **DMA read** transfers data from the memory to the I/O device.
 - A **DMA write** transfers data from an I/O device to memory.
- The system contains separate memory and I/O control signals.
- Hence the Memory & the I/O are controlled simultaneously

- The DMA controller provides memory with its address, and the controller signal selects the I/O device during the transfer.
- Data transfer speed is determined by speed of the memory device or a DMA controller.
- In many cases, the DMA controller **SLOWS** the speed of the system when transfers occur.
- The serial PCI (Peripheral Component Interface) Express bus transfers data at rates exceeding DMA transfers.
- This in modern systems has made DMA is less important.

The 8237 DMA controller

- Supplies memory and I/O with control signals and addresses during DMA transfer
- 4-channels (expandable)
 - 0: DRAM refresh
 - 1: Free
 - 2: Floppy disk controller
 - 3: Free
- 1.6MByte/sec transfer rate
- 64 KByte section of memory address capability with single programming
- “fly-by” controller (data does not pass through the DMA-only memory to I/O transfer capability)
- Initialization involves writing into each channel:
 - i) The address of the first byte of the block of data that must be transferred (called the base address).
 - ii) The number of bytes to be transferred (called the word count).

ACOE255

Microprocessors I - Frederick University

7

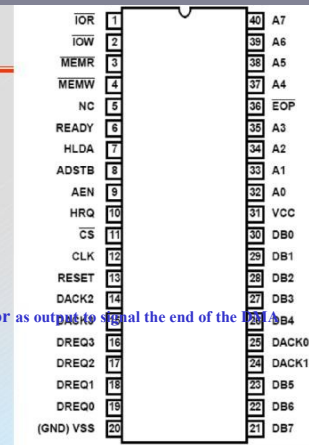
3

8237 DMA Controller

- The 8237 DMA controller supplies the memory and I/O with control signals and memory address information during the DMA transfer.
- The 8237 is capable of DMA transfers at rates of up to 1.6M bytes per second.
- Each channel is capable of addressing a full 64K-byte section of memory and can transfer up to 64K bytes with a single programming.

8237 pins

- CLK: System clock
- CS': Chip select (decoder output)
- RESET: Clears registers, sets mask register
- READY: 0 for inserting wait states
- HLDA: Signals that the μp has relinquished buses
- DREQ3 – DREQ0: DMA request input for each channel
- DB7-DB0: Data bus pins
- IOR': Bidirectional pin used during programming and during a DMA write cycle
- IOW': Bidirectional pin used during programming and during a DMA read cycle
- EOP': End of process is a bidirectional signal used as input to terminate a DMA process or as output to signal the end of the DMA transfer
- A3-A0: Address pins for selecting internal registers
- A7-A4: Outputs that provide part of the DMA transfer address
- HRQ: DMA request output
- DACK3-DACK0: DMA acknowledge for each channel.
- AEN: Address enable signal
- ADSTB: Address strobe
- MEMR': Memory read output used in DMA read cycle
- MEMW': Memory write output used in DMA write cycle



ACOE255

Microprocessors I - Frederick University

8

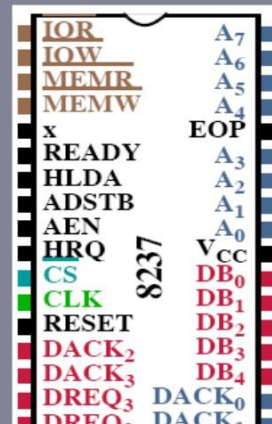
4

Important signal pins

➤ **DREQ₃ – DREQ₀ (DMA request):** Used to request a DMA transfer for a particular DMA channel.

➤ **DACK₃ – DACK₀ (DMA channel acknowledge):** Acknowledges a channel DMA request from a device.

▢ **HRQ (Hold request):** Requests a DMA transfer.



➤ **HLDA (Hold acknowledge)** signals the 8237 that the microprocessor has relinquished control of the address, data and control buses.

➤ **MEMW (Memory write):** Used as an output to cause memory to write data during a DMA write cycle.

▢ **MEMR (Memory read):** Used as an output to cause memory to read data during a DMA read cycle

- ▮ **A3 – A0** : address pins select an internal register during programming and provide part of the DMA transfer address during DMA operation.
- **A7 – A4** : address pins are outputs that provide part of the DMA transfer address during a DMA operation.
- ▮ **DB0 – DB7** : data bus, connected to microprocessor and are used during the programming DMA controller.

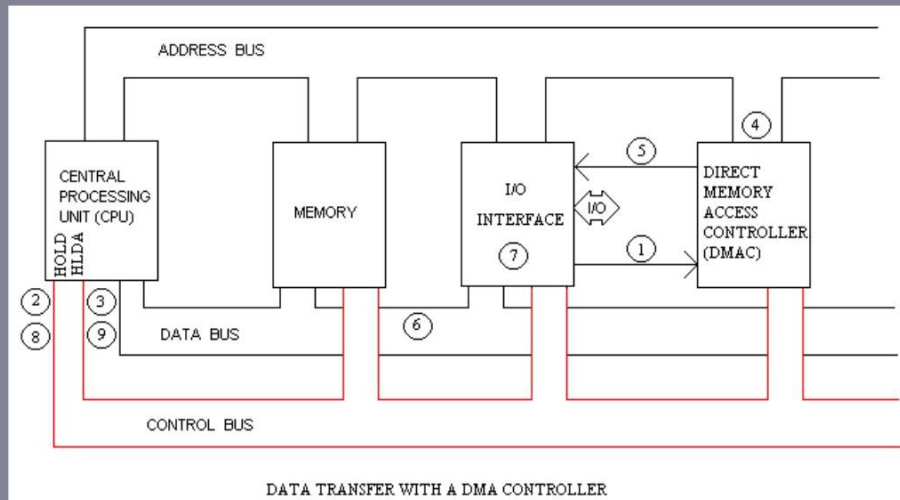
Basic DMA operation :-

- The direct memory access (DMA) I/O technique provides direct access to the memory while the microprocessor is temporarily disabled.
- ▢ A DMA controller temporarily borrows the address bus, data bus, and control bus from the microprocessor and transfers the data bytes directly between an I/O port and a series of memory locations.
- ▢ The DMA transfer is also used to do high-speed memory-to memory transfers.
- ▢ Two control signals are used to request and acknowledge a DMA transfer in the microprocessor-based system.

- The HOLD signal is a bus request signal which asks the microprocessor to release control of the buses after the current bus cycle.

- ▢ The HLDA signal is a bus grant signal which indicates that the microprocessor has indeed released control of its buses by placing the buses at their high-impedance states.
- ▢ The HOLD input has a higher priority than the INTR or NMI interrupt inputs.

Data transfer with a DMA Controller



Data transfer with DMA controller

- During a block input byte transfer, the following sequence occurs as the data byte is sent from the interface to the memory:
- The interface sends the DMA controller a request for DMA service.
- A Bus request is made to the HOLD pin (active High) on the 8086 microprocessor and the controller gains control of the bus.
- A Bus grant is returned to the DMA controller from the Hold Acknowledge (HLDA) pin (active High) on the 8086 microprocessor.

➤ The DMA controller places contents of the address register onto the address bus. ➤ The controller sends the interface a DMA acknowledgment, which tells the interface to put data on the data bus. (For an output it signals the interface to latch the next data placed on the bus.)

➤ The data byte is transferred to the memory location indicated by the address bus. ➤ The interface latches the data.

- The Bus request is dropped, the HOLD pin goes Low, and the controller relinquishes the bus.
- The Bus grant from the 8086 microprocessor is dropped and the HLDA pin goes Low.
- The address register is incremented by 1.
- The byte count is decremented by 1.
- If the byte count is non-zero, return to step 1, otherwise stop

Register Organization of 8237

8237 Internal Registers

CAR

- ▢ The current address register holds a 16-bit memory address used for the DMA transfer.
- ▢ Each channel has its own current address register for this purpose.
- ▢ When a byte of data is transferred during a DMA operation, CAR is either incremented or decremented depending on how it is programmed.

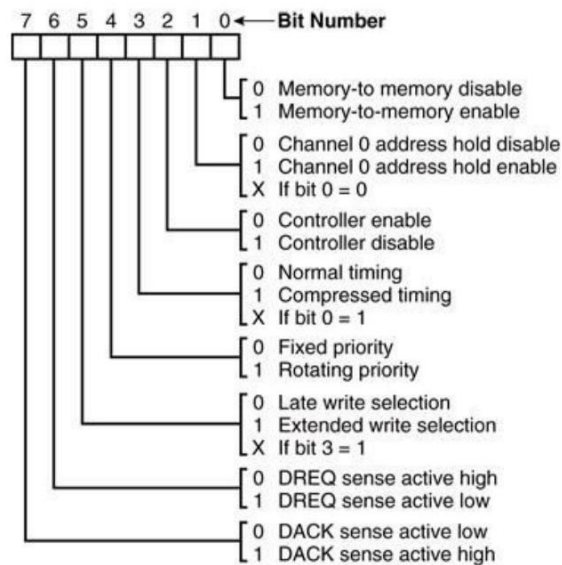
CWCR

- ▢ The current word count register programs a channel for the number of bytes to transferred during a DMA action.

CR

- ▢ The command register programs the operation of the 8237DMAcontroller.
- ▢ The register uses bit position 0 to select the memory-to-memoryDMAtransfer mode.
 - ▢ Memory-to-memory DMA transfers use DMA channel 0tohold the source address
 - ▢ DMAchannel 1 holds the destination address

command register.

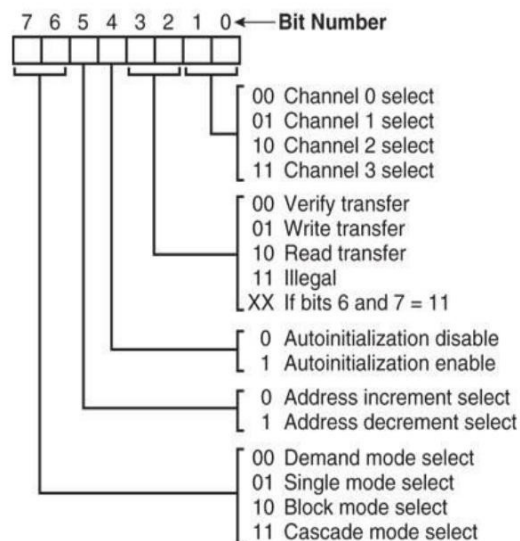


BA and BWC

- ▢ The base address (BA) and base word count (BWC) registers are used when auto-initialization is selected for a channel.
- ▢ In auto-initialization mode, these registers are used to reload the CAR and CWCR after the DMA action is completed.

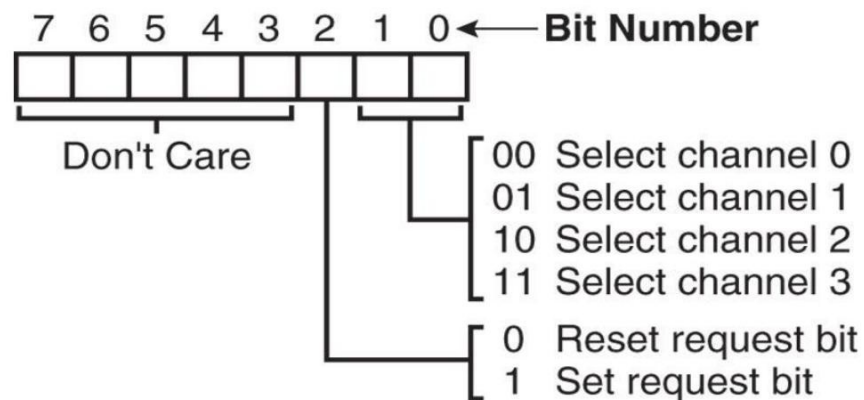
MR

- ▢ The **mode register** programs the mode of operation for a channel.
- ▢ Each channel has its own mode register as selected by bit positions 1 and 0.
- ▢ Remaining bits of the mode register select operation, auto-initialization, increment/decrement, and mode for the channel



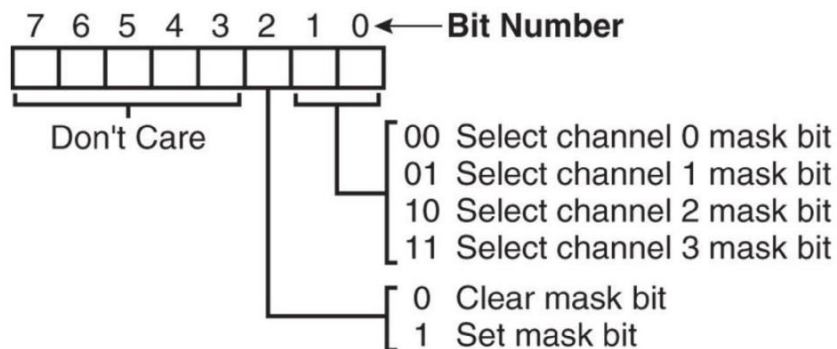
BR

- The **bus request register** is used to request a DMA transfer via software.
- very useful in memory-to-memory transfers, where an external signal is not available to begin the DMA transfer



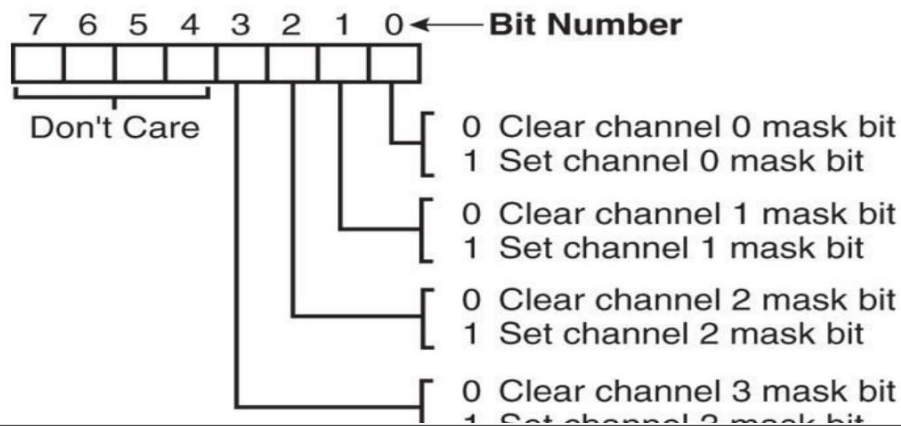
MRSR

- The **mask register set/reset** sets or clears the channel mask.
- if the mask is set, the channel is disabled
- the **RESET** signal sets all channel masks to disable them



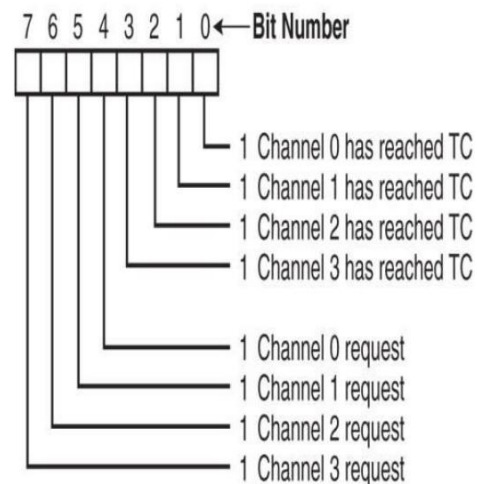
MSR

- The **mask register** clears or sets all of the masks with one command instead of individual channels, as with the MRSR.



SR

- The **status register** shows status of each DMA channel. The TC bits indicate if the channel has reached its terminal count (transferred all its bytes).
- When the terminal count is reached, the DMA transfer is terminated for most modes of operation.
- The request bits indicate whether the DREQ input for a given channel is active.



1. Current Address Register

- Each of 4 DMA channels of 8237 has a 16-bit current address register that holds the current memory address, being accessed during DMA transfer.
- The address is automatically incremented or decremented after each transfer & resulting address value is again stored current address register.
- This can be byte – wise programmed by CPU i.e. lower byte 1st & higher byte later.

2.Current Word Register

- Each channel has a 16- bit current word register that holds the number of data byte transfers of be carried out.
- The word count is decremented after each transfer & new value is again stored back to the current word register .
- When count becomes zero an EOP signal will be generated . This can be written in successive bytes by the CPU ,in program mode

3. Base Address & Base Word Count Register

- Each channel has a pair of these register . These maintains an original copy of the resp. initial current address register & current word register (before increment or decrement) resp.
- These are automatically written along with current register .
- These cannot be read by the CPU . The contents of these register are used internally for auto – initialization.

4. Command Register

- This is 8 – bit controls the complete operation of 8237.
- This can be programmed by the CPU and cleared by a reset operation .

5.Mode Register

- Each of channel has 8 bit mode register . This is written by the CPU program mode.
- Bits 0 or 1 of mode register determine which of the four channel mode register is to be written . ➤ The bits 2 & 3 indicate the type of DMA transfer. ➤ Bit 4 indicates whether auto- initialization is selected or not , while bit 5 indicates whether address increment or decrement mode is selected .

6.Request Register

- Each channel has a request register bit associated with it , in the request register . ➤ These are nonmaskable & subject to prioritization by the priority resolving network of 8237 .
- Each bit is set or reset under program control or is cleared upon generation of a TC or an external EOP. This register is cleared by reset.

7.Mask Register

- Each of 4 channel has a mask bit which can be set under program control to disable the incoming DREQ request at the specific channel . ➤ This bit is set when corresponding channel produces an EOP signal , if channel is not programmed for auto – initialization .
- The register is set to FFH after a reset operation. This disables all the DMA request till the mask register is cleared

8.Temporary Register

- The temporary register holds data during memory to memory data transfers .
- After the completion of the transfer operation , the last word transferred remains in the temporary register till it is cleared by the reset operation .

9. Status Register

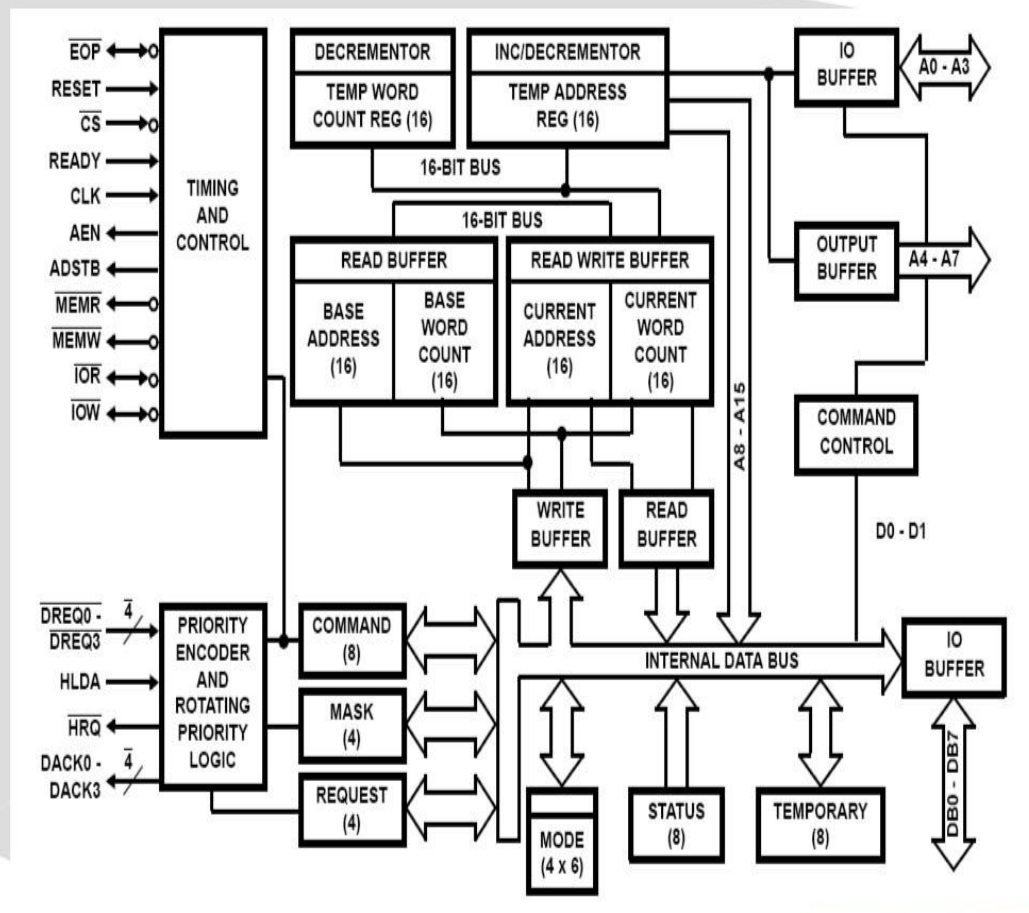
- Status register keeps the track of the all DMA channel pending request & status of their terminal count .
- The bit Do-D3 are updates every time , the corresponding channel reaches TC or external EOP occurs . These are cleared upon reset and also on each status read operation .

OPERATION	A3	A2	A1	A0	$\overline{\text{IOR}}$	$\overline{\text{IOW}}$
Read Status Register	1	0	0	0	0	1
Write Command Register	1	0	0	0	1	0
Read Request Register	1	0	0	1	0	1
Write Request Register	1	0	0	1	1	0
Read Command Register	1	0	1	0	0	1
Write Single Mask Bit	1	0	1	0	1	0
Read Mode Register	1	0	1	1	0	1
Write Mode Register	1	0	1	1	1	0
Set First/Last F/F	1	1	0	0	0	1
Clear First/Last F/F	1	1	0	0	1	0
Read Temporary Register	1	1	0	1	0	1
Master Clear	1	1	0	1	1	0
Clear Mode Reg. Counter	1	1	1	0	0	1
Clear Mask Register	1	1	1	0	1	0
Read All Mask Bits	1	1	1	1	0	1
Write All Mask Bits	1	1	1	1	1	0

- **Clear First/Last Flip-Flop** - This command is executed prior to writing or reading new address or word count information to the 82C37. This command initializes the flipflop to a known state (low byte first) so that subsequent accesses to register contents by the microprocessor will address upper and lower bytes in the correct sequence.
- **Set First/Last Flip-Flop** - This command will set the flip-flop to select the high byte first on read and write operations to address and word count registers.
- **Master Clear** - This software instruction has the same effect as the hardware Reset. The Command, Status, Request, and Temporary registers, and Internal First/Last Flip-Flop and mode register counter are cleared and the Mask register is set. The 82C37A will enter the idle cycle.
- **Clear Mask Register** - This command clears the mask bits of all four channels, enabling them to accept DMA requests.
- **Clear Mode Register Counter** - Since only one address location is available for reading the Mode registers, an internal two-bit counter has been

included to select Mode registers during read operation. To read the Mode registers, first execute the Clear Mode Register Counter command, then do consecutive reads until the desired channel is read. Read order is channel 0 first, channel 3 last. The lower two bits on all Mode registers will read as ones.

8237 block diagram



Initiating a DMA transaction

- **Save the current interrupt status and disable interrupts by executing the CLI instruction**
- **Disable the channel that will be used for the transaction**
- **Reset the flip-flop by writing a value of 0X to the register**
- **Set the Mode Register**
- **Set the Page Register**
- **Set the Offset Register**
- **Set the Block Size Register**
- **Enable the channel that will be used for the transaction**
- **Restore the interrupt status**

ACOE255

Microprocessors I - Frederick University

17

Programming the 8237

- **First program the address and count registers first:**
 - 1. **Clear the F/L flip-flop with a clear F/L command**
 - 2. **Disable the channel**
 - 3. **Program the LSB and then MSB of the address**
 - 4. **Program the LSB and then MSB of the count**
- **select the mode of operation**
- **Enable channel**

ACOE255

Microprocessors I - Frederick University

18

25

Data Transfer modes

1) Single Transfer Mode

- ▢ In Single Transfer mode the device is programmed to make one transfer only.
- ▢ The word count will be decremented and the address decremented or incremented following each transfer.
- ▢ When the word count "rolls over" from zero to FFFFH, a Terminal Count (TC) will cause an Auto initialize if the channel has been programmed to do so.

2) Block Transfer Mode

- In Block Transfer mode the device is activated by DREQ to continue making transfers during the service until a TC, caused by word count going to FFFFH, or an external End of Process (EOP) is encountered.
- DREQ need only be held active until DACK becomes active. Again, an Auto initialization will occur at the end of the service if the channel has been programmed for it.

3) Demand transfer mode

- Device continues transfer until a TC is reached or an external EOP is detected or the DREQ signal goes inactive .
- After the I/O device is able to catch up , the service may be re-established activating DREQ signal again.
- Only EOP generated by TC or external EOP can cause the auto-initialization and only if it is programmed for .

4) Cascade mode

- In this mode ,more than one 8237 can be connected together to provide more than four DMA channels .
- The HRQ & HLDA signals from additional 8237s are connected with DREQ & DACK pins of channel of the host rsp.
- The priority of the DMA requests may be preserved at each level.

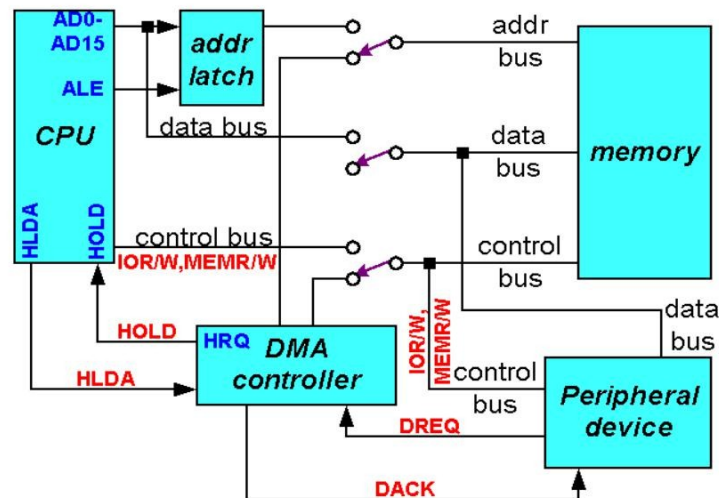
5)Memory to memory transfer

- To perform the transfer of the block of data from one set of memory address to another one, this mode is used.
- The transfer is initialize by setting DREQ0 using software command .
- The 8237 sends HRQ signal to CPU as usual & when HLDA signal is activated by CPU ,device starts operating in block transfer mode to read the data from file.
- The channel 0 current register acts as a source pointer.
- The byte read data from memory is stored in an internal temporary register of 8237.
- The channel 1 current register acts as a destination pointer to write the data from temporary register to the destination memory.
- The pointers are automatically incremented or decremented , depending upon the programming .

The diagram illustrates the system architecture of the 8085 microprocessor. It shows the 8085 microprocessor connected to Memory, an Interface, and an I/O Device. The system uses three main buses: Control, Address, and Data. The 8085 microprocessor has pins labeled HOLD and HLDA. The DMA controller is connected to the Interface, which in turn connects to the I/O Device. Red arrows indicate the flow of data and control signals between the components.

The diagram illustrates the 8085 microprocessor system architecture. The CPU is connected to Memory and a Peripheral device. The CPU's AD0-AD15 bus is connected to the Memory's address bus. The CPU's ALE signal is connected to the Memory's data bus. The CPU's HOLD signal is connected to the DMA controller's HOLD signal. The CPU's HLDA signal is connected to the DMA controller's HLDA signal. The DMA controller's HRQ signal is connected to the CPU's IOR/W, MEMR/W signal. The DMA controller's DREQ signal is connected to the Peripheral device's DACK signal. The DMA controller's control bus is connected to the Peripheral device's control bus. The DMA controller's data bus is connected to the Peripheral device's data bus. The DMA controller's IOR/W, MEMR/W signal is connected to the Peripheral device's IOR/W, MEMR/W signal. The DMA controller's control bus is connected to the Peripheral device's control bus. The DMA controller's data bus is connected to the Peripheral device's data bus. The DMA controller's IOR/W, MEMR/W signal is connected to the Peripheral device's IOR/W, MEMR/W signal.

When DMA operates:



8251

10.10 SERIAL COMMUNICATION INTERFACE

Serial Communication Port: I/O interface, used to connect peripheral units, such as CRT terminals, modems, and printers, to a microcomputer.

It permits data to be transferred between two units using just two data lines

One line is used for transmitting data and the other for receiving data.

Types of serial data communications.

Synchronous communications.

Asynchronous communications.

Synchronous vs. Asynchronous

A communication protocol is a convention for data transmission that include such functions as timing, control, formatting, and data presentation. There are two categories depending on the clocking of the data on the serial link:

- Synchronous protocols--each successive datum in a stream of data is governed by a master clock and appears at a specific interval in time.
- Asynchronous protocols--successive data appear in the data stream at arbitrary times, with no specific clock control governing the relative delays between data.

There are special IC chips made for serial data communications. These chip is called UART (universal asynchronous receiver transmitter) and USART (universal synchronous-asynchronous receiver-transmitter) 8251.

Synchronous communications

The receiver and transmitter sections of the two pieces of equipment communicating with each other must run synchronously.

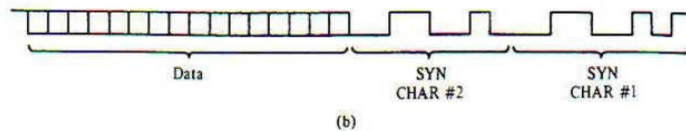
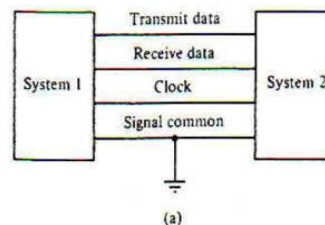
To initiate synchronous transmission, the transmitter first sends out synchronization characters to the receiver.

The receiver reads the synchronization bit pattern and compares it to a known sync pattern. Once they are identified as being the same, the receiver begins to read character data off the data line.

Transfer of data continues until the complete block of data is received.

If large blocks of data are being sent, the synchronization characters may be periodically resent to assure that synchronization is maintained.

Used in applications where high speed data transfer is required.



32

Asynchronous communications

The asynchronous method of communication eliminates the need for the Clock signal.

The simplest form of an asynchronous communication interface could consist of a Receive data, Transmit data, and Signal common communication line.

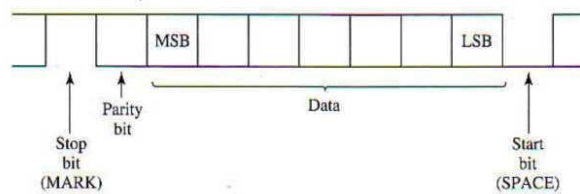
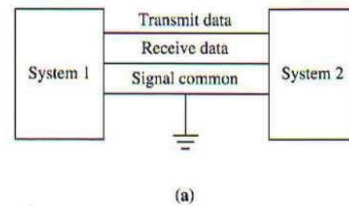
The data to be transmitted are sent out one character at a time, and at the receiver examining synchronization bits that are included at the beginning and end of each character performs end of the communication line synchronization.

The synchronization bit at the beginning of the character is called the Start bit, and that at the end of the character the stop bit.

Depending on the communications scheme, 1, 1.5, or 2 stop bits can be used.

The bits of the character are embedded between the start and stop bits.

7-bit ASCII can be used and parity added as an eighth bit for higher reliability in transmission.



Baud Rate and the Baud-Rate Generator

Baud rate: The rate at which data transfers take place over the receive and transmit lines.

By baud rate we mean the number of bits of data transferred per second.

some of the common data transfer rates

300 bps
1200 bps
9600 bps

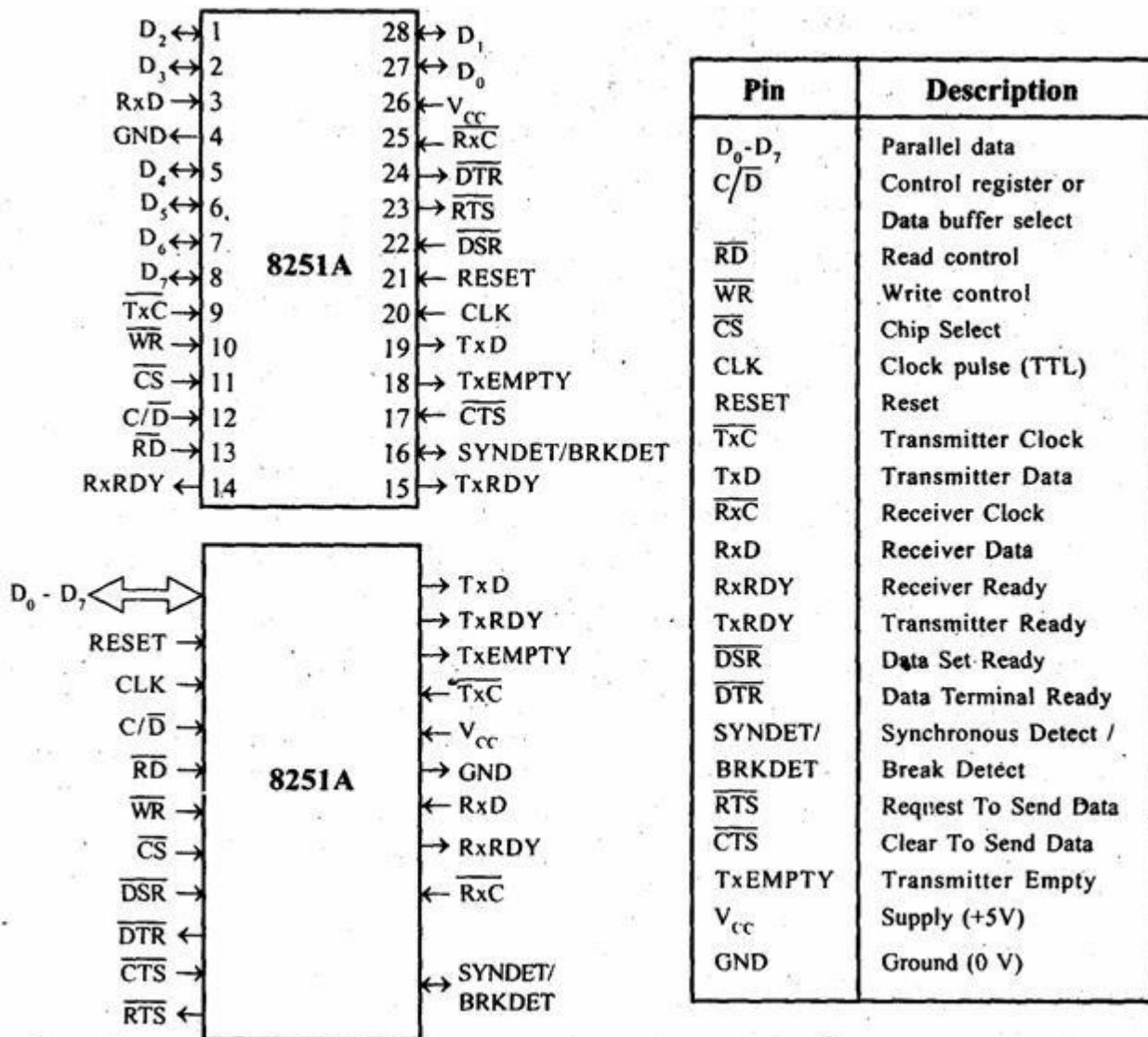
Baud rate is set by a part of the serial communication interface called the

Baud rate generator.

The baud rate at which data are transferred determines the bit time, that is, the amount of time each bit of data is on the communication line. At 300 baud rate, the bit time is found to be 3.33 ms.

PIN DESCRIPTION

- ▣ The 8251A is a programmable serial communication interface chip designed for synchronous and asynchronous serial data communication.
- ▣ It supports the serial transmission of data.
- ▣ It is packed in a 28 pin DIP.



Read/Write control logic:

- ▣ The Read/Write Control logic interfaces the 8251A with CPU, determines the functions of the 8251A according to the control word written into its control register.
- ▣ It monitors the data flow.

- This section has three registers and they are control register, status register and data buffer.
- The active low signals RD, WR, CS and C/D(Low) are used for read/write operations with these three registers.
- When C/D(low) is high, the control register is selected for writing control word or reading status word.
- When C/D(low) is low, the data buffer is selected for read/write operation.
- When the reset is high, it forces 8251A into the idle mode.
- The clock input is necessary for 8251A for communication with CPU and this clock does not control either the serial transmission or the reception rate.

Transmitter section:

- The transmitter section accepts parallel data from CPU and converts them into serial data.
- The transmitter section is double buffered, i.e., it has a buffer register to hold an 8-bit parallel data and another register called output register to convert the parallel data into serial bits.
- When output register is empty, the data is transferred from buffer to output register. Now the processor can again load another data in buffer register.
- If buffer register is empty, then TxRDY is goes to high.
- If output register is empty then TxEMPTY goes to high.
- The clock signal, TxC (low) controls the rate at which the bits are transmitted by the USART.
- The clock frequency can be 1,16 or 64 times the baud rate.

Receiver Section:

- The receiver section accepts serial data and convert them into parallel data
- The receiver section is double buffered, i.e., it has an input register to receive serial data and convert to parallel, and a buffer register to hold the parallel data.
- When the RxD line goes low, the control logic assumes it as a START bit, waits for half a bit time and samples the line again.
- If the line is still low, then the input register accepts the following bits, forms a character and loads it into the buffer register.
- The CPU reads the parallel data from the buffer register.
- When the input register loads a parallel data to buffer register, the RxRDY line goes high.
- The clock signal RxC (low) controls the rate at which bits are received by the USART.

- During asynchronous mode, the signal SYNDET/BRKDET will indicate the break in the data transmission.
- During synchronous mode, the signal SYNDET/BRKDET will indicate the reception of synchronous character.

MODEM Control:

- The MODEM control unit allows to interface a MODEM to 8251A and to establish data communication through MODEM over telephone lines.
- This unit takes care of handshake signals for MODEM interface.

8251A	USART
-------	-------

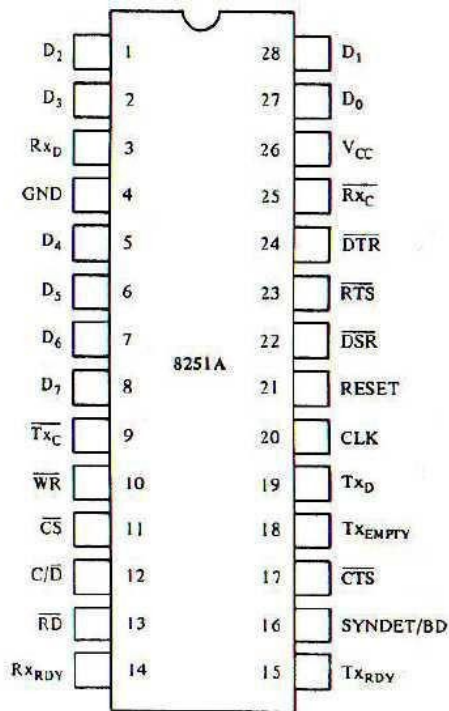
Includes four key sections:

the bus interface section, which consists of the data bus buffer and read/write control logic blocks.

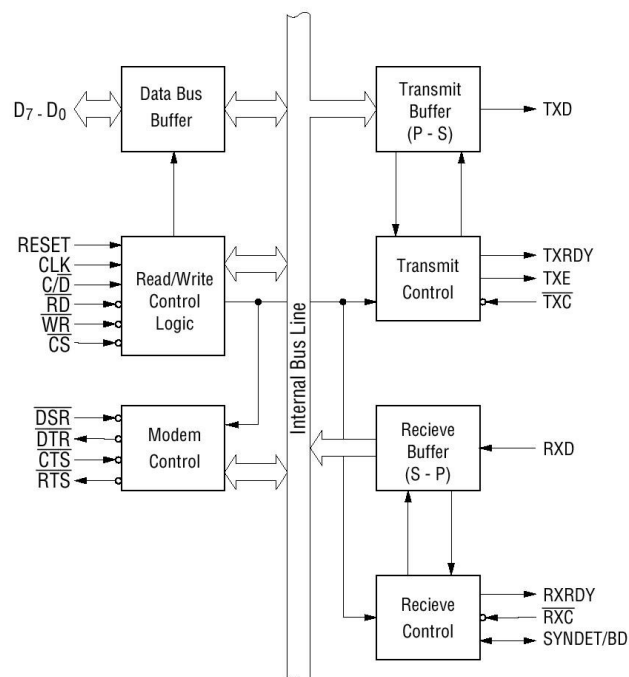
the transmit section, which consists of the transmit buffer and transmit control block.

the receive section, which consists of the receive buffer and receive control block.

the modem-control section.

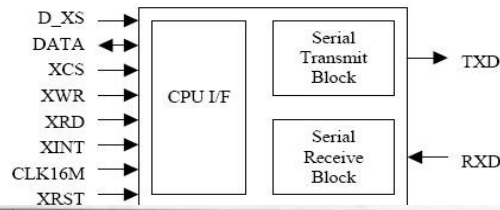


8251 Block Diagram



- The UART is a universal asynchronous receiver/transmitter, which is modeled on the real-world Intel® 8251 peripheral interface adapter component. In the model we are considering, the UART consists of three main blocks.

- a serial transmitter block
- a serial receiver block
- a CPU interface I/F block.



Port	Type	Description	Width
D_XS	Input	CPU control signal	1
DATA	InOut	Data received from or sent to the CPU	8
XCS	Input	Chip Select	1
XWR	Input	CPU write control signal	1
XRD	Input	CPU read control signal	1
CLK16M	Input	16 MHz system clock	1
XRST	Input	System reset	1
XINT	Output	Interrupt Factor	1

C/D	RD	WR	CS	Operation
0	0	1	0	8251A Data → Data bus
0	1	0	0	Data bus → 8251A Data
1	0	1	0	Status → Data bus
1	1	0	0	Data bus → Control
X	1	1	0	Data bus → 3-State
X	X	X	1	Data bus → 3-State

The receiver section

Is responsible for reading the serial bit stream of data at the RxD (receive data) input and converting it to parallel form.

When a mark voltage level is detected on this line, indicating a start bit, the receiver enables a counter

As the counter increments to a value equal to one-half a bit time, the logic level at the RxD line is sampled again.

If it is still at the mark level, a valid start pulse has been detected.

Then RxD is examined every time the counter increments through another bit time.

This continues until a complete character is assembled and the stop bit is read.

After this, the complete character is transferred into the receive-data register.

During reception of a character

the receiver automatically checks the character data for parity, framing, or overrun errors.

If one of these conditions occurs, it is flagged by setting a bit in the status register.

Then the $RxRDY$ (receiver ready) output is switched to the 1 logic level.

This signal is sent to the microprocessor to tell it that a character is available and should be read from the receive-data register.

$RxRDY$ is automatically reset to logic 0 when the MPU reads the contents of the receive_data register.

Through software, the 8251A can be set up to internally divide the Clock signal input at Rx_c by 1, 16, or 64 to obtain the desired baud rate.

The transmitter section

does the opposite of the receiver section.

It receives parallel character data from the MPU over the data bus.

The character is then automatically framed with the start bit, appropriate parity bit and the correct number of stop bits and put into the transmit-data buffer register.

Then, the serial output on the TxD line will be sent.

The $TxRDY$ output switches to logic 1. This signal can be returned to the MPU to tell it that another character should be output to the transmitter section.

In most applications, the transmitter receiver operate on the same baud rate. Therefore, the same baud-rate generator supplies both Rx_c and Tx_c .

Configuration of 8251A

8251 can be configured for various modes of operation through software.

It has three internal control registers:

**mode-control register
command register
status register.**

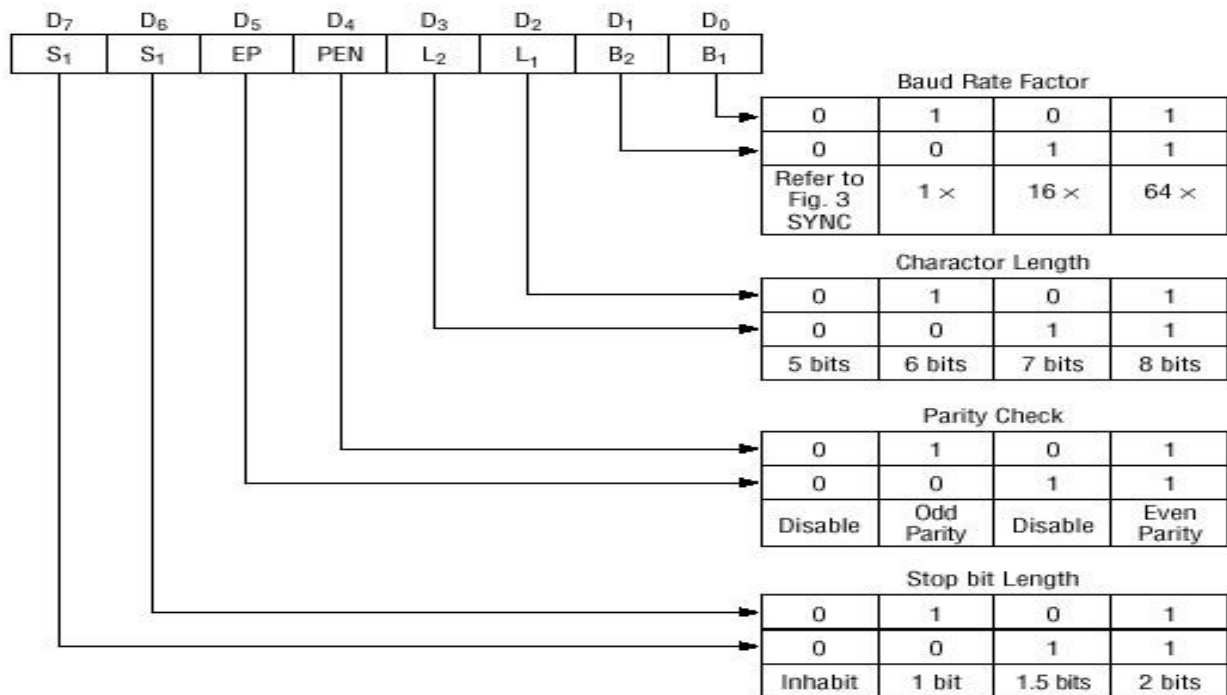
1. MODE INSTRUCTION WORD

This format defines the Baud rate, Character length, Parity and Stop bits required to work with asynchronous data communication. By selecting the appropriate baud factor sync mode, the 8251 can be operated in Synchronous mode.

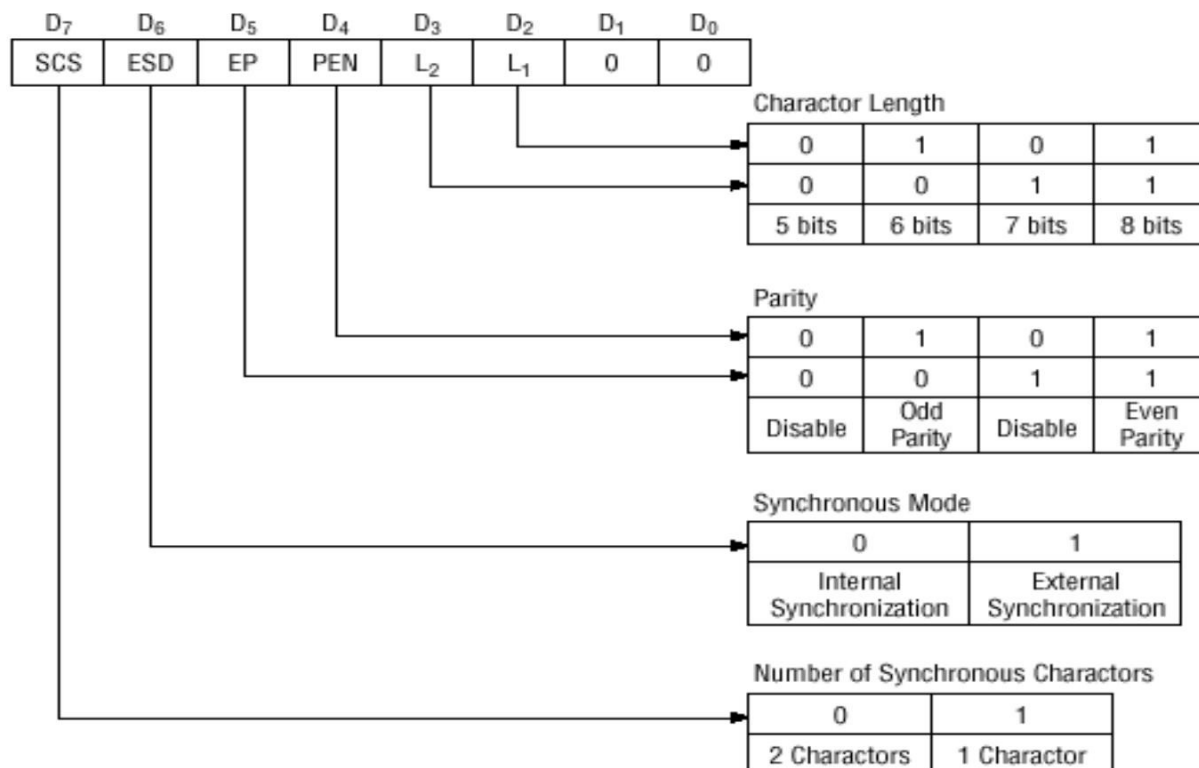
Initializing 8251 using the mode instruction to the following conditions

8 Bit data
No Parity
Baud rate Factor (16X)
1 Stop Bit

Mode Instruction (Asynchronous)



Mode Instruction (Synchronous)

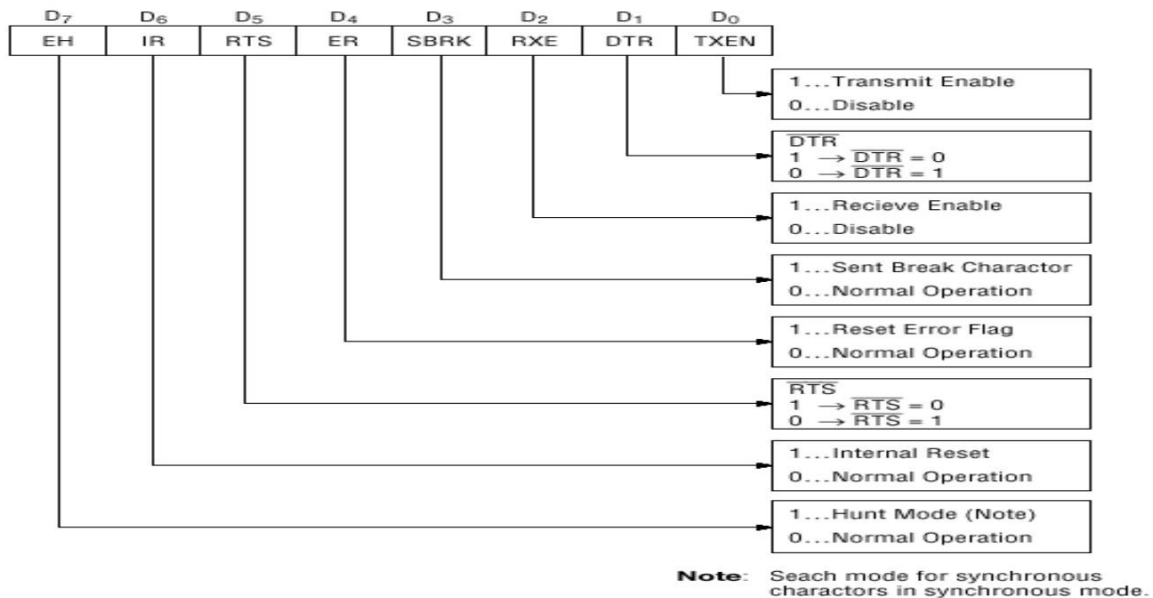


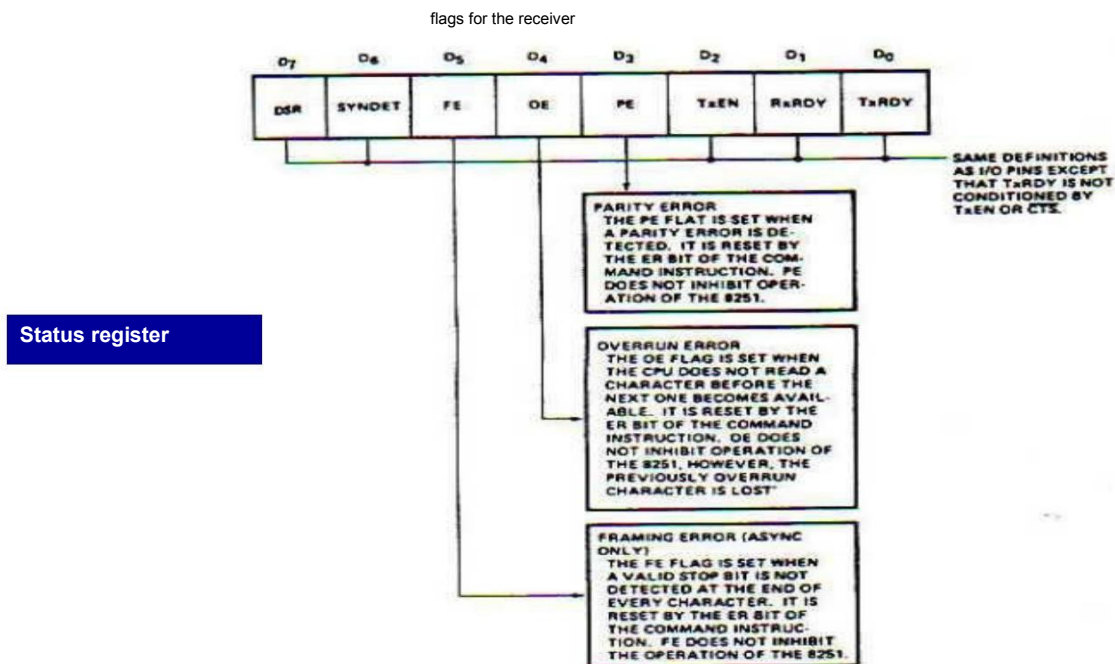
T_{XEN} and R_{XEN} are enable bits for the transmitter and receiver. Since both the receiver and transmitter can operate simultaneously, these two bits can both be set. R_{XEN} is actually an enable signal to the R_{XRDY} signal. It does not turn the receiver section on and off. The receiver runs at all times, but if R_{XEN} is set to 0, the 8251A does not signal the MPU that a character has been received by switching R_{XRDY} to logic 1. The same is true for T_{XEN} . It enables the T_{XRDY} signal.

2. COMMAND INSTRUCTION WORD

This format defines a status word that is used to control the actual operation of 8251. All control words written into 8251 after the mode instruction will load the command instruction.

The command instructions can be written into 8251 at any time in the data block during the operation of the 8251. to return to the mode instruction format, the master reset bit in the command instruction word can be set to initiate an internal reset operation which automatically places the 8251 back into the mode instruction format. Command instructions must follow the mode instructions or sync characters.



**NOTE:**

1. TxRDY status bit has different meanings from the TxRDY output pin. The former is not conditioned by $\overline{\text{CTS}}$ and TxEN; the latter is conditioned by both $\overline{\text{CTS}}$ and TxEN.

i.e. TxRDY status bit = DB Buffer Empty

$$\text{TxRDY pin out} = \text{DB Buffer Empty} \cdot (\overline{\text{CTS}} = 0) \cdot (\text{TxEN} = 1)$$

INTERFACING WITH INTEL 8251A (USART)

- The 8251A can be either memory mapped or I/O mapped in the system.
- 8251A in I/O mapped in the system is shown in the figure.
- Using a 3-to-8 decoder generates the chip select signals for I/O mapped devices.
- The address lines A4, A5 and A6 are decoded to generate eight chip select signals (IOCS-0 to IOCS-7) and in this, the chip select signal IOCS-2 is used to select 8251A.
- The address line A7 and the control signal IO / M(low) are used as enable for decoder.
- The address line A0 of 8085 is connected to C/D(low) of 8251A to provide the internal addresses.
- The data lines D0 - D7 are connected to D0 - D7 of the processor to achieve parallel data transfer.
- The RESET and clock signals are supplied by the processor. Here the processor clock is directly connected to 8251A. This clock controls the parallel data transfer between the processor and 8251A.
- The output clock signal of 8085 is divided by suitable clock dividers like programmable timer 8254 and then used as clock for serial transmission and reception.

8279

Programmable Keyboard/Display Interface

- **Intel's 8279 is a general purpose** keyboard display controller that simultaneously drives the display of a system and interfaces a keyboard with the CPU, leaving it free for its routine task.

The keyboard display controller chip 8279 provides

- A set of four scan lines and eight return lines for interfacing keyboard
- A set of eight output lines for interfacing display

Keyboard segment

- **Scans the keyboard, detects key press transmits to CPU the characteristic's of key**
- Connected to a 64 contact key matrix
- Keyboard entries are debounced and stored in FIFO
- Interrupt signal is generated with each entry

Display segment

- It puts data from the CPU to display devices
- 16 character scanned display
- 16x8 R/W memory (RAM)
- Right entry or left entry

I/O Control and Data Buffers

- The I/O control section controls the flow of data to/from the 8279
- The I/O section is enabled only if CS is low.
- The pins A0, RD and WR select the command, status or data read/write operations carried out by the CPU with 8279.
- The data buffers interface the external bus of the system with internal bus of 8279.

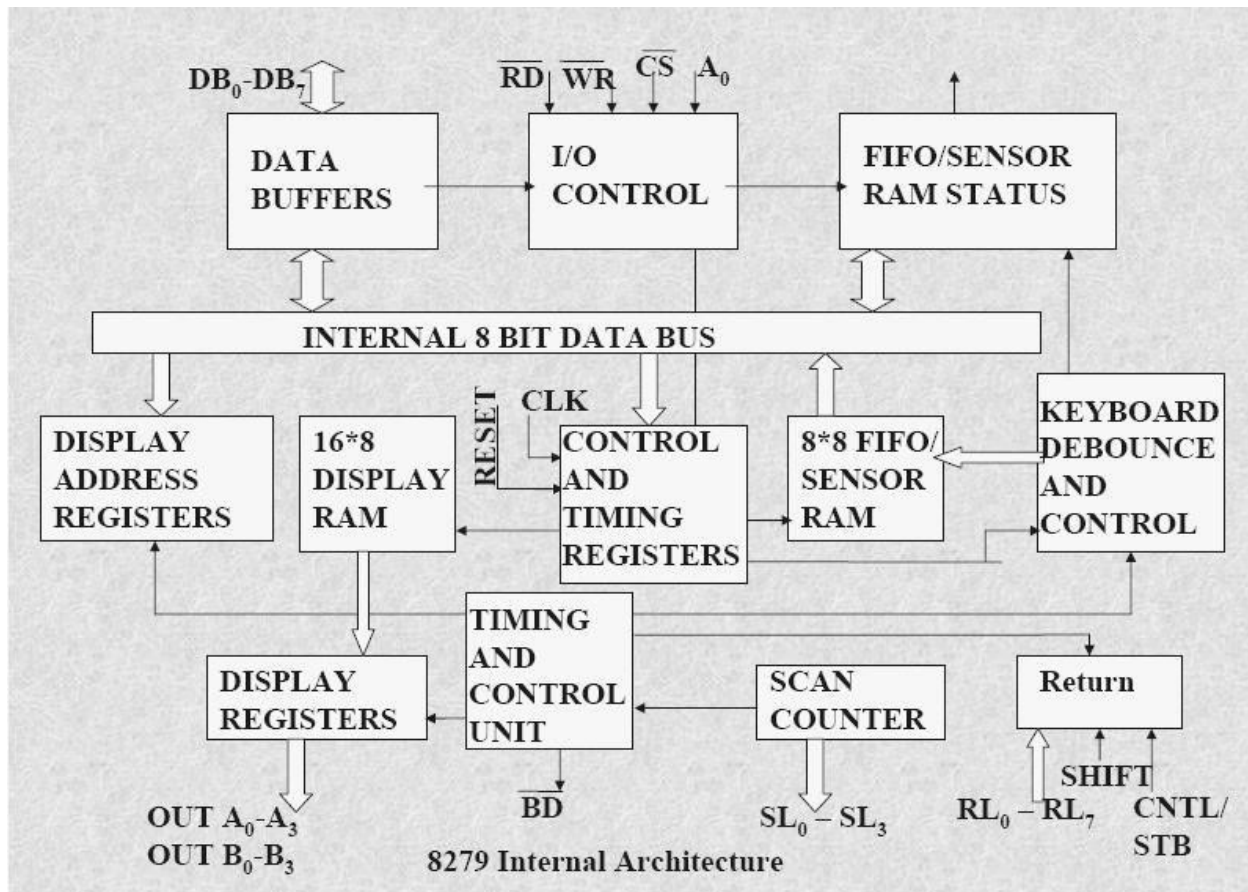
Control and Timing Register and Timing Control

- These registers store the keyboard and display modes and other operating conditions programmed by CPU.
- The registers are written with A0=1 and WR=0. The Timing and control unit controls the basic timings for the operation of the circuit.
- Scan counter divide down the operating frequency of 8279 to derive scan keyboard and scan display frequencies

Scan Counter

- The scan counter has **two modes** to scan the key matrix and refresh the display.
- In the **encoded mode**, the counter provides binary count that is to be externally decoded to provide the scan lines for keyboard and display
- Four externally decoded scan lines may drive up to 16 displays.
- In the **decode scan mode**, the counter internally decodes the least significant 2 bits and provides a decoded 1 out of 4 scan on SL0-SL3

- Four internally decoded scan lines may drive up to 4 displays.
- The keyboard and display both are in the same mode at a time.



Return Buffers and Keyboard De-bounce and Control:

- This section scans for a key closure row wise. If a key closer is detected, the keyboard debounce unit debounces the key entry (i.e. wait for 10 ms).
- After the debounce period, if the key continues to be detected, The code of key is directly transferred to the sensor RAM along with SHIFT and CONTROL key status.

Display Address Registers and Display RAM :

- The display address register holds the address of the word currently being written or read by the CPU to or from the display RAM.
- The contents of the registers are automatically updated by 8279 to accept the next data entry by CPU.

FIFO/Sensor RAM and Status Logic:

- **In keyboard or strobed input mode, this block acts as 8-byte first-in-first out (FIFO) RAM.**

- Each key code of the pressed key is entered in the order of the entry and in the mean time read by the CPU, till the RAM become empty.
- The status logic generates an interrupt after each FIFO read operation till the FIFO is empty.
- **In scanned sensor matrix mode, this unit acts as sensor RAM.**
 - Each row of the sensor RAM is loaded with the status of the corresponding row of sensors in the matrix.
 - If a sensor changes its state, the IRQ line goes high to interrupt the CPU.

Operating modes of 8279**INPUT (keyboard) MODES**

- Scanned keyboard mode
- Scanned sensor matrix mode
- Strobed input **mode**

2 key lock out

- If two keys are pressed within a debounce cycle (simultaneously), no key is recognized till one of them remains closed and the other is released. The last key, that remains depressed is considered as single valid key depression

N – key roll over

- When a key is pressed, the debounce circuit waits for 2 keyboards scans and then checks whether the key is still depressed. If it is still depressed, the code is entered in FIFO RAM.

OUTPUT (display) MODES

- Display scan
- Display entry

Display Scan :

- **In this mode 8279 provides 8 or 16 character multiplexed displays those can be organized as dual 4- bit or single 8-bit display units.**

Display Entry (Right entry or Left entry mode)

- **8279** allows options for data entry on the displays.
- The display data is entered for display either from the right side or from the left side.

Example :

Left Entry Mode (TYPE WRITER)

Right Entry Mode (CALCULATOR)

8279 contains the following features:

- **Simultaneous and independent scanning of a keyboard and refresh of a display,**
- **significantly offloading these functions from the microprocessor.**

Keyboard section:

- 8-character Keyboard FIFO
- 2-Key Lockout or N-key Rollover with Contact Debounce
- Interrupt Output on Key Entry
- Programmable Keyboard Scan & Debounce rates

Display Section:

- Dual 8- or 16-Numeric Display
- Single 16-Character Display
- Right or Left Entry 16-Byte Display RAM with address auto increment
- Programmable display refresh rate

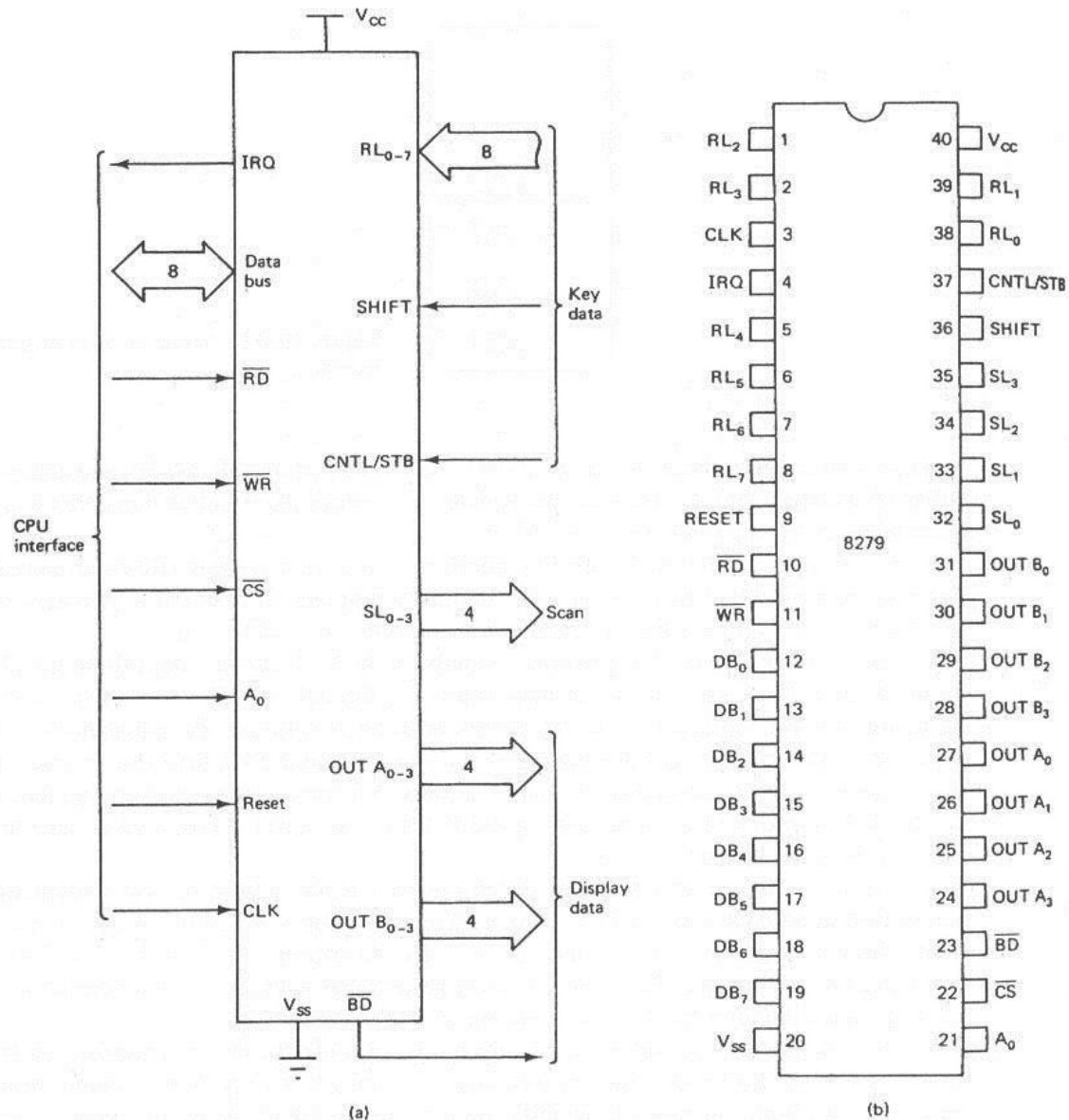


Figure 10-73 (a) Block diagram of the 8279. (Reprinted by permission of Intel Corporation. Copyright/Intel Corp. 1987) (b) Pin layout. (Reprinted by permission of Intel Corporation. Copyright/Intel Corp. 1987)

Pinout Definition 8279

- P A0: Selects data (0) or control/status (1) for reads and writes between micro and 8279.
- P BD: Output that blanks the displays.
- P CLK: Used internally for timing. Max is 3 MHz.
- P CN/ST: Control/strobe, connected to the control key on the keyboard.
- P CS: Chip select that enables programming, reading the keyboard, etc.
- P DB7-DB0: Consists of bidirectional pins that connect to data bus on micro.
- P IRQ: Interrupt request, becomes 1 when a key is pressed, data is available.
- P OUT A3-A0/B3-B0: Outputs that sends data to the most significant/least significant nibble of display.
- P RD(WR): Connects to micro's IORC or RD signal, reads data/status registers.
- P DB7-DB0: Consists of bidirectional pins that connect to data bus on micro.
- P RESET: Connects to system RESET.
- P RL7-RL0: Return lines are inputs used to sense key depression in the keyboard matrix.
- P Shift: Shift connects to Shift key on keyboard.
- P SL3-SL0: Scan line outputs scan both the keyboard and displays

Table 1. Pin Description

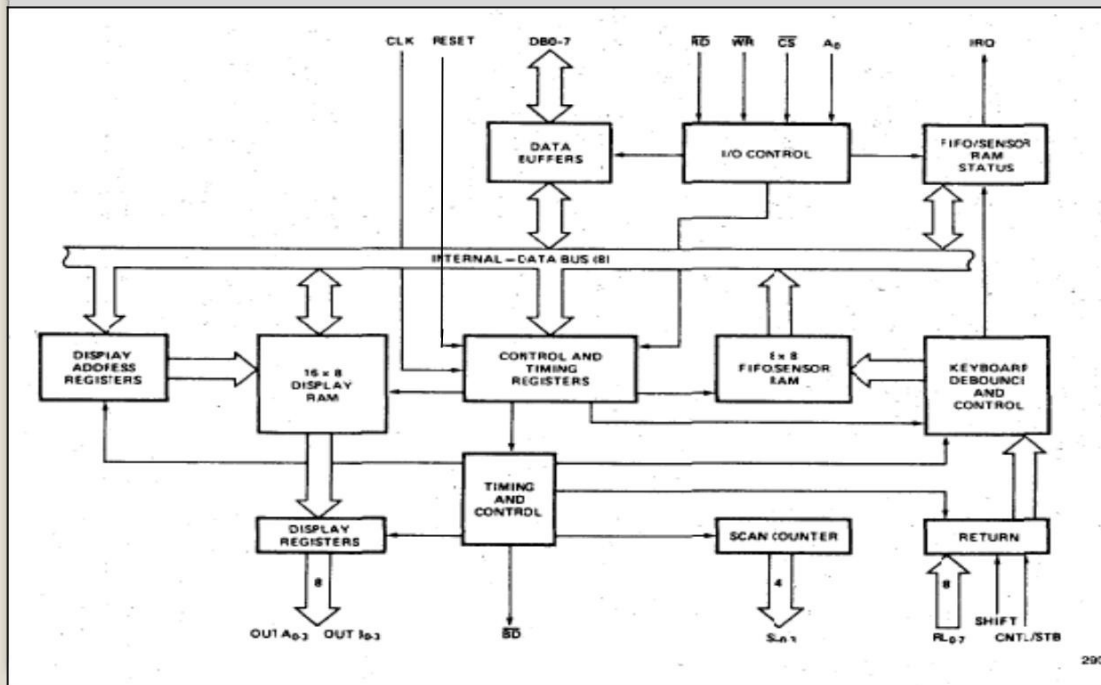
Symbol	Pin No.	Name and Function
DB ₀ -DB ₇	19-12	BI-DIRECTIONAL DATA BUS: All data and commands between the CPU and the 8279 are transmitted on these lines.
CLK	3	CLOCK: Clock from system used to generate internal timing.
RESET	9	RESET: A high signal on this pin resets the 8279. After being reset the 8279 is placed in the following mode: 1) 16 8-bit character display—left entry. 2) Encoded scan keyboard—2 key lockout. Along with this the program clock prescaler is set to 31.
CS	22	CHIP SELECT: A low on this pin enables the interface functions to receive or transmit.
A ₀	21	BUFFER ADDRESS: A high on this line indicates the signals in or out are interpreted as a command or status. A low indicates that they are data.
RD, WR	10-11	INPUT/OUTPUT READ AND WRITE: These signals enable the data buffers to either send data to the external bus or receive it from the external bus.
IRQ	4	INTERRUPT REQUEST. In a keyboard mode, the interrupt line is high when there is data in the FIFO/Sensor RAM. The interrupt line goes low with each FIFO/Sensor RAM read and returns high if there is still information in the RAM. In a sensor mode, the interrupt line goes high whenever a change in a sensor is detected.
V _{SS} , V _{CC}	20, 40	GROUND AND POWER SUPPLY PINS.
SL ₀ -SL ₃	32-35	SCAN LINES: Scan lines which are used to scan the key switch or sensor matrix and the display digits. These lines can be either encoded (1 of 16) or decoded (1 of 4).
RL ₀ -RL ₇	38, 39, 1, 2, 5-8	RETURN LINE: Return line inputs which are connected to the scan lines through the keys or sensor switches. They have active internal pullups to keep them high until a switch closure pulls one low. They also serve as an 8-bit input in the Strobed Input mode.
SHIFT	36	SHIFT: The shift input status is stored along with the key position on key closure in the Scanned Keyboard modes. It has an active internal pullup to keep it high until a switch closure pulls it low.
CNTL/STB	37	CONTROL/STROBED INPUT MODE: For keyboard modes this line is used as a control input and stored like status on a key closure. The line is also the strobe line that enters the data into the FIFO in the Strobed Input mode. (Rising Edge). It has an active internal pullup to keep it high until a switch closure pulls it low.
OUT A ₀ -OUT A ₃ OUT B ₀ -OUT B ₃	27-24 31-28	OUTPUTS: These two ports are the outputs for the 16 x 4 display refresh registers. The data from these outputs is synchronized to the scan lines (SL ₀ -SL ₃) for multiplexed digit displays. The two 4 bit ports may be blanked independently. These two ports may also be considered as one 8-bit port.
BD	23	BLANK DISPLAY: This output is used to blank the display during digit switching or by a display blanking command.

8279 KEYBOARD AND DISPLAY INTERFACING

Features:

- ▣ It is designed by Intel
 - ▣ It supports 64 contact key matrix with **two more keys "CONTROL" and "SHIFT"**
 - ▣ It provides 3 operating modes
 1. Scanned keyboard mode
 2. Scanned sensor matrix mode
 3. Strobed Input mode.
 - ▣ It has inbuilt debounce key .
-
- ▣ It provides 16 byte display RAM to display 16 digits and interfacing 16 digits.
 - ▣ It provides two output modes:
 1. Left entry (Typewriter type).
 2. Right entry (Calculator type).
 - ▣ Simultaneous keyboard and display operation facility allows to interleave keyboard and display software.
 - ▣ The interrupt output of 8279 can be used to tell CPU that the key press is detected, this eliminates the need of software polling.

Block Diagram



- It consists 4 main section.
- 1. CPU interface and control section.
- 2. Scan section
- 3. Keyboard Section
- 4. Display section.

CPU INTERFACE AND CONTROL SECTION:

It consists of

- 1. Data buffers
- 2. I/O control
- 3. Control and timing registers.
- 4. Timing and control logic.

Data Buffers:

- 8-bit bidirectional buffer.
- Used to connect the internal data bus and external data bus.

I/O control:

- I/O control section uses the A0, CS, RD and WR signals to controls the data flow.
- The data flow is enabled by CS=0 otherwise it is the high impedance state.
- A0=0 means the data is transferred.
- A0=1 means status or command word is transferred.

- I/O control signals listed below

A0	RD	WR	Interpretation
0	1	0	Data from CPU to 8279
0	0	1	Data from 8279 to CPU
1	1	0	Command word from CPU to 8279
1	0	1	Status word from 8279 to CPU

TIMING AND CONTROL REGISTERS:

- Store the keyboard and display modes and others operating condition programmed by the CPU.
- The modes are programmed by sending proper command $A_0=1$.

TIMING AND CONTROL:

- It consist timing counter chain.
- First counter is divided by N prescalar that can be programmed to give an internal frequency of 100 KHz.

- ▣ The other counter is divide by basic internal frequency .Listed below

Parameter	Timings
Keyboard and time	5.1 msec
Keyboard and debounce time	10.3 msec
Key scan time	80 μ sec
Display scan time	10.3 msec
Digit ON time	480 μ sec
Blanking time	160 μ sec
Internal clock time	10 μ sec

Scan Section

- ▮ It has two modes, 1. Encoded mode
- 2. Decoded mode.

ENCODED MODE:

- ▮ It provide binary count from 0000 to 1111 by four scan lines(SC_3 - SC_0)by active high inputs.
- ▮ It is externally decoded to provide 16 scan lines

- ▢ Display use all 16 lines to interface 16 digit 7 segment display.
- ▢ But keyboard use only 8 scan lines out of 16 lines.
- ▢ DECODED MODE:
 - ▢ In this mode ,the internal decoder decodes the least 2 significant bits.
 - ▢ It is provide four possible combination from (SC₀-SC₃) such as 1110 ,1101 , 1011 and 0111.
 - ▢ This four active low outputs line is used to directly to interface 4 –digit 7-segment display ,8*4 matrix keyboard

Keyboard section

- ▢ This is consist of,
- ▢ Return buffers.
- ▢ Keyboard debounce control.
- ▢ FIFO / sensor RAM.
- ▢ FIFO / sensor RAM status.

RETURN BUFFERS:

- ▢ 8 return lines(RL₇-RL₀) are buffered and latched by when each row scan in scanned keyboard or sensor matrix mode.
- ▢ In strobed mode ,the contents of return lines are transferred to FIFO Ram.

KEYBOARD DEBOUNCE AND CONTROL:

- ▣ It is enabled only when keyboard mode is selected.
- ▣ In this mode , return lines are scanned whether any keys are closed in the row.
- ▣ If debounce circuit is detect any closed switch it wait about 10 msec.
- ▣ It is continued , the status of SHIFT and CONTROL keys are transferred into RAM.

- ▣ **FIFO/SENSOR RAM:**

- ▣ This is a dual function of 8*8 RAM.



- In scanned key board mode and Strobed input mode , It is FIFO.
- Each new entry is written into successive RAM position and read in the order of entry.
- In sensor matrix mode it is a sensor RAM.
- Each sensor RAM is loaded with corresponding sensor RAM status.

- FIFO/SENSOR RAM status:
- This is used to tell the status of FIFO/SENSOR RAM.
- The status of logic also makes IRQ signal is High , When FIFO is empty.

Display section:

It consists of,

1. Display RAM.
2. Display Address registers.
3. Display registers.

DISPLAY RAM:

- It is a 16*8 RAM.
- Which stores 16 digits display codes.
- It can be accessed by CPU directly.
- In Decoded mode,8279 uses only first four location of Display RAM.
- In Encoded mode,8279 uses only first eight location of Display RAM.
- And all 16 location for 16 digits display.

DISPLAY ADDRESS REGISTERS:

- ▢ Used to hold address of the byte currently write or read by the CPU and scan count value.
- ▢ In auto increment mode, address in the register is automatically incremented for each write or read.

DISPLAY REGISTERS:

- ▢ It is a Two 4-bit registers such as , A and B.
- ▢ They hold the bit patterns of character to be displayed.
- ▢ The content of display registers A and B can B blanked and inhibited individually.

Operating modes

▢ It is two types, 1. Input modes.

2. Display modes.

INPUT MODES:

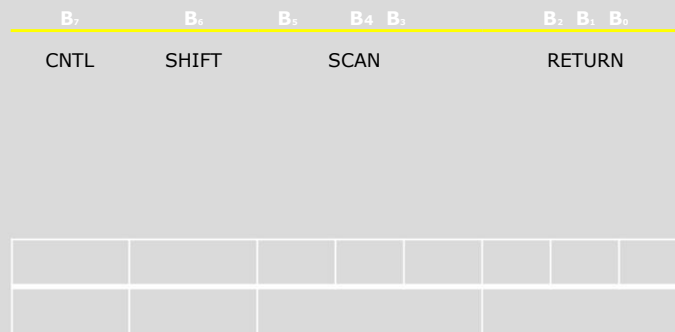
- It is basically 3 types, 1. Scanned keyboard.
- 2. Scanned sensor matrix.
- 3. Strobed mode.

SCANNED KEYBOARD:

Key board can be scanned in two ways. 1.Encoded Scan
2.Decoded Scan.

ENCODED SCAN:

- ▣ In this scan, scan lines (SL₂-SL₀) are decoded externally to provide 8 scan lines.
- ▣ Additionally it provides 8 return lines.
- ▣ So the size of matrix keyboard is 8*8 (i.e Scan * Return)=64.
- ▣ When the key is pressed , it is stored the status of return lines , Scan lines ,SHIFT and CNTL/STB keys into FIFO RAM.
- ▣ The Scanned keyboard structure is,

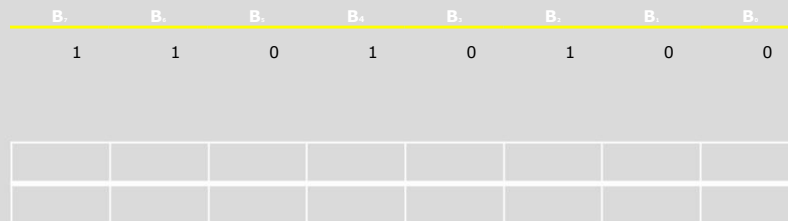


Example:

Find the key code for given condition below:
CNTL/STB SHIFT keys are open.

The pressed keys are to scan lines 2 and return lines 4.

SOLUTION:



- ▣ CNTL=1
- ▣ SHIFT=1
- ▣ Scan mode=010 (Scan line 2)
- ▣ Return mode=100 (Return line 4)
- ▣ Key code =D4 H

DECODED SCAN:

- ▣ In this mode ,internal decoder decodes the least significant bits of scan lines (SC3-SC0).
- ▣ That is provide the four combination such as 1110,1101,1011 and 0111.
- ▣ So the maximum size of keyboard is $8 \times 4 = 32$.
- ▣ The key code is similar to encoded code , only bit 5 (B₅) is always zero.

2-KEY LOCKOUT:

- ▢ In this mode, the two key depression is not allowed.
- ▢ When any key is depressed, the debounce logic is set and 8279 checks for any key depress next two scans.
- ▢ Three possible condition to avoid debouncing:
- ▢ Condition 1:
 - ▢ If other key depression is not found during next two scan, it is a single key is depressed .Then the status of key code is entered into FIFO RAM along with the status of CNTL and SHIFT lines

- ▢ If FIFO RAM is empty , The CPU is entry the data.
- ▢ If FIFO RAM is full , The CPU does not entry the data.
- Condition 2:
 - ▢ If any other key depress is encountered , no entry to the FIFO can occur.
 - ▢ When the key is released after that only Entry will be allowed.
- Condition3:
 - ▢ If the two key is pressed in simultaneously in a debounce cycle, both depression is not considered.

▮ N-KEY ROLLOVER:

- ▮ Each key depression is treated as independently from all others.

SCANNED SENSOR MATRIX:

- ▮ In this mode, image of the sensor matrix is kept in the sensor RAM.
- ▮ The status of sensor switches are input directly to the sensor RAM.
- ▮ 8279 scans row one by one and store the status of each row in the corresponding memory location.
- ▮ STROBED INPUT MODE:
- ▮ The data is entered from Returned lines.

Display modes:

- ▮ It is basically two types,
 1. Left entry (Type writer mode).
 2. Right entry (Calculator mode).

LEFT ENTRY:

- ▮ In this mode, 8279 display characters from left to right.
- ▮ Like a typewriter.

AUTOINCREMENT IN LEFT ENTRY:

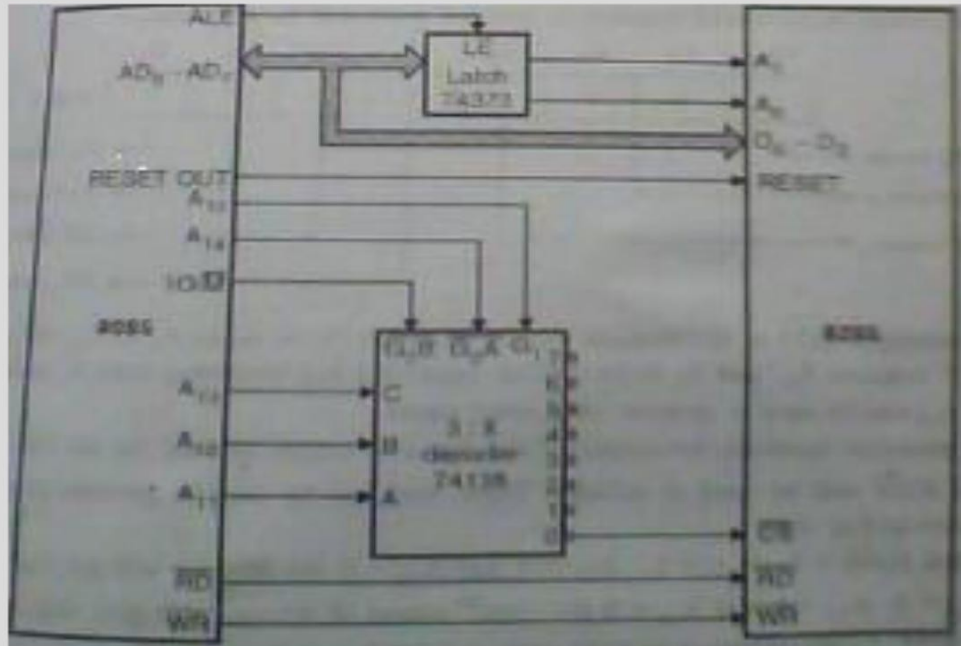
- ▮ In left entry mode, Autoincrement flag is set after each operation display RAM address is incremented.

RIGHT ENTRY:

- ▮ In this mode , 8279 display characters from Right to left.
- ▮ Like a Calculator.

AUTOINCREMENT IN RIGHT ENTRY:

- ▮ In right entry mode , Auto increment flag is set after each operation display RAM address is incremented.



Interfacing with 8279 with 8085

Interfacing with 8051

13.4 Interfacing Keyboard

For interfacing keyboard to the microprocessor/microcontroller based systems, usually push button keys are used. These push button keys when pressed, bounces a few times, closing and opening the contacts before providing a steady reading, as shown in the Fig.13.13. Reading taken during bouncing period may be faulty. Therefore, microprocessor/microcontroller must wait until the key reach to a steady state; this is known as **key debounce**.

The problem of key bounce can be eliminated using key debounce technique, either hardware or software.

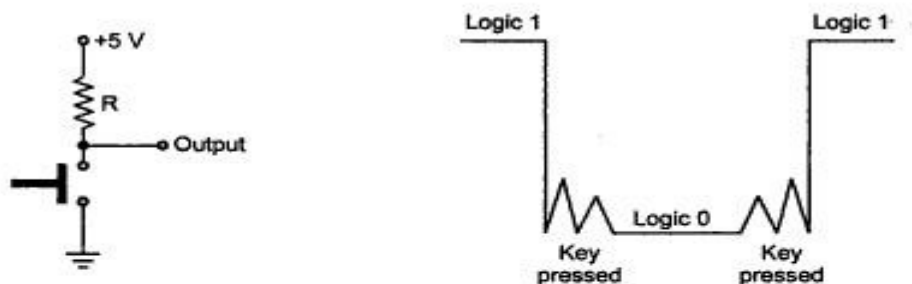


Fig. 13.13 Bouncing of key switch

13.4.1 Key Debounce using Hardware

Key position	a	b	Y	c	d	\bar{Y}
A	0	0	1	1	1	0
B	1	1	0	0	0	1
Between A and B	1	\bar{Y}	No change	Y	1	No change

Table 13.7

Fig. 13.14 shows the circuit diagram of key debounce. It consists of flip-flop. The output of flip-flop shown in Fig. 13.14 is logic 1 when key is at position A (unpressed) and it is logic 0 when key is at position B, as shown in Table 13.7. It is important to note that, when key is in between A and B, output does not change, preventing bouncing of key output. In other words we can say that output does not change during transition period, eliminating key debouncing.

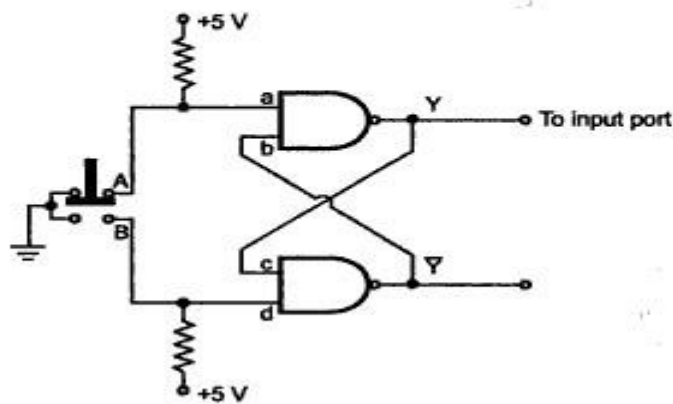


Fig. 13.14

13.4.3 Simple Keyboard Interface

Fig. 13.16 shows simple keyboard interface.

Here eight keys are individually connected to specific pins of port P1. Each port pin gives the status of key connected to that pin. When port pin is logic 1, key is open, otherwise key is closed.

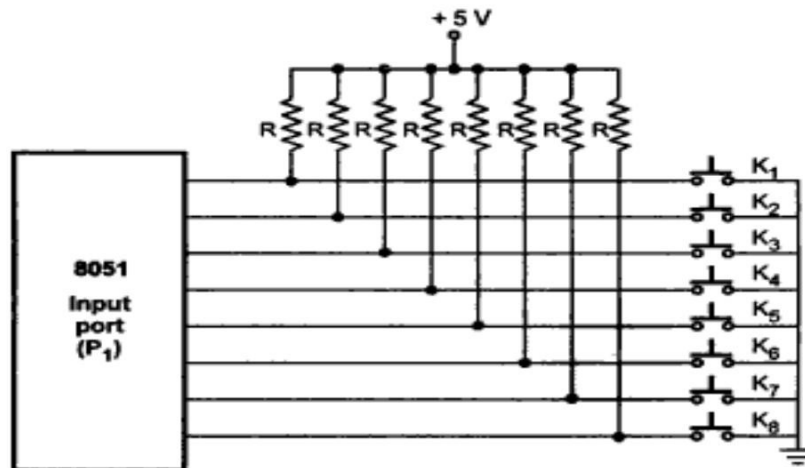


Fig. 13.16 Simple keyboard interface

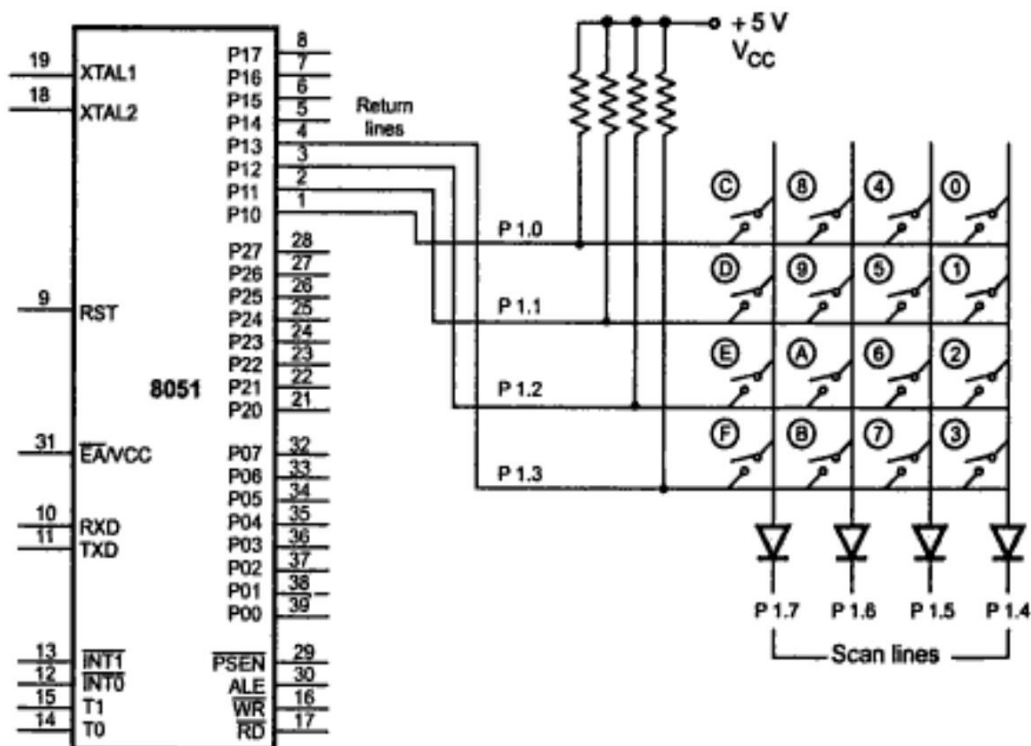


Fig. 13.19 4 × 4 matrix keyboard connected to port 1 of 8051

The Fig. 13.30 shows the interfacing of a 20 character \times 2 line LCD module with the 8051. As shown in the Fig. 13.30, the data lines are connected to the port 1 of 8051 and control lines RS, $\overline{R/W}$ and E are driven by 3.2, 3.3 and 3.1 lines of port 3, respectively. The voltage at V_{EE} pin is adjusted by a potentiometer to adjust the contrast of the LCD.

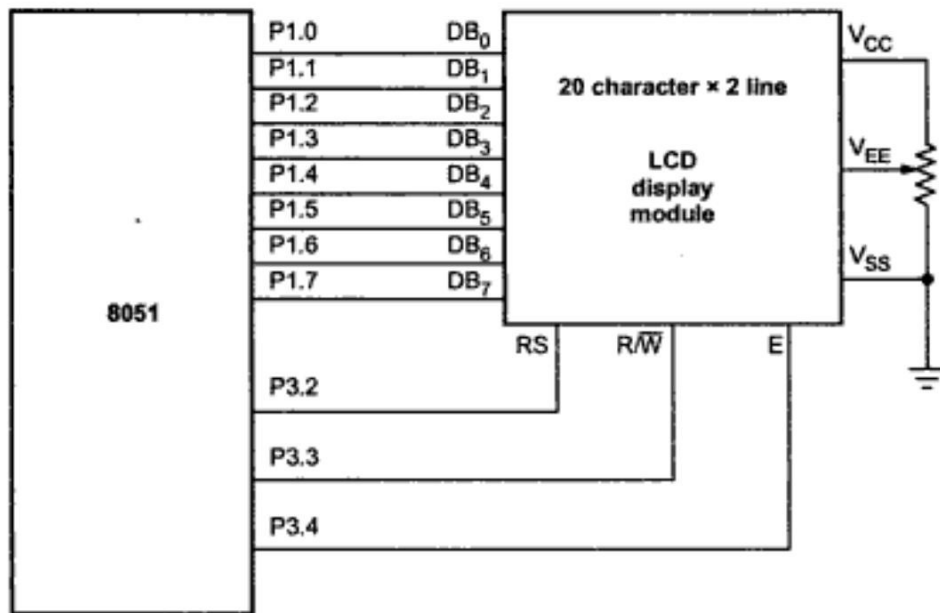
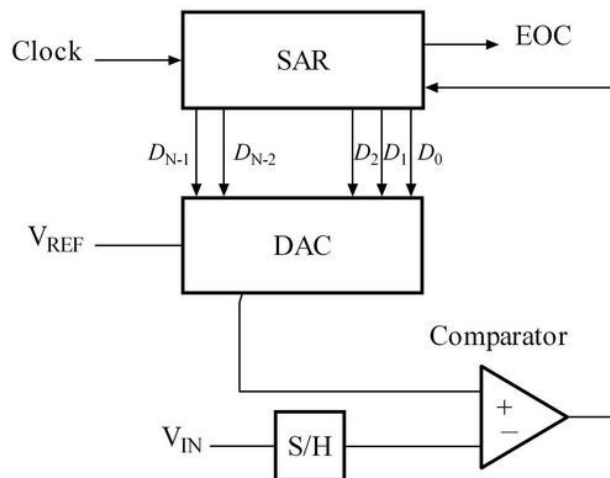


Fig. 13.30 Interfacing LCD module with 8051

ADC

- ADC0808/ADC0809 8-Bit μ P Compatible A/D Converters with 8-Channel Multiplexer
- The 8-bit A/D converter uses successive approximation as the conversion technique.
- The 8-channel multiplexer can directly access any of 8-single-ended analog signals.
- **Key Specifications:**
 - Resolution 8 Bits
 - Single Supply 5 VDC
 - Low Power 15 mW
 - Conversion Time 100 μ s

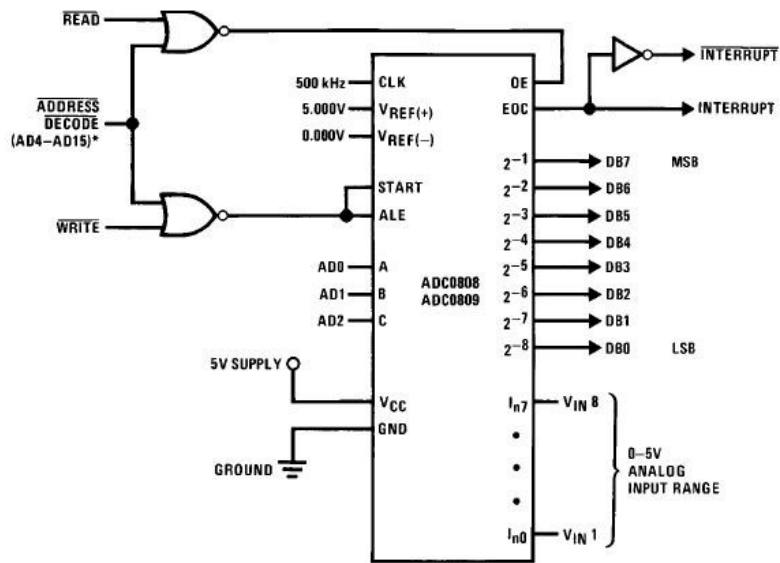
successive approximation



The successive approximation circuit typically consists of four block:

- . **A sample and hold circuit to acquire the input voltage (V_{in}).**
- . **An analog voltage comparator that compares V_{in} to the output of the internal DAC and outputs the result of the comparison to the successive approximation register (SAR).**
- . **A successive approximation register block designed to supply an approximate digital code of V_{in} to the internal DAC.**
- . **An internal reference DAC that supplies the comparator with an analog voltage equivalent of the digital code output of the SAR for comparison with V_{in} .**

The successive approximation register is initialized so that the (MSB) is equal to a digital 1. This code is fed into the DAC which then supplies the analog equivalent of this digital code ($V_{ref}/2$) into the comparator circuit for comparison with the sampled input voltage. If this analog voltage exceeds V_{in} the comparator causes the SAR to reset this bit; otherwise, the bit is left a 1. Then the next bit is set to 1 and do the same test, continuing this binary search until every bit in the SAR has been tested. The resulting code is the digital approximation of the sampled input voltage and is finally output by the DAC at the end of the conversion (EOC).



13.7 Interfacing DAC to 8051

In this section, we are going to see the interfacing of DAC 1408, 8-bit R/2R ladder type DAC with 8051. We begin with the details of IC DAC 1408.

13.7.1 IC DAC 1408

The 1408 is an 8-bit R/2R ladder type D/A converter compatible with TTL and CMOS logic. It is designed to use where the output current is linear product of an eight-bit digital word.

Fig. 13.31 shows the pin diagram and block diagram for IC 1408 DAC.

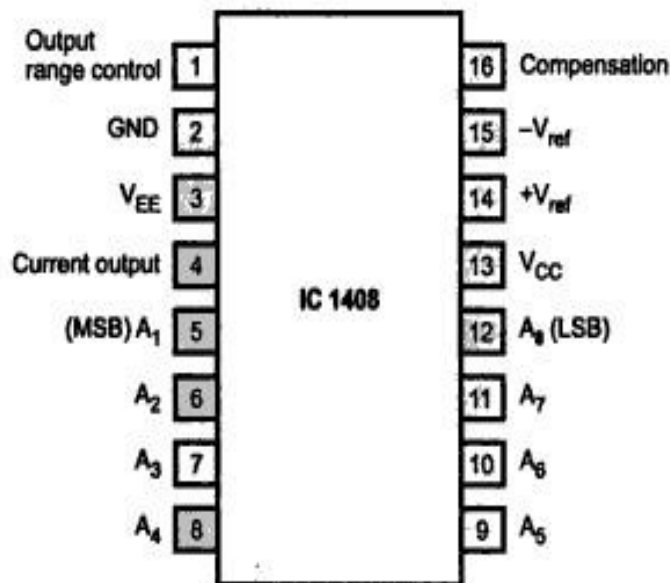


Fig. 13.31 (a) Pin diagram

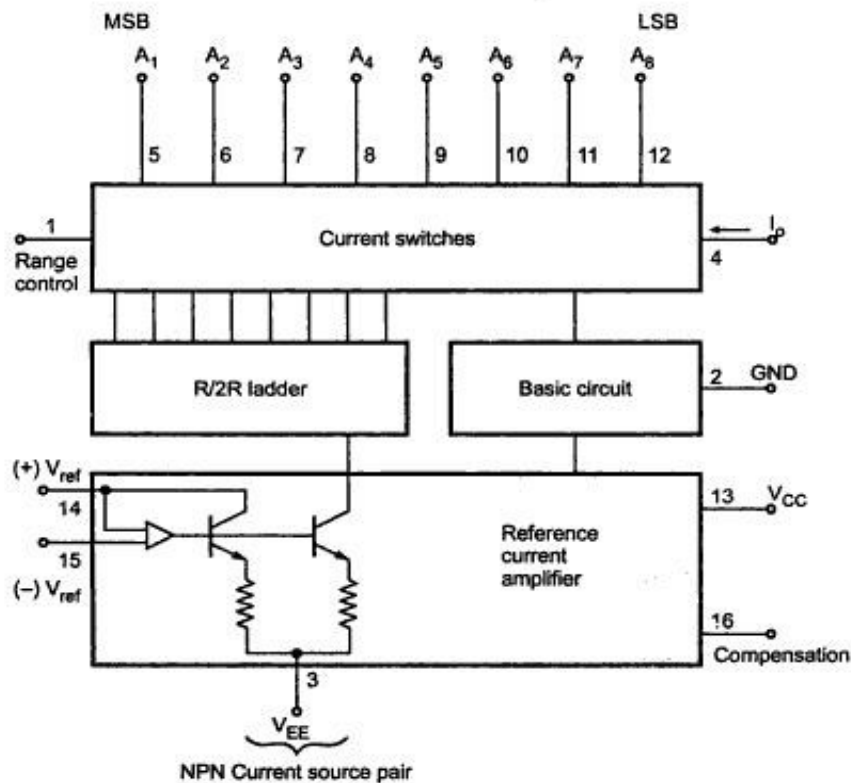


Fig. 13.31 (b) Block diagram

The IC 1408 consists of a reference current amplifier, an R/2R ladder and eight high speed current switches. It has eight input data lines A_1 (MSB) through A_8 (LSB) which control the positions of current switches.

It requires 2 mA reference current for full scale input and two power supplies $V_{CC} = +5\text{ V}$ and $V_{EE} = -15\text{ V}$ (V_{EE} can range from -5 V to -15 V).

The voltage V_{ref} and resistor R_{14} determines the total reference current source and R_{15} is generally equal to R_{14} to match the input impedance of the reference current amplifier.

Fig. 13.32 shows a typical circuit for IC 1408.

Copyrighted material

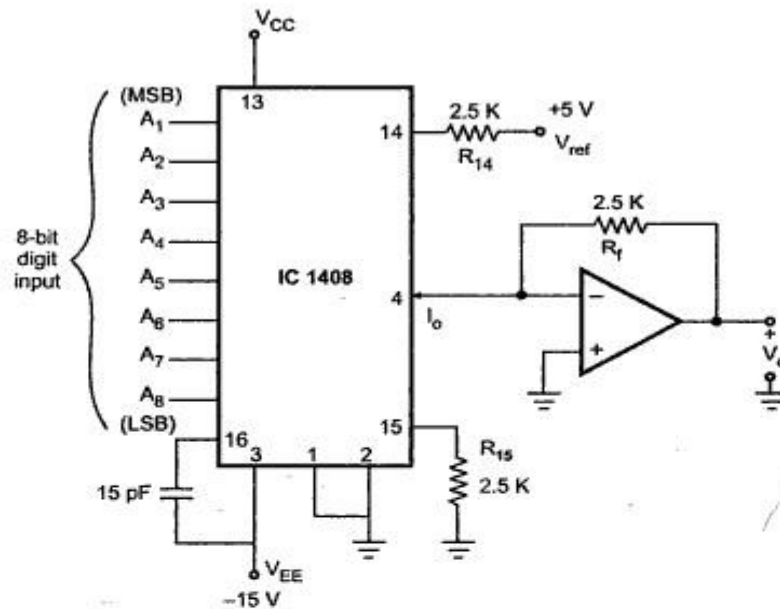


Fig. 13.32 Typical circuit for IC 1408

The output current I_o can be given as

$$I_o = \frac{V_{ref}}{R_{14}} \left(\frac{A_1}{2} + \frac{A_2}{4} + \frac{A_3}{8} + \frac{A_4}{16} + \frac{A_5}{32} + \frac{A_6}{64} + \frac{A_7}{128} + \frac{A_8}{256} \right)$$

Note : Input A_1 through A_8 can be either 0 or 1. Therefore, for typical circuit full scale current can be given as,

$$\begin{aligned} I_o &= \frac{5}{2.5 \text{ K}} \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64} + \frac{1}{128} + \frac{1}{256} \right) \\ &= \frac{2 \text{ mA} \times 255}{256} = 1.992 \text{ mA} \end{aligned}$$

It shows that the full scale output current is always 1 LSB less than the reference current source of 2 mA. This output current is converted into voltage by I to V converter. The output voltage for full scale input can be given as,

$$V_o = 1.992 \times 2.5 \text{ K} = 4.98 \text{ V}$$

Note : The arrow on the pin 4 shows the output current direction. It is inward. This means that IC 1408 sinks current. At $(0000\ 0000)_2$ binary input it sinks zero current and at $(1111\ 1111)_2$ binary input it sinks 1.992 mA.

The circuit shown in the Fig. 13.32 gives output in the unipolar range. When digital input is 00H, the output voltage is 0 V and when digital input is FFH $(1111\ 1111)_2$, the output voltage is + 5 V. This circuit can be modified to give bipolar output.

Copyrighted material

Microprocessor & Microcontroller System 13 - 37 Interfacing to External World

Fig. 13.33 shows the circuit for giving output in the bipolar range. Here, resistor R_B (5 K) is connected between V_{ref} and the output terminal of IC 1408. This gives a constant current source of 1 mA.

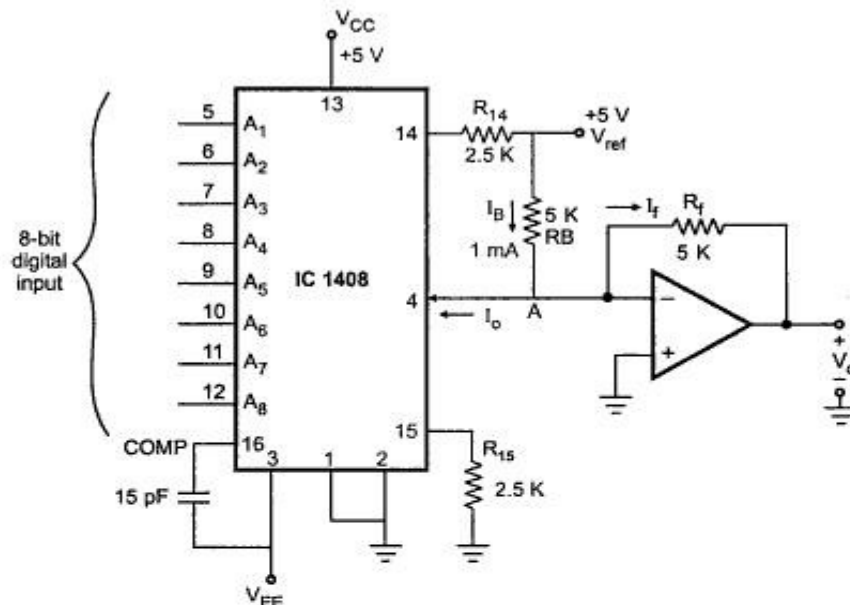


Fig. 13.33 Interfacing DAC in the bipolar range

The circuit operation can be observed for three conditions :

Condition 1 : For binary input (00H)

When binary input is 00H, the output current I_o at pin 4 is zero. Due to this current flowing through R_B (1 mA) flows through R_f giving $V_o = -5$ V.

Condition 2 : For binary input 80H

When binary input is 80H, the output current I_o at pin 4 is 1 mA. By applying KCL at node A we get,

$$-I_B + I_o + I_f = 0$$

Substituting values of I_B and I_o we get,

$$-(1 \text{ mA}) + (1 \text{ mA}) + I_f = 0$$

$$\therefore I_f = 0$$

and therefore the output voltage is zero.

Copyrighted material

Microprocessor & Microcontroller System 13 - 38 Interfacing to External World**Condition 3 : For binary input FFH**

When binary input is FFH, the output current I_o at pin 4 is 2 mA. By applying KCL at node A we get,

$$-I_B + I_o + I_f = 0$$

substituting values of I_B and I_o we get,

$$-(1 \text{ mA}) + (2 \text{ mA}) + I_f = 0$$

$$\therefore I_f = -1 \text{ mA}$$

Therefore, the output voltage is + 5 V. In this way, circuit shown in the Fig. 13.33 gives output in the bipolar range.

13.7.1.1 Important Electrical Characteristics for IC 1408

- Reference current : 2 mA
- Supply voltage : + 5 V_{CC} and - 15 V V_{EE}
- Setting time : 300 ns
- Full scale output current : 1.992 mA
- Accuracy : 0.19 %

13.7.2 Interfacing DAC 1408 / DAC 0808 with 8051

Fig. 13.34 shows the interfacing of DAC 0808 with 8051 microcontroller based system.

We now see how different waveforms can be generated using this circuit.

Square Wave

To generate square wave first we have to output FF and then 00 on port 1 of 8051. The port 1 is connected as an input to the DAC 0808. According to frequency requirement delay is provided between the two outputs.

Program :

```

MOV SP, #08H      ; Initialize Stack Pointer
REPEAT: MOV PI, #0FFH ; Load all 1's in Port 1
        LCALL DELAY  ; Call delay routine
        MOV P, #00H  ; Load all 0's in Port 1
        LCALL DELAY  ; Call delay routine
        LJMP REPEAT  ; Repeat
DELAY:   MOV R0, #0FFH ; Load delay count
BACK:   DJNZ R0, BACK  ; Decrement and check whether delay
                        ; count is zero if not repeat the
                        ; operation
                        RET ; Return to main program

```

Copyrighted material

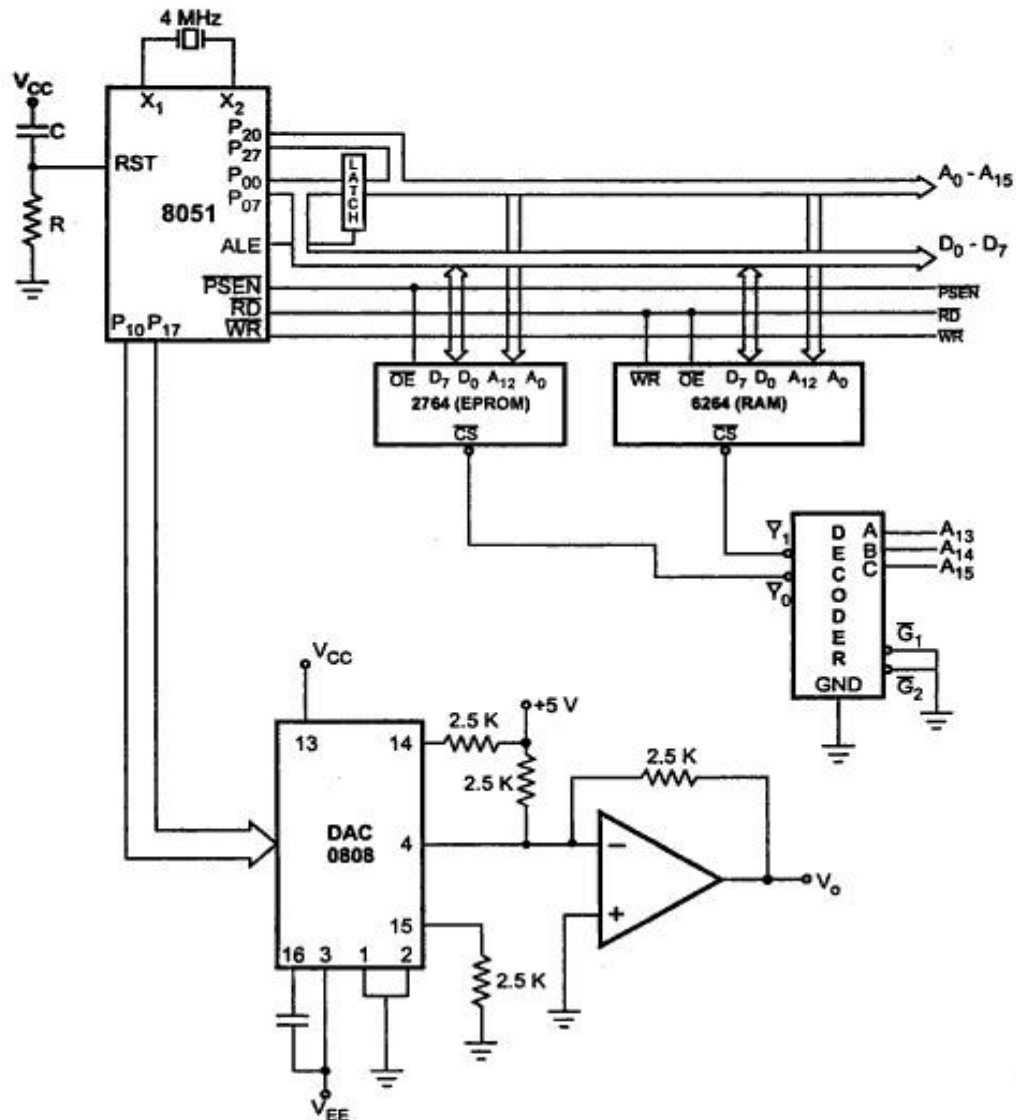


Fig. 13.34

Triangular Wave

To generate triangular wave we have to output data from 00 initially, and it should be incremented upto FF. When it reaches FF it should be decremented upto 00.

Copyrighted material

Microprocessor & Microcontroller System 13 - 40 Interfacing to External World

Program :

```

M
MOV SP, #08H          ; Initialize Stack Pointer
MOV R0, #00H
REPEAT: MOV P1, R0      ; Send digital data to the input
                        ; of DAC 0808
        INC R0          ; Increment digital data
        CJNE R0, #0FFH, REPEAT ; Check digital data for peak
                        ; output if not repeat
REPEAT1: MOV P1, R0     ; Send digital data to the input
                        ; of DAC 0808
        DJNZ R0, REPEAT1 ; Decrement digital data and
                        ; check digital data for least
                        ; output if not repeat.
        LJMPL REPEAT

```

Sine Wave generate :

To generate sine wave we have to output digital equivalent values which will represent sine wave as shown in the Fig. 13.35. Digital data 00H represents -2.5 V. 7FH represents 0 V and FFH represents +2.5 V. The digital equivalent for sine wave can be calculated as follows.

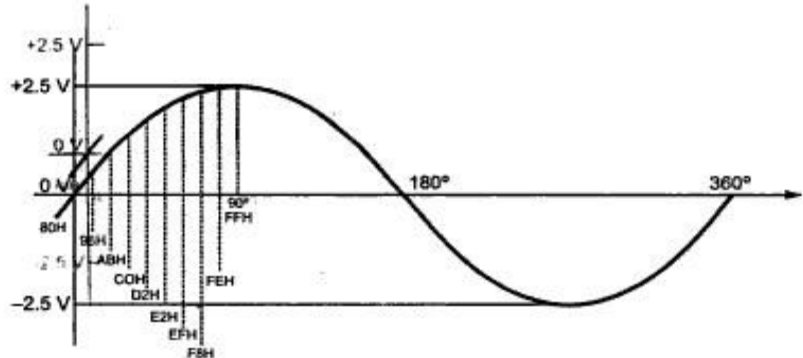


Fig. 13.35

We know that $\sin 0^\circ = 0$ and $\sin 90^\circ = 1$. The range $\sin 0^\circ$ to $\sin 90^\circ$ is distributed over digital range of 7FH to FFH i.e. $(FFH - 7FH)$ 128 decimal steps. Therefore, taking 128 as an offset we can write,

Digital equivalent value (DEV) for $\sin x = (128 + 128 \times \sin x)$

where x is an angle in degrees and digital value is in decimal.

Lookup table shows the digital equivalent values for sine wave.

Copyrighted material

Microprocessor & Microcontroller System 13 - 42 Interfacing to External World

```

AGAIN:  MOV DPTR, #LUT    ; Initialize pointer to lookup table
        MOV R0, #25H      ; Initialize counter
BEGIN:  CLR A
        MOVC A, @A+DPTR   ; Get data from lookup table
        MOV P1, A         ; Send data to port 1
        INC DPTR          ; point to next lookup table entry
        DJNZ R0, BEGIN    ; Decrement counter and go to BEGIN
                        ; if not zero
        SJMP AGAIN       ; Repeat the entire process
        ORG 300
LUT:    DB 80H, 96H, ABH, .....
        DB .....
        DB ..... 40H, 54H, 6AH, 80H

```

Note :

In above program we have to store entries from given lookup table instead of dots.

13.8 Interfacing ADC to 8051

In this section, we are going to see the interfacing of ADC 0803/0805 and 0808/0809 with 8051. We begin with the details of IC ADC 0803/0805 and ADC 0808/0809.

13.8.1 ADC 0804 Family

The ADC 0803, ADC 0804 and ADC 0805 are CMOS 8-bit successive-approximation analog to digital converters. These devices are design to operate from common microprocessor control buses, with tri-state output latches driving the data bus, and are identical except for accuracy.

Pin Diagram

Fig. 13.36 shows the pin diagram of ADC 0803, ADC 0804 and ADC 0805. IN+ and IN- inputs allow application of differential input voltage which has high common mode rejection and eliminates offset due to the zero input analog voltage value. The devices can

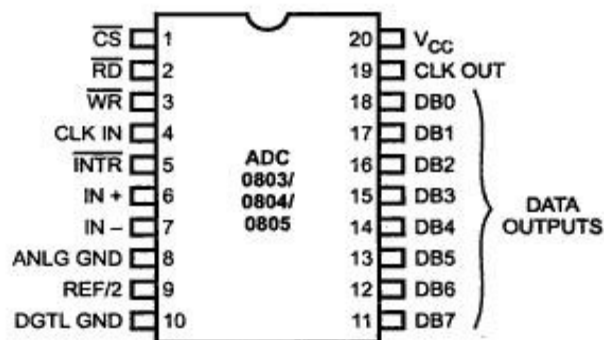


Fig. 13.36 Pin diagram of ADC 0803/0804/0805

Copyrighted material

Microprocessor & Microcontroller System 13 - 43 Interfacing to External World

operate with an external clock signal or, the on chip clock generator can be used independently by adding an external resistor and capacitor to set the time period.

Features

- 8-bit successive approximation ADC
- Conversion time 100 μ s
- Access time 135 ns.
- It has an on-chip clock generator.
- It does not require any zero adjustment.
- It operates on single 5 V power supply.
- Output meet TTL voltage level specifications.

Operation

When the \overline{WR} input goes low, the internal successive approximation register (SAR) is reset. As long as both \overline{CS} and \overline{WR} remain low, the analog to digital converter will remain in its reset state. One to eight clock periods after \overline{CS} or \overline{WR} makes a low-to-high transition, conversion starts. The \overline{INTR} signal is held high during conversion process. After conversion, \overline{INTR} goes low which is used as end of conversion signal. By making \overline{CS} and \overline{RD} signals low, an output can be read through DB_0 to DB_7 data signals.

Analog Inputs

As mentioned earlier, there are two analog inputs to ADC 0803/0804/0805 : $IN+$ and $IN-$. These inputs are connected to an internal operational amplifier and are differential inputs. The differential input rejects the common mode noise and eliminates offset at zero analog input voltage.

The Fig. 13.37 shows a few ways to use these differential inputs. The Fig 13.37 (a) shows the one way to use single input that can vary between 0 V and +5 V. Using another way variable voltage can be applied to the $V_{in(-)}$ pin so that the zero reference for $V_{in(+)}$ can be adjusted, as shown in Fig. 13.37 (b).

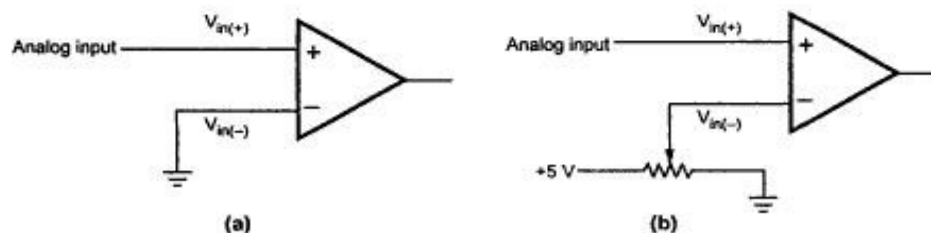
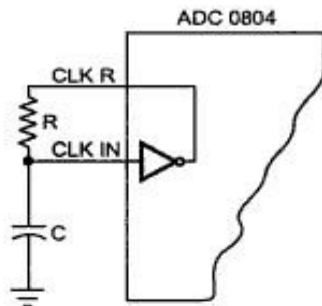


Fig. 13.37 Ways to use differential inputs

Copyrighted material

Microprocessor & Microcontroller System**13 - 44****Interfacing to External World****Clock Signal****Fig. 13.38 Clock circuit**

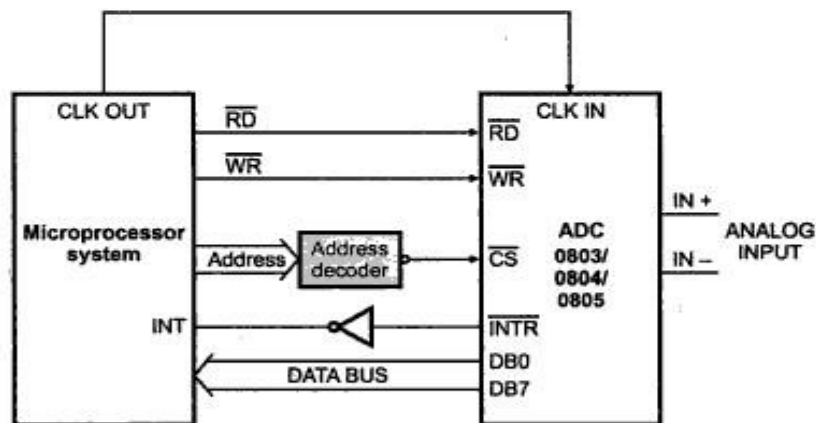
The ADC 0803/0804/0805 requires a clock source ranging 100 kHz to 1460 kHz for operation. The clock signal can be applied external or it can be generated with an RC circuit, as shown in the Fig. 13.38. When the RC circuit, shown in Fig. 13.38 is used to generate the clock, the clock frequency is given as,

$$F_{\text{clock}} = \frac{1}{1.1 RC}$$

Note : To have a minimum conversion time clock frequency should be kept nearer to 1460 kHz.

Typical Interface

Fig. 13.39 shows the interfacing of ADC 0803, ADC 0804 and ADC 0805 with microprocessor/microcontroller system.

**Fig. 13.39 Interfacing of ADC 0803/0804/0805 with microprocessor/microcontroller system****13.8.2 ADC 0808/0809 Family**

The ADC 0808 and ADC 0809 are monolithic CMOS devices with an 8-channel multiplexer. These devices are also designed to operate from common microprocessor/microcontroller control buses, with tri-state output latches driving the data bus. The main features of these devices are :

Copyrighted material

Microprocessor & Microcontroller System**13 - 45****Interfacing to External World****Features**

- 8-bit successive approximation ADC.
- 8-channel multiplexer with address logic.
- Conversion time 100 μ s.
- It eliminates the need for external zero and full-scale adjustments.
- Easy to interface to all microprocessors.
- It operates on single 5 V power supply.
- Output meet TTL voltage level specifications.

Pin Diagram

Fig. 13.40 shows pin diagram of 0808/0809 ADC.

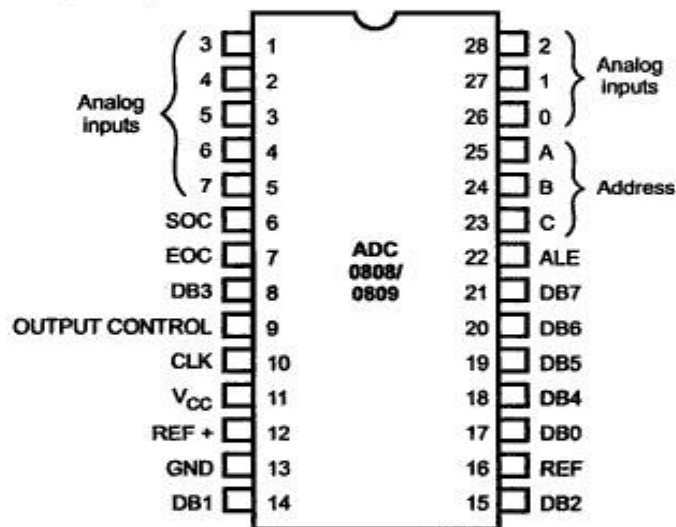


Fig. 13.40 Pin diagram of ADC 0808/0809

Operation

ADC 0808/0809 has eight input channels, so to select desired input channel, it is necessary to send 3-bit address on A, B and C inputs. The address of the desired channel is sent to the multiplexer address inputs through port pins. After atleast 50 ns, this address must be latched. This can be achieved by sending ALE signal. After another 2.5 μ s, the start of conversion (SOC) signal must be sent high and then low to start the conversion process. To indicate end of conversion ADC 0808/0809 activates EOC signal. The microprocessor system can read converted digital word through data bus by enabling the output enable signal after EOC is activated. This is illustrated in Fig. 13.41.

Copyrighted material

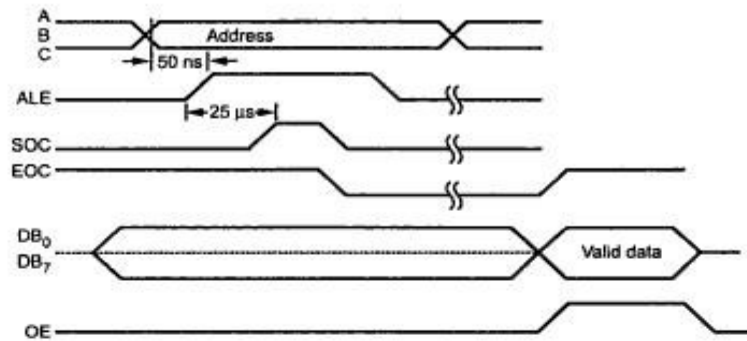
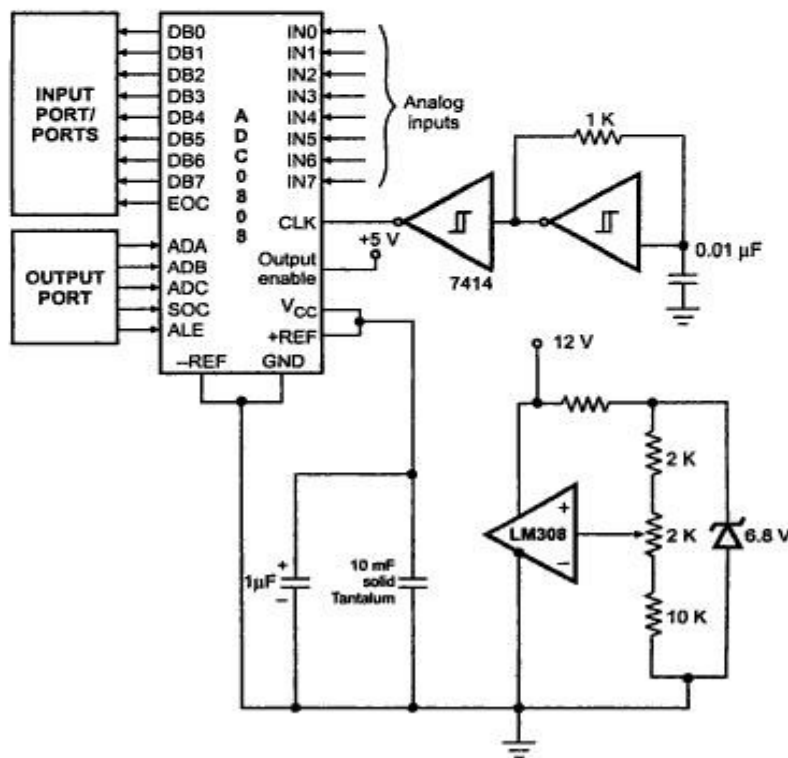
Microprocessor & Microcontroller System 13 - 46 Interfacing to External World**Fig. 13.41 Timing waveforms for ADC 0808****Interfacing**

Fig. 13.42 shows typical interfacing circuit for ADC 0808 with microprocessor/microcontroller system.

**Fig. 13.42 Typical interface for 0808/0809**

Copyrighted material

Microprocessor & Microcontroller System 13 - 49 Interfacing to External World

```

CLR TF0          ; Clear timer 0 flag
DJNZ R0, BACK    ; Decrement counter and if
                  ; not zero repeat
RET              ; Return

```

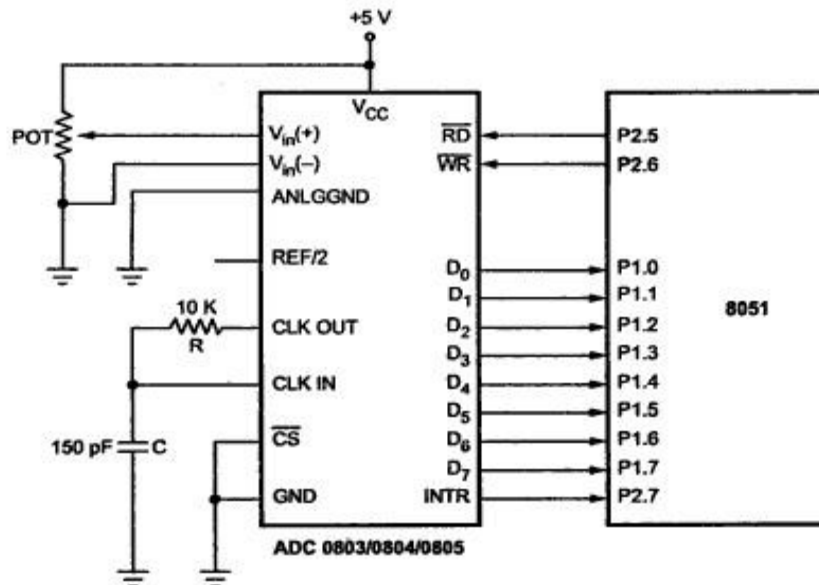
Note : Timer 0 gives a delay of 50 ms. To get a delay of 1 sec, such delay is executed 20 times.

13.8.3 Interfacing of ADC 0803/0804/0805 with 8051

The Fig. 13.44 shows the interfacing of ADC 0803/0804/0805 with 8051 using port 1 and port 2. Here, port 1 is used to read digital data from ADC and port 2 is used to provide control signals to ADC 0803/0804/0805. Potentiometer is used to adjust V_{in+} voltage. The clock signal is provided using internal clock generator and two external components, resistor and capacitor as shown in the Fig. 13.44. The frequency of such clock can be determined by,

$$f = \frac{1}{1.1 RC}$$

The typical values are $R = 10 \text{ k}\Omega$ and $C = 150 \text{ pF}$. With these values we get approximately 606 kHz clock frequency. In such case, the conversion time is around 110 μs .

**Fig. 13.44****AD Conversion Program**

```

MOV P1, #0FFH    ; configure Port1 as input
BACK: CLR P2.6    ; [Make WR = 0 and

```

Copyrighted material