16BECS403A **Microprocessors and Microcontrollers** **L T P C**
3 0 0 3

Course Objectives:

- To introduce the architecture and programming of 8085 microprocessor.
- To introduce the interfacing of peripheral devices with 8085 microprocessor.
- To introduce the architecture and programming of 8086 microprocessor.
- To introduce the architecture, programming and interfacing of 8051 micro controller.

Learning outcomes:

This course is designed to introduce the basic concepts of Microprocessor and Microcontroller and provide an understanding of the basic concepts of various Interfacing and Applications. The course also helps the students to develop the ability to designing a programming for various applications.

UNIT I Microprocessor- 8086 **9**

Register Organization -Architecture-Signals-Memory Organization-Bus Operation-I/O Addressing Minimum Mode-Maximum Mode-Timing Diagram-Interrupts - Service Routines – I/O and Memory Interfacing concepts.

UNIT II Programming of 8086 **9**

Addressing Modes-Instruction format-Instruction set-Assembly language programs in 8086. RISC architecture – introduction to ARM Programming-register configuration and instruction set – sample program.

UNIT III Interfacing Devices **9**

Programmable Peripheral Interface (8255) - Programmable Interval Timer (8254) - Programmable Interrupt Controller (8259A) - Programmable DMA Controller (8257) - Programmable Communication Interface (8251A) –Programmable Keyboard and Display Controller (8279).

UNIT IV Microcontroller-8051 **9**

Register Set-Architecture of 8051 microcontroller- I/O and memory addressing-Interrupts-Instruction set- Addressing modes. .

UNIT V Programming And Interfacing Of 8051 **9**

Timer-Serial Communication-Interrupts Programming-Interfacing to External Memory-Interfacing to ADC, LCD, DAC, Keyboard and stepper motor.

Total Hours: 45

Text Books:

1. K. Ray and K. M. Bhurchandi, Advanced Microprocessors and Peripherals. Tata McGraw Hill, New Delhi,3rd edition,2013.
2. Douglas V.Hall, Microprocessor and Interfacing Programming and Hardware. Tata McGraw Hill, New Delhi 2007

References:

1. Muhammad Ali Mazidi, Janice Gillispie Mazidi, Rolin D.MCKinlay The 8051 Microcontroller and Embedded Systems Pearson Education, New Jersy,2nd edition,2009
2. Krishna Kant, Microprocessor and Microcontroller Architecture, programming and system design using 8085, 8086, 8051 and 8096, PHI, New Delhi, 2008
3. Kenneth J.Ayala, The 8051 Microcontroller, Thompson Delmar Learning, New Delhi, 2007
4. Barry B.Brey, The Intel Microprocessors Architecture, Programming and Interfacing,
5. Pearson Education, New Delhi, 2007

Websites:

1. http://www.8052.com/tut8051http://www.eastaughs.fsnet.co.uk/cpu/index.htm
2. http://www.webphysics.davidson.edu/faculty/dmb/py310/**8085**.pdf
3. http://www.aust.edu/cse/moinul/8086_lectures.pdf
4. http://www.cache.com.hk/datasheetC8255ovview.html

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Established Under Section 3 of UGC Act 1956)
Pollachi Main Road, Eachanari Post, Coimbatore – 641 021. INDIA
Phone : 0422-6471113-5, 6453777 Fax No : 0422 -2980022-3
Email : info@karpagam.com Web : www.kahedu.edu.in

NAME OF THE STAFF : REGI SARAL A
DESIGNATION : ASSISTANT PROFESSOR
SUBJECT : MICROPROCESSORS AND MICROCONTROLLERS
SUBJECT CODE : 16BECS403A

S.NO	DESCRIPTION OF PORTIONS TO BE COVERED	TEACHING AIDS	HOURS
	UNIT I MICROPROCESSOR 8086		
1	Register Organization Signals	R(1)-page No.130-132	1
2	8086 Architecture	R(1)-page No.128-135	2
3	Memory Organization	R(1)-page No.142-146	1
4	Bus Operation-I/O Addressing Minimum Mode-Maximum Mode	R(1)-page No.146-149	1
5	Timing Diagram-Interrupts	R(1)-page No.169-177	1
6	Service Routines – I/O and Memory Interfacing concepts.	R(1)-page No.164-168	2
	UNIT II PROGRAMMING OF 8086		
7	Addressing Modes of 8086	R(1)-page No.188-197	2
8	Instruction format-Instruction set	R(1)-page No. 203-240	2
9	Assembly language programs in 8086. RISC Architecture	T(1)-page No.3.1-4.35 http://youtu.be/dw6fUVJZKL8	2
10	Introduction to ARM Programming	http://youtu.be/XojE13qeiTE	1
11	Register configuration and instruction set	http://youtu.be/7LqPJGnBPMM	1
12	Sample program	T(1)-page No.3.1-4.35	1
	UNIT III INTERFACING DEVICES		
13	Programmable Peripheral Interface (8255)	R(1)-page No.249-253	2
14	Programmable Interval Timer (8254)	R(1)-page No.313-319	1
15	ProgrammableInterrupt Controller (8259A)	T(1)-page No.8.30-8.38	2

16	Programmable DMA Controller (8257)	T(1)-page No.11.5-11.10	2
17	Programmable CommunicationInterface (8251A)	http://youtu.be/ODqkBoRvYUU	1
18	Programmable Keyboard and Display Controller (8279).	R(1)-page No.281-291	2
	UNIT IV MICROCONTROLLER-8051		
19	Register Set of 8051	R(1)-page No.4.24-4.30	1
20	Architecture of 8051 microcontroller	R(1)-page No.419-423	2
21	I/O and memory addressing	R(2)-page No.73-77	1
22	Interrupts- Instruction set	R(1)-page No.465-470 R(1)-page No.494-516	2
23	Addressing modes	R(1)-page No.488-493	1
	UNIT V PROGRAMMING AND INTERFACING OF 8051		
24	Timer-Serial Communication of 8051	R(2)-page No.80-91	2
25	Interrupts Programming	R(2)-page No.92-96	1
26	Interfacing to External Memory of 8051 micro controller	R(2)-page No.78-80	1
27	Interfacing toADC, LCD, DAC, Keyboard and stepper motor.	R(2)-page No.261-294 R(2)-page No.299-303	3

Text Books:

1. Douglas V.Hall, Microprocessor and Interfacing: Programming and Hardware. Tata McGraw Hill, New Delhi 2007

References:

1. Krishna Kant, Microprocessor and Microcontroller Architecture, programming and system design using 8085, 8086, 8051 and 8096, PHI, New Delhi, 2008
2. Kenneth J.Ayala, The 8051 Microcontroller, Thompson Delmar Learning, New Delhi, 2007

UNIT1

MICROPROCESSOR- 8086

Microprocessor- 8086

Register Organization -Architecture-Signals-Memory Organization-Bus Operation-I/O Addressing Minimum Mode-Maximum Mode-Timing Diagram-Interrupts - Service Routines – I/O and Memory Interfacing concepts.

1.1 Register organization of 8086

8086 has a powerful set of registers containing general purpose and special purpose registers. All the registers of 8086 are 16-bit registers. The general purpose registers, can be used either 8-bit registers or 16-bit registers. The general purpose registers are either used for holding the data, variables and intermediate results temporarily or for other purpose like counter or for storing offset address for some particular addressing modes etc. The special purpose registers are used as segment registers, pointers, index registers or as offset storage registers for particular addressing modes. We will categorize the register set into four groups as follows:

General data registers:

The registers AX, BX, CX, and DX are the general 16-bit registers.

AX Register:

Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16- bit register AX. AL in this case contains the low-order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations, rotate and string manipulation.

BX Register:

This register is mainly used as a base register. It holds the starting base location of a memory region within a data segment. It is used as offset storage for forming physical address in case of certain addressing mode.

CX Register:

It is used as default counter or count register in case of string and loop instructions.

DX Register:

Data register can be used as a port number in I/O operations and implicit operand or destination in case of few instructions. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

Segment registers:

To complete 1Mbyte memory is divided into 16 logical segments. The complete 1Mbyte memory segmentation. Each segment contains 64Kbyte of memory. There are four segment registers.

Code segment (CS)

It has 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions. It is used for addressing a memory location in the code segment of the memory, where the executable program is stored.

Stack segment (SS)

is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction. It is used for addressing stack segment of memory. The stack segment is that segment of memory, which is used to store stack data.

Data segment (DS)

is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions. It points to the data segment memory where the data is resided.

Extra segment (ES)

is a 16-bit register containing address of 64KB segment, usually with program data. By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. ES register can be changed directly using POP and LES instructions. It also refers to segment which essentially is another data segment of the memory. It also contains data.

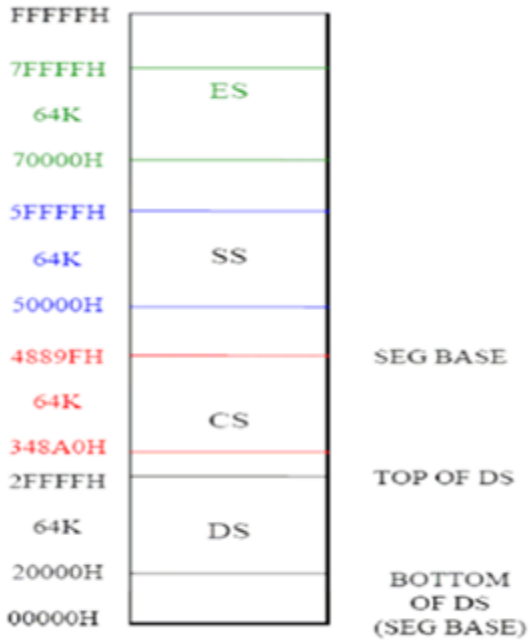


Fig.1 Memory Segmentation

1.2 Architecture of 8086Microprocessor

Features:

1. Intel 8086 was launched in 1978.
2. It was the first 16-bitmicroprocessor.
3. This microprocessor had major improvement over the execution speed of8085.
4. It is available as 40-pin Dual-Inline-Package(DIP).
5. It is available in three versions: a.8086 (5MHz)
   b. 8086-2 (8 MHz)
   c. 8086-1 (10 MHz)
6. It consists of 29,000transistors.

Bus Interface Unit (BIU):

The function of BIU is to

- Fetch the instruction or data from memory.
- Write the data to memory.
- Write the data to the port.
- Read data from the port.

Instruction Queue

1. To increase the execution speed, BIU fetches as many as six instruction bytes a head to time from memory.
2. All six bytes are then held in first in first out 6 byte register called instruction queue.
3. Then all bytes have to be given to EU one by one.
4. This pre fetching operation of BIU may be in parallel with execution operation of EU, which improves the speed execution of the instruction.

Execution Unit (EU)

The functions of execution unit are

- To tell BIU where to fetch the instructions or data from.
- To decode the instructions.
- To execute the instructions.

The EU contains the control circuitry to perform various internal operations. A decoder in EU decodes the instruction fetched memory to generate different internal or external control signals required to perform the operation. EU has 16-bit ALU, which can perform arithmetic and logical operations on 8-bit as well as 16-bit.

Architecture of 8086 Microprocessor

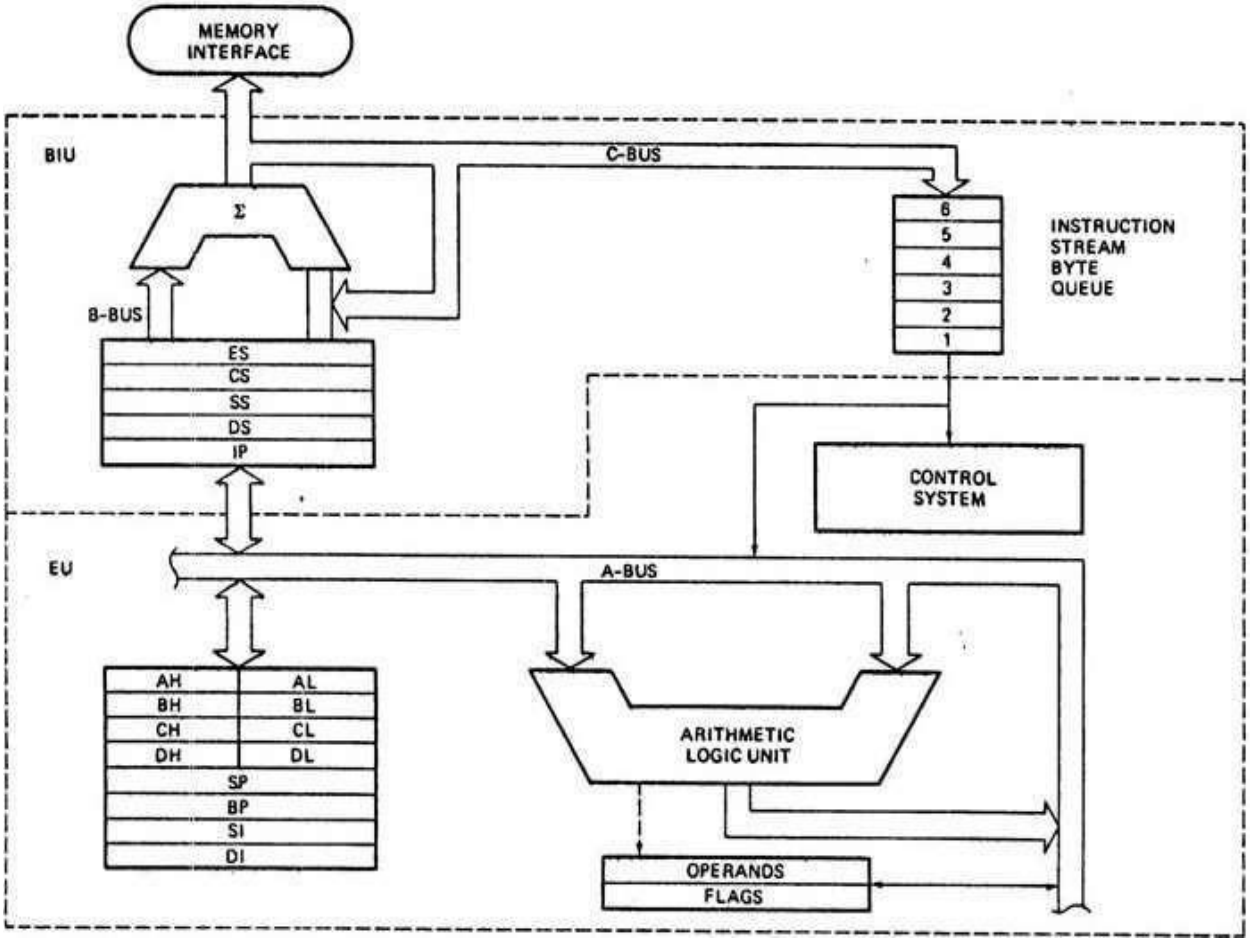


Fig.2 Architecture of 8086 Microprocessor

General Purpose Registers of8086

These registers can be used as 8-bit registers individually or can be used as 16-bit in pair to have AX,BX, CX, and DX.

AX Register: AX register is also known as accumulator register that stores operands for arithmetic operation like divided, rotate.

BX Register: This register is mainly used as a base register. It holds the starting base location of a memory region within a data segment.

CX Register: It is defined as a counter. It is primarily used in loop instruction to store loop counter.

DX Register: DX register is used to contain I/O port address for I/O instruction.

Segment Registers

Additional registers called segment registers generate memory address when combined with other in the microprocessor. In 8086 microprocessor, memory is divided into 4 segments as follow:

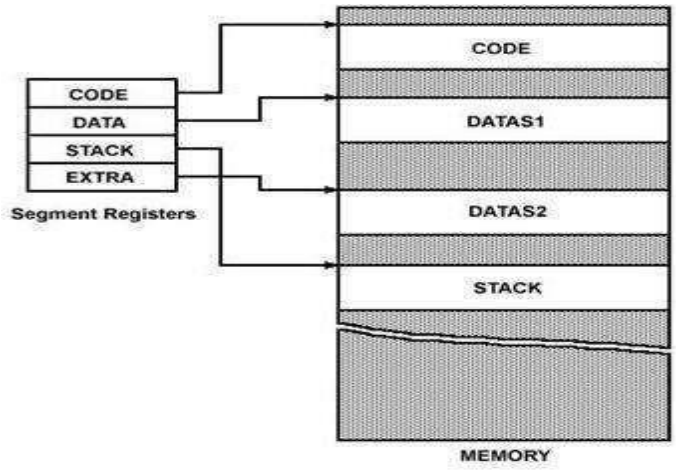


Fig.3 Segment Register of 8086 Microprocessor

Code Segment (CS): The CS register is used for addressing a memory location in the Code Segment of the memory, where the executable program is stored.

Data Segment (DS): The DS contains most data used by program. Data are accessed in the Data Segment by an offset address or the content of other register that holds the offset address.

Stack Segment (SS): SS defined the area of memory used for the stack.

Extra Segment (ES): ES is additional data segment that is used by some of the string to hold the destination data.

Flag Registers of8086

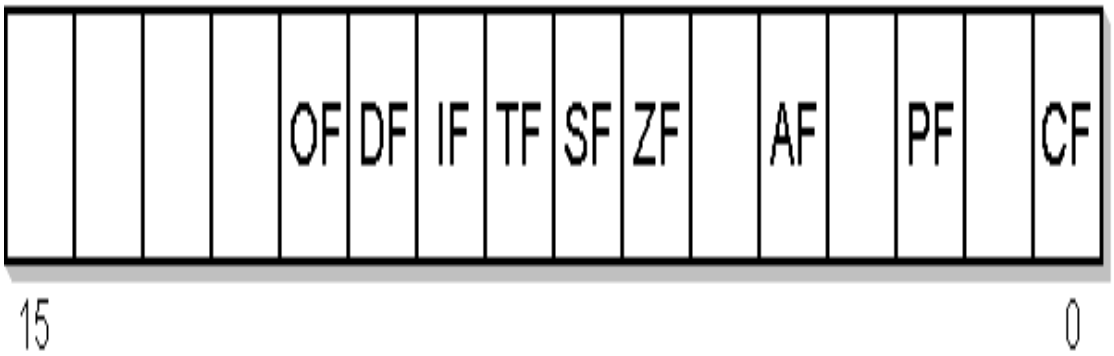


Fig.4 Flag register of 8086 Microprocessor

Flags Register determines the current state of the processor. They are modified automatically by CPU after mathematical operations, this allows to determine the type of the result, and to determine conditions to transfer control to other parts of the program. 8086 has 9 flags and they are divided into two categories:

1. Conditional Flags
2. Control Flags

(1) Conditional Flags

Conditional flags represent result of last arithmetic or logical instruction executed. Conditional flags are as follows:

Carry Flag (CF)

This flag indicates an overflow condition for unsigned integer arithmetic. It is also used in multiple-precision arithmetic.

Auxiliary Flag (AF):

If an operation performed in ALU generates a carry/barrow from lower nibble (i.e. D_0 D_3) to upper nibble (i.e. D_4 – D_7), the AF flag is set i.e. carry given by D_3 bit to D_4 is AF flag. This is not a general-purpose flag, it is used internally by the processor to perform Binary to BCD conversion.

Parity Flag (PF):

This flag is used to indicate the parity of result. If lower order 8-bits of the result contains even number of 1"s, the Parity Flag is set and for odd number of 1"s, the Parity Flag is reset.

Zero Flag (ZF):

It is set; if the result of arithmetic or logical operation is zero else it is reset.

Sign Flag (SF):

In sign magnitude format the sign of number is indicated by MSB bit. If the result of operation is negative, sign flag is set.

Overflow Flag (OF):

It occurs when signed numbers are added or subtracted. An OF indicates that the result has exceeded the capacity of machine.

2.Control Flags

Control flags are set or reset deliberately to control the operations of the execution unit.

Control flags are as follows:

1.Trap Flag(TP):

a. It is used for single step control.

b. It allows user to execute one instruction of a program at a time for debugging.

c. When trap flag is set, program can be run in single step mode.

2.Interrupt Flag(IF):

a. It is an interrupt enable/disable flag.

b. If it is set, the maskable interrupt of 8086 is enabled and if it is reset, the interrupt is disabled.

c. It can be set by executing instruction sit and can be cleared by executing CLI instruction.

3.Direction Flag (DF):

a. It is used in string operation.

b. If it is set, string bytes are accessed from higher memory address to lower memory address.

When it is reset, the string bytes are accessed from lower memory address to higher memory address.

1.3 Signal Description of 8086 Microprocessor

The 8086 Microprocessor is a 16-bit CPU available in 3 clock rates, i.e. 5, 8 and 10MHz, packaged in a 40 pin CERDIP or plastic package. The 8086 Microprocessor operates in single processor or multiprocessor configurations to achieve high performance. The pin configuration is as shown in fig1. Some of the pins serve a particular function in minimum mode (single processor mode) and others function in maximum mode (multiprocessor mode) configuration.

The 8086 signals can be categorized in three groups. The first are the signals having common functions in minimum as well as maximum mode, the second are the signals which have special functions in minimum mode and third are the signals having special functions for maximum mode The following signal description are common for both the minimum and maximum modes.

AD15-AD0:

These are the time multiplexed memory I/O address and data lines. Address remains on the lines during T1 state, while the data is available on the data bus during T2, T3, TW and T4. Here T1, T2, T3, T4 and TW are the clock states of a machine cycle. TW is await state. These lines are active high and float to a tristate during interrupt acknowledge and local bus hold acknowledge cycles.

A19/S6, A18/S5, A17/S4, A16/S3:

These are the time multiplexed address and status lines. During T1, these are the most significant address lines or memory operations. During I/O operations, these lines are low. During memory or I/O operations, status information is available on those lines for T2, T3, TW and T4 .The status of the interrupt enable flag bit(displayed on S5) is updated at the beginning of each clock cycle. The S4 and S3 combined indicate which segment register is presently being used for memory accesses as shown in Table 1

These lines float to tri-state off (tristate) during the local bus hold acknowledge. The status line S6 is always low(logical). The address bits are separated from the status bits using latches controlled by the ALE signal.

Table 1

S4	S3	Indication
0	0	Alternate Data
0	1	Stack
1	0	Code or none
1	1	Data

$\overline{BHE}$ /S_7 – Bus High Enable /Status:

The bus high enable signal is used to indicate the transfer of data over the higher order (D15-D8) data bus as shown in Table 2. It goes low for the data transfers over D15-D8 and is used to derive chip selects of odd address memory bank or peripherals. $\overline{BHE}$ is low during T1 for read, write and interrupt acknowledge cycles, when- ever a byte is to be transferred on the higher byte of the data bus. The status information is available during T2, T3 and T4. The signal is active low and is tristated during 'hold'. It is low during T1 for the first pulse of the interrupt acknowledge cycle.

Table 2

$\overline{BHE}$	A_0	Indication
0	0	Whole Word
0	1	Upper byte from or to odd address
1	0	Upper byte from or to even address
1	1	None

$\overline{RD}$ - Read: Read signal, when low, indicates the peripherals that the processor is performing a memory or I/O read operation. $\overline{RD}$ is active low and shows the state for T2, T3, TW of any read cycle. The signal remains tristated during the 'hold acknowledge'.

READY:

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

INTR-interrupt Request:

This is a level triggered input. This is sampled during the last clock cycle of each instruction to determine the availability of the request. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle. This can be internally masked by resetting the interrupt enable flag. This signal is active high and internally synchronized.

TEST:

This input is examined by a 'WAIT' instruction. If the TEST input goes low, execution will continue, else, the processor remains in an idle state. The input is synchronized internally during each clock cycle on leading edge of clock.

NMI-Non-maskable Interrupt:

This is an edge-triggered input which causes a Type2 interrupt. The NMI is not maskable internally by software. A transition from low to high initiates the interrupt response at the end of the current instruction. This input is internally synchronized.

RESET:

This input causes the processor to terminate the current activity and start execution from FFFF0H. The signal is active high and must be active for at least four clock cycles. It restarts execution when the RESET returns low. RESET is also internally synchronized.

CLK-Clock Input:

The clock input provides the basic timing for processor operation and bus control activity. Its an asymmetric square wave with 33% duty cycle. The range of frequency for different 8086 versions is from 5MHz to 10MHz.

VCC :

+5V power supply for the operation of the internal circuit. GND ground for the internal circuit.

MN/MX :

The logic level at this pin decides whether the processor is to operate in either minimum (single processor) or maximum (multiprocessor) mode. The following pin functions are for the minimum mode operation of 8086.

M/IO -Memory/IO:

This is a status line logically equivalent to S2 in maximum mode. When it is low, it indicates the CPU is having an I/O operation, and when it is high, it indicates that the CPU is having a memory operation. This line becomes active in the previous T4 and remains active till final T4 of the current cycle. It is tristated during local bus "hold acknowledge".

$\overline{INTA}$ **- Interrupt Acknowledge:**

This signal is used as a read strobe for interrupt acknowledge cycles. In other words, when it goes low, it means that the processor has accepted the interrupt. It is active low during T2, T3 and TW of each interrupt acknowledge cycle.

ALE-Address latch Enable:

This output signal indicates the availability of the valid address on the address/data lines, and is connected to latch enable input of latches. This signal is active high and is never tristated.

DT/$\overline{R}$-Data Transmit/Receive:

This output is used to decide the direction of data flow through the transreceivers (bidirectional buffers). When the processor sends out data, this signal is high and when the processor is receiving data, this signal is low. Logically, this is equivalent to S1 in maximum mode. Its timing is the same as M/I/O. This is tristated during 'hold acknowledge'.

$\overline{DEN}$ - Data Enable

This signal indicates the availability of valid data over the address/data lines. It is used to enable the transreceivers (bidirectional buffers) to separate the data from the multiplexed address/data signal. It is active from the middle ofT2 until the middle of T4 DEN is tristated during 'hold acknowledge' cycle.

1.4 8086 Memory Organization

Segmented Memory

Two types of memory organization are used:

- Linear addressing where the entire memory is available to the processor at all the times (Motorola 68000family).
- Segmented addressing where the memory space is divided into several segments and the processor is limited to access program instructions and data in specific segments.

8086 Memory Organization

Each memory location 8086 is a byte while the 8086 is a 16-bits microprocessor?
Many of the 8086 ‘s operation codes are single bytes. I/O devices like printers, terminals and modems are designed to transfer ASCII encoded data. The 8086 has a 20-bits address bus, allowing to access a memory of 2^{20} = 1 M locations.

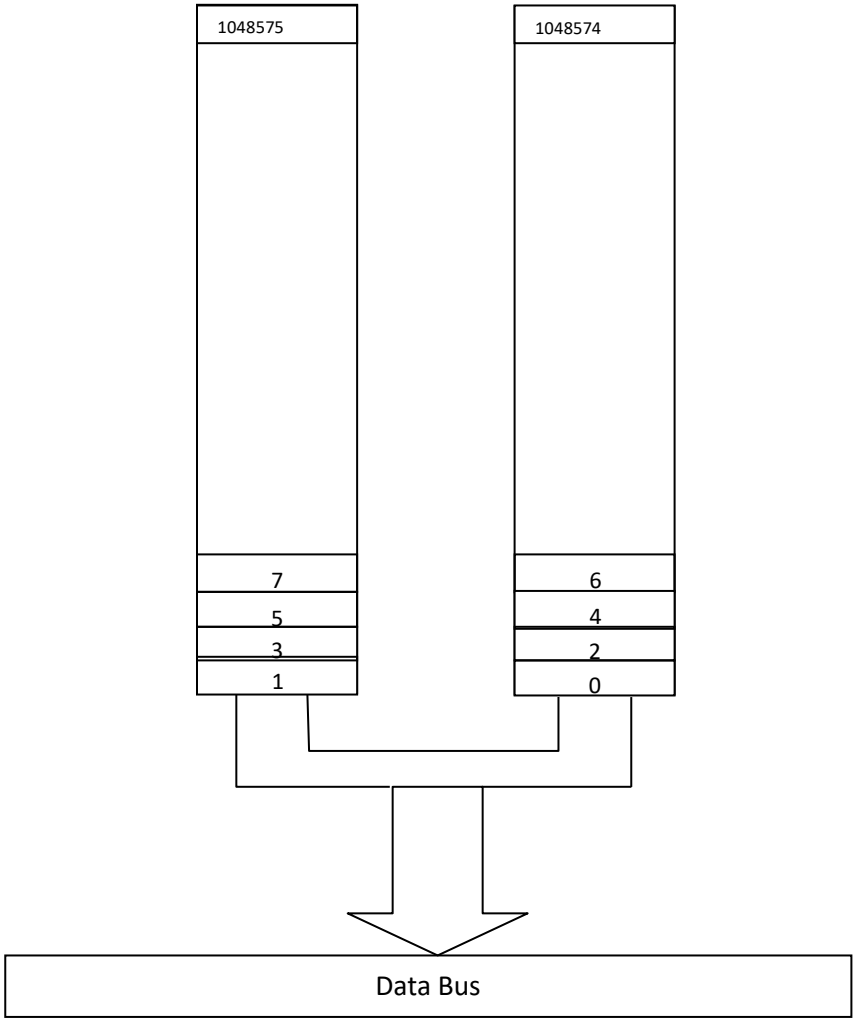


Fig.5 Memory organization of 8086 Microprocessor

Segment Registers

Within the 1MB of memory space, the 8086 defines four 64 K memory blocks: The code segment, stack segment, data segment, extra segment.

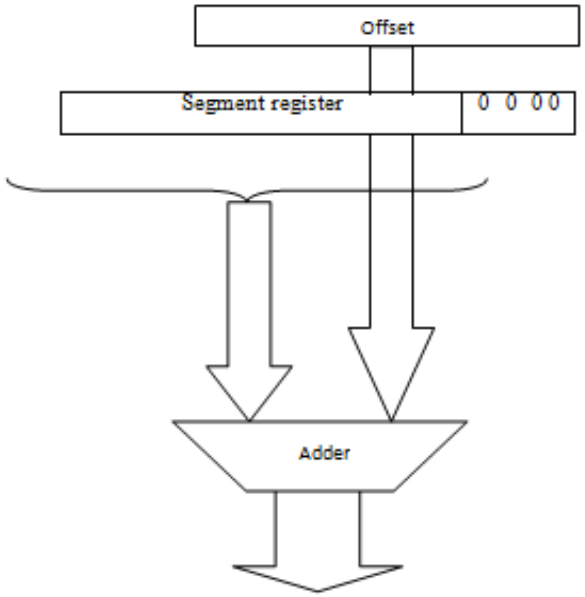
Each of these blocks of memory is used differently by the processor.

- The code segment holds the program instruction codes.
- The data segment holds the data of the program.
- The extra segment is an extra data segment (often used for shared data).
- The stack segment is used to store interrupt and subroutine return addresses.

There are four segment registers CS, DS, ES and SS and each of them defines the starting address of the corresponding segment. Each segment register is 16 bits wide while the address bus is 20 bits wide. The BIU takes care of this by appending four 0s to the low order bits of the segment register.

Logical and Physical Addresses

Addresses within a segment cab be ranged from address 0 to address FFFFh. This correspond to the 64K length of the segment. An address within a segment is called an offset or logical address. For example, logical address 20h in the code segment shown above actually correspond to the real address E0000h+ 20h= E0020h. this real address is called the physical address. Physical address is the 20 bits address that being output bus the BIU on the address bus.



Types of memory Reference	Default Segment	Alternate Segment	Offset (Logical Address)
Instruction fetch	CS	None	IP
Stack operation	SS	None	SP
General data	DS	CS, ES, SS	Effective address
String source	DS	CS, ES, SS	SI
String destination	ES	None	DI
BX used as pointer	DS	CS, ES, SS	Effective address
BP used as pointer	SS	CS, ES, SS	Effective address

1.9 8086 interrupt

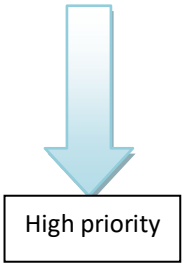
Introduction

An interrupt is the method of processing the microprocessor by peripheral device. An interrupt is used to cause a temporary halt in the execution of program. Microprocessor responds to the interrupt with an interrupt service routine, which is short program or subroutine that instructs the microprocessor on how to handle the interrupt.

There are two basic type of interrupt, maskable and non-maskable, non- maskable interrupt requires an immediate response by microprocessor, it usually used for serious circumstances like power failure. A maskable interrupt is an interrupt that the microprocessor can ignore depending upon some predetermined upon some predetermined condition defined by status register.

Interrupt can divide to five group:

1. hardware interrupt
2. Non-maskable interrupt
3. Software interrupt
4. Internal interrupt
5. Reset



Hardware, software and internal interrupt are service on priority basis. each interrupt is given a different priority level by assign it a type number. Type 0 identifies the highest-priority and type 255 identifies the lowest- priority interrupt.

The 80x86 chips allow up to 256 vectored interrupts. This means that you can have up to 256 different sources for an interrupt and the 80x86 will directly call the service routine for that interrupt without any software processing. This is in contrast to non vectored interrupts that transfer control directly to a single interrupt service routine, regardless of the interrupt source.

The 80x86 provides a 256 entry interrupt vector table beginning at address 0:0 in memory. This is a 1K table containing 256 4-byte entries. Each entry in this table contains a segmented address that points at the interrupt service routine in memory. The lowest five types are dedicated to specific interrupts such as the divide by zero interrupt and thenonmaskableinterrupt.Thenext27interrupttypes,from5to31are reserved by Intel for use in future microprocessors. The upper 224 interrupt types, from32 to 255, are available to use for hardware and software interrupts.

When an interrupt occurs (shown in figure 6), regardless of source, the 80x86 does the following:

1. The CPU pushes the flags register onto the stack.
2. The CPU pushes a far return address (segment: offset) onto the stack, segment value first.
3. The CPU determines the cause of the interrupt (i.e., the interrupt number) and fetches the four byte interrupt vector from address 0:vector*4.
4. The CPU transfers control to the routine specified by the interrupt vector table entry.

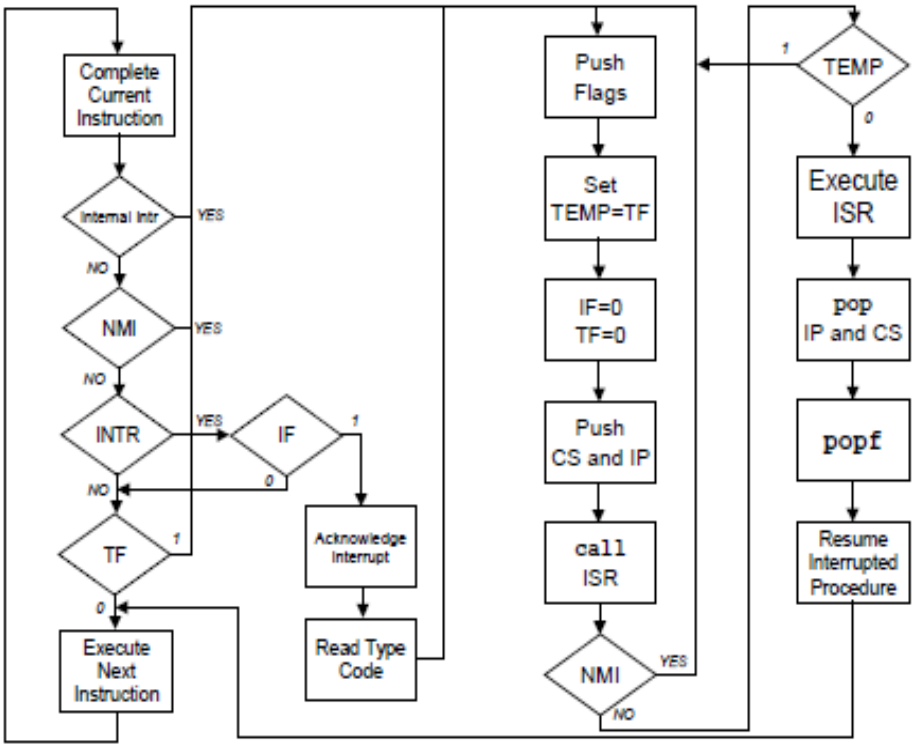


Fig.6 block diagram to interrupt handle

When the interrupt service routine wants to return control, it must execute an IRET (interrupt return) instruction. The interrupt return pops the far return address and the flags off the stack. Note that executing a far return is insufficient since that would leave the flags on the stack.

Hardware interrupt

The primary sources of interrupts, however, are the PCs timer chip, keyboard, serial ports, parallel ports, disk drives, CMOS real-time clock, mouse, sound cards, and other peripheral devices. These devices connect to an Intel 8259A programmable interrupt controller (PIC) that prioritizes the interrupts and interfaces with the 80x86 CPU. The 8259A chip adds considerable complexity to the software that processes interrupts.

programmable interrupt controller

The 8259A programmable interrupt controller chip accepts interrupts from up to eight different devices, which shown in figure (2), If any one of the devices requests service, the 8259 will toggle an interrupt output line (connected to the CPU) and pass a programmable interrupt vector to the CPU. You can cascade (show in figure(3))the device to support up to 64 devices by connecting nine 8259s together: eight of the devices with eight inputs each whose outputs become the eight inputs of the ninth device. A typical PC uses two of these devices to provide 15 interrupt inputs (seven on the master PIC with the eight input coming from the slave PIC to process its eight inputs)7. The sections following this one will describe the devices connected to each of those inputs, for now we will concentrate on what the 8259 does with those inputs. Nevertheless, for the sake of discussion, the following table lists the interrupt sources on the PC:

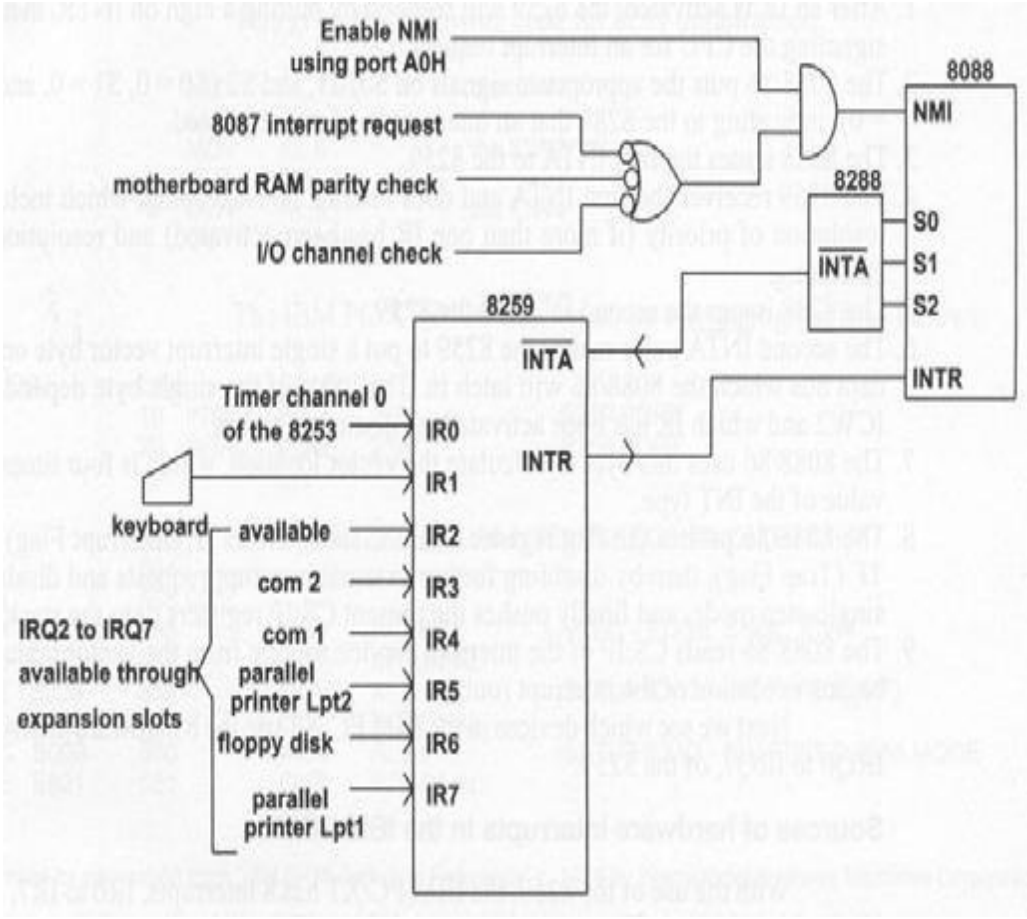


Fig.7general block diagram 8086 interrupt

NON-MASKABLE INTERRUPT(NMI)

The processor provides a single non-maskable interrupt pin (NMI) which has higher priority than the maskable interrupt request pin (INTR). A typical use would be to activate a power failure routine. The NMI is edge- triggered on a LOW-to-HIGH transition. The activation of this pin causes a type 2 interrupt. NMI is required to have a duration in the HIGH state of greater than two CLK cycles, but is not required to be synchronized to the clock. Any high-going transition of NMI is latched on-chip and will be serviced at the end of the current instruction or between whole moves of a block-type instruction. Worst case response to NMI would be for multiply, divide, and variable shift instructions. There is no specification on the occurrence of the low-going edge; it may occur before, during, or after the servicing of NMI. Another
high-going edge triggers another response if it occurs after the start of the NMI procedure. The signal must be free of logical spikes in general and be free of bounces on the low-going edge to avoid triggering extraneous responses.

MASKABLEINTERRUPT

Whenever an external signal activates the INTR pin, the microprocessor will be interrupted only if interrupts are enabled using set interrupt Flag instruction. If the interrupts are disabled using clear interrupt Flag instruction, the microprocessor will not get interrupted even if INTR is activated. That is, INTR can be masked. INTR is a non vectored interrupt, which means, the 8086 does not know where to branch to service the interrupt. The 8086 has to be told by an external device like a Programmable Interrupt controller regarding the branch. Whenever the INTR pin is activated by an

I/O port, if Interrupts are enabled and NMI is not active at that time, the microprocessor finishes the current instruction that is being executed and gives out a '0' on INTA pin twice. When INTA pin goes low for the first time, it asks the external device to get ready. In response to the second INTA the microprocessor receives the 8 bit, say N, from a programmable Interrupt controller. The action taken is as follows.

1. Complete the current instruction.
2. Activates INTA output, and receives type Number, sayN
3. Flag register value, CS value of the return address & IP value of the return address are pushed on to the stack.
4. IP value is loaded from contents of word location N x4.
5. CS is loaded from contents of the next word location.
6. 2Interrupt Flag and trap Flag are reset to0.

At the end of the ISS, there will be an IRET instruction. This performs popping off from the stack top to IP, CS and Flag registers. Finally, the register values which are also saved on the stack at the start of ISS, are restored from the stack and a return to the interrupted program takes place using the IRET instruction.

Table 1:type of interrupt

Vector No.	Mnemonic	Description	Source
0	#DE	Divide Error	DIV and IDIV instructions.
1	#DB	Debug	Any code or data reference.
2		NMI Interrupt	Non-maskable external interrupt.
3	#BP	Breakpoint	INT 3 instruction.
4	#OF	Overflow	INTO instruction.
5	#BR	BOUND Range Exceeded	BOUND instruction.
6	#UD	Invalid Opcode (UnDefined Opcode)	UD2 instruction or reserved opcode.[1]
7	#NM	Device Not Available (No Math Coprocessor)	Floating-point or WAIT/FWAIT instruction.
8	#DF	Double Fault	Any instruction that can generate an exception, an NMI, or an INTR.

Vector No.	Mnemonic	Description	Source
9		CoProcessor Segment Overrun (reserved)	Floating-point instruction.[2]
10	#TS	Invalid TSS	Task switch or TSS access.
11	#NP	Segment Not Present	Loading segment registers or accessing system segments.
12	#SS	Stack Segment Fault	Stack operations and SS register loads.
13	#GP	General Protection	Any memory reference and other protection checks.
14	#PF	Page Fault	Any memory reference.
15		(Intel reserved. Do not use.)	
16	#MF	Floating-Point Error (Math Fault)	Floating-point or WAIT/FWAIT instruction.
17	#AC	Alignment Check	Any data reference in memory.[3]
18	#MC	Machine Check	Error codes (if any) and source are model dependent.[4]
19-31		(Intel reserved. Do not use.)	
32-255		Maskable Interrupts	External interrupt from INTR pin or INT *n* instruction.

1. The UD2 instruction was introduced in the Pentium® Pro processor.
2. Intel Architecture processors after the Intel386™ processor do not generate this exception.
3. This exception was introduced in the Intel486™ processor.
4. This exception was introduced in the Pentium processor and enhanced in the Pentium Pro processor.

Software interrupt Instructions

There are instructions in 8086 which cause an interrupt. They are

- INT instructions with type number specified.
- INT 3, Break Point Interrupt instruction.
- INTO, Interrupt on overflow instruction.

These are instructions at the desired places in a program. When one of these instructions is executed a branch to an ISS takes place. Because their execution results in a branch to an ISS, they are called interrupts. Software Interrupt instructions can be used to test the working of the various Interrupt handlers- For example, we can execute INTO instruction to execute type 0 ISS, without really having to divide a number by 0. Similarly, we can execute INT 2 instruction to test NMIISS.

INT-Interrupt Instruction with Type number Specified

The mnemonic for this is INT. It is a 2 byte instruction. The first byte provides the op-code and

the second byte the Interrupt type number. Op-code for this instruction isCDH The execution of an INT instruction, say INTN, when N is the value in the range 00H to FFH, results in the following:

1. Flag register value is pushed on to the stack.
2. CS value of the Return address and IP value of the Return address are pushed on to the stack.
3. IP is loaded from the contents of the word location N x4.
4. CS is loaded from the contents of the next word location.
5. Interrupt Flag and Trap Flag are reset to0.

Thus a branch to the ISS take place. During the ISS, interrupt are disabled because the Interrupt flag is reset to 0.At the end of the ISS, there will be an IRET instruction. Thus a return back to the interrupted program takes place with Flag registers unchanged.

INT 3-Break Point Interrupt Instruction

When a break point is inserted, the system executes the instructions up to the breakpoint, and then goes to the break point procedure. Unlike the single-Step feature which stops execution after each instruction, the breakpoint feature executes all the instructions up to the inserted breakpoint and then stops execution. The mnemonic for the instruction is INT3. It is a 1 byte instruction Op-code for this is CCH.

The execution of INT3 instruction results in the following.

1. Flag register value is pushed on to the Stack.
2. CS value of the return address and IP value of the return address are pushed onto the Stack.
3. IP is loaded from the contents of the word location 3x4 =0000CH.
4. CS is loaded from the contents of the next word location.
5. Interrupt Flag and Trap Flag are reset to0.

Thus a branch to the ISS takes place. During the ISS, interrupts are disabled because Interrupt flag is reset to 0. At the end of the ISS, there will be an IRET instruction to return back to the interrupted program. A break point interrupt service procedure usually saves all the register contents on the Stack. Depending upon the system, it may then send the

register contents to the CRTdisplay and wait for the next command from the user.

INTO - Interrupt on over flow instruction

The 8086 overflow flag, OF, will be set if the signed result of an arithmetic operation on two signed numbers is too large to be represented in the destination register or memory location. For example, if we add the 8-bit signed number 01101100 and the 8- bit signed number 01010001, the signed result will be 10111101. This is correct if we add unsigned binary numbers, but it is not the correct signed result. There are two ways to detect and respond to an overflow error in a program. One way is to put the jump if overflow instruction, JO, immediately after the arithmetic instruction. If the overflow flag is Set, execution will jump to the address specified in the JO instruction. At this address an error routine may be put which respond to the overflow. The second way is to put them INTO instruction immediately after the arithmetic Instruction in the program. The mnemonic for the instruction is INTO. It is a 1 byte instruction. The op-code for this is CEH. It is a conditional interrupt instruction. Only if the overflow flag is Set, a branch takes place to an interrupt handler whose interrupt type

number is 4. If the overflow flag is reset, the execution continues with the next instruction. The execution of INTO results in the following.

6.Flag register values are pushed on to the Stack.

7.CS value of the return address and IP value of the return address and IP value of the return address are pushed on to the stack.

8.IP is loaded from the contents of word location 4x4 =00010H.

9.CS is loaded from the contents of next word location.

10. Interrupt flag and Trap flag are reset to0.

Thus a branch to ISS takes place. During the ISS, interrupts are disabled. At the end of ISS, there will be an IRET instruction, returning back to the interrupted program. Instructions in the ISS procedure perform the desired response to the error condition.

11. RESET

Processor initialization or start up is accomplished with activation (HIGH) of the RESET pin which it shows in table (2). The 8086 RESET is required to be HIGH for greater than 4 CLK cycles. The 8086 will terminate operations on the high-going edge of RESET and will remain dormant as long as RESET is HIGH. The low-going transition of RESET triggers an internal reset sequence for approximately 10 CLK cycles. After this interval the 8086 operates normally beginning with the instruction in absolute location FFFF0H.

Table 2:process initialization register content

CPU Asset	Content
FLAGS Register	0000h
IP	0000h
CS	ffffh
DS	0000h
SS	0000h
ES	0000h
Instruction Queue	Empty

1.11 I/O Interfacing Techniques

Input/output devices can be interfaced with microprocessor systems in two ways :

1. I/O mapped I/O
2. Memory mapped I/O

1. I/O mapped I/O :

8086 has special instructions IN and OUT to transfer data through the input/output ports in I/O mapped I/O system. The IN instruction copies data from a port to the Accumulator. If an 8-bit port is read data will go to AL and if 16-bit port is read the data will go to AX. The OUT instruction copies a byte from AL or a word from AX to the specified port. The M/IO signal is always low when 8086 is executing these instructions. In this address of I/O device is 8-bit or 16-bit. It is 8-bit for Direct addressing and 16-bit for Indirect addressing.

2. Memory mapped I/O

In this type of I/O interfacing, the 8086 uses 20 address lines to identify an I/O device. The I/O device is connected as if it is a memory device. The 8086 uses same control signals and instructions to access I/O as those of memory, here RD and WR signals are activated indicating memory bus cycle.

Problem :

Interface an input port 74LS245 to read the status of the switches SW1 to SW8. the switches when shorted, input a '1' else input a '0' to the microprocessor system. Store the status in register BL. The address of the port is 0740H

Solution :

The hardware interface circuit is shown in figure. The address, control and data lines are assumed to be readily available at the microprocessor system The ALP is given as follows :

```
MOV BL, 00H ; clear BL for status
MOV DX, 0740H ; 16-bit Port address in DX
IN AL,DX ; Read Port 0740H for switch positions.
MOV BL,AL ; Store status of switches from AL into BL
HLT ; Stop
```

Problem :

Design an interface of input port 74LS245 to read the status of switches SW1 to SW8 and output port 74LS373 with 8086. display the number of key that is pressed with the help of output port on 7 segment display.

Solution : Status of the switches is first read into the AL. Displaying the shorted switch number in the 7 segment display. Instead of using 16 address lines, one may use only A3– A0.

QUESTION BANK

PART – B (16 MARKS)

1. Explain the internal hardware architecture of 8086 microprocessor with neat diagram?
2. Write short note about assembler directives?
3. Write a note about stack, procedures and macros?
4. Define interrupt and their two classes? Write in detail about interrupt service routine?
5. Explain byte and string manipulation with examples?
6. Write an ALP to find the largest number and smallest number in the array?
7. Explain the Architecture of 8086 microprocessor with neat diagram
8. With a neat sketch describe the Minimum and Maximum mode of operation of 8086.
9. Discuss in detail the various signals of 8086.
10. Write short notes on addressing memory.
11. Write notes on addressing input and output devices?
12. Explain in detail about 8086 memory banks and associated signals for byte and word operations.
13. Explain about System Bus Structure with suitable timing diagram.
14. Discuss in detail about Interrupt Priority Management.
15. Explain the various Bus Arbitration Schemes.
16. Draw the bus cycle timing diagram for I/O read and I/O write in minimum mode for 8086 processor.
17. Describe the functional block diagram of 8086 Microprocessor with register Organization.
18. Draw the bus cycle timing diagram for I/O read and I/O write in maximum mode for 8086 processor.
19. Draw the bus cycle timing diagram for the memory read and memory write in minimum mode for 8086
20. i) Explain external memory addressing.
    ii) Draw timing diagram of
        (i) 16 bit Read at odd address 0A2F8H 0BC07H.
        (ii) 8 bit Read at odd address 0A2F7H.
        (iii) 16 bit Read at even address 0A2F7H 0A2F8H.
21. Explain in 8086 signals and timing in detail with neat diagram.
22. Explain in detail about the minimum and maximum mode configurations of 8086.
23. Why 8086 is called a pipelined processor justify.
24. Explain external memory addressing.
25. Draw the bus cycle timing diagram for the memory read and memory write in maximum mode for 8086

UNIT II

PROGRAMMING OF 8086

Programming of 8086

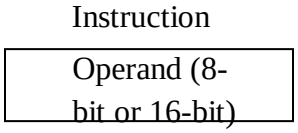
Addressing Modes-Instruction format-Instruction set-Assembly language programs in 8086. RISC architecture – introduction to ARM Programming-register configuration and instruction set – sample program.

2.1 8086 ADDRESSING MODES:

Addressing modes refers to the mechanism by which the operands are specified for an operation. Depending on the type of operands used, there are 7 addressing modes available.

1. Immediate Mode:

In this mode, the instruction contains 8-bit or 16-bit immediate data along with the mnemonic. Data can be moved to 8-bit or 16-bit register, or memory location using the address or register.

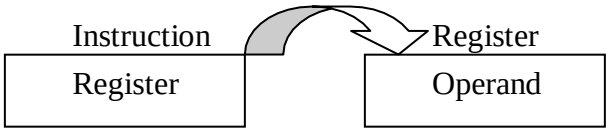


Ex: MOV AX,1234 h, ANDAL,3FH, MOV DX,0ABCDH
MOV[2000h],25 **MOV AL,3373(invalid), MOV DS, 1234(invalid)**

Note: a) Any data must start with a decimal digit (0 to 9). Hex digits A to F are treated as characters. So, Hex nos must be prefixed with a Zero. (Ex: MOV CL, 0A2H)
b) Immediate data cannot be copied into Segment registers.
c) Size of the data (byte or word) must match the size of registers (8-bit or16-bit)

2. Register Mode:

In this mode, the operands are available in general purpose registers. Instruction specifies the register name.



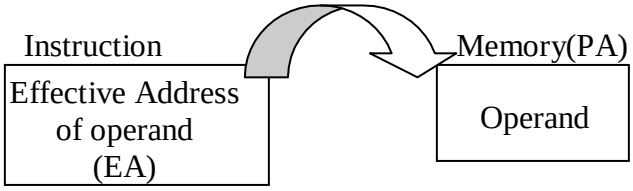
Ex: MOVCL,DL ; ADD BL,AL; **MOV AX, CL(invalid)**
XOR CX, DX ; DECBX ;

Memory Operands:

3. Direct Mode:

Here, the instruction contains 16-bit offset (Effective address) (EA) of the memory location.

The Physical Address (PA) is calculated by 8086 using EA and Segment Register; and then the operand will be fetched from memory. Default register is DS.



PA = EA + (DS * 10H)

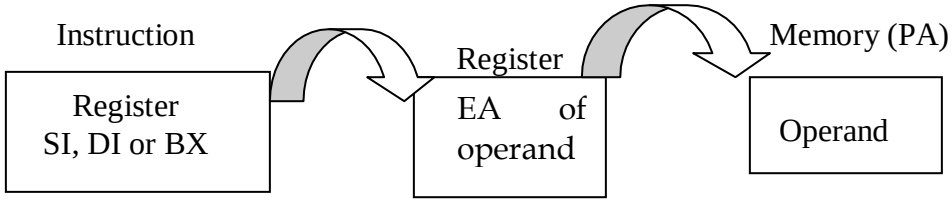
Ex: MOVBL,NUM ; MOV[3455H],AL ; MOV CX, ES: 20h, MOV NUM2,CX

4. Indirect Modes

8086 provides many indirect addressing modes (where one operand is in memory) which are extensively used in programming.

a) Register Indirect Mode:

In this mode, the instruction contains a 16-bit register name which contains the EA. Using this EA, PA is calculated. Default segment register for memory is DS. Only BX, SI and DI



register can be used to hold the EA.

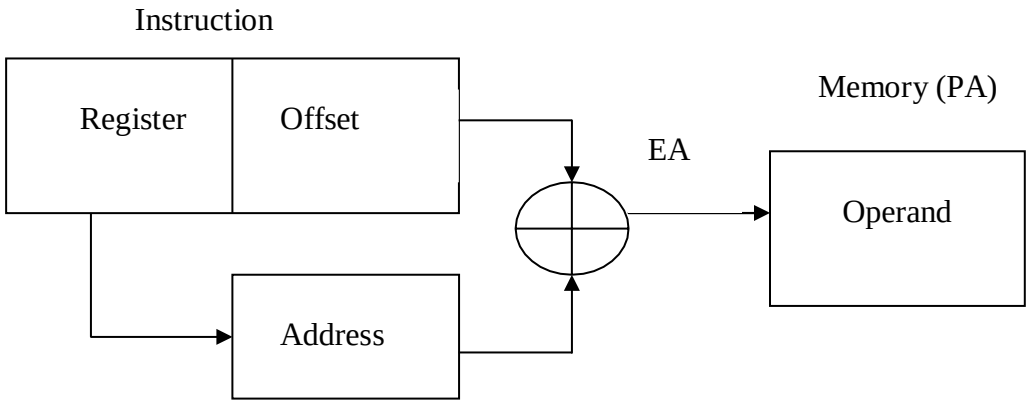
Ex: MOV AL, BYTE PTR [BX] ; INC WORD PTR [DI], **MOV [BX], [SI] (invalid)**
MUL BYTE PTR [SI] ; AND CX, WORD PTR [BX], MOV BYTE PTR [BX], 50h
ADD CX, [AX](invalid)

b) Register Relative Mode or Indexed Mode: (Register Indirect with Displacement)

In this mode, the instruction contains a 16-bit register name (which contains an address) and a signed displacement. Adding the register contents with displacement gives the EA. Using this EA, PA is calculated. Default segment register for memory is DS. Only BX, SI, DI and BP register can be used.

PA = EA + (DS * 10H)

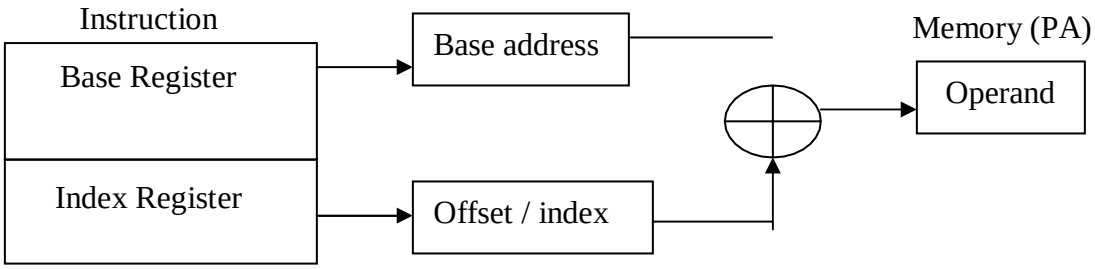
EA= { (BX), (BP), (SI), (DI) } + { 8-bit displacement(sign-extended) or 16- bit signed displacement }



Ex: MOV AL, BYTE PTR 50 [BX] ; DEC WORD PTR [DI+30] MOV
100H WORD PTR[BX],1224H ; MOV CX, -22H[SI]

c) Base plus Indexed Mode:

In this mode, the instruction contains a 16-bit Base register name (which contains the base address) and a 16-bit Index register name. Adding the contents of both registers, EA is obtained. Then, PA is calculated. Default segment register for memory is DS. Only BX, BP and SI, DI registers can be used.



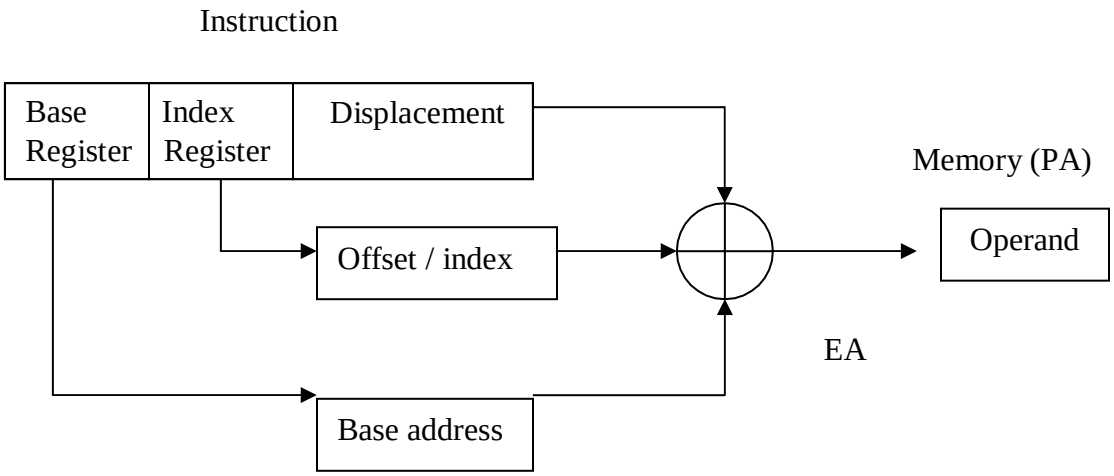
EA= (BX)or(BP) + (SI) or(DI)

PA = EA + (DS * 10H)

Ex: MOV [BX][SI],1234H, SUB AL, BYTE PTR[BX][DI]

d) Relative Based ñ Indexed Mode:

In this mode, the instruction contains a 16-bit Base register name (which contains the base address), a 16-bit Index register name and a displacement. Adding the contents of both registers along with the displacement, EA is obtained. Then, PA is calculated. Default segment register for memory is DS. Only BX, BP and SI, DI registers can be used.



EA= (BX)or(BP) + { (SI)or(DI) + { 8-bit sign-extended displacement or 16-bit signed displacement }

PA = EA + (DS * 10H)

Ex: MOV 50h [BX] [SI],1234H MOV -30H [BP] [SI],DX

2.2 Microprocessor - 8086 Instruction Sets

The 8086 microprocessor supports 8 types of instructions −

- Data Transfer Instructions
- Arithmetic Instructions
- Bit Manipulation Instructions
- String Instructions
- Program Execution Transfer Instructions (Branch & Loop Instructions)
- Processor Control Instructions
- Iteration Control Instructions
- Interrupt Instructions

Data Transfer Instructions

These instructions are used to transfer the data from the source operand to the destination operand. Following are the list of instructions under this group −

Instruction to transfer a word

- **MOV** − Used to copy the byte or word from the provided source to the provided destination.

- **PUSH** − Used to put a word at the top of the stack.
- **POP** − Used to get a word from the top of the stack to the provided location.
- **PUSH** − Used to put all the registers into the stack.
- **POP** − Used to get words from the stack to all registers.
- **XCHG** − Used to exchange the data from two locations.
- **XLAT** − Used to translate a byte in AL using a table in the memory.

Instructions for input and output port transfer

- **IN** − Used to read a byte or word from the provided port to the accumulator.
- **OUT** − Used to send out a byte or word from the accumulator to the provided port.

Instructions to transfer the address

- **LEA** − Used to load the address of operand into the provided register.
- **LDS** − Used to load DS register and other provided register from the memory
- **LES** − Used to load ES register and other provided register from the memory.

Instructions to transfer flag registers

- **LAHF** − Used to load AH with the low byte of the flag register.
- **SAHF** − Used to store AH register to low byte of the flag register.
- **PUSHF** − Used to copy the flag register at the top of the stack.
- **POPF** − Used to copy a word at the top of the stack to the flag register.

Arithmetic Instructions

These instructions are used to perform arithmetic operations like addition, subtraction, multiplication, division, etc.

Following is the list of instructions under this group −

Instructions to perform addition

- **ADD** − Used to add the provided byte to byte/word to word.
- **ADC** − Used to add with carry.
- **INC** − Used to increment the provided byte/word by 1.

- **AAA** − Used to adjust ASCII after addition.
- **DAA** − Used to adjust the decimal after the addition/subtraction operation.

Instructions to perform subtraction

- **SUB** − Used to subtract the byte from byte/word from word.
- **SBB** − Used to perform subtraction with borrow.
- **DEC** − Used to decrement the provided byte/word by 1.
- **NPG** − Used to negate each bit of the provided byte/word and add 1/2's complement.
- **CMP** − Used to compare 2 provided byte/word.
- **AAS** − Used to adjust ASCII codes after subtraction.
- **DAS** − Used to adjust decimal after subtraction.

Instruction to perform multiplication

- **MUL** − Used to multiply unsigned byte by byte/word by word.
- **IMUL** − Used to multiply signed byte by byte/word by word.
- **AAM** − Used to adjust ASCII codes after multiplication.

Instructions to perform division

- **DIV** − Used to divide the unsigned word by byte or unsigned double word by word.
- **IDIV** − Used to divide the signed word by byte or signed double word by word.
- **AAD** − Used to adjust ASCII codes after division.
- **CBW** − Used to fill the upper byte of the word with the copies of sign bit of the lower byte.
- **CWD** − Used to fill the upper word of the double word with the sign bit of the lower word.

Bit Manipulation Instructions

These instructions are used to perform operations where data bits are involved, i.e. operations like logical, shift, etc.

Following is the list of instructions under this group −

Instructions to perform logical operation

- **NOT** − Used to invert each bit of a byte or word.
- **AND** − Used for adding each bit in a byte/word with the corresponding bit in another byte/word.
- **OR** − Used to multiply each bit in a byte/word with the corresponding bit in another byte/word.
- **XOR** − Used to perform Exclusive-OR operation over each bit in a byte/word with the corresponding bit in another byte/word.
- **TEST** − Used to add operands to update flags, without affecting operands.

Instructions to perform shift operations

- **SHL/SAL** − Used to shift bits of a byte/word towards left and put zero(S) in LSBs.
- **SHR** − Used to shift bits of a byte/word towards the right and put zero(S) in MSBs.
- **SAR** − Used to shift bits of a byte/word towards the right and copy the old MSB into the new MSB.

Instructions to perform rotate operations

- **ROL** − Used to rotate bits of byte/word towards the left, i.e. MSB to LSB and to Carry Flag [CF].
- **ROR** − Used to rotate bits of byte/word towards the right, i.e. LSB to MSB and to Carry Flag [CF].
- **RCR** − Used to rotate bits of byte/word towards the right, i.e. LSB to CF and CF to MSB.
- **RCL** − Used to rotate bits of byte/word towards the left, i.e. MSB to CF and CF to LSB.

String Instructions

String is a group of bytes/words and their memory is always allocated in a sequential order.

Following is the list of instructions under this group −

- **REP** − Used to repeat the given instruction till CX ≠ 0.
- **REPE/REPZ** − Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.
- **REPNE/REPNZ** − Used to repeat the given instruction until CX = 0 or zero flag ZF = 1.
- **MOVS/MOVSB/MOVSW** − Used to move the byte/word from one string to another.

- **COMS/COMPSB/COMPSW** − Used to compare two string bytes/words.
- **INS/INSB/INSW** − Used as an input string/byte/word from the I/O port to the provided memory location.
- **OUTS/OUTSB/OUTSW** − Used as an output string/byte/word from the provided memory location to the I/O port.
- **SCAS/SCASB/SCASW** − Used to scan a string and compare its byte with a byte in AL or string word with a word in AX.
- **LODS/LODSB/LODSW** − Used to store the string byte into AL or string word into AX.

Program Execution Transfer Instructions (Branch and Loop Instructions)

These instructions are used to transfer/branch the instructions during an execution. It includes the following instructions −

Instructions to transfer the instruction during an execution without any condition −

- **CALL** − Used to call a procedure and save their return address to the stack.
- **RET** − Used to return from the procedure to the main program.
- **JMP** − Used to jump to the provided address to proceed to the next instruction.

Instructions to transfer the instruction during an execution with some conditions −

- **JA/JNBE** − Used to jump if above/not below/equal instruction satisfies.
- **JAE/JNB** − Used to jump if above/not below instruction satisfies.
- **JBE/JNA** − Used to jump if below/equal/ not above instruction satisfies.
- **JC** − Used to jump if carry flag CF = 1
- **JE/JZ** − Used to jump if equal/zero flag ZF = 1
- **JG/JNLE** − Used to jump if greater/not less than/equal instruction satisfies.
- **JGE/JNL** − Used to jump if greater than/equal/not less than instruction satisfies.
- **JL/JNGE** − Used to jump if less than/not greater than/equal instruction satisfies.
- **JLE/JNG** − Used to jump if less than/equal/if not greater than instruction satisfies.
- **JNC** − Used to jump if no carry flag (CF = 0)
- **JNE/JNZ** − Used to jump if not equal/zero flag ZF = 0

- **JNO** − Used to jump if no overflow flag OF = 0
- **JNP/JPO** − Used to jump if not parity/parity odd PF = 0
- **JNS** − Used to jump if not sign SF = 0
- **JO** − Used to jump if overflow flag OF = 1
- **JP/JPE** − Used to jump if parity/parity even PF = 1
- **JS** − Used to jump if sign flag SF = 1

Processor Control Instructions

These instructions are used to control the processor action by setting/resetting the flag values.

Following are the instructions under this group −

- **STC** − Used to set carry flag CF to 1
- **CLC** − Used to clear/reset carry flag CF to 0
- **CMC** − Used to put complement at the state of carry flag CF.
- **STD** − Used to set the direction flag DF to 1
- **CLD** − Used to clear/reset the direction flag DF to 0
- **STI** − Used to set the interrupt enable flag to 1, i.e., enable INTR input.
- **CLI** − Used to clear the interrupt enable flag to 0, i.e., disable INTR input.

Iteration Control Instructions

These instructions are used to execute the given instructions for number of times. Following is the list of instructions under this group −

- **LOOP** − Used to loop a group of instructions until the condition satisfies, i.e., CX = 0
- **LOOPE/LOOPZ** − Used to loop a group of instructions till it satisfies ZF = 1 & CX = 0
- **LOOPNE/LOOPNZ** − Used to loop a group of instructions till it satisfies ZF = 0 & CX = 0
- **JCXZ** − Used to jump to the provided address if CX = 0

Interrupt Instructions

These instructions are used to call the interrupt during program execution.

- **INT** − Used to interrupt the program during execution and calling service specified.

- **INTO** − Used to interrupt the program during execution if OF = 1
- **IRET** − Used to return from interrupt service to the main program

2.4 RISC Architecture

RISC (Reduced Instruction Set Computer) is used in portable devices due to its power efficiency. RISC is a type of microprocessor architecture that uses highly-optimized set of instructions. RISC does the opposite, reducing the cycles per instruction at the cost of the number of instructions per program Pipelining is one of the unique feature of RISC. It is performed by overlapping the execution of several instructions in a pipeline fashion. It has a high performance advantage over CISC.

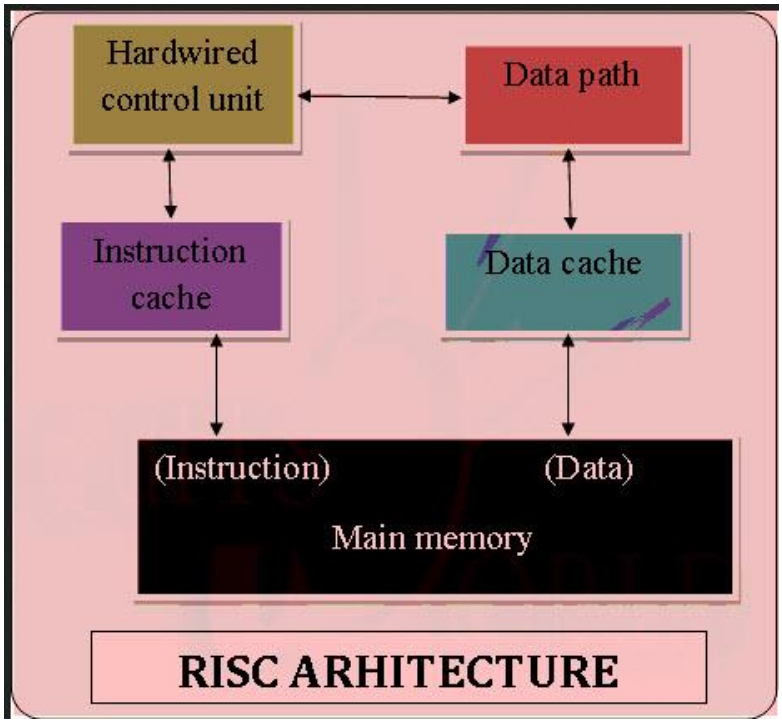


Fig 1 RISC Architecture

RISC processors take simple instructions and are executed within a clock cycle

RISC ARCHITECTURE CHARACTERISTICS

- Simple Instructions are used in RISC architecture.
- RISC helps and supports few simple data types and synthesize complex data types.
- RISC utilizes simple addressing modes and fixed length instructions for pipelining.
- RISC permits any register to use in any context.
- One Cycle Execution Time
- The amount of work that a computer can perform is reduced by separating “LOAD” and “STORE” instructions.
- RISC contains Large Number of Registers in order to prevent various number of interactions with memory.

- In RISC, Pipelining is easy as the execution of all instructions will be done in a uniform interval of time i.e. one click.
- In RISC, more RAM is required to store assembly level instructions.
- Reduced instructions need a less number of transistors in RISC.
- RISC uses Harvard memory model means it is Harvard Architecture.
- A compiler is used to perform the conversion operation means to convert a high-level language statement into the code of its form.

RISC & CISC Comparison

CISC	RISC
It is prominent on Hardware	It is prominent on the Software
It has high cycles per second	It has low cycles per second
It has transistors used for storing Instructions which are complex	More transistors are used for storing memory
LOAD and STORE memory-to-memory is induced in instructions	LOAD and STORE register-register are independent
It has multi-clock	It has a single - clock

MUL instruction is divided into three instructions
"LOAD" – moves data from the memory bank to a register
"PROD" – finds product of two operands located within the registers
"STORE" – moves data from a register to the memory banks
The main difference between RISC and CISC is the number of instructions and its complexity.

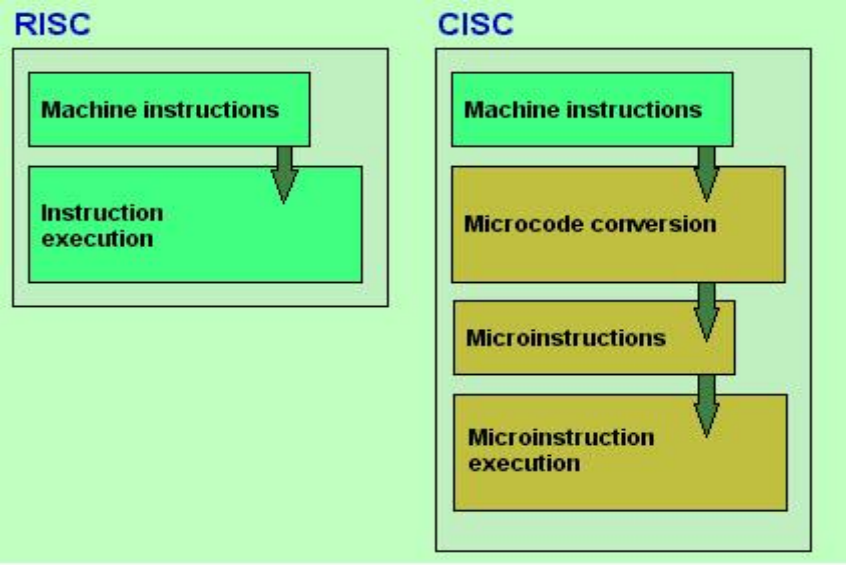


Fig 2 RISC Vs CISC

SEMANTIC GAP

Both RISC and CISC architectures have been developed as an attempt to cover the semantic gap.

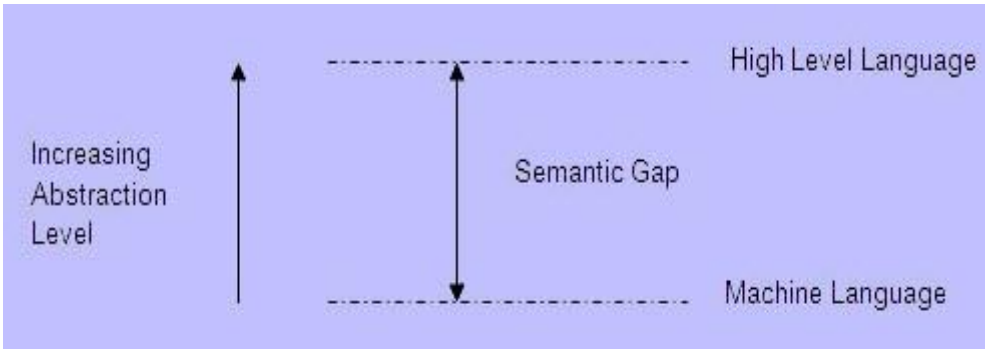


Fig 3 Semantic Gap

With an objective of improving efficiency of software development, several powerful programming languages have come up, viz., Ada, C, C++, Java, etc. They provide a high level of abstraction, conciseness and power. By this evolution the semantic gap grows. To enable efficient compilation of high level language programs, CISC and RISC designs are the two options.

CISC designs involve very complex architectures, including a large number of instructions and addressing modes, whereas RISC designs involve simplified instruction set and adapt it to the real requirements of user programs.

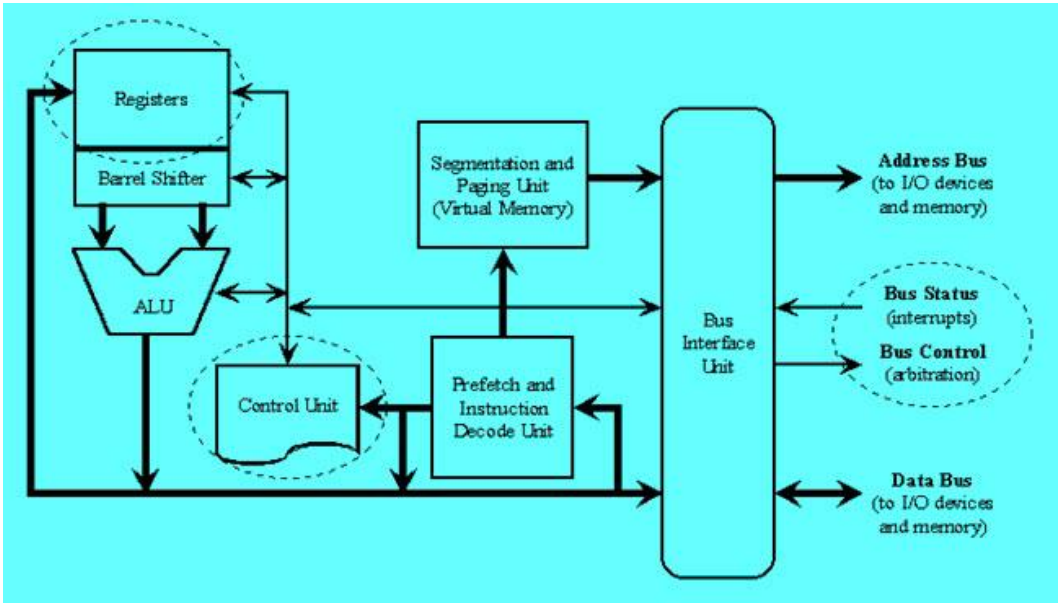


Fig 4 CISC and RISC Design

Multiplication of two Numbers in Memory

If the main memory is divided into areas that are numbered from row1:column 1 to row 5 :column 4. The data is loaded into one of four registers (A, B, C, or D). To find multiplication of two numbers- One stored in location 1:3 and other stored in location 4:2 and store back result in 1:3.

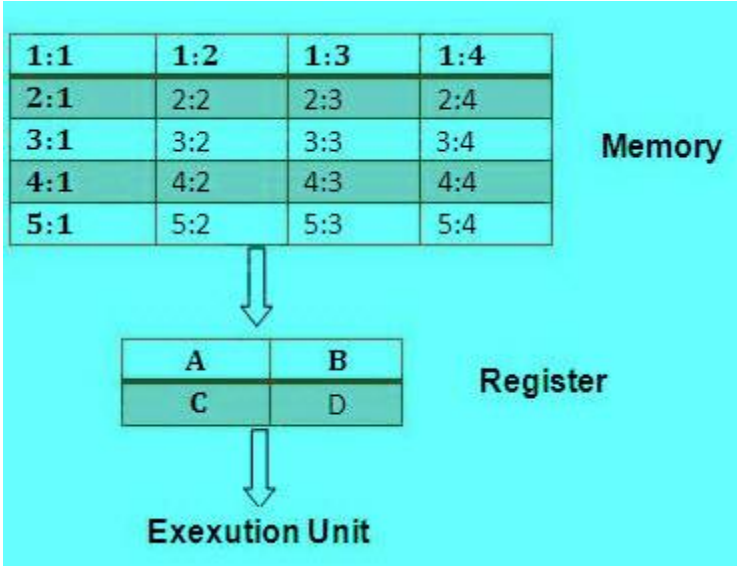


Fig 5 Multiplication of Two Numbers

The Advantages and Disadvantages of RISC and CISC

The Advantages of RISC architecture

- RISC architecture has a set of instructions, so high-level language compilers can produce more efficient code
- It allows freedom of using the space on microprocessors because of its simplicity.
- Many RISC processors use the registers for passing arguments and holding the local variables.
- RISC functions use only a few parameters, and the RISC processors cannot use the call instructions, and therefore, use a fixed length instruction which is easy to pipeline.
- The speed of the operation can be maximized and the execution time can be minimized. Very less number of instructional formats, a few numbers of instructions and a few addressing modes are needed.

The Disadvantages of RISC architecture

- Mostly, the performance of the RISC processors depends on the programmer or compiler as the knowledge of the compiler plays a vital role while changing the CISC code to a RISC code
- While rearranging the CISC code to a RISC code, termed as a code expansion, will increase the size. And, the quality of this code expansion will again depend on the compiler, and also on the machine's instruction set.

- The first level cache of the RISC processors is also a disadvantage of the RISC, in which these processors have large memory caches on the chip itself. For feeding the instructions, they require very fast memory systems.

Advantages of CISC architecture

- Microprogramming is easy assembly language to implement, and less expensive than hard wiring a control unit.
- The ease of micro coding new instructions allowed designers to make CISC machines upwardly compatible:
- As each instruction became more accomplished, fewer instructions could be used to implement a given task.

Disadvantages of CISC architecture

- The performance of the machine slows down due to the amount of clock time taken by different instructions will be dissimilar
- Only 20% of the existing instructions is used in a typical programming event, even though there are various specialized instructions in reality which are not even used frequently.
- The conditional codes are set by the CISC instructions as a side effect of each instruction which takes time for this setting – and, as the subsequent instruction changes the condition code bits – so, the compiler has to examine the condition code bits before this happens.

TEXT BOOK:

1. Douglas V.Hall, Microprocessor and Interfacing Programming and Hardware. Tata McGraw

Hill, New Delhi 2007.

REFERENCES BOOK:

1. Krishna Kant, Microprocessor and Microcontroller Architecture, programming and system

design using 8085, 8086, 8051 and 8096, PHI, New Delhi, 2008

QUESTION BANK

PART-B (16 MARKS)

1. Explain the various addressing modes of 8086 microprocessor with examples?
2. Explain Data transfer, arithmetic and branch instructions?
3. Write an 8086 ALP to find the sum of numbers in the array of 10 elements?
4. Write in detail about instruction formats and instruction execution timing?
5. Write an ALP to find the largest number and smallest number in the array? 6.Write a short note about Loop, NOP and HLT instructions
7. Write a short note about Flag manipulation, logical and shift& rotate instructions?
8. Explain the direct addressing modes and indirect addressing modes of 8086 with example.
9. Assume that the accumulator contents data bytes 88 hand instruction MOV C, A 4FH is fetched. List the steps decoding and executing the instruction
10. Write a Program to Perform the following functions and verify the output steps: a. Load the number 5CH in register D b. Load the number 9E H in register C . Increment the Contents of register C by one. d. Add the contents of register C and D and Display the sum at output port1.
11. Write an assembly language program to find out the largest number from a given unordered array of 8 bit numbers, stored in the locations starting from a known address.
12. With suitable examples explain 8086 addressing modes in detail.
13. Write 8086 assembly language program to SORT an array of 10 bytes in Descending order.
14. Write an 8086 ALP to perform 32 bit binary addition?
15. Write an 8085 ALP to convert the hexadecimal value to decimal value
16. List the addressing modes in 8086 and explain with suitable examples.
17. Describe the Different types of instruction in 8086.
18. Explain in detail about the Logical and Arithmetic group of 8086 instruction set with suitable examples.
19. Explain the various assembler directives
20. Write a program for 8 bit addition and subtraction
21. Write a program for 8 bit addition, subtraction, multiplication and division
22. Explain the assembler directives and procedures with examples.
23. Write a Program for 8 bit addition and subtraction and find largest number using 8086.
24. Explain the various ways by which the operand can be addressed with examples.
25. Describe the Instruction Set of ARM with examples.
26. Explain the internal hardware architecture of RISC with neat diagram?

UNIT III

INTERFACING DEVICES

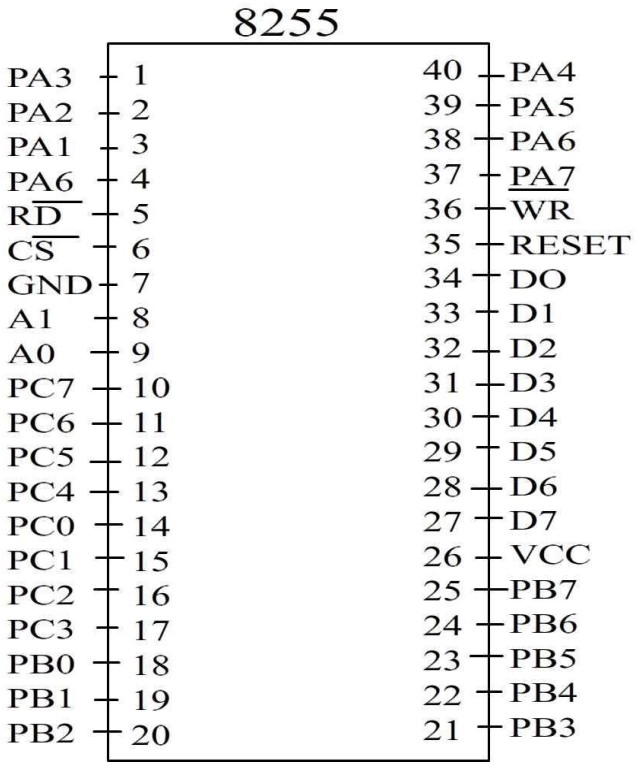
Interfacing Devices

Programmable Peripheral Interface (8255) - Programmable Interval Timer (8254) – Programmable Interrupt Controller (8259A) - Programmable DMA Controller (8257) - Programmable Communication Interface (8251A) –Programmable Keyboard and Display Controller (8279).

3.1 INTEL 8255:(Programmable Peripheral Interface)

The 8255A is a general purpose programmable I/O device designed for use with Intel microprocessors. It consists of three 8-bit bidirectional I/O ports (24I/O lines) that can be configured to meet different system I/O needs. The three ports are PORT A, PORT B & PORT C. Port A contains one 8-bit output latch/buffer and one 8-bit input buffer. Port B is same as PORT A or PORT B. However, PORT C can be split into two parts PORT C lower (PC_0-PC_3) and PORT C upper (PC_7-PC_4) by the control word. The three ports are divided in two groups Group A (PORT A and upper PORT C) Group B (PORT B and lower PORT C). The two groups can be programmed in three different modes. In the first mode (mode 0), each group may be programmed in either input mode or output mode (PORT A, PORT B, PORT C lower, PORT C upper). In mode 1, the second's mode, each group may be programmed to have 8-lines of input or output (PORT A or PORT B) of the remaining 4-lines (PORT C lower or PORT C upper) 3-lines are used for hand shaking and interrupt control signals. The third mode of operation (mode 2) is a bidirectional bus mode which uses 8-line (PORT A only for a bidirectional bus and five lines (PORT C upper 4 lines and borrowing one from other group) for handshaking.

The 8255 is contained in a 40-pin package, whose pin out is shown below:



PIN Names

RESET –Reset input

$\overline{CS}$-Chip selected

$\overline{RD}$- Read input

$\overline{WR}$- Write input A_0

A_1– port address

$PA_7 – PA_0$– PORT A

$PB_7 – PB_0$– PORT B $PC_7 –$

PC_0– PORT C VCC -+5v

GND - Ground

Fig 1 PIN Diagram of 8255

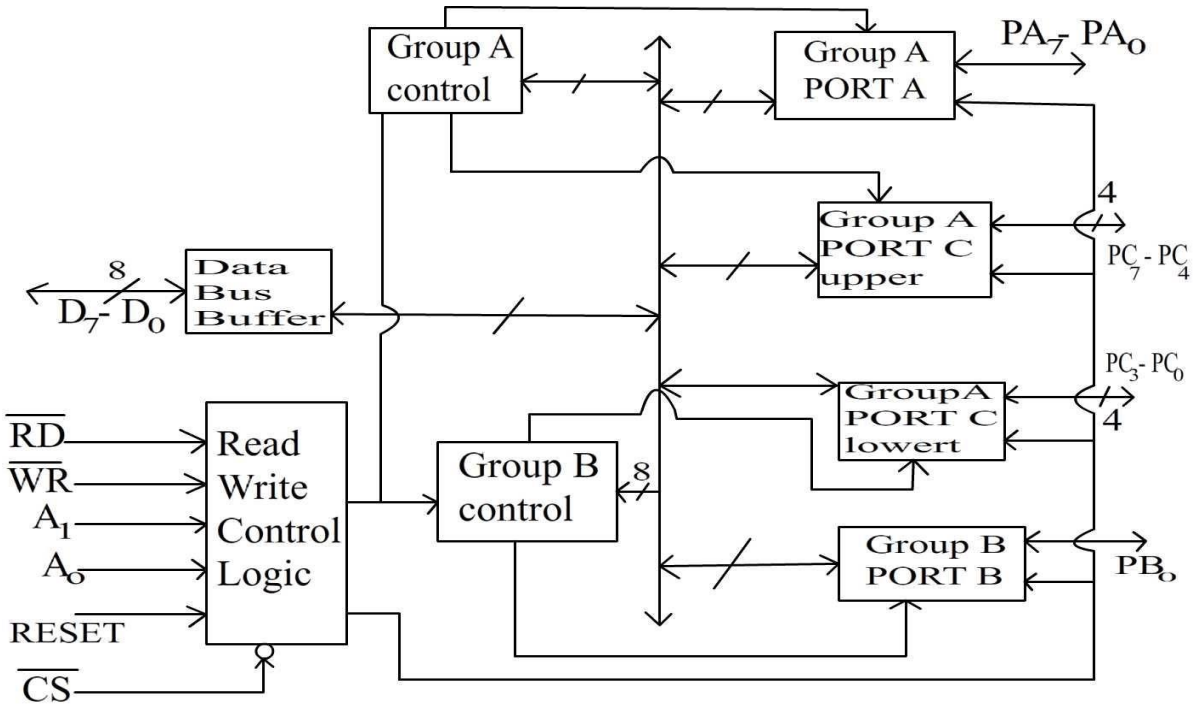


Fig 2 Block Diagram of 8255

Functional Description:

This support chip is a general purpose I/O component to interface peripheral equipment to the microcomputer system bus. It is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer:

It is a tri-state 8-bit buffer used to interface the chip to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer. The data lines are connected to BDB of p.

Read/Write and logic control:

The function of this block is to control the internal operation of the device and to control the transfer of data and control or status words. It accepts inputs from the CPU address and control buses and in turn issues command to both the control groups.

Chip Select: $\overline{CS}$

A low on this input selects the chip and enables the communication between the 8255 A& the CPU. It is connected to the output of address decode circuitry to select the device when it $\overline{RD}$ (Read). A low on this input enables the 8255 to send the data or status information to the CPU on the data bus.

(Write): $\overline{WR}$

A low on this input pin enables the CPU to write data or control words into the 8255

A_1, A_0 port select:

These input signals, in conjunction with the $\overline{RD}$ and $\overline{WR}$ inputs, control the selection of one of the three ports or the control word registers. They are normally connected to the least significant bits of the address bus (A_0 andA_1).

Following Table gives the basic operation,

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	Input operation
0	0	0	1	0	PORT A Data bus
0	1	0	1	0	PORT B Data bus
1	0	0	1	0	PORT C Data bus
0	0	1	0	0	Output operation Data bus PORTA
0	1	1	0	0	Data bus PORT B
1	0	1	0	0	Data bus PORT C
1	1	1	0	0	Data bus control

RESET:

A high on this input pin clears the control register and all ports (A, B & C) are initialized to input mode. This is connected to RESET OUT of 8255. This is done to prevent destruction of circuitry connected to port lines. If port lines are initialized as output after a power up or reset, the port might try to output into the output of a device connected to same inputs might destroy one or both of them.

PORTs A, B and C:

The 8255A contains three 8-bit ports (A, B and C). All can be configured in a variety of functional characteristic by the system software.

PORTA:

One 8-bit data output latch/buffer and one 8-bit data input latch.

PORT B:

One 8-bit data output latch/buffer and one 8-bit data input buffer.

PORT C:

One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input).

This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signals inputs in conjunction with ports A and B.

Group A & Group B control:

The functional configuration of each port is programmed by the system software. The control words outputted by the CPU configure the associated ports of the each of the two groups. Each control block accepts command from Read/Write content logic receives control words from the internal data bus and issues proper commands to its associated ports.

Control Group A – Port A& Port C upper Control Group B – Port B & Port C lower

The control word register can only be written into No read operation if the control word register is allowed.

Operation Description:

Mode selection:

There are three basic modes of operation that can be selected by the system software.

Mode 0: Basic Input/output

Mode 1: Strobes Input/output

Mode 2: Bi-direction bus

When the reset input goes HIGH all ports are set to mode'0' as input which means all 24 lines are in high impedance state and can be used as normal input. After the reset is removed the 8255A remains in the input mode with no additional initialization. During the execution of the program any of the other modes may be selected using a single output instruction.

The modes for PORT A& PORT B can be separately defined, while PORT C is divided into two portions as required by the PORT A and PORT B definitions. The ports are thus

divided into two groups Group A& Group B. All the output register, including the status flip-flop will be reset whenever the mode is changed. Modes of the two group may be combined for any desired I/O operation e.g. Group A in mode '1' and group B in mode'0'.

The basic mode definitions with bus interface and the mode definition format are given in fig (a) & (b),

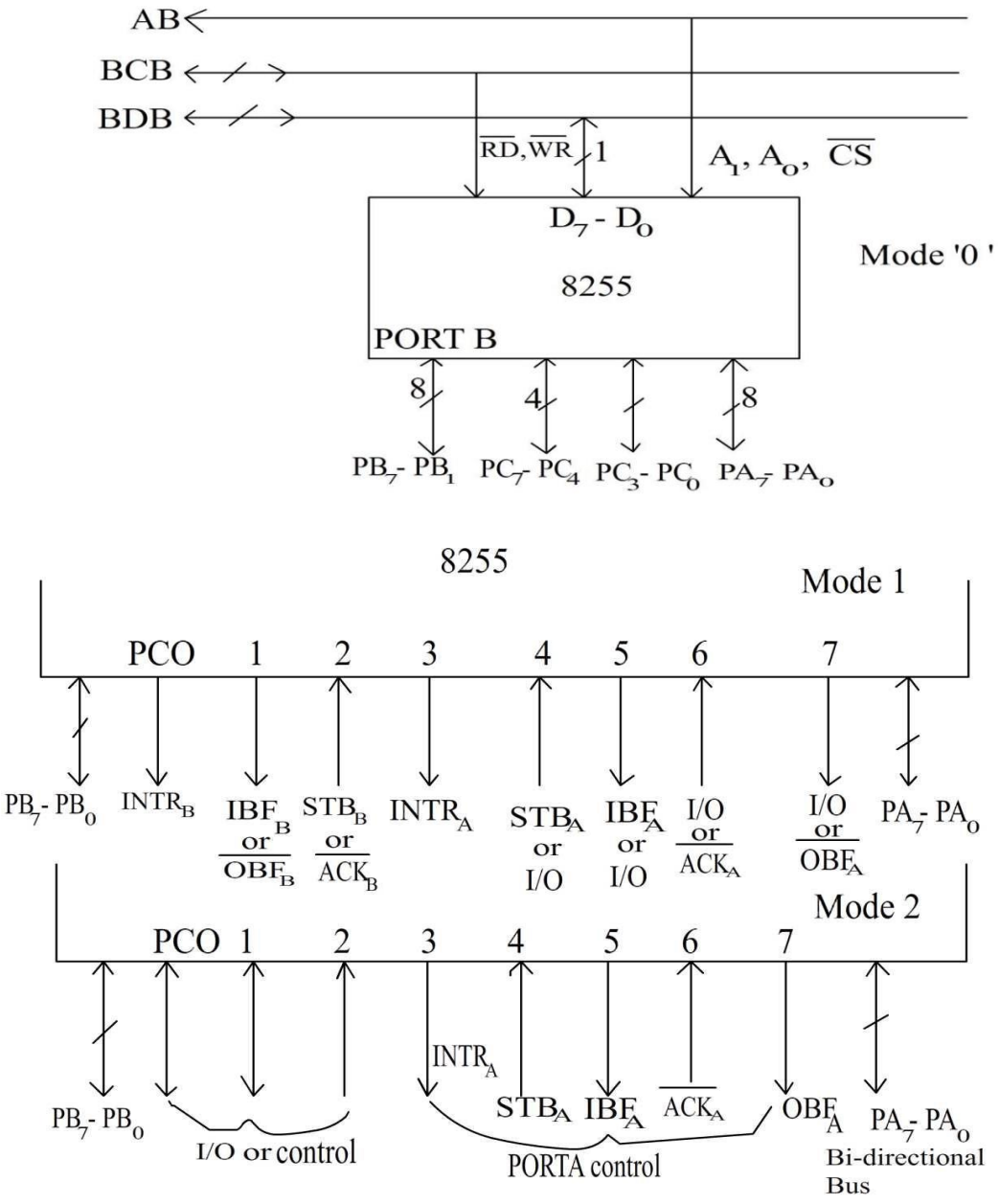


Fig 3 Mode operation of 8255

2.4 PROGRAMMABLE DMA CONTROLLER - INTEL 8257

- It is a device to transfer the data directly between IO device and memory without through the CPU. So it performs a high-speed data transfer between memory and I/O device.
- The features of 8257 is,

1. The 8257 has four channels and so it can be used to provide DMA to four I/O devices
2. Each channel can be independently programmable to transfer up to 64kb of data by DMA.
3. Each channel can be independently perform read transfer, write transfer and verify transfer.

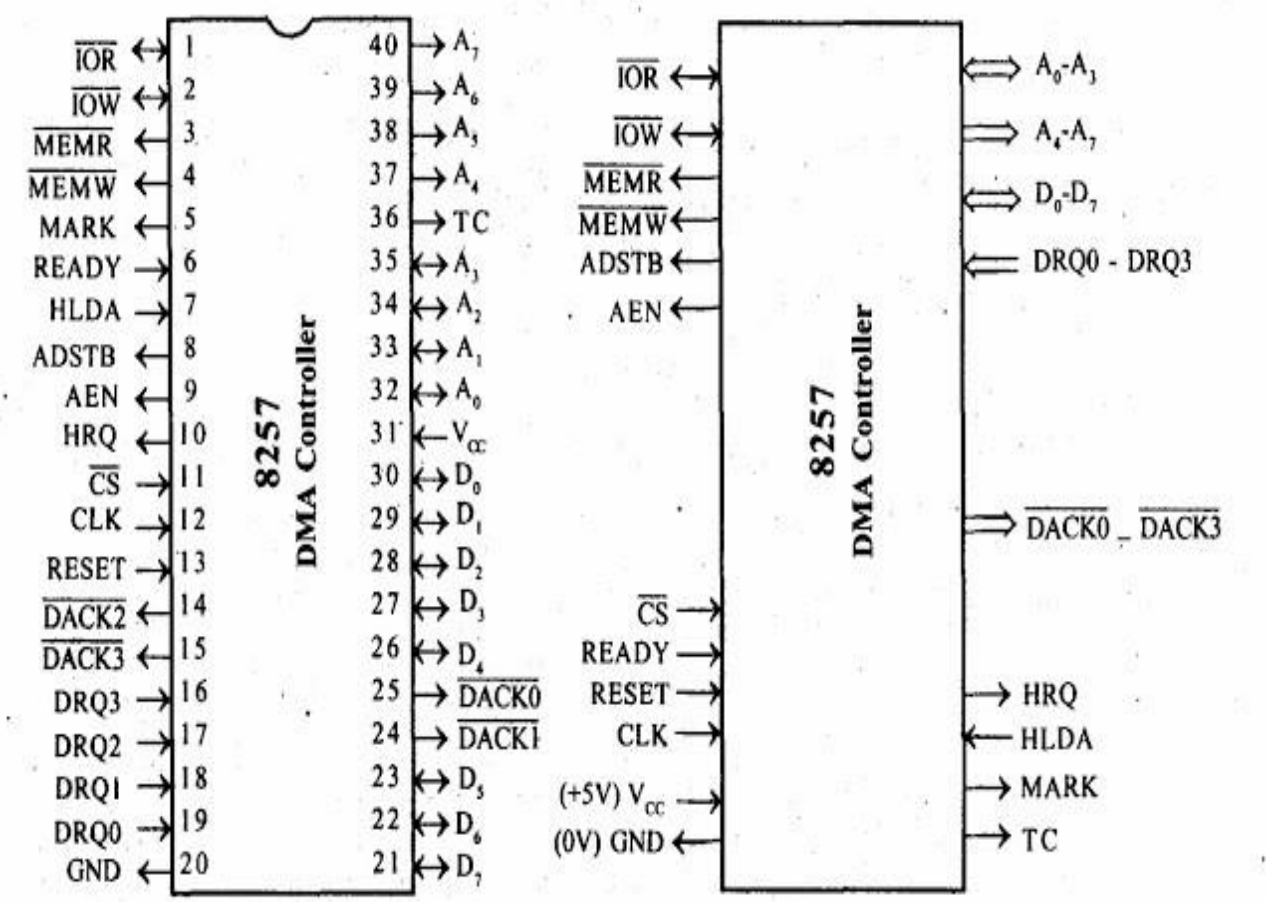


Fig 4 PIN Diagram of 8257

Functional Block Diagram of 8257:

- The functional block diagram of 8257 is shown in fig 5.

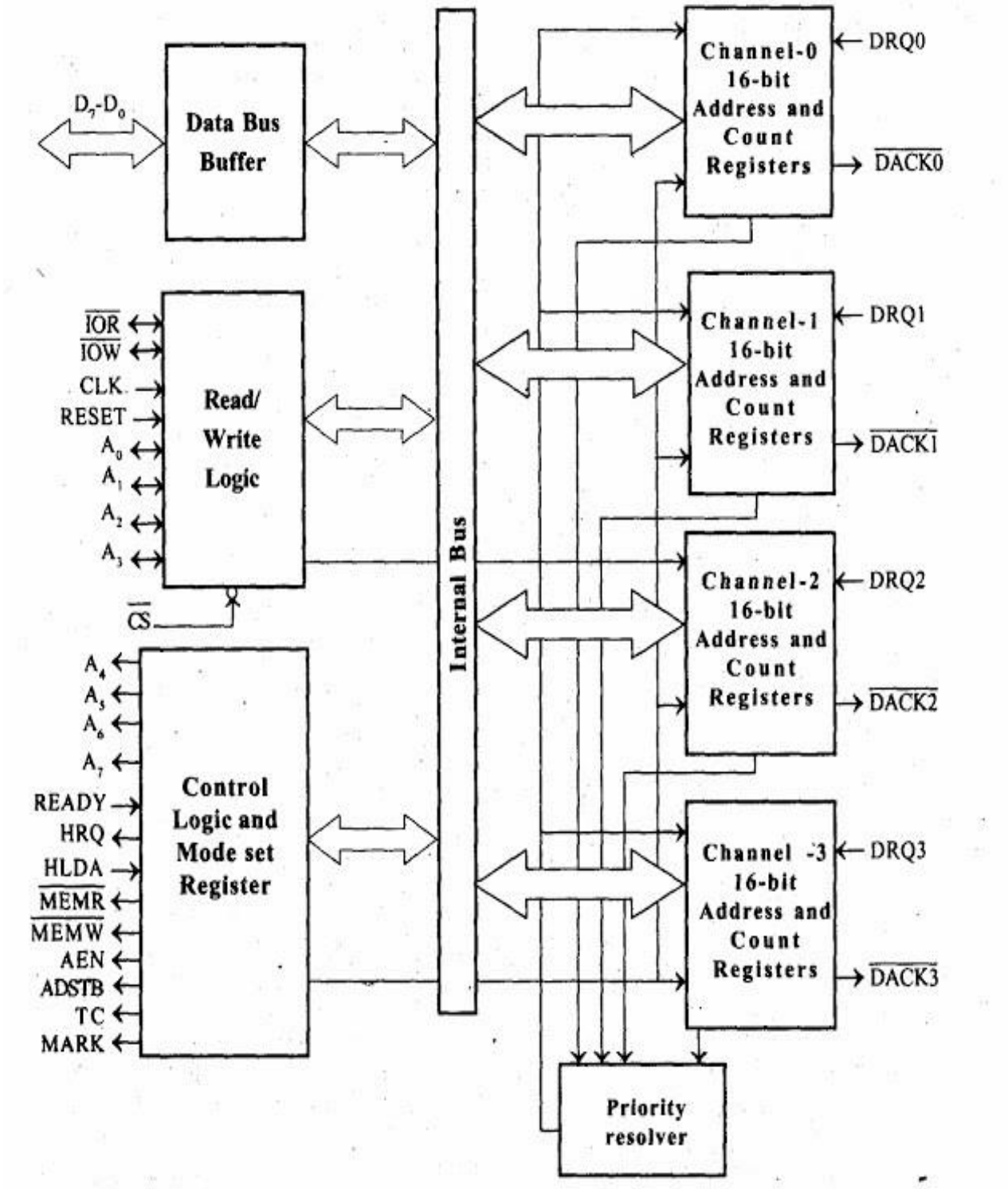


Fig 5 Block Diagram of 8257

- The functional blocks of 8257 are data bus buffer, read/write logic, control logic, priority resolver and four numbers of DMA channels.
- Each channel has two programmable 16-bit registers named as address register and count register
- Address register is used to store the starting address of memory location for DMA data transfer.
- The address in the address register is automatically incremented after every read/write/verify transfer.
- The count register is used to count the number of byte or word transferred by DMA
- The format of count register is,

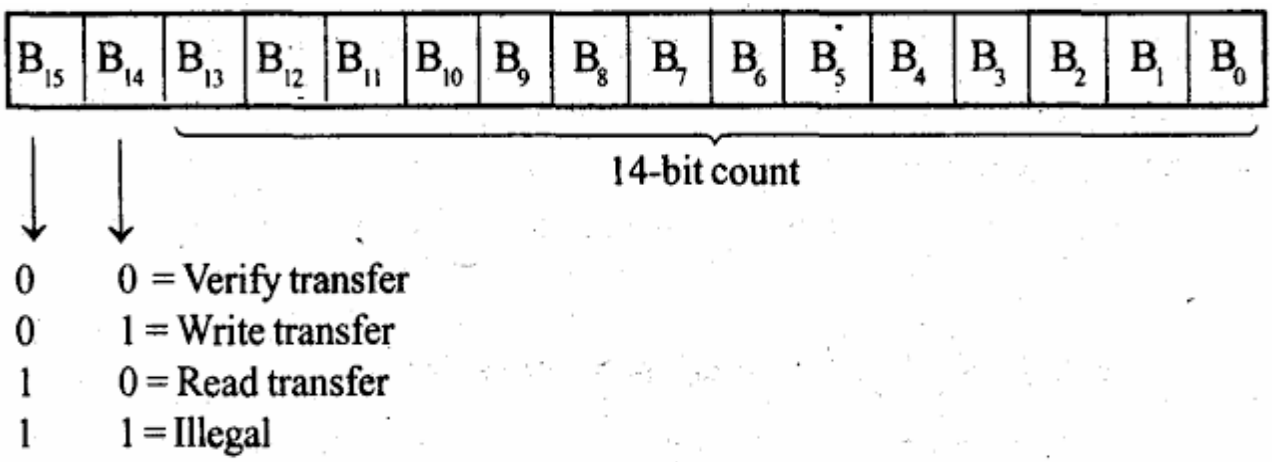


Fig 6 Count Register

- 14-bits B0-B13 is used to count value and a 2-bits is used for indicate the type of DMA transfer (Read/Write/Veri1 transfer).
- In read transfer the data is transferred from memory to I/O device.
- In write transfer the data is transferred from I/O device to memory.
- Verification operations generate the DMA addresses without generating the DMA memory and I/O control signals.
- The 8257 has two eight bit registers called mode set register and status register.
- The format of mode set register is,

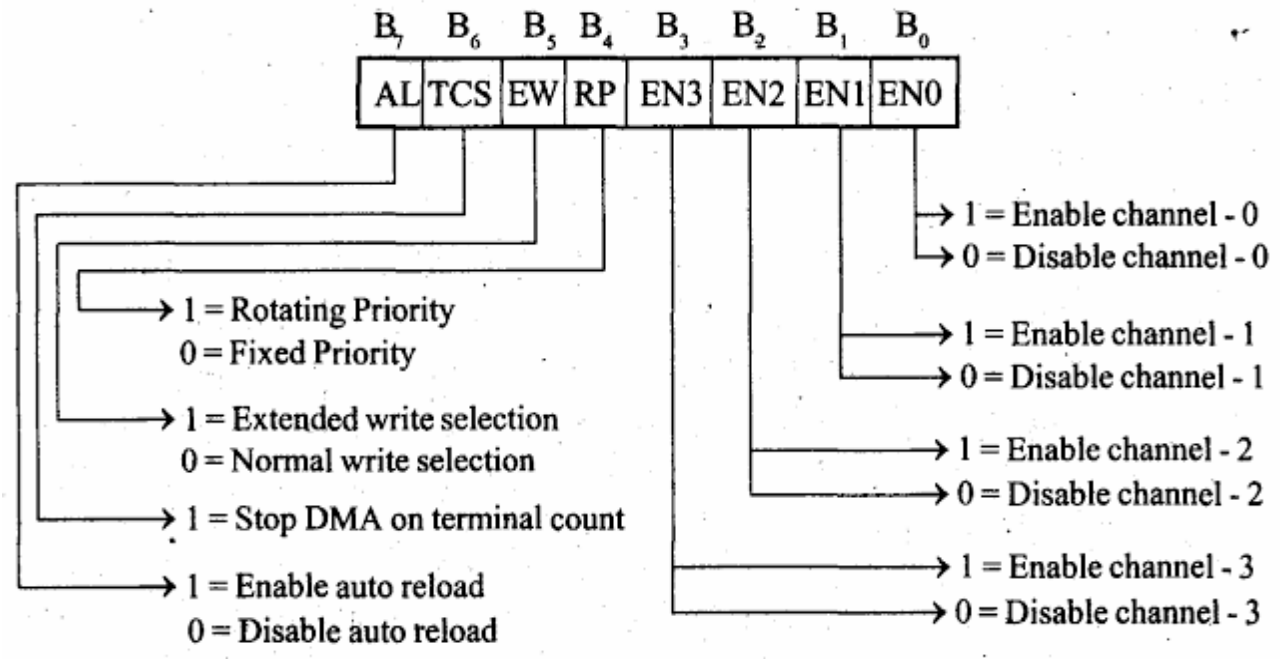


Fig 7 Format of mode set register

- The use of mode set register is,
  1. Enable/disable a channel.
  2. Fixed/rotating priority
  3. Stop DMA on terminal count.
  4. Extended/normal write time.
  5. Auto reloading ofchannel-2.
- The bits B0, B1, B2, and B3 of mode set register are used to enable/disable channel -0, 1,2 and 3 respectively. A one in these bit position will enable a particular channel and a zero will disable it
- If the bit B4 is set to one, then the channels will have rotating priority and if it zero then the channels wilt have fixed priority.

In rotating priority after servicing a channel its priority is made as lowest.
In fixed priority the channel-0 has highest priority and channel-2 has lowest priority.

- If the bit B5 is set to one, then the timing of low write signals (MEMW and IOW) will be extended.
- If the bit B6 is set to one then the DMA operation is stopped at the terminal count.
- The bit B7 is used to select the auto load feature for DMAchannel-2.

- When bit B7 is set to one, then the content of channel-3 count and address registers are loaded in channel-2 count and address registers respectively whenever the channel-2 reaches terminal count. When this mode is activated the number of channels available for DMA reduces from four to three.
- The format of status register of 8257 is shown in fig.

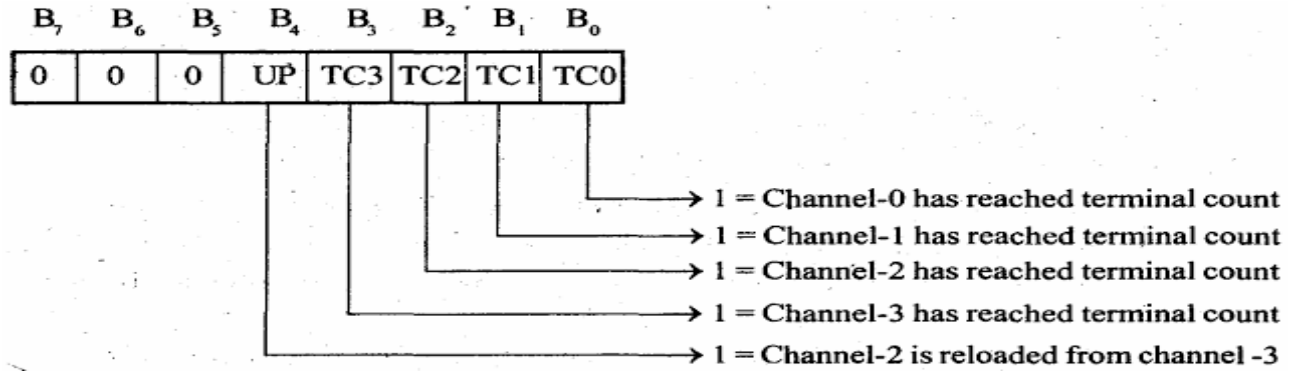


Fig 8 Format of Status register

- The bit B0, B1, B2, and B3 of status register indicates the terminal count status of channel-0, 1,2 and 3 respectively. A one in these bit positions indicates that the particular channel has reached terminal count.
- These status bits are cleared after a read operation by microprocessor.
- The bit B4 of status register is called update flag and a one in this bit position indicates that the channel-2 register has been reloaded from channel-3 registers in the auto load mode of operation.
- The internal addresses of the registers of 8257 are listed in table.

Register	Address			
	A_3	A_2	A_1	A_0
Channel-0 DMA address register	0	0	0	0
Channel-0 Count register	0	0	0	1
Channel-1 DMA address register	0	0	1	0
Channel-1 Count register	0	0	1	1
Channel-2 DMA address register	0	1	0	0
Channel-2 Count register	0	1	0	1
Channel-3 DMA address register	0	1	1	0
Channel-3 Count register	0	1	1	1
Mode set register (Write only)	1	0	0	0
Status register (Read only)	1	0	0	0

2.5 DATA COMMUNICATION

- Data communications refers to the ability of one computer to exchange data with another computer or a peripheral
- Standard data communication interfaces and standards are needed
- Centronic's parallel printer interface
- RS-232 defines a serial communications standard
- 8251 USART (Universal Synchronous/Asynchronous Receiver/Transmitter) is the key component for converting parallel data to serial form and vice versa

Two types of serial data communications are widely used

Asynchronous communications
Synchronous communications

Communication Modes

When data is transmitted between two piece of equipment, 3 modes of communication are used

- Simplex

Data is transmitted in one direction only

- Half Duplex

This is used when to devices wants information alternatively, but one after another

- Full Duplex

This is used when data is to be exchanged between two devices in both directions simultaneously

Introduction

- 8251 is a USART (Universal Synchronous Asynchronous Receiver Transmitter) for serial data communication.
- Programmable peripheral designed for synchronous /asynchronous serial data communication, packaged in a 28-pinDIP.
- Receives parallel data from the CPU &transmits serial data after conversion.
- Also receives serial data from the outside& transmits parallel data to the CPU after conversion.

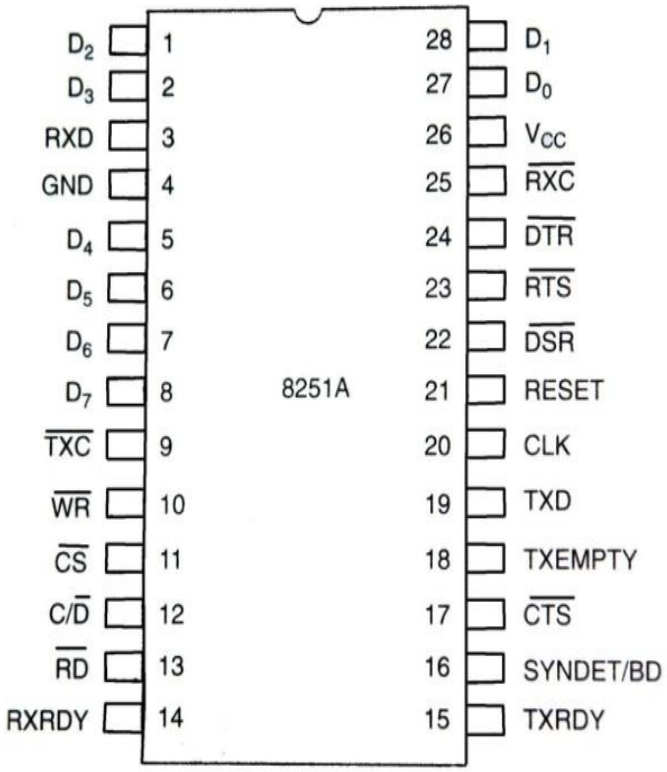


Fig 9 PIN Diagram of 8251

SIGNALS OF 8251:

CS – Chip Select : When signal goes low, the 8251A is selected by the MPU for communication.

C/D – Control/Data : When signal is high, the control or status register is addressed; when it is low, data buffer is addressed. (Control register & status register are differentiated by WR and RD signals)

WR : When signal is low, the MPU either writes in the control register or sends output to the data buffer.

RD : When signal goes low, the MPU either reads a status from the status register or accepts data from data buffer.

RESET : A high on this signal reset 8252A& forces it into the idle mode.

CLK : Clock input, usually connected to the system clock for communication with the microprocessor.

SECTION OF 8251:

- Data Bus buffer
- Read/Write Control Logic
- Modem Control
- Transmitter
- Receiver

Data Bus Buffer

D0-D7 : 8-bit data bus used to read or write status, command word or data from or to the8251A

Read/Write Control logic

Includes a control logic, six input signals & three buffer registers: Data register, control register & status register.

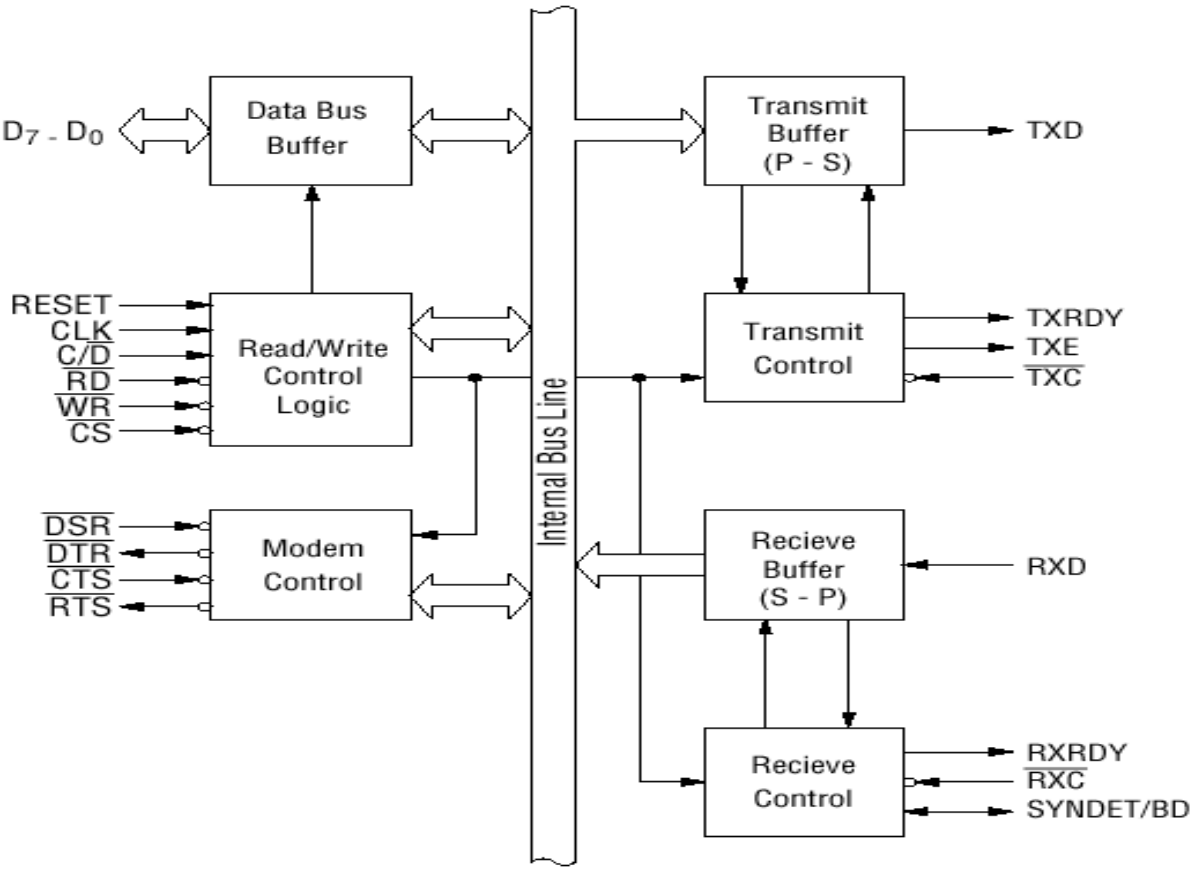


Fig 10 Block Diagram of 8251

Control logic : Interfaces the chip with MPU, determines the functions of the chip according to the control word in the control register & monitors the dataflow.

Modem Control

DSR - Data Set Ready : Checks if the Data Set is ready when communicating with a modem.

DTR - Data Terminal Ready : Indicates that the device is ready to accept data when the 8251 is communicating with a modem.

CTS - Clear to Send: If its low, the 8251A is enabled to transmit the serial data provided the enable bit in the command byte is set to'1'.

RTS - Request to Send Data : Low signal indicates the modem that the receiver is ready to receive a data byte from the modem.

Transmitter

Accepts parallel data from MPU & converts them into serial data. Has two registers

- Buffer register : To hold eight bits
- Output register :To convert eight bits into a stream of serial bits.

The MPU writes a byte in the buffer register.

Whenever the output register is empty; the contents of buffer register are transferred to output register.

Transmitter section consists of three output &one input signals

TxD - Transmitted Data Output : Output signal to transmit the data to peripherals

TxC- Transmitter Clock Input : Input signal, controls the rate of transmission.

TxRDY - Transmitter Ready : Output signal, indicates the buffer register is empty and the USART is ready to accept the next data byte.

TxE - Transmitter Empty : Output signal to indicate the output register is empty and the USART is ready to accept the next data byte.

Receiver Section

Accepts serial data on the RxD pin and converts them to parallel data.

Has two registers:

1. Receiver input register

Buffer register

When RxD goes low, the control logic assumes it is a start bit, waits for half bit time, and samples the line again. If the line is still low,the input register accepts the following data, and loads it into buffer register at the rate determined by the receiver clock.

RxRDY - Receiver Ready Output: Output signal, goes high when the USART has a character in the buffer register & is ready to transfer it to theMPU.

RxD - Receive Data Input : Bits are received serially on this line converted into a parallel byte in the receiver input register.

RxC - Receiver Clock Input : Clock signal that controls the rateat which bits are received by the USART.

Control Register

16-bit register for a control word consist of two independent bytes namely mode word & command word.

Mode word : Specifies the general characteristics of operation such as baud, parity, number of bits etc.

Command word : Enables the data transmission and reception.

Register can be accessed as an output port when the Control/Data pin is high.

Status register

Check the ready status of the peripheral.

Status word in the status register provides the information concerning register status and transmission errors.

Data register

Used as an input and output port when the C/D is low

CS	C/D	WR	RD	Operation
0	0	1	0	MPU reads data from data buffer MPU writes data
0	0	0	1	from data buffer MPU writes a word to control register
0	1	0	1	MPU reads a word from status register Chip is not
0	1	1	0	selected for any operation

8251 COMMAND WORD

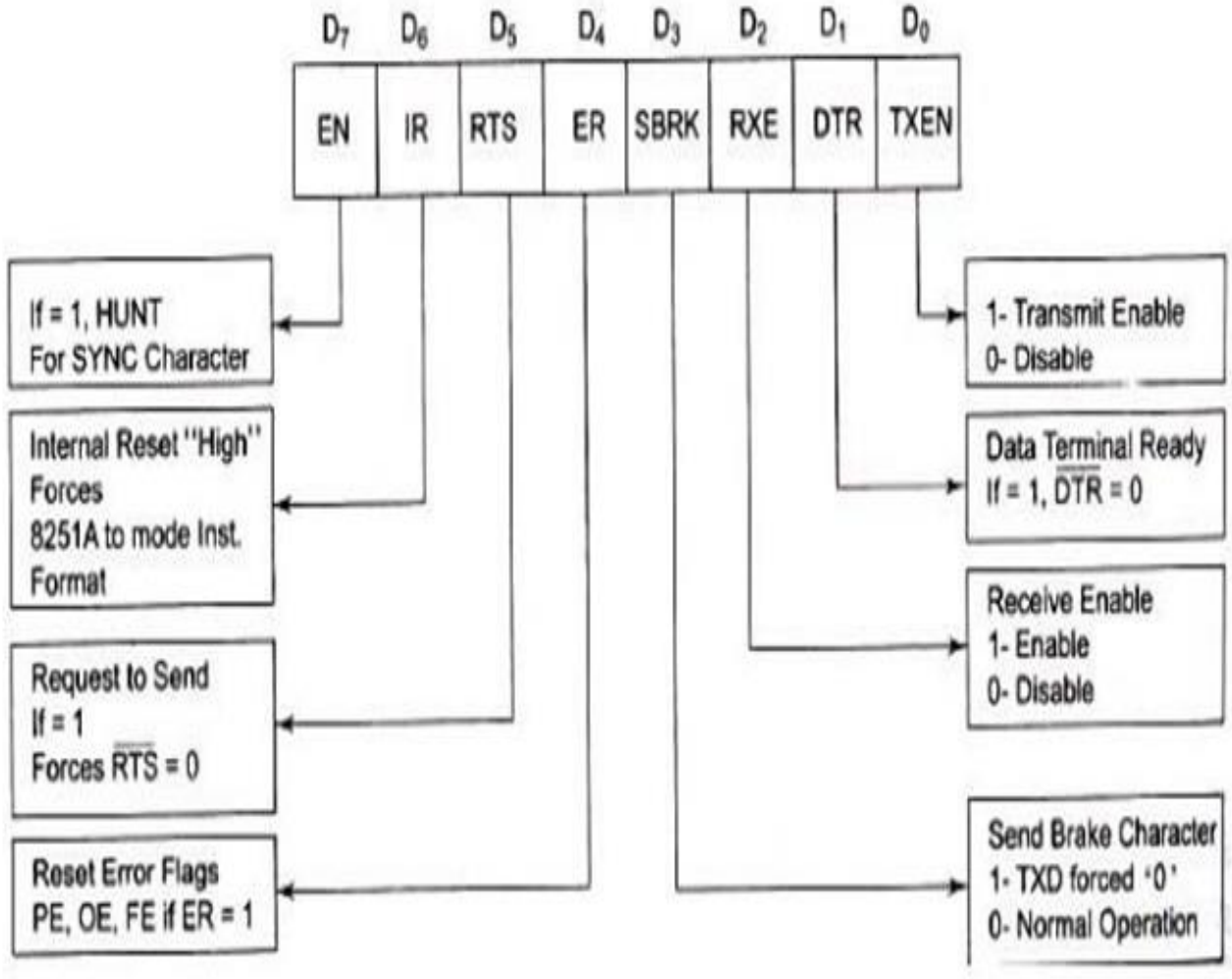


Fig 11 Format of Command Word

- Initializing the **TxEN** bit to 1 will enable the transmitter section of the 8251A and the **TxRDY** output.
- Initializing **DTR/** bit to 1 will cause the DTR/ output of the 8251A to be asserted low. This signal is used to tell a modem that a PC or terminal is operational.
- Initializing **RxE** bit to 1 will enable the RxRDY output of the 8251A.
- Initializing **SBRK** bit to 1 will cause the 8251A to output characters of 0's including start bits, data bits, and parity bits (**break character**). A break character is used to indicate the end of block of transmitted data.
- Initializing **ER** bit to 1 will cause the 8251A to reset the **parity**, **overrun**, and **framing** error flags in the 8251A status register.
- Initializing **RTS** bit to 1 will cause the 8251A to assert its **request-to-send** (**RTS/**) output low. This signal is sent to a modem to ask whether a modem and the receiving system are ready for a data character to be sent.
- Initializing **IR** bit to 1 will cause 8251A to be internally reset. After the software- reset command, a new mode word must be sent.
- Initializing **EH** bit to 1 will cause 8251A to enter hunt mode (search for **SYN** characters, and is used only in synchronous mode.

2.6 INTEL 8279 MICROPROCESSOR - KEYBOARD/DISPLAY CONTROLLER

The INTEL 8279 is specially developed for interfacing keyboard and display devices to 8085/8086/8088 microprocessor based system. The important features of 8279 are,

- Simultaneous keyboard and display operations.
- Scanned keyboard mode.
- Scanned sensor mode.
- 8-character keyboard FIFO.

- 1 6-characterdisplay.
- Right or left entry 1 6-byte display RAM.
- Programmable scan timing.

Pin Definition 8279

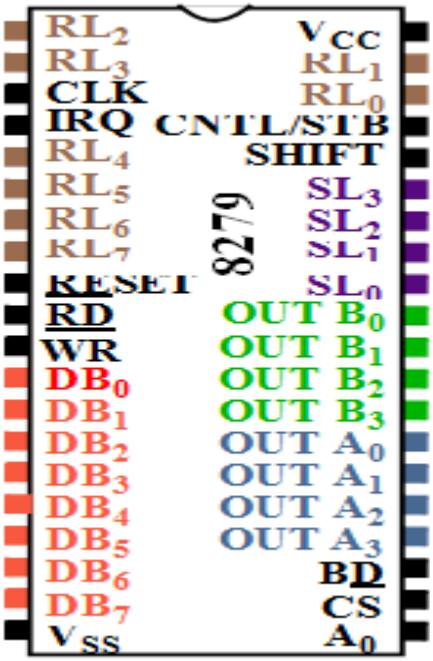


Fig 12 PIN Diagram of 8279

- A0: Selects data (0) or control/status (1) for reads and writes between micro and 8279.
- BD: Output that blanks the displays.
- CLK: Used internally for timing. Max is 3MHz.
- CN/ST: Control/strobe, connected to the control key on the keyboard
- CS: Chip select that enables programming, reading the keyboard, etc.
- DB_7-DB_0: Consists of bidirectional pins that connect to data bus on micro.
- IRQ: Interrupt request, becomes 1 when a key is pressed, data is available.
- OUT A_3-A_0/B_3-B_0: Outputs that sends data to the most significant/least significant nibble of display.
- RD(WR): Connects to micro's IORC or RD signal, reads data/status registers.
- RESET: Connects to system RESET.
- RL_7-RL_0: Return lines are inputs used to sense key depression in the keyboard matrix.

- Shift: Shift connects to Shift key on keyboard.
- SL_3-SL_0: Scan line outputs scan both the keyboard and displays.

Block diagram of 8279:

- The functional block diagram of 8279 is shown.

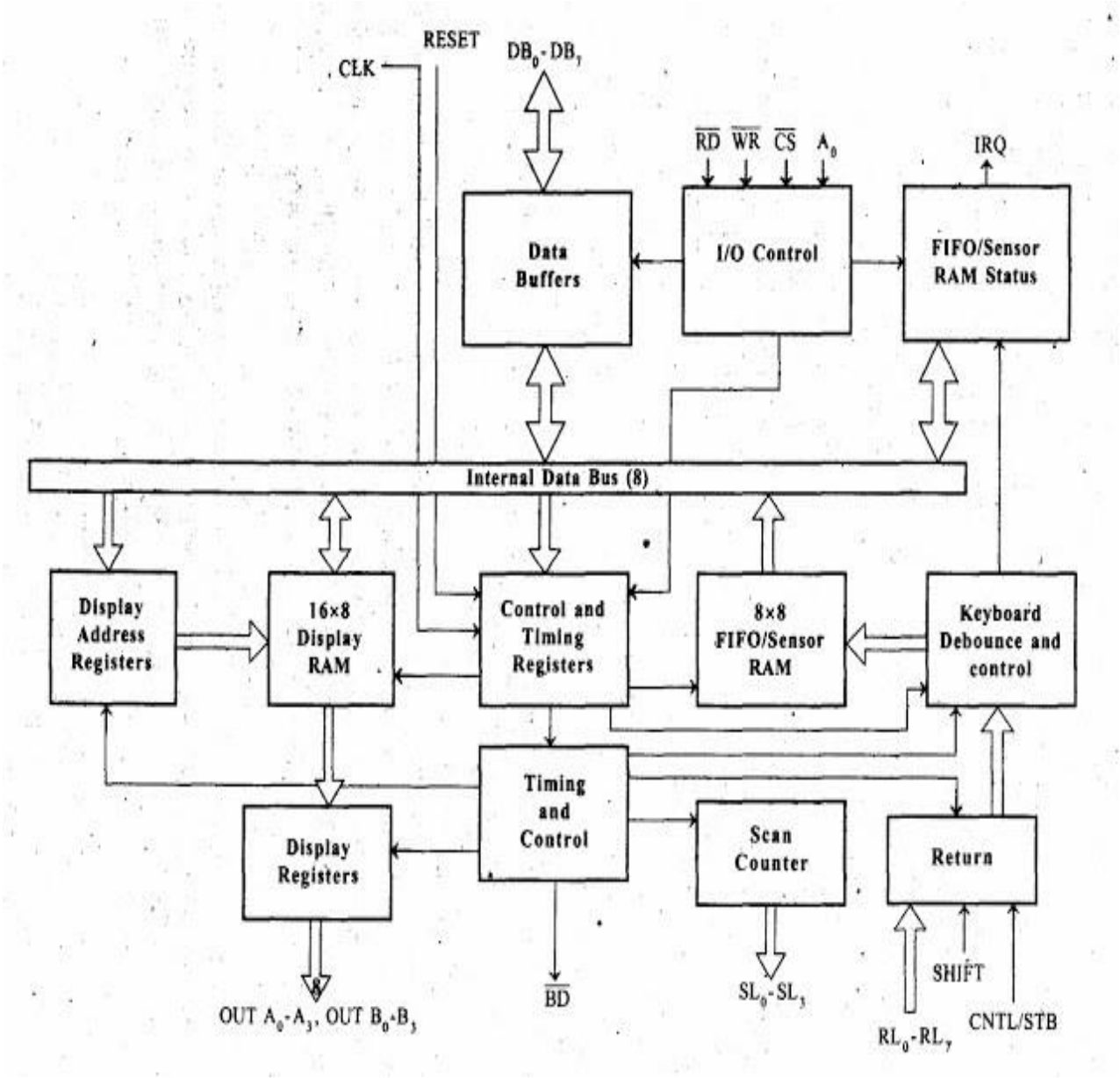
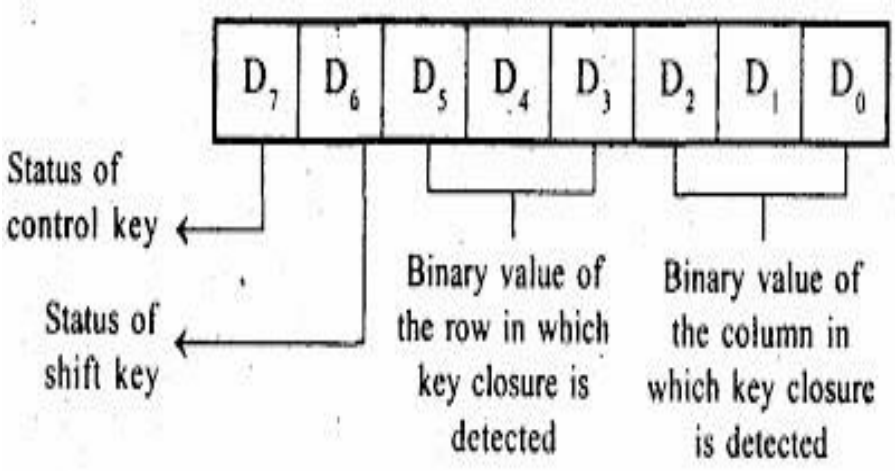


Fig 13 Block Diagram of 8279

- The four major sections of 8279 are keyboard, scan, display and CPU interface.

Keyboard section:

- The keyboard section consists of eight return lines RL0 – RL7 that can be used to form the columns of a keyboard matrix.
- It has two additional input : shift and control/strobe. The keys are automatically debounced.
- The two operating modes of keyboard section are 2-key lockout and N-key rollover.
- In the 2-key lockout mode, if two keys are pressed simultaneously, only the first key is recognized.
- In the N-key rollover mode simultaneous keys are recognized and their codes are stored in FIFO.
- The keyboard section also have an 8 x 8 FIFO (First In First Out) RAM.
- The FIFO can store eight key codes in the scan keyboard mode. The status of the shift key and control key are also stored along with key code. The 8279 generate an interrupt signal when there is an entry in FIFO. The format of key code entry in FIFO for scan keyboard mode is,



- In sensor matrix mode the condition (i.e., open/close status) of 64 switches is stored in FIFO RAM. If the condition of any of the switches changes then the 8279 asserts IRQ as high to interrupt the processor.

Display section:

- The display section has eight output lines divided into two groups A0-A3 andB0-B3.
- The output lines can be used either as a single group of eight lines or as two groups of four lines, in conjunction with the scan lines for a multiplexed display.

- The output lines are connected to the anodes through driver transistor in case of common cathode 7- segment LEDs.
- The cathodes are connected to scan lines through driver transistors.
- The display can be blanked by BD (low)line.
- The display section consists of 16 x 8 display RAM. The CPU can read from or write into any location of the display

RAM. Scan section:

- The scan section has a scan counter and four scan lines, SL0 toSL3.
- In decoded scan mode, the output of scan lines will be similar to a 2-to-4decoder.
- In encoded scan mode, the output of scan lines will be binary count, and so an external decoder should be used to convert the binary count to decoded output.
- The scan lines are common for keyboard and display.
- Thescanlinesareusedtoformtherowsofamatrixkeyboardandalsoconnectedtodigitdriversofa multiplexed display, to turn ON/OFF.

CPU interface section:

- The CPU interface section takes care of data transfer between 8279 and the processor.
- This section has eight bidirectional data lines DB0 to DB7 for data transfer between 8279 and CPU.
- It requires two internal address A =0 for selecting data buffer and A = 1 for selecting control registerof8279.
- The control signals WR (low), RD (low), CS (low) and A0 are used for read/write to8279.
- It has an interrupt request line IRQ, for interrupt driven data transfer with processor.
- The 8279 require an internal clock frequency of 100 kHz. This can be obtained by dividing the input clock by an internal prescaler.
- The RESET signal sets the 8279 in 16-character display with two -key lockout keyboard modes.

TEXT BOOK:

1. Douglas V.Hall, Microprocessor and Interfacing Programming and Hardware. Tata McGraw Hill, New Delhi 2007.

REFERENCES BOOK:

1. Krishna Kant, Microprocessor and Microcontroller Architecture, programming and system design using 8085, 8086, 8051 and 8096, PHI, New Delhi, 2008

QUESTION BANK

PART - B

1. With neat block diagram explain Programmable peripheral interface
2. Explain the serial communication interface 8251.
3. Explain the working of DMA controller with neat diagram.
4. Explain the functions of Programmable interrupt controller 8259A.
5. Two memory locations R1 and R2 store 07H and 3FH respectively. Exchange the values in these memory location using direct and indirect addressing.
6. Explain the various multiprocessor configurations.
7. Bring about the features of 8251.
8. Discuss how 8251 is used for serial communication of data.
9. With neat sketch explain the functions of 8251.
10. Types of data transfer IN 8251.
11. Explain the block of Programmable Peripheral Interface (8255) commenting about its comment and states words.
12. Explain the various modes of operation of Programmable Peripheral Interface (8255).
13. Draw the block diagram of 8279 and explain the function of each.
14. With the help of neat diagram explain how 8251 is interfaced with 8086 and used for serial communication.
15. Discuss the silent feature of 8259 and explain the block diagram of 8259-programmable interrupts controllers.
16. What is DMA? Explain the DMA based data transfer using DMA controller.
17. Explain the Mode 2 operations of 8255 Parallel communication interface and Draw the Block diagram of 8255 and explain the control word.
18. Discuss the features of programmable timer and explain its different modes of operation.

19. What can you say about I/O interface &explain in detail with diagrams.
20. (a)What do you think about traffic light control system? Explain with neat diagram. (b)Write an assembly language program for interfacing 8279 with 8086.
21. Draw the Block diagram and explain the operations of 8251 serial communication interface.
22. Explain the transmission and reception of serial data using 8251 indicating the function of various registers in it.
23. Draw the Block diagram and explain the operations of 8255 Parallel communication interface.
24. With a neat block diagram, explain the operation of 8259 programmable interrupt controller.
25. Draw the Block diagram of 8257 DMA controller and explain its operations.

UNIT IV

MICROCONTROLLER-8051

Microcontroller-8051

Register Set-Architecture of 8051 microcontroller- I/O and memory addressing-Interrupts-Instruction set-Addressing modes.

4.1 Register set 8051

There are 21 Special function registers (SFR) in 8051 micro controller and this includes Register A, Register B, Processor Status Word (PSW), PCON etc etc. There are 21 unique locations for these 21 special function registers and each of these register is of 1 byte size. Some of these special function registers are bit addressable (which means you can access 8 individual bits inside a single byte), while some others are only byte addressable. Let's take a look at them in detail.

Register A/Accumulator

The most important of all special function registers, that's the first comment about Accumulator which is also known as ACC or A. The Accumulator (sometimes referred to as Register A also) holds the result of most of arithmetic and logic operations. ACC is usually accessed by direct addressing and its physical address is E0H. Accumulator is both byte and bit addressable. You can understand this from the figure shown below. To access the first bit (i.e bit 0) or to access accumulator as a single byte (all 8 bits at once), you may use the same physical address E0H. Now if you want to access the second bit (i.e bit 1), you may use E1H and for third bit E2H and so on.

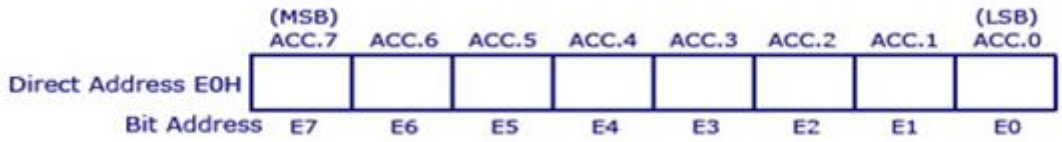


Fig 1 Accumulator Register A

Register B

The major purpose of this register is in executing multiplication and division. The 8051 micro controller has a single instruction for multiplication (MUL) and division (DIV). If you are familiar with 8085, you may now know that multiplication is repeated addition, whereas division is repeated subtraction. While programming 8085, you may have written a loop to execute repeated addition/subtraction to perform multiplication and division. Now here in 8051 you can do this with a single instruction.

Ex: MUL A,B – When this instruction is executed, data inside A and data inside B is multiplied and answer is stored in A.

Note: For MUL and DIV instructions, it is necessary that the two operands must be in A and B.

Register B is also byte addressable and bit addressable. To access bit o or to access all 8 bits (as a single byte), physical address F0 is used. To access bit 1 you may use F1 and so on. Please take a look at the picture below.

Note: Register B can also be used for other general purpose operations.

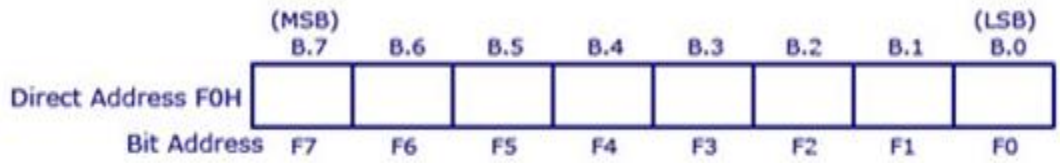


Fig 2 Register B

Port Registers

4 Input/ Output ports named P0, P1, P2 and P3 has got four corresponding port registers with same name P0, P1, P2 and P3. Data must be written into port registers first to send it out to any other external device through ports. Similarly any data received through ports must be read from port registers for performing any operation. All 4 port registers are bit as well as byte addressable. Take a look at the figure below for a better understanding of port registers.

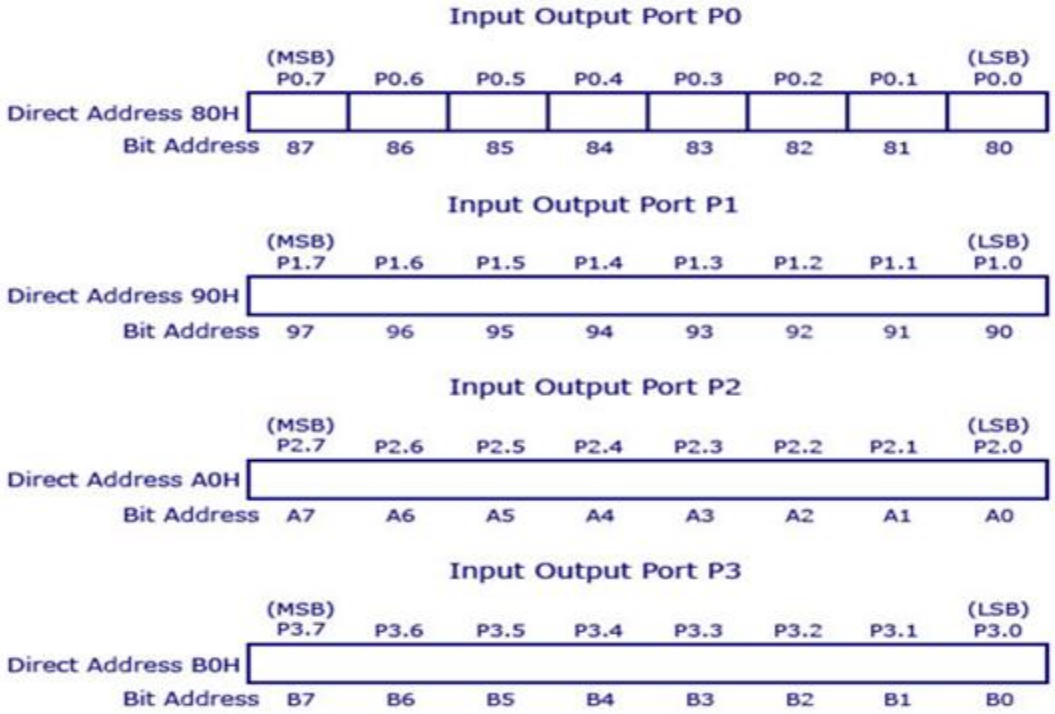


Fig 3 Port Registers

From the figure:-

- The physical address of port 0 is 80
- The physical address of port 1 is 90
- And that of port 2 is A0
- And that of port 3 is B0

Stack Pointer

stack pointer represents a pointer to the the system stack. Stack pointer is an 8 bit register, the direct address of SP is 81H and it is only byte addressable, which means you cant access individual bits of stack pointer. The content of the stack pointer points to the last stored location of system stack. To store something new in system stack, the SP must be incremented by 1 first and then execute the “store” command. Usually after

a system reset SP is initialized as 07H and data can be stored to stack from 08H onwards. This is usually a default case and programmer can alter values of SP to suit his needs.

Power Management Register (PCON)

As the name indicates, this register is used for efficient power management of 8051 micro controller. Commonly referred to as PCON register, this is a dedicated SFR for power management alone. From the figure below you can observe that there are 2 modes for this register :- Idle mode and Power down mode.

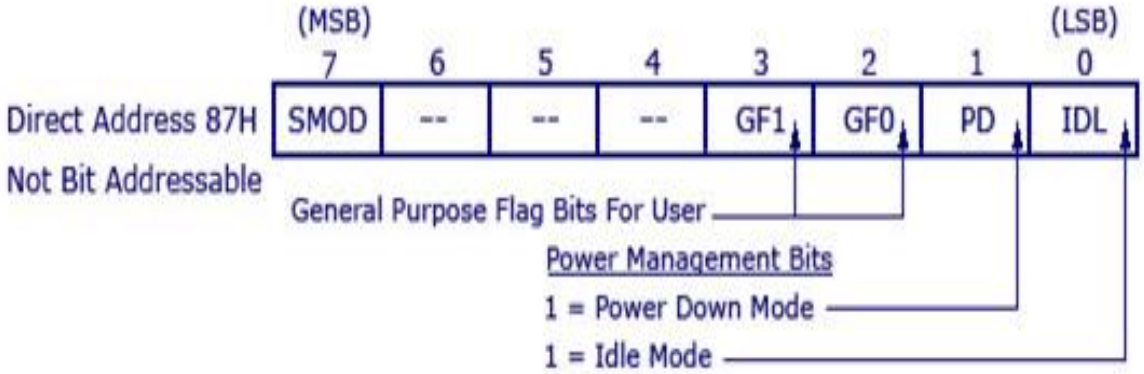


Fig 4 Register PCON

Setting bit 0 will move the micro controller to Idle mode and Setting bit 1 will move the micro controller to Power down mode.

Processor Status Word (PSW)

Commonly known as the PSW register, this is a vital SFR in the functioning of micro controller. This register reflects the status of the operation that is being carried out in the processor. The picture below shows PSW register and the way register banks are selected using PSW register bits – RS1 and RS0. PSW register is both bit and byte addressable. The physical address of PSW starts from D0H. The individual bits are then accessed using D1, D2 … D7. The various individual bits are explained below.

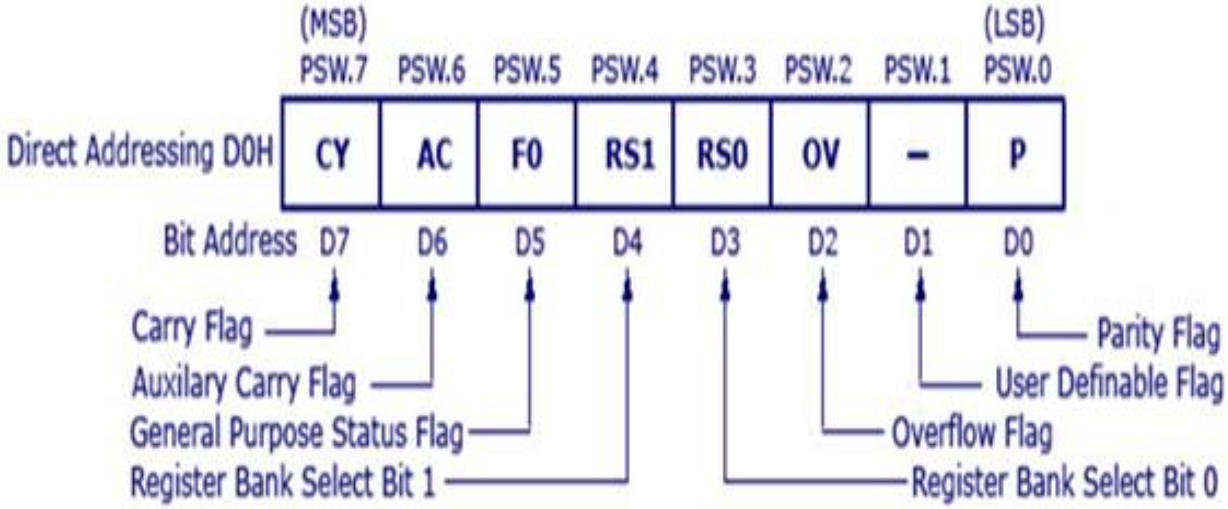


Fig 5 Processor status word

Bit No	Bit ymbol	Direct Address	Name	Function
0	P	D0	Parity	This bit will be set if ACC has odd number of 1's after an operation. If not, bit will remain cleared.
1	–	D1		User definable bit
2	OV	D2	Overflow	OV flag is set if there is a carry from bit 6 but not from bit 7 of an Arithmetic operation. It's also set if there is a carry from bit 7 (but not from bit 6) of Acc
3	RS0	D3	Register Bank select bit 0	LSB of the register bank select bit. Look for explanation below this table.
4	RS1	D4	Register Bank select bit 1	MSB of the register bank select bits.
5	F0	D5	Flag 0	User defined flag
6	AC	D6	Auxiliary carry	This bit is set if data is coming out from bit 3 to bit 4 of Acc during an Arithmetic operation.
7	CY	D7	Carry	s set if data is coming out of bit 7 of Acc during an Arithmetic operation.

At a time registers can take value from R0,R1…to R7. You may already know there are 32 such registers. So how you access 32 registers with just 8 variables to address registers? Here comes the use of register banks. There are 4 register banks named 0,1,2 and 3. Each bank has 8 registers named from R0 to R7. At a time only one register bank can be selected. Selection of register bank is made possible through PSW register bits PSW.3 and PSW.4, named as RS0 and RS1.These two bits are known as register bank select bits as they are used to select register banks. The picture will talk more about selecting register banks.

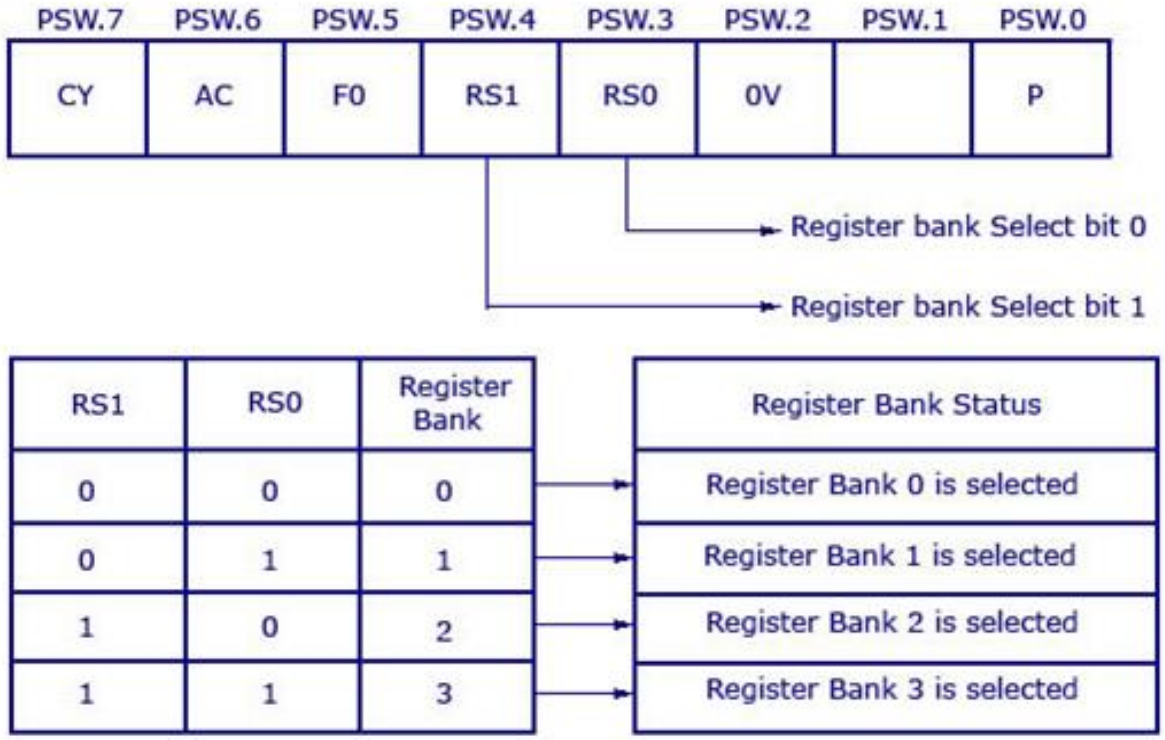


Fig 6 Processor Status Word

SFR	Address	Function
DPH	83	Data pointer registers (High). Only byte addressing possible.
DPL	82	Data pointer register (Low). Only byte addressing possible.
IP	B8	Interrupt priority. Both bit addressing and byte addressing possible.
IE	A8	Interrupt enable. Both bit addressing and byte addressing possible.
SBUF	99	Serial Input/Output buffer. Only byte addressing is possible.
SCON	98	Serial communication control. Both bit addressing and byte addressing possible.
TCON	88	Timer control. Both bit addressing and byte addressing possible.

TH0	8C	Timer 0 counter (High). Only byte addressing is possible.
TL0	8A	Timer 0 counter (Low). Only byte addressing is possible.
TH1	8D	Timer 1 counter (High). Only byte addressing is possible.
TL1	8B	Timer 1 counter (Low). Only byte addressing is possible.
TMOD	89	Timer mode select. Only byte addressing is possible.

4.2 ARCHITECTURE OF 8051 MICRO CONTROLLER

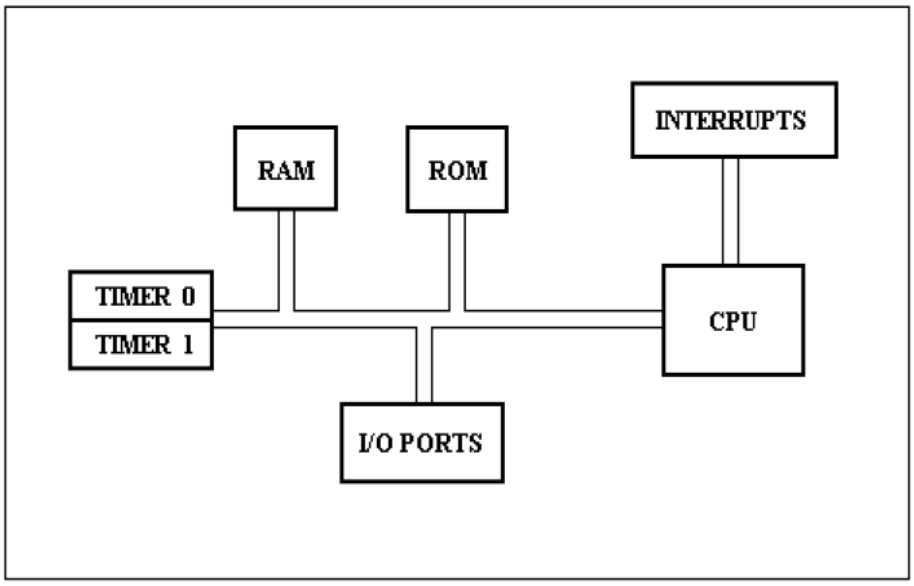


Fig 7 Architecture of 8051

Memory Organization

- Logical separation of program and data memory

-Separate address spaces for Program (ROM) and Data (RAM) Memory

-Allow Data Memory to be accessed by 8-bit addresses quickly and manipulated by 8-bit CPU

Program Memory

-Only be read, not written to

-The address space is 16-bit, so maximum of 64K bytes

-Up to 4K bytes can be on-chip (internal) of 8051 core

-PSEN (Program Store Enable) is used for access to external Program Memory

Data Memory

-Includes 128 bytes of on-chip Data Memory which are more easily accessible directly by its instructions

-There is also a number of Special Function Registers (SFRs)

-Internal Data Memory contains four banks of eight registers and a special 32- byte long segment which is bit addressable by 8051 bit-instructions

-External memory of maximum 64K bytes is accessible by "movx"

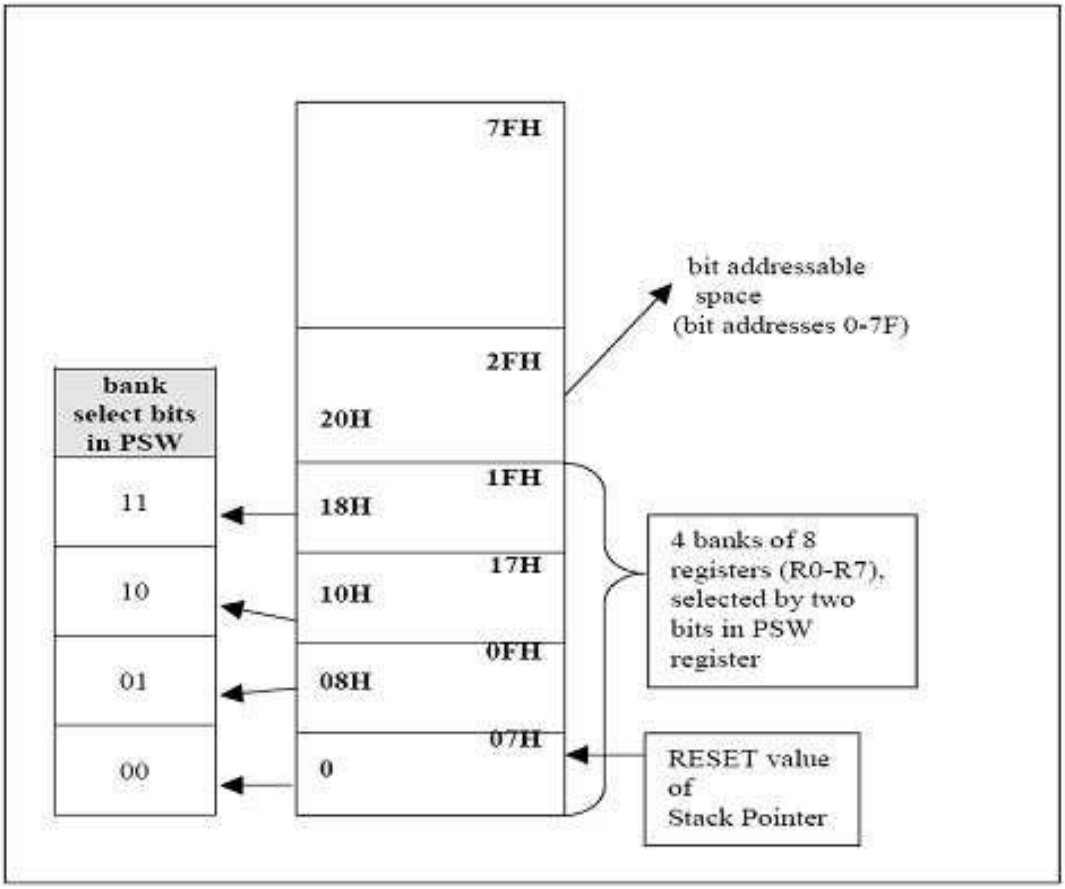


Fig 8 Internal Data Memory of 8051

Interrupt Structure

- The 8051 provides 4 interrupt sources
- Two external interrupts
- Two timer interrupts

Port Structure

- The 8051 contains four I/O ports
- All four ports are bidirectional
- Each port has SFR (Special Function Registers P0 through P3) which works like latch, an output driver and an input buffer
- Both output driver and input buffer of Port 0 and output driver of Port 2 are used for accessing external memory
- Accessing external memory works like this
- Port 0 outputs the low byte of external memory address (which is time- multiplexed with the byte being written or read)
- Port 2 outputs the high byte (only needed when the address is 16 bits wide)
- Port 3 pins are multifunctional
- The alternate functions are activated with the 1 written in the corresponding bit in the ports

Port Pin	Alternate Function
P3.2	~INT0 (external interrupt)
P3.3	~INT1 (external interrupt)
P3.4	T0 (Timer/Counter 0 external input)
P3.5	T1 (Timer/Counter 1 external input)
P3.6	~WR (external Data Memory writestrobe)
P3.7	~RD (external DataMemory readstrobe)

Timer/Counter

- The 8051 has two 16-bit Timer/Counter registers
- Timer0
- Timer1
- Both can work either as timers or event counters
- Both have four different operating modes

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 8051 instruction set is classified into the following three groups according to word size:

1. One-word or 1-byteinstructions
2. Two-word or 2-byteinstructions
3. Three-word or 3-byteinstructions

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.

These instructions are 1-byte instructions performing three different tasks. In the first instruction, both operand registers are specified. In the second instruction, the operand
B is specified and the accumulator is assumed. Similarly, in the third instruction, the accumulator is assumed to be the implicit operand. These instructions are stored in 8- bit binary format in memory; each requires one memory location.

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.

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.

4.3 I/O AND MEMORY ADDRESSING

4.3.1 I/O Ports

- The 8051 contains four I/O ports

- All four ports are bidirectional
- Each port has SFR (Special Function Registers P0 through P3) which works like a latch, an output driver and an input buffer
- Both output driver and input buffer of Port 0 and output driver of Port 2 are used for accessing external memory
- Accessing external memory works like this
- Port 0 outputs the low byte of external memory address (which is time- multiplexed with the byte being written or read)
- Port 2 outputs the high byte (only needed when the address is 16 bits wide)
- Port 3 pins are multifunctional
- The alternate functions are activated with the 1 written in the corresponding bit in the port SFR

Alternate Functions of Port 3 pins

Port Pin	Alternate Function
P3.2	~INT0 (external interrupt)
P3.3	~INT1 (external interrupt)
P3.4	T0 (Timer/Counter 0 external input)
P3.5	T1 (Timer/Counter 1 external input)
P3.6	~WR (external Data Memory writestrobe)
P3.7	~RD (external DataMemory readstrobe)

4.3.2 8051 Memory Organization

The 8051 microcontroller's memory is divided into Program Memory and Data Memory. Program Memory (ROM) is used for permanent saving program being executed, while Data Memory (RAM) is used for temporarily storing and keeping intermediate results and variables.

Program Memory (ROM)

Program Memory (ROM) is used for permanent saving program (CODE) being executed. The memory is read only. Depending on the settings made in compiler, program memory may also used to store a constant variables. The 8051 executes programs stored in program memory only. code memory type specifies is used to refer to program memory.8051 memory organization allows external program memory to be added. How does the microcontroller handle external memory depends on the pin EA logical state.

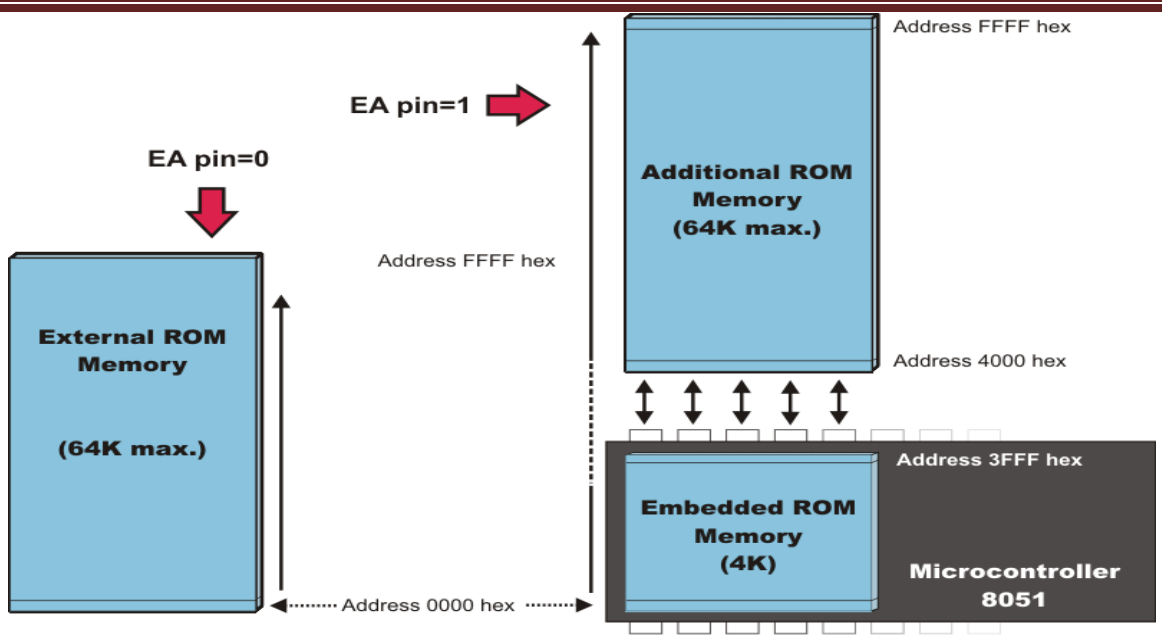


Fig 9 Memory organization

Internal Data Memory

Up to 256 bytes of internal data memory are available depending on the 8051 derivative. Locations available to the user occupy addressing space from 0 to 7Fh, i.e. first 128 registers and this part of RAM is divided in several blocks. The first 128 bytes of internal data memory are both directly and indirectly addressable. The upper 128 bytes of data memory (from 0x80 to 0xFF) can be addressed only indirectly.

Since internal data memory is used for CALL stack also and there is only 256 bytes splited over few different memory areas fine utilizing of this memory is crucial for fast and compact code. See types efficiency also. Memory block in the range of 20h to 2Fh is bit-addressable, which means that each bit being there has its own address from 0 to 7Fh. Since there are 16 such registers, this block contains in total of 128 bits with separate addresses (Bit 0 of byte 20h has the bit address 0, and bit 7 of byte 2Fh has the bit address 7Fh). Three memory type specifies can be used to refer to the internal data memory: **data**, **idata**, and **bdata**.

Fig 10 RAM Memory

External Data Memory

Access to external memory is slower than access to internal data memory. There may be up to 64K Bytes of external data memory. Several 8051 devices provide on-chip XRAM space that is accessed with the same instructions as the traditional external data space. This XRAM space is typically enabled via proper setting of SFR register and overlaps the external memory space. Setting of that register must be manually done in code, before any access to external memory or XRAM space is made.
The mikroC PRO for 8051 has two memory type specifies that refers to external memory space: **xdata** and **pdata**.

SFR Memory

The 8051 provides 128 bytes of memory for Special Function Registers (SFRs). SFRs are bit, byte, or word-sized registers that are used to control timers, counters, serial I/O, port I/O, and peripherals.

4.4 Interrupt Structure

- 8051 provides 4 interrupt sources
- 2 external interrupts
- 2 timer interrupts
- They are controlled via two SFRs, IE and IP
- Each interrupt source can be individually enabled or disabled by setting or clearing a bit in IE (Interrupt Enable). IE also exists a global disable bit, which can be cleared to disable all interrupts at once
- Each interrupt source can also be individually set to one of two priority levels by setting or clearing a bit in IP (Interrupt Priority)
- A low-priority interrupt can be interrupted by high-priority interrupt, but not by another low-priority one
- A high-priority interrupt can't be interrupted by any other interrupt source
- If interrupt requests of the same priority level are received simultaneously, an internal polling sequence determines which request is serviced, so within each priority lever there is a second priority structure
- This internal priority structure is determined by the polling sequence, shown in the following table

Source	Priority Within Level
IE0	highest
TF0	
IE1	
TF1	lowest

External Interrupt

■External interrupts ~INT0 and ~INT1 have two ways of activation

■Level-activated

■Transition-activated

■This depends on bits IT0 and IT1 inTCON

■The flags that actually generate these interrupts are bits IE0 and IE1 in TCON

■On-chip hardware clears that flag that generated an external interrupt when the service routine is vectored to, but only if the interrupt was transition-activated

■When the interrupt is level-activated, then the external requesting source is controlling the request flag, not the on-chip hardware

Handling Interrupt

■When interrupt occurs (or correctly, when the flag for an enabled interrupt is found to be set (1)), the interrupt system generates an LCALL to the appropriate location in Program Memory, unless some other conditions block the interrupt

■Several conditions can block an interrupt

■An interrupt of equal or higher priority level is already in progress

■The current (polling) cycle is not the final cycle in the execution of the instruction in progress

■The instruction in progress is RETI or any write to IE or IP registers

■If an interrupt flag is active but not being responded to for one of the above conditions, must be still active when the blocking condition is removed, or the denied interrupt will not be serviced

■Next step is saving the registers on stack. The hardware-generated LCALL causes only the contents of the Program Counter to be pushed onto the stack, and reloads the PC with the beginning address of the service routine

■In some cases it also clears the flag that generated the interrupt, and in other cases it doesn‟t. It clears an external interrupt flag (IE0 or IE1) only if it was transition- activated.

■Having only PC be automatically saved gives programmer more freedom to decide how much time to spend saving other registers. Programmer must also be more careful with proper selection, which register to

save.

■The service routine for each interrupt begins at a fixed location. The interrupt locations are spaced at 8-byte interval, beginning at 0003H for External Interrupt 0, 000BH for Timer 0, 0013H for External Interrupt 1 and 001BH for Timer1.

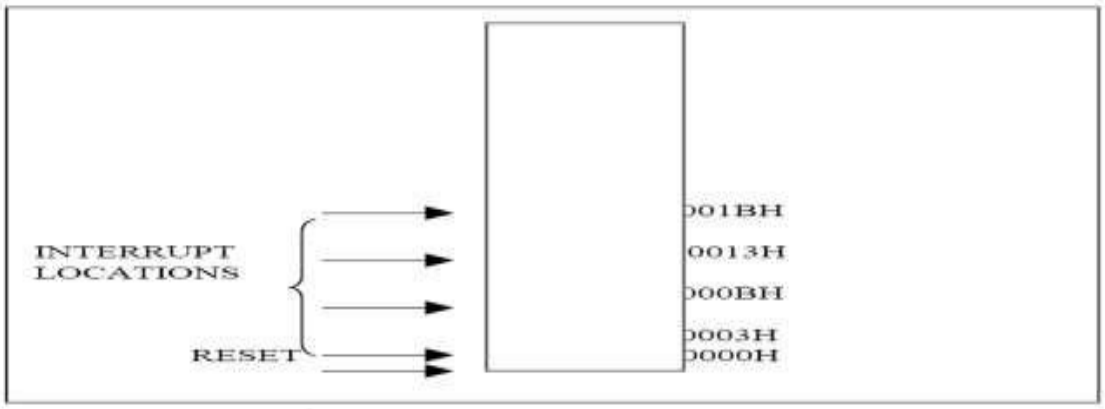


Fig 11 Interrupt Location in 8051 Program Memory

4.5 INSTRUCTION SET

Arithmetic Instructions

Mnemonic	Operation	Addressing modes	Execution time
ADD A, <byte>	A = A + <byte>	Dir, Ind, Reg, Imm	1
ADDC A, <byte>	A = A + <byte> + C	Dir, Ind, Reg, Imm	1
SUBB A, <byte>	A = A - <byte> - C	Dir, Ind, Reg, Imm	1
INC A	A = A + 1	Accumulator only	1
INC <byte>	<byte> = <byte> + 1	DPTR only	1
INC DPTR	DPTR = DPTR + 1	Dir, Ind, Reg	2
DEC A	A = A – 1	Accumulator only	1
DEC <byte>	<byte> = <byte> - 1	Dir, Ind, Reg	1
MUL AB	B:A=B x A	ACC and B only	4
DIV AB	A = Int [A/B] B = Mod [A/Bl	ACC and B only	4
DA A	Decimal Adjust	Accumulator only	1

Logical Instructions

Mnemonic	Operation	Addressing modes	Execution time
ANL A, <byte>	A = A .AND. <byte>	Dir, Ind, Reg, Imm	1
ANL A, <byte>	<byte> = <byte> .AND. A	Dir	1
ANL <byte>, #data	<byte> = <byte> .AND. #data	Dir	2
ORL A, <byte>	A = A .OR. <byte>	Dir, Ind, Reg, Imm	1
ORL A, <byte>	<byte> = <byte> .OR. A	Dir	1
ORL <byte>, #data	<byte> = <byte> .OR. #data	Dir	2
XRL A, <byte>	A = A .XOR. <byte>	Dir, Ind, Reg, Imm	1
XRL A, <byte>	<byte> = <byte> .XOR. A	Dir	1
XRL <byte>, #data	<byte> = <byte> .XOR. #data	Dir	2
CRL A	A = 00H	Accumulator only	1
CPL A	A = .NOT. A	Accumulator only	1
RL A	Rotate ACC Left 1 bit	Accumulator only	1
RLC A	Rotate Left through Carry	Accumulator only	1
RR A	Rotate ACC Right 1 bit	Accumulator only	1
RRC A	Rotate Right through Carry	Accumulator only	1
SWAP A	Swap Nibbles in A	Accumulator only	1

Data Transfer Instructions that access the Internal Data Memory

Mnemonic	Operation	Addressing modes	Execution time
MOV A, <src>	A = <src>	Dir, Ind, Reg, Imm	1
MOV <dest>, A	<dest> = A	Dir, Ind, Reg	1
MOV <dest>, <src>	<dest> = <src>	Dir, Ind, Reg, Imm	2
MOV DPTR, #data 16	DPTR = 16-bit immediate constant	Imm	2
PUSH <src>	INCSP: MOV "@'SP', <src>	Dir	2
POP <dest>	MOV <dest>, "@SP": DECSP	Dir	2
XCH A, <byte>	ACC and <byte> exchange data	Dir, Ind, Reg	1
XCHD A, @Ri	ACC and @Ri exchange low nibbles	Ind	1

Data Transfer Instructions that access the External Data Memory

Address width	Mnemonic	Operation	Execution time
8 bits	MOVX A, @Ri	Read external RAM @Ri	2
8 bits	MOVX @Ri, A	Write external RAM @Ri	2
16 bits	MOVX A, @DPTR	Read external RAM @DPTR	2
16 bits	MOVX @DPTR, A	Write external RAM @DPTR	2

Look up Tables

Mnemonic	Operation	Execution time
MOVC A, @A+DPTR	Read Program Memory at (A + DPTR)	2
MOVC A, @A+PC	Read Program Memory at (A + PC)	2

Boolean Instructions

Mnemonic	Operation	Execution time
ANL C, bit	C = C .AND. bit	2
ANL C, /bit	C = C .AND. .NOT. bit	2
ORL C, bit	C = C .OR. bit	2
ORL C, /bit	C = C .OR. .NOT. bit	2
MOV C, bit	C = bit	1
MOV bit, C	bit = C	2
CRL C	C = 1	1
CRL bit	bit = 0	1
SETB C	C = 1	1
SETB bit	bit = 1	1
CPL C	C = .NOT. C	1
CPL bit	bit = .NOT. bit	1
JC rel	Jump if C = 1	2
JNC rel	Jump if C = 0	2
JB bit, rel	Jump if bit = 1	2
JNB bit, rel	Jump if bit = 0	2
JBC bit, rel	Jump if bit = 1; CLR bit	2

Jump Instructions

Mnemonic	Operation	Execution time
ANL C, bit	C = C .AND. bit	2
ANL C, /bit	C = C .AND. .NOT. bit	2
ORL C, bit	C = C .OR. bit	2
ORL C, /bit	C = C .OR. .NOT. bit	2
MOV C, bit	C = bit	1
MOV bit, C	bit = C	2
CRL C	C = 1	1
CRL bit	bit = 0	1
SETB C	C = 1	1
SETB bit	bit = 1	1
CPL C	C = .NOT. C	1
CPL bit	bit = .NOT. bit	1
JC rel	Jump if C = 1	2
JNC rel	Jump if C = 0	2
JB bit, rel	Jump if bit = 1	2
JNB bit, rel	Jump if bit = 0	2
JBC bit, rel	Jump if bit = 1; CLR bit	2

4.6 Addressing Modes of 8051

The 8051 provides a total of five distinct addressing modes.

(1)immediate

(2)register

(3)direct

(4)register indirect

(5)indexed

(1) Immediate Addressing Mode

- The operand comes immediately after the op-code.
- The immediate data must be preceded by the pound sign,"#".

```
MOV A,#25H          ;load 25H into A
MOV R4,#62          ;load the decimal value 62 into R4
MOV B,#40H          ;load 40H into B
MOV DPTR,#4521H     ;DPTR=4512H
```

(2) Register Addressing Mode

- Register addressing mode involves the use of registers to hold the data tobe manipulated

```
MOV A,R0  ;copy the contents of R0 into A
MOV R2,A  ;copy the contents of A into R2
ADD A,R5  ;add the contents of R5 to contents of A
ADD A,R7  ;add the contents of R7 to contents of A
MOV R6,A  ;save accumulator in R6
```

(3) Direct Addressing Mode

- It is most often used to access RAM locations 30 - 7FH.
- This is due to the fact that register bank locations are accessed by the register names of R0 - R7.
- There is no such name for other RAM locations so must use direct addressing
- In the direct addressing mode, the data is in a RAM memory location whose address is known, and this address is given as a part of the instruction

```
MOV R0,40H    ;save content of RAM location 40H in R0
MOV 56H,A     ;save content of A in RAM location 56H
MOV R4,7FH    ;move contents of RAM location 7FH to R4
```

```
MOV A,4          ;is same as
MOV A,R4         ;which means copy R4 into A

MOV A,7          ;is same as
MOV A,R7         ;which means copy R7 into A
```

(4) *Register Indirect* Addressing Mode

- A register is used as a pointer to the data.
- If the data is inside the CPU, only registers R0 and R 1 are used for this purpose.
- R2 - R7 cannot be used to hold the address of an operand located in RAM when using indirect addressing mode.
- When RO and R 1 are used as pointers they must be preceded by the @sign.

```
MOV A,@R0  ;move contents of RAM location whose
           ;address is held by R0 into A
MOV @R1,B  ;move contents of B into RAM location
           ;whose address is held by R1
```

(5) Indexed Addressing Mode

- Indexed addressing mode is widely used in accessing data elements of look-up table entries located in the program ROM space of the8051.

The instruction used for this purpose is:

- MOVC A, @A+DPTR
- The 16-bit register DPTR and register A are used to form the address of the data element stored in on-chip ROM.
- Because the data elements are stored in the program (code) space ROM of the 8051, the instruction MOVC is used instead of MOV. The "C" means code.
- In this instruction the contents of A are added to the 16-bit register DPTR to form the 16- bit address of the needed data

TEXT BOOK:

1. Douglas V.Hall, Microprocessor and Interfacing Programming and Hardware. Tata McGraw Hill, New Delhi 2007.

REFERENCES BOOK:

1. Krishna Kant, Microprocessor and Microcontroller Architecture, programming and system design using 8085, 8086, 8051 and 8096, PHI, New Delhi, 2008
2. Kenneth J.Ayala, The 8051 Microcontroller, Thompson Delmar Learning, New Delhi, 2007

QUESTION BANK

PART B (16MARKS)

1. Explain the memory and I/O interface of 8051.

2. Discuss the peripheral interface of 8051.

3. Explain the memory structure of 8051.

4. Read and test P1 to see whether it has the value 45H. If its does, send 99H to P2. Otherwise, it stays cleared.

5. Explain the architecture of 8051 with its diagram.

6. Explain the I/O pins ports and circuit details of 8051 with its diagram.

7. Write an 8051ALP to create a square wave 66%duty cycle on bit3 of port 1.

8.With example explain the arithmetic and logic instruction of 8051 microcontroller.

9. With example explain the different instruction set of 8051 microcontroller.

10. Write a program based on 8051 instruction set to pack array of unpacked BCD digits.

11. Explain the different addressing modes of 8051.

12. Write a program to bring in data in serial form and send it out in parallel form using 8051.

13. Explain the data types and assembler directives of 8051.

14. Explain about the register banks and special function register of 8051 in detail.

15. List the addressing modes in 8051 and explain with suitable examples.

16. Describe the architecture of 8051 with neat diagram

17. Explain in detail about the Logical and Arithmetic group of 8051 instruction set with suitable examples.

18. Explain memory organization and I/O addressing in 8051.

19. Explain in detail about the Jump Instructions and Arithmetic group of 8051 instruction set with suitable examples.

20. Explain in detail about the Jump Instructions and Data Transfer of 8051 instruction set with suitable examples.

21. Explain in detail about the Look up Tables and Boolean Instructions of 8051 instruction set with suitable examples.

22. What is meant by register instructions addressing mode? Explain.

23. Differentiate between program memory and data memory.

24. Explain the internal hardware architecture of 8051 Microcontroller with neat diagram?

25. Explain in 8051 signals and timing in detail with neat diagram.

UNIT V

PROGRAMMING AND INTERFACING OF 8051

Programming And Interfacing Of 8051
Timer-Serial Communication-Interrupts Programming-Interfacing to External Memory-Interfacing to ADC, LCD, DAC, Keyboard and stepper motor.

5.1 Timers

The 8051 comes equipped with two timers, both of which may be controlled, set, read, and configured individually. The 8051 timers have three general functions: 1) Keeping time and/or calculating the amount of time between events, 2) Counting the events themselves, or 3) Generating baud rates for the serial port.

one of the primary uses of timers is to measure time. We will discuss this use of timers first and will subsequently discuss the use of timers to count events. When a timer is used to measure time it is also called an "interval timer" since it is measuring the time of the interval between two events.

Timer SFR

8051 has two timers which each function essentially the same way. One timer is TIMER0 and the other is TIMER1. The two timers share two SFRs (TMOD and TCON) which control the timers, and each timer also has two SFRs dedicated solely to itself (TH0/TL0 and TH1/TL1).

SFR Name	Description	SFR Address
TH0	Timer 0 High Byte	8Ch
TL0	Timer 0 Low Byte	8Ah
TH1	Timer 1 High Byte	8Dh
TL1	Timer 1 Low Byte	8Bh
TCON	Timer Control	88h
TMOD	Timer Mode	89h

13-bit Time Mode (mode 0)

Timer mode "0" is a 13-bit timer. This is a relic that was kept around in the 8051 to maintain compatibility with its predecessor, the 8048. Generally the 13-bit timer mode is not used in new development.

When the timer is in 13-bit mode, TLx will count from 0 to 31. When TLx is incremented from 31, it will "reset" to 0 and increment THx. Thus, effectively, only 13 bits of the two timer bytes are being used: bits 0-4 of TLx and bits 0-7 of THx. This also means, in essence, the timer can only contain 8192 values. If you set a 13-bit timer to 0, it will overflow back to zero 8192 machine cycles later.

Again, there is very little reason to use this mode and it is only mentioned so you wont be surprised if you ever end up analyzing archaeic code which has been passed down through the generations (a generation in a programming shop is often on the order of about 3 or 4 months).

16-bit Time Mode (mode 1)

Timer mode "1" is a 16-bit timer. This is a very commonly used mode. It functions just like 13-bit mode except that all 16 bits are used.

TLx is incremented from 0 to 255. When TLx is incremented from 255, it resets to 0 and causes THx to be incremented by 1. Since this is a full 16-bit timer, the timer may contain up to 65536 distinct values. If you set a 16-bit timer to 0, it will overflow back to 0 after 65,536 machine cycles.

8-bit Time Mode (mode 2)

Timer mode "2" is an 8-bit auto-reload mode. What is that, you may ask? Simple. When a timer is in mode 2, THx holds the "reload value" and TLx is the timer itself. Thus, TLx starts counting up. When TLx reaches 255 and is subsequently incremented, instead of resetting to 0 (as in the case of modes 0 and 1), it will be reset to the value stored inTHx.

Split Timer Mode (mode3)

Timer mode "3" is a split-timer mode. When Timer 0 is placed in mode 3, it essentially becomes two separate 8-bit timers. That is to say, Timer 0 is TL0 and Timer 1 is TH0. Both timers count from 0 to 255 and overflow back to 0. All the bits that are related to Timer 1 will now be tied to TH0.

While Timer 0 is in split mode, the real Timer 1 (i.e. TH1 and TL1) can be put into modes 0, 1 or 2 normally--however, you may not start or stop the real timer 1 since the bits that do that are now linked to TH0. The real timer 1, in this case, will be incremented every machine cycle no matter what.

USING TIMERS AS EVENTCOUNTERS

We've discussed how a timer can be used for the obvious purpose of keeping track of time. However, the 8051 also allows us to use the timers to count events.

How can this be useful? Let's say you had a sensor placed across a road that would send a pulse every time a car passed over it. This could be used to determine the volume of traffic on the road. We could attach this sensor to one of the 8051's I/O lines and constantly monitor it, detecting when it pulsed high and then incrementing our counter when it went back to a low state. This is not terribly difficult, but requires some code. Let's say we hooked the sensor to P1.0; the code to count cars passing would look something like this:

```
JNBP1.0,$     ;If a car hasn't raised the signal, keep waiting
JBP1.0,$      ;The line is high which means the car is on the sensor right now INC COUNTER ;The car has passed completely, so we count it
```

5.2 SERIAL DATA COMMUNICATION IN 8051 MICROCONTROLLER

- The fastest way of transmitting data, within a microcomputer is parallel data transfer.
- For transferring data over long distances, however, parallel data transmission requires too many wires.
- For long distance transmission, data is usually converted from parallel form to serial form so that it can be sent on a single wire or pair of wires.
- Serial data received from a distant source is converted to parallel form and it can be easily transferred on the microcomputer buses.
- The types of communication systems are,

1. Simplex 2.Half-duplex 3.Full-duplex

- In simplex communication, data can be transmitted only in one direction, ie. data from sensors to processor. Eg : commercial radio stations.
- In half-duplex transmission, data can be transmitted in either direction between two systems, but can occur only in one direction at a time. Eg : two-way radio system, where one user always listens while the other talks because the receiver circuitry is turned off during transmit.
- In full duplex, the data can be send and received at the same time. Eg : A normal phone conversation.
- Serial data can be sent by two ways. They are,
  1. Synchronous communication
  2. Asynchronous communication

- In synchronous transmission, data are transmitted in block at a constant rate. The start and end of a block are identified with specific bytes or bit patterns.
- In asynchronous transmission, data is transmitted one by one.
- The beginning of a data character is indicated by the line going low for 1 bit time. This bit is called a start bit.
- The data bits are then sent out on the line one after the other. Note that the least- significant bit is sent out first. Depending on the system, the data word may consist of 5, 6, 7 or 8 bits.
- Following the data bits is a parity bit, which is used to check for errors in received data.
- The line is returned high for at least 1-bit time to identify the end of the character. This always-high bit is referred to as a stop bit. Some systems may use 2 stop bits.
- The bit format for asynchronous data transmission is,

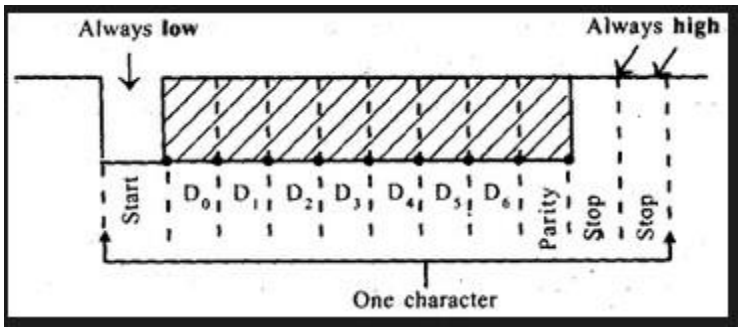


Fig 1 Bit Format for asynchronous transmission

- The term baud rate is used to indicate the rate at which serial data is being transferred.

Baud rate = 1/ time for a bit cell

- A device such as INTEL 8251A, which can be programmed to do either asynchronous or synchronous communication, is often called USART(Universal Synchronous Asynchronous Receiver Transmitter).
- A device such as the National 1NS8250, which can only do asynchronous communication, is often referred to as a Universal Asynchronous Receiver Transmitter(UART).
- For sending serial data over long distances the standard telephone system is a convenient path, because the wiring and connections are already in place.
- Standard phone lines often referred to as switched lines because any two points can be connected

together through a series of switches and have a bandwidth of about 300 to 3000Hz.

- But digital signals require very large bandwidth (typically 5 MHz). Therefore, digital signals cannot be sent directly over standard phone lines. So, the digital signals are converted to audio-frequency tones, which are in the frequency range.

- The device used to do this conversion and to convert transmitted tones back to digital information is called a MODEM.

- Modems and other equipment used to send serial data over long distances are known as data communication equipment or DCE. The terminals and computers that are sending or receiving the serial data are referred to as data terminal equipment or DTE.

RS-232C serial data standard:

- In serial I/O, data can be transmitted as either current or voltage.

- When data are transmitted as voltage, the commonly used standard is known as RS-232C.

- This standard, proposes a maximum of 25 signals for the bus used for serial data transfer.

- The 25 signals of RS-232Care,

Pin Number	Common Name	RS-232C Name	Description	Signal Direction On DCE
1	-	AA	Protective Ground	-
2	TxD	BA	Transmitted Data	IN
3	RxD	BB	Received Data	OUT
4	$\overline{RTS}$	CA	Request to send	IN
5	$\overline{CTS}$	CB	Clear to send	OUT
6	$\overline{DSR}$	CC	Data Set ready	OUT
7	GND	AB	Signal ground (Common return)	-
8	$\overline{CD}$	CF	Received line signal detector	OUT
9		-	Reserved for Data set testing	-
10		-	Reserved for Data set testing	-
11		-	Unassigned	-
12		SCF	Secondary Received Line signal Detector	OUT
13		SCB	Secondary clear to send	OUT
14		SBA	Secondary Transmitted Data	IN
15		DB	Transmission signal element timing (DCE source)	OUT
16		SBF	Secondary Received data	OUT
17		DD	Receiver signal element timing (DCE source)	OUT
18		-	Unassigned	-
19		SCA	Secondary request to send	IN
20	$\overline{DTR}$	CD	Data terminal ready	IN
21		CG	Signal quality detector	OUT
22		CE	Ring indicator	OUT
23		CH/CI	Data signal rate selector (DTE/DCE Source)	IN/OUT
24		DA	Transmit signal element timing (DTE source)	IN
25			Unassigned	

- In practice the first 9-signals are sufficient for most of the serial data transmission scheme and so the RS-232C bus signals are terminated on a D-type 9- pin connector.

- When all the 25 signals are used, then RS-232C serial bus is terminated ona 25-pin connector.

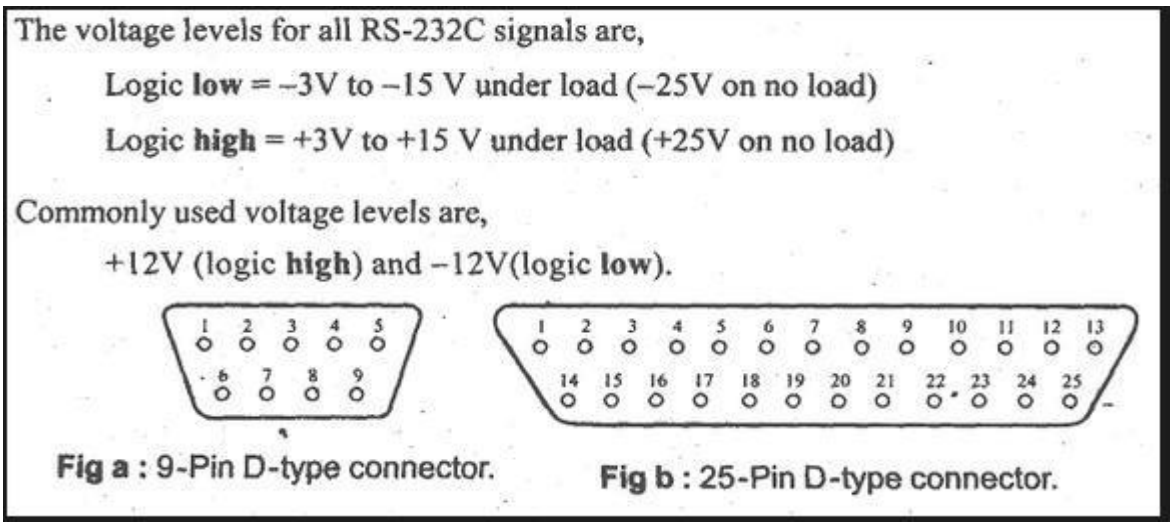


Fig 2 PIN D Connector

- The RS-232C signal levels are not compatible with TTL logic levels. Hence for interfacing TTL devices, level converters or RS-232C line drivers are employed.

- The popularly used level converters are,

1. MC1488 - TTL to RS-232C level converter.

2. MC1489 - RS-232C to TTL level converter.

3. MAX 232 - Bidirectional level converter.(Max 232 is equivalent to a combination of MC 1488 and MCI 489 in single IC)

- The pin diagram of MAX 232is,

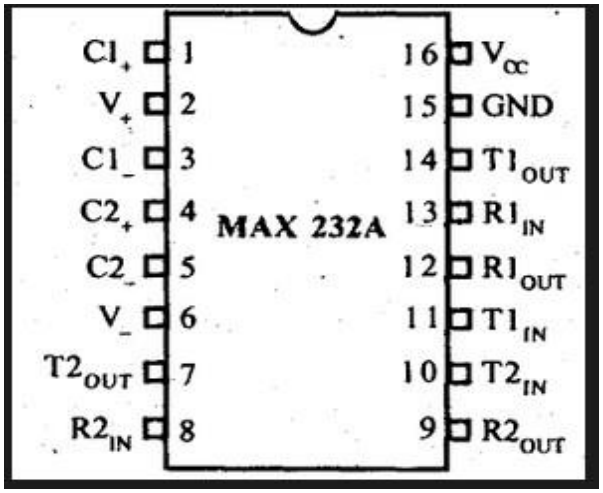


Fig 3 PIN Diagram of MAX 232

- For MAX 232 all capacitor should be1µF.
- The voltage rating of all capacitor should above10V.

5.5 INTERFACING TO ADC, LCD, DAC, KEYBOARD AND STEPPER MOTOR.

5.5.1 ADC0808/ADC0809

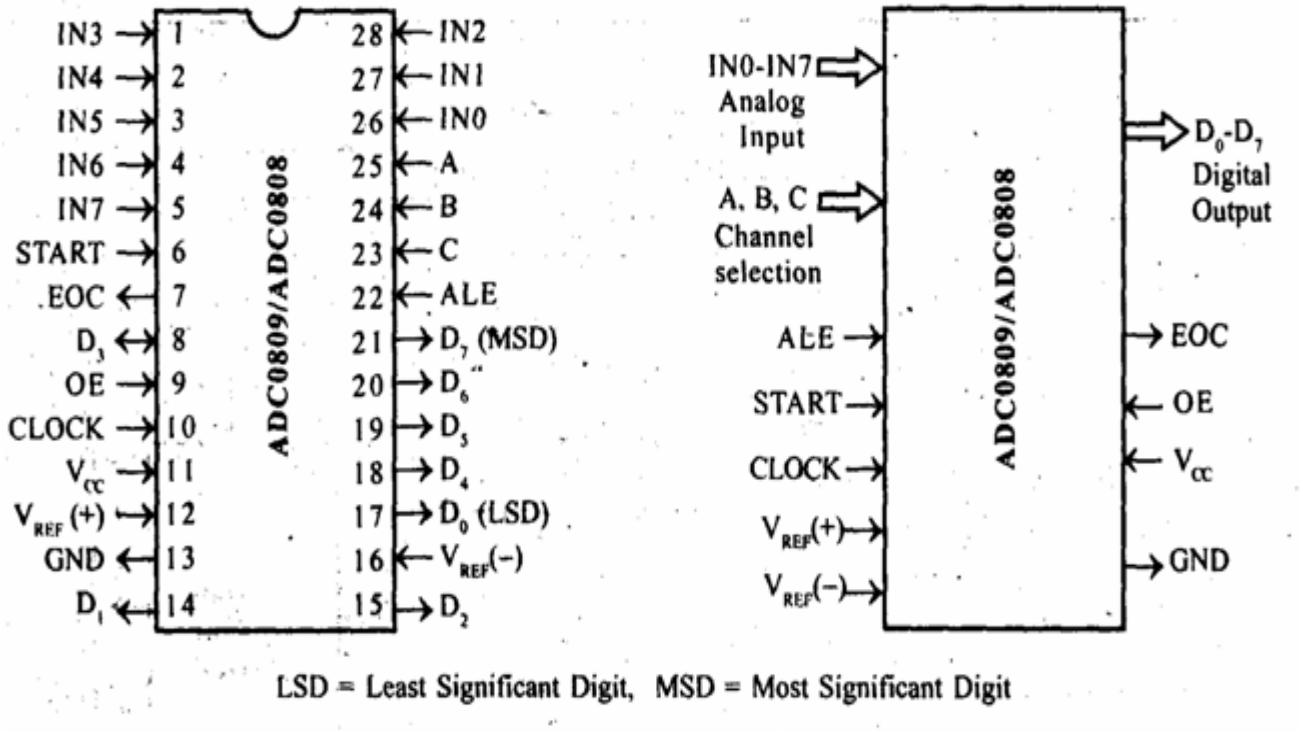


Fig 4 PIN Diagram of ADC

ADC0808/0809 IO Description

- IN0-IN7: Analog Input channels
- D0-D7: Data Lines
- A, B, C: Analog Channel select lines; A is LSB and C is MSB
- OE: Output enable signal
- ALE: Address Latch Enable
- EOC: End of Conversion signal
- V_{ref+}/V_{ref-}: Differential Reference voltage input
- Clock: External ADC clock input

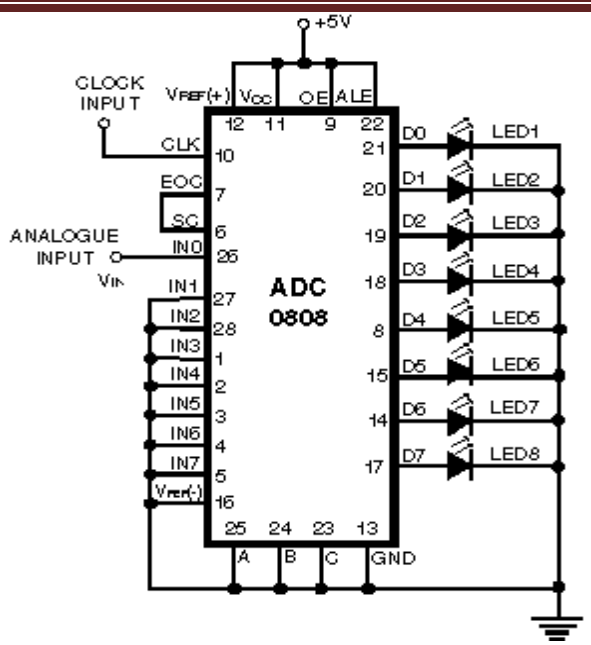


Fig 5 ADC 0808

Normally analogue-to-digital converter (ADC) needs interfacing through a microprocessor to convert analogue data into digital format. This requires hardware and necessary software, resulting in increased complexity and hence the total cost. The circuit of A-to-D converter shown here is configured around ADC 0808, avoiding the use of a microprocessor. The ADC 0808 is an 8-bit A-to-D converter, having data lines D0-D7. It works on the principle of successive approximation. It has a total of eight analogue input channels, out of which any one can be selected using address lines A, B and C. Here, in this case, input channel IN0 is selected by grounding A, B and C address lines.

Usually the control signals EOC (end of conversion), SC (start conversion), ALE (address latch enable) and OE (output enable) are interfaced by means of a microprocessor. However, the circuit shown here is built to operate in its continuous mode without using any microprocessor. Therefore the input control signals ALE and OE, being active-high, are tied to Vcc (+5 volts). The input control signal SC, being active-low, initiates start of conversion at falling edge of the pulse, whereas the output signal EOC becomes high after completion of digitization. This EOC output is coupled to SC input, where falling edge of EOC output acts as SC input to direct the ADC to start the conversion.

As the conversion starts, EOC signal goes high. At next clock pulse EOC output again goes low, and hence SC is enabled to start the next conversion. Thus, it provides continuous 8-bit digital output corresponding to instantaneous value of analogue input. The maximum level of analogue input voltage should be appropriately scaled down below positive reference (+5V) level.

The ADC 0808 IC requires clock signal of typically 550 kHz, which can be easily derived from an Astable multi-vibrator constructed using 7404 inverter gates. In order to visualize the digital output, the row of eight LEDs (LED1 through LED8) have been used, where in each LED is connected to respective data lines D0 through D7. Since ADC works in the continuous mode, it displays digital output as soon as analogue input is applied. The decimal equivalent digital output value D for a given analogue input voltage Vin can be calculated from the relationship.

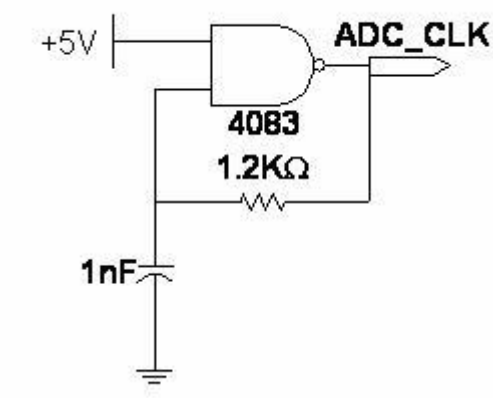


Fig 6 ADC0808 Clock input

Interfacing ADC0808/ADC0809 with 8051

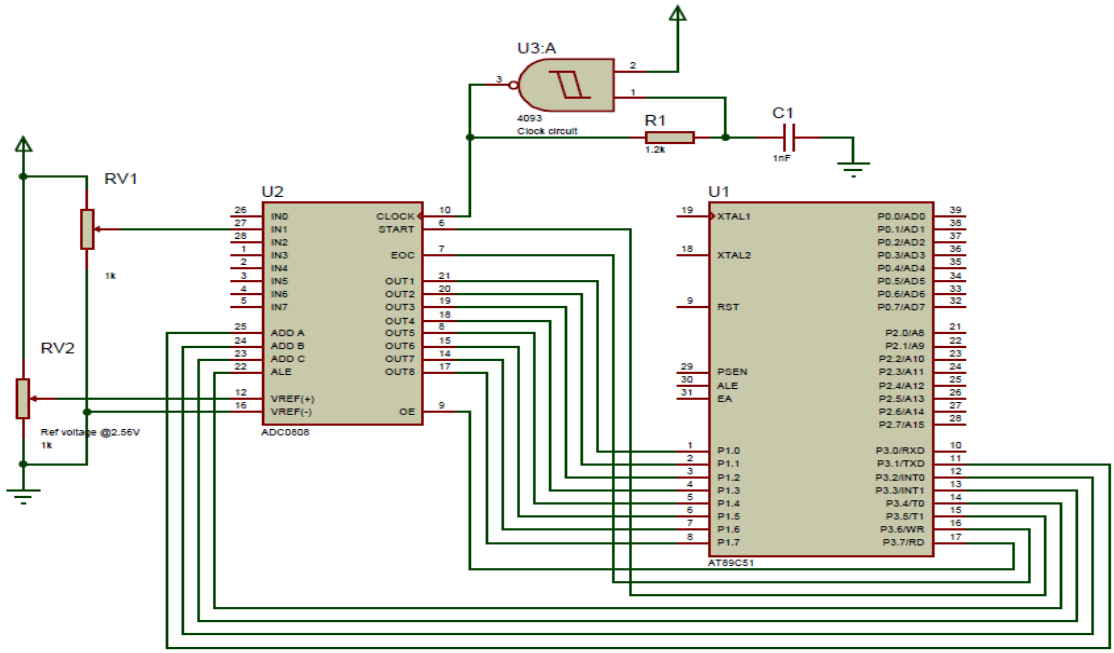


Fig 7 ADC0808 and 8051 connection diagram

5.5.2 LCD display interfacing a 16×2 LCD with 8051 microcontroller

16×2 LCD module.

16×2 LCD module is a very common type of LCD module that is used in 8051 based embedded projects. It consists of 16 rows and 2 columns of 5×7 or 5×8 LCD dot matrices. The module were are talking about here is type number JHD162A which is a very popular one . It is available in a 16 pin package with back light ,contrast adjustment function and each dot matrix has 5×8 dot resolution. The pin numbers, their name and corresponding functions are shown in the table below.

Pin No:	Name	Function
1	VSS	This pin must be connected to the ground
2	VCC	Positive supply voltage pin (5V DC)
3	VEE	Contrast adjustment
4	RS	Register selection
5	R/W	Read or write
6	E	Enable
7	DB0	Data
8	DB1	Data
9	DB2	Data
10	DB3	Data
11	DB4	Data
12	DB5	Data
13	DB6	Data
14	DB7	Data
15	LED+	Back light LED+
16	LED-	Back light LED-

VEE pin is meant for adjusting the contrast of the LCD display and the contrast can be adjusted by varying the voltage at this pin. This is done by connecting one end of a POT to the Vcc (5V), other end to the Ground and connecting the center terminal (wiper) of of the POT to the VEE pin. See the circuit diagram for better understanding.

The JHD162A has two built in registers namely data register and command register. Data register is for placing the data to be displayed , and the command register is to place the commands. The 16×2 LCD

module has a set of commands each meant for doing a particular job with the display. We will discuss in detail about the commands later. High logic at the RS pin will select the data register and Low logic at the RS pin will select the command register. If we make the RS pin high and the put a data in the 8 bit data line (DB0 to DB7) , the LCD module will recognize it as a data to be displayed . If we make RS pin low and put a data on the data line, the module will recognize it as a command.

R/W pin is meant for selecting between read and write modes. High level at this pin enables read mode and low level at this pin enables write mode.

E pin is for enabling the module. A high to low transition at this pin will enable the module.

DB0 to DB7 are the data pins. The data to be displayed and the command instructions are placed on these pins.

LED+ is the anode of the back light LED and this pin must be connected to Vcc through a suitable series current limiting resistor. LED- is the cathode of the back light LED and this pin must be connected to ground.

16×2 LCD module commands.

16×2 LCD module has a set of preset command instructions. Each command will make the module to do a particular task. The commonly used commands and their function are given in the table below.

Command	Function
0F	LCD ON, Cursor ON, Cursor blinking ON
01	Clear screen
02	Return home
04	Decrement cursor
06	Increment cursor
0E	Display ON ,Cursor blinking OFF
80	Force cursor to the beginning of 1st line

C0	Force cursor to the beginning of 2nd line
38	Use 2 lines and 5×7 matrix
83	Cursor line 1 position 3
3C	Activate second line
08	Display OFF, Cursor OFF
C1	Jump to second line, position1
OC	Display ON, Cursor OFF
C1	Jump to second line, position1
C2	Jump to second line, position2

LCD initialization.

The steps that has to be done for initializing the LCD display is given below and these steps are common for almost all applications.

- Send 38H to the 8 bit data line for initialization
- Send 0FH for making LCD ON, cursor ON and cursor blinking ON.
- Send 06H for incrementing cursor position.
- Send 01H for clearing the display and return the cursor.

Sending data to the LCD.

The steps for sending data to the LCD module is given below. I have already said that the LCD module has pins namely RS, R/W and E. It is the logic state of these pins that make the module to determine whether a given data input is a command or data to be displayed.

- Make R/W low.
- Make RS=0 if data byte is a command and make RS=1 if the data byte is a data to be displayed.
- Place data byte on the data register.
- Pulse E from high to low.

- Repeat above steps for sending another data.

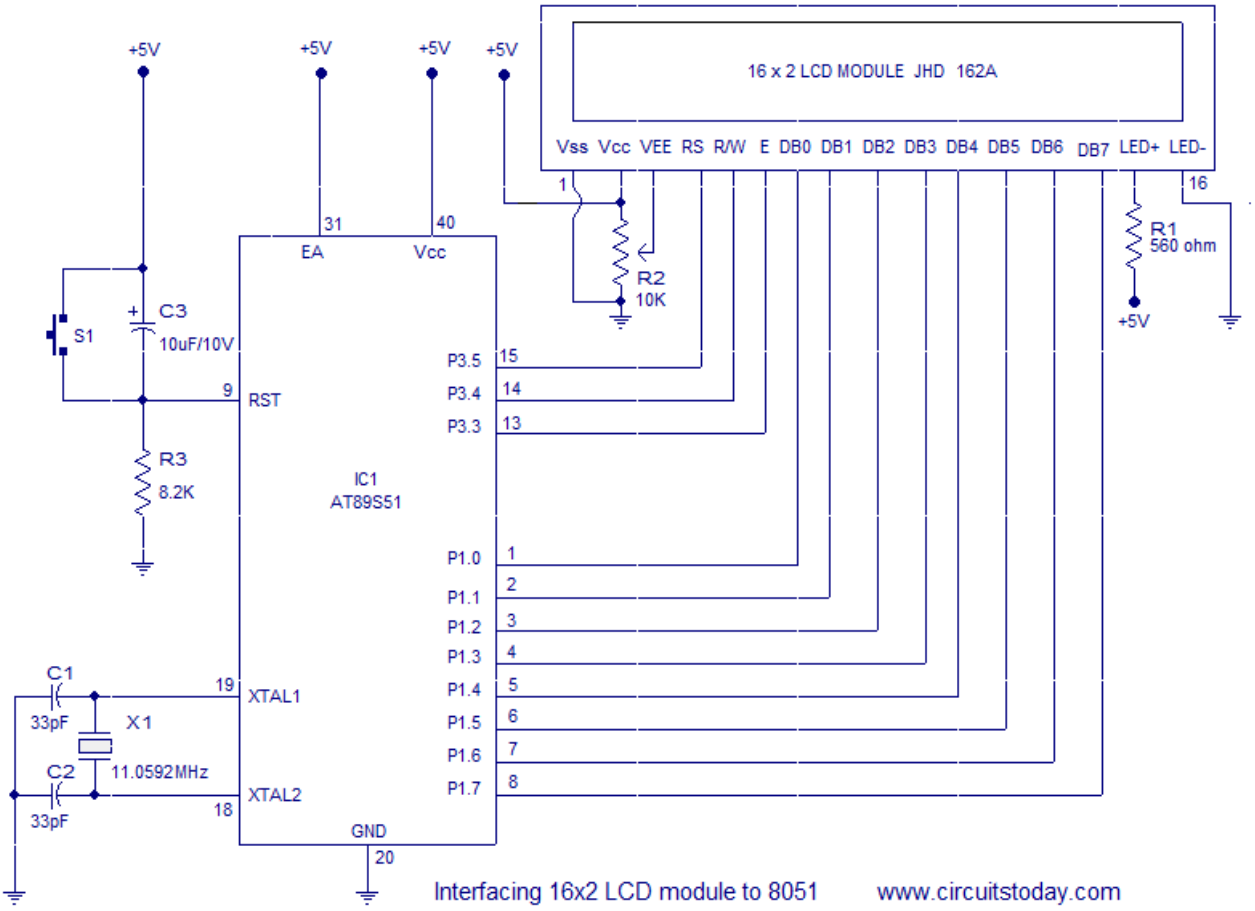


Fig 8 Interfacing 16×2 LCD module to 8051

The circuit diagram given above shows how to interface a 16×2 LCD module with AT89S1 microcontroller. Capacitor C3, resistor R3 and push button switch S1 forms the reset circuitry. Ceramic capacitors C1,C2 and crystal X1 is related to the clock circuitry which produces the system clock frequency. P1.0 to P1.7 pins of the microcontroller is connected to the DB0 to DB7 pins of the module respectively and through this route the data goes to the LCD module. P3.3, P3.4 and P3.5 are connected to the E, R/W, RS pins of the microcontroller and through this route the control signals are transferredto the LCD module. Resistor R1 limits the current through the back light LED and so do the back light intensity. POT R2 is used for adjusting the contrast of the display.

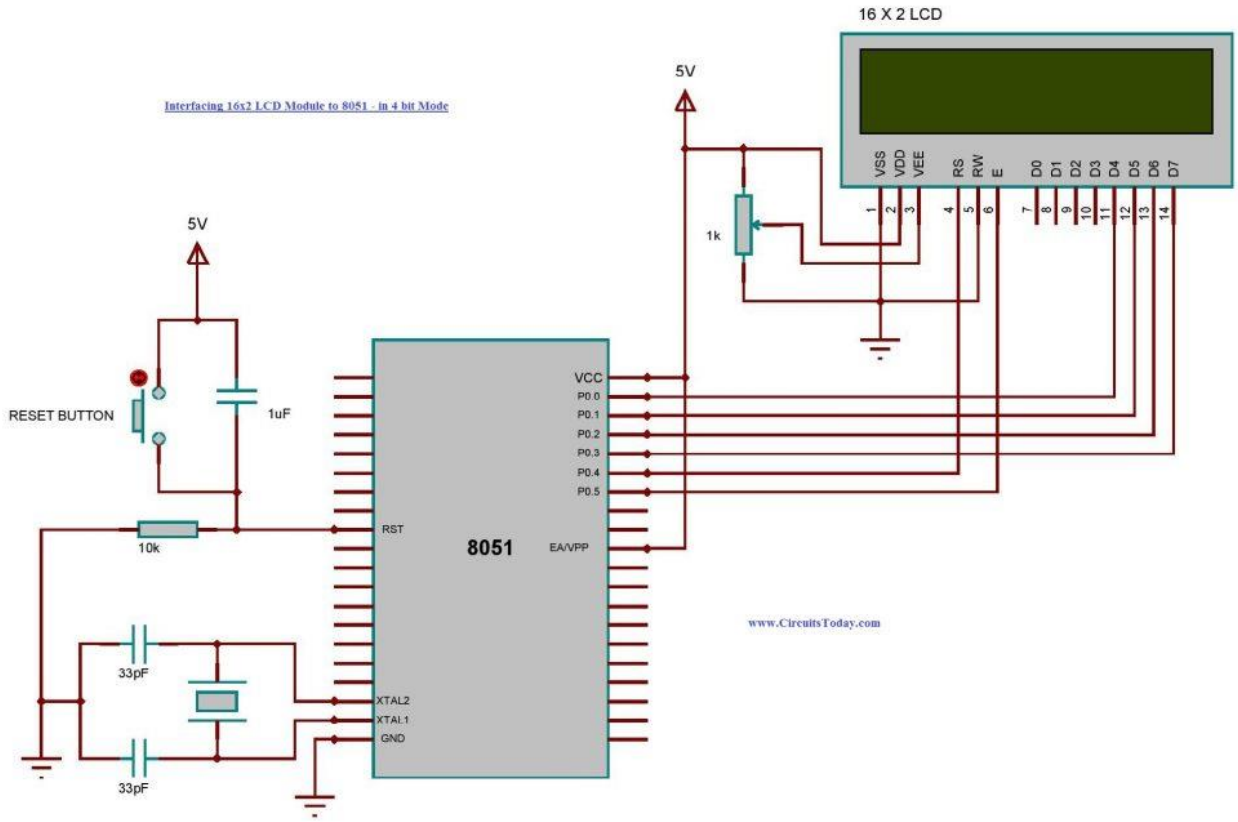
Interfacing LCD Module to 8051 in 4 Bit Mode (using only 4 pins of a port)

The microcontroller like 8051 has only limited number of GPIO pins (GPIO – general purpose input output). So to design complex projects we need sufficient number of I/O pins . An LCD module can be interfaced with a microcontroller either in **8 bit mode (**as seen above**)** or in 4 bit mode. 8 bit mode is the conventional mode which uses 8 data lines and RS, R/W, E pins for functioning. However 4 bit mode uses

only 4 data lines along with the control pins. This will saves the number of GPIO pins needed for other purpose.

Objectives

- Interface an LCD with 8051 in 4 bit mode
- Use a single port of the microcontroller for both data and control lines of the LCD.



LCD Module to 8051 – 4 Bit Mode

As shown in the circuit diagram, port 0 of the controller is used for interfacing it with LCD module. In 4 bit mode only 4 lines D4-D7, along with RS, R/W and E pins are used. This will save us 4 pins of our controller which we might employ it for other purpose. Here we only need to write to the LCD module. So the R/W pin can be ground it as shown in the schematic diagram. In this way the total number of pins can be reduced to 6. In 4 Bit mode the data bytes are split into two four bits and are transferred in the form of a nibble. The data transmission to a LCD is performed by assigning logic states to the control pins RS and E. The reset circuit, oscillator circuit and power supply need to be provided for the proper working of the circuit.

5.5.3 Interfacing DAC with 8051

- Microcontroller are used in wide variety of applications like for measuring and control of physical quantity like temperature, pressure, speed, distance, etc.

- In these systems microcontroller generates output which is in digital form but the controlling system requires analog signal as they don't accept digital data thus making it necessary to use DAC which converts digital data into equivalent analog voltage.

- In the figure shown, we use 8-bit DAC 0808. This IC converts digital data into equivalent analog Current. Hence we require an I to V converter to convert this current into equivalent voltage.

- According to theory of DAC Equivalent analog output is given as:

$$V0=Vref\left[\frac{D0}{2}+\frac{D1}{4}+\frac{D2}{8}+\frac{D3}{16}+\frac{D4}{32}+\frac{D5}{64}+\frac{D6}{128}+\frac{D7}{256}\right]$$

Ex:
1. IF data =00H [00000000], Vref= 10V

$$V0=10\left[\frac{0}{2}+\frac{0}{4}+\frac{0}{8}+\frac{0}{16}+\frac{0}{32}+\frac{0}{64}+\frac{0}{128}+\frac{0}{256}\right]$$

Therefore, V0= 0 Volts.
2. If data is 80H [10000000], Vref= 10V

$$V0=10\left[\frac{1}{2}+\frac{0}{4}+\frac{0}{8}+\frac{0}{16}+\frac{0}{32}+\frac{0}{64}+\frac{0}{128}+\frac{0}{256}\right]$$

Therefore, V0= 5 Volts.

Different Analog output voltages for different Digital signal is given as:

DATA	OUTPUT VOLTAGE
00H	0V
80H	5V
FFH	10V

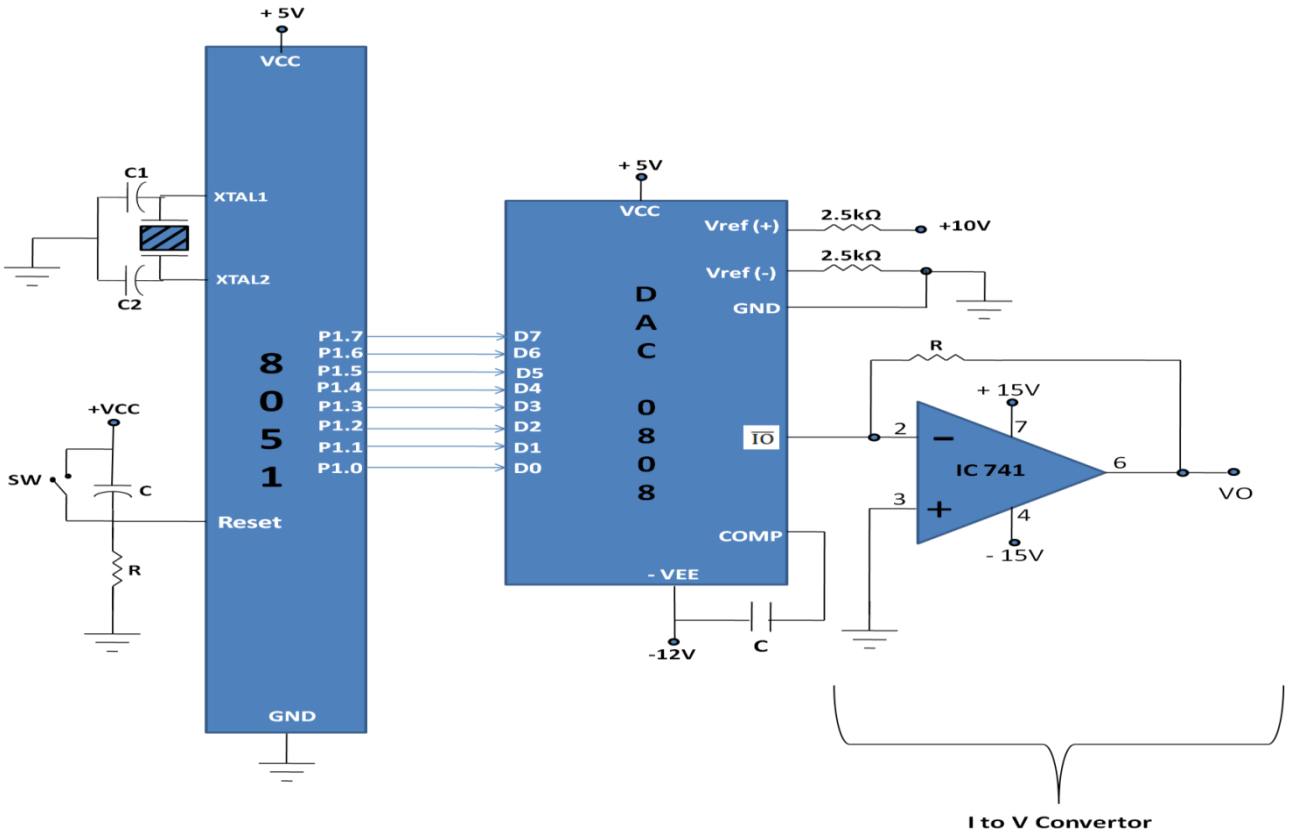


Fig 9 Interfacing DAC to 8051

5.5.4 Interfacing Keyboard to 8051Microcontroller

The key board here we are interfacing is a matrix keyboard. This key board is designed with a particular rows and columns. These rows and columns are connected to the microcontroller through its ports of the micro controller 8051. We normally use 8*8 matrix key board. So only two ports of 8051 can be easily connected to the rows and columns of the key board.

Whenever a key is pressed, a row and a column gets shorted through that pressed key and all the other keys are left open. When a key is pressed only a bit in the port goes high. Which indicates microcontroller that the key is pressed. By this high on the bit key in the corresponding column is identified.

Once we are sure that one of key in the key board is pressed next our aim is to identify that key. To do this we firstly check for particular row and then we check the corresponding column the key board.

To check the row of the pressed key in the keyboard, one of the row is made high by making one of bit in the output port of 8051 high . This is done until the row is found out. Once we get the row next out job is to find out the column of the pressed key. The column is detected by contents in the input ports with

the help of a counter. The content of the input port is rotated with carry until the carry bit is set.

The contents of the counter is then compared and displayed in the display. This display is designed using a seven segment display and a BCD to seven segment decoder IC 7447.

The BCD equivalent number of counter is sent through output part of 8051 displays the number of pressed key.

Interfacing Keyboard to 8051 Microcontroller

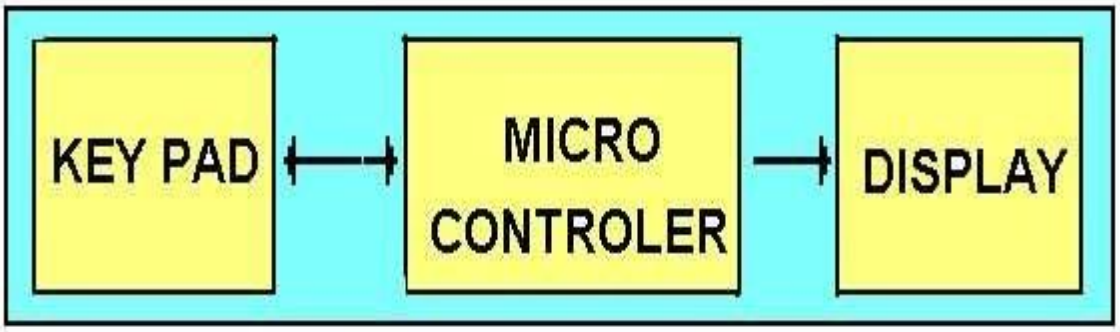


Fig 10 Interfacing Keyboard to 8051

Circuit Diagram of Interfacing Keyboard to 8051

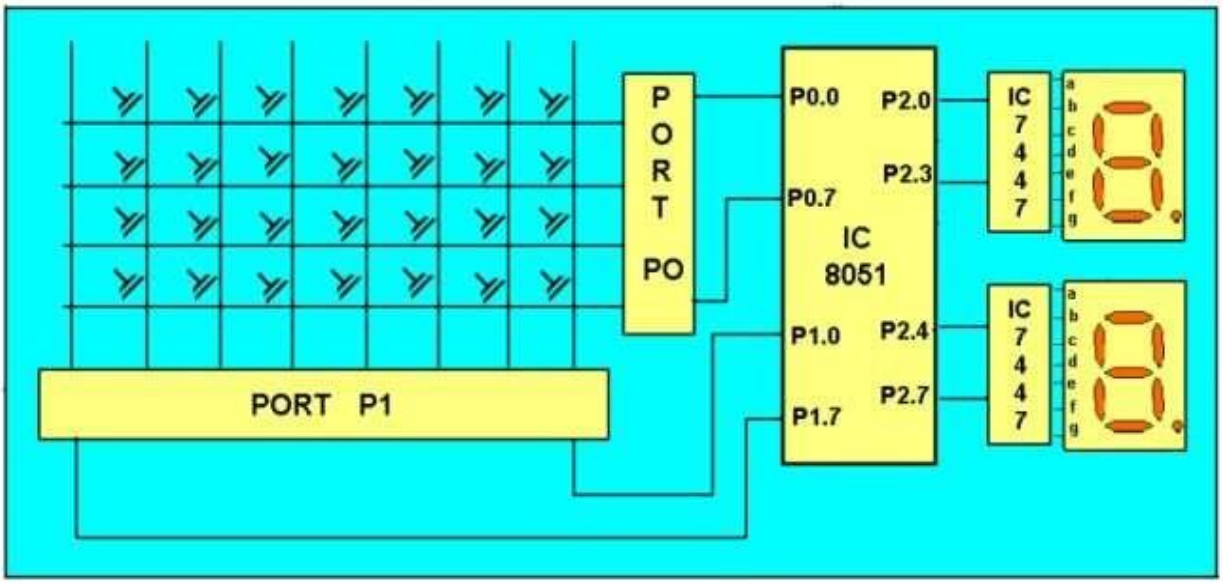


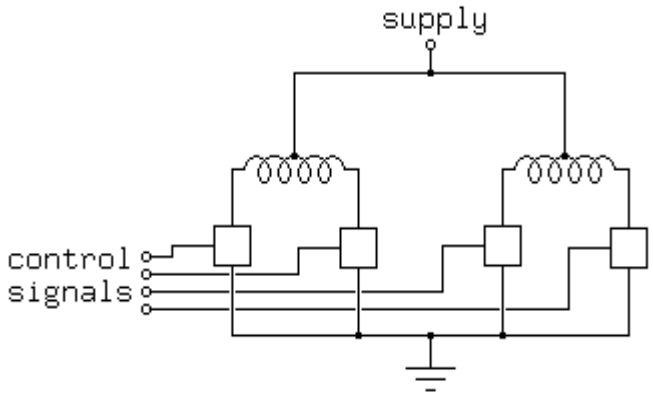
Fig 11 Circuit Diagram of Interfacing Keyboard to 8051

5.5.5 Interfacing Stepper Motor with8051Microcontroller

Step motor is the easiest to control. It's handling simplicity is really hard to deny - all there is to do is to bring the sequence of rectangle impulses to one input of step controller and direction information to another input. Direction information is very simple and comes down to "left" for logical one on that pin and

"right" for logical zero. Motor control is also very simple - every impulse makes the motor operating for one step and if there is no impulse the motor won't start. Pause between impulses can be shorter or longer and it defines revolution rate. This rate cannot be infinite because the motor won't be able to "catch up" with all the impulses (documentation on specific motor should contain such information). The picture below representstheschemeforconnectingthestepmotortomicrocontrollerandappropriateprogram code follows.

The key to driving a stepper is realizing how the motor is constructed. A diagram shows the representation of a 4 coil motor, so named because 4 coils are used to cause the revolution of the drive shaft. Each coil must be energized in the correct order for the motor to spin.



Step angle

It is angle through which motor shaft rotates in one step. step angle is different for different motor . selection of motor according to step angle depends on the application , simply if you require small increments in rotation choose motor having smaller step angle.

No of steps require to rotate one complete rotation = 360 deg. / step angle in deg.

INTERFACING TO 8051.

To cause the stepper to rotate, we have to send a pulse to each coil in turn. The 8051 does not have sufficient drive capability on its output to drive each coil, so there are a number of ways to drive a stepper,

Stepper motors are usually controlled by transistor or driver IC like ULN2003.

Driving current for each coil is then needed about 60mA at +5V supply. A Darlington

Transistor array, ULN2003 is used to increase driving capacity of the8051 chip Four 4.7k resistors help the 8051 to provide more sourcing current from the +5V supply

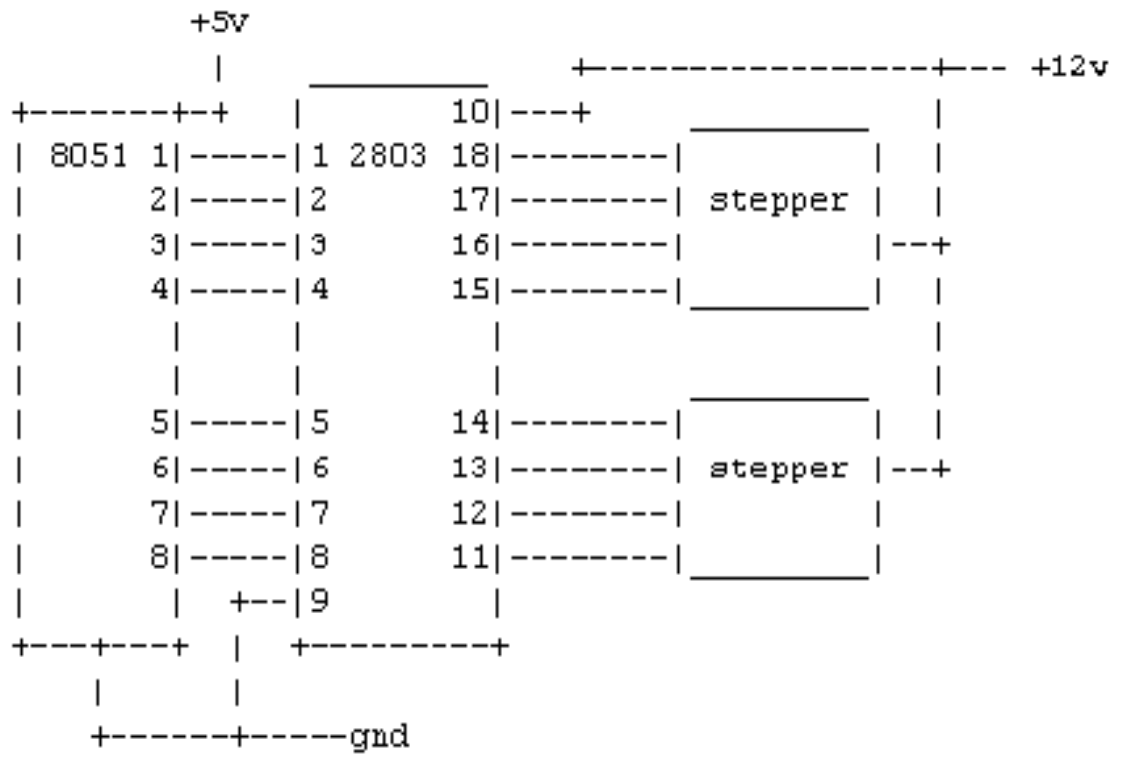


Fig 12 Circuit Diagram of Interfacing stepper to 8051

CONTROLLING STEPPER MOTOR WITH TWO PORT

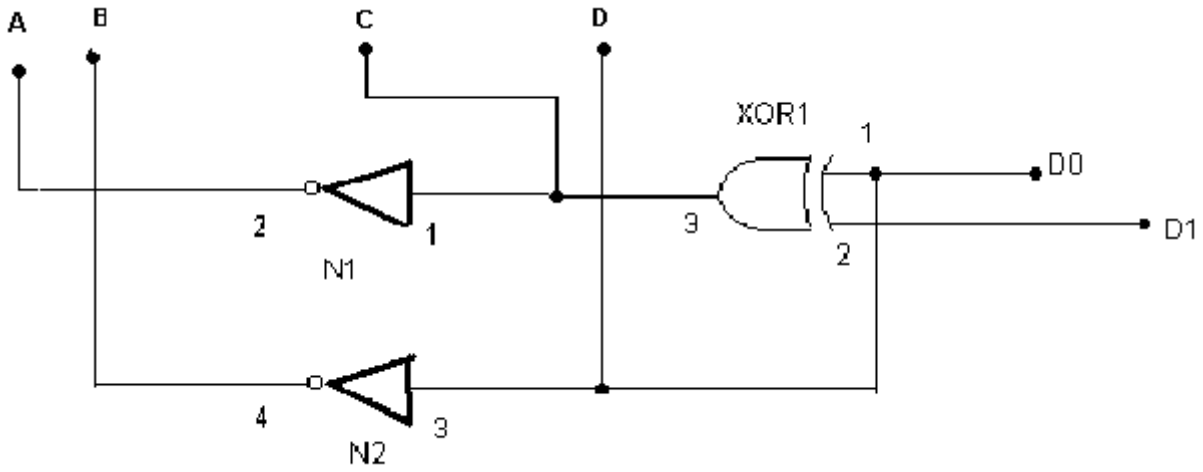


Fig 13 Controlling Stepper motor with two port

TEXT BOOK:

1. Douglas V.Hall, Microprocessor and Interfacing Programming and Hardware. Tata McGraw Hill, New Delhi 2007.

REFERENCES BOOK:

2. Kenneth J.Ayala, The 8051 Microcontroller, Thompson Delmar Learning, New Delhi, 2007

QUESTION BANK
PART B (16MARKS)

1. Explain the different serial communication modes in 8051.
2. States various modes available for timer in 8051.
3. Explain memory organization and I/O addressing in 8051.
4. Explain the different serial communication modes in 8051.
5. Explain the memory structure of 8051.
6. Discuss the peripheral interface of 8051.
7. Explain the memory structure of 8051.
8. How do you explain with diagram to interface a stepper motor with 8051 microcontroller and explain also write an 8051 ALP to run the stepper motor in both forward and reverse direction with delay.
9. How would you describe LCD interface. Explain in detail.
10. State in your own words 8051 serial port programming ? Write short notes on I2C interface.
11. How would you explain about external memory interfacing to 8051
12. How would you show your understanding on 8051 timer and counter programming.
13. (a)State in your own words ADC interfacing using 8051 and explain with diagram.
    (b)Can you recall DAC interfacing using 8051and explain with diagram.
14. What is your opinion on the keyboard interfacing using 8051
15. (a) Can you elaborate on the sensor interfacing using 8051.
    (b) How would you describe seven segment LED interface.