MICROPROCESSORS & MICROCONTROLLERS

LECTURE NOTES

B.TECH

(III YEAR – II SEM)

(2021-22)

Prepared by:

Ms. D Asha, Associate Professor

Department of Electronics and Communication Engineering

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956
(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – ‘A’ Grade - ISO 9001:2015 Certified) Maisammaguda, Dhulapally (Post Via. Kompally), Secunderabad – 500100, Telangana State, India

MALLA REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

III Year B.Tech. ECE-II Sem **L T/P/D C**

5 -/ - /- 4

(R18A0414) MICROPROCESSORS AND MICROCONTROLLERS

OBJECTIVES:

1. To understand the basics of microprocessors and microcontrollers architectures and its functionalities
2. To develop an in-depth understanding of the operation of microprocessors and microcontrollers, machine language programming & interfacing techniques.
3. To design and develop Microprocessor/ microcontroller based systems for real time applications using low level language like ALP.

UNIT -I:

8086 Architecture: Architecture of 8086, Register Organization, Programming Model, Memory Segmentation, Physical Memory Organization, Signal descriptions of 8086- Common Function Signals, Minimum and Maximum mode signals, Timing diagrams.

UNIT -II:

Instruction Set and Assembly Language Programming of 8086: Addressing modes, Instruction Set, Assembler Directives, Procedures, Macros, and Simple Programs involving Logical, Branch and Call Instructions, Sorting, Evaluating Arithmetic Expressions, String Manipulations.

UNIT -III:

I/O Interface: 8255 PPI, Various Modes of Operation and Interfacing to 8086, D/A and A/D Converter, Stepper motor, Interfacing of DMA controller 8257 ,Memory Interfacing to 8086, Interrupt Structure of 8086, Interrupt Vector Table, Interrupt Service Routine.

Communication Interface: Serial Communication Standards, Serial Data Transfer Schemes, 8251 USART Architecture and Interfacing.

UNIT -IV:

Introduction to Microcontrollers: Overview of 8051 Microcontroller, Architecture, I/O Ports, Memory Organization, Addressing Modes and Instruction set of 8051, Simple Programs, memory interfacing to 8051

UNIT -V:

8051 Real Time Control: Programming Timer Interrupts, Programming External Hardware Interrupts, Programming the Serial Communication Interrupts, Programming 8051 Timers and Counters.

ARM Processor : Fundamentals, Registers, current program status register, pipeline concept.

TEXT BOOKS:

1. D. V. Hall, Microprocessors and Interfacing, TMGH, 2nd Edition 2006.
2. Kenneth. J. Ayala, The 8051 Microcontroller, 3rd Ed., Cengage Learning.
3. ARM System Developer's Guide: Designing and Optimizing System Software- Andrew N. Sloss, Dominic Symes, Chris Wright, Elsevier Inc., 2007

REFERENCE BOOKS:

1. Advanced Microprocessors and Peripherals – A. K. Ray and K.M. Bhurchandani, TMH, 2nd Edition 2006.
2. The 8051Microcontrollers, Architecture and Programming and Applications -K.Uma Rao, Andhe Pallavi, Pearson, 2009.
3. Micro Computer System 8086/8088 Family Architecture, Programming and Design - Liu and GA Gibson, PHI, 2nd Ed.
4. Microcontrollers and Application - Ajay. V. Deshmukh, TMGH, 2005.

OUTCOMES:

After going through this course the student will be able to

1. The student will learn the internal organization of popular 8086/8051 microprocessors/microcontrollers.
2. The student will learn how to interface peripherals to microprocessors/microcontrollers.
3. The students will learn the design of microprocessors/microcontrollers-based systems

UNIT -I
8086 Architecture

- **Architecture of 8086**
- **Register Organization**
- **Programming Model**
- **Memory addresses**
- **Memory Segmentation**
- **Physical Memory Organization**
- **Signal descriptions of 8086- Common Function Signals**
- **Minimum and Maximum mode signals**
- **Timing diagrams**

UNIT-I

8086 Architecture

Introduction to Microprocessors

A microprocessor is a computer processor which incorporates the functions of a computer's central processing unit (CPU) on a single integrated circuit (IC), or at most a few integrated circuits

The microprocessor is a multipurpose, clock driven, register based, digital-integrated circuit which accepts binary data as input, processes it according to instructions stored in its memory, and provides results as output. Microprocessors contain both combinational logic and sequential digital logic. Microprocessors operate on numbers and symbols represented in the binary numeral system.

Generation of Microprocessors:

- **INTEL 4004 (1971)**
  - 4-bit microprocessor
  - 4 KB main memory
  - 45 instructions
  - PMOS technology
  - was first programmable device which was used in calculators
- **INTEL 8008 (1972)**
  - 8-bit version of 4004
  - 16 KB main memory
  - 48 instructions
  - PMOS technology
  - Slow
- **Intel 8080** (1973)
  - 8-bit microprocessor
  - 64 KB main memory
  - 2 microseconds clock cycle time
  - 500,000 instructions/sec
  - 10X faster than 8008
  - NMOS technology
  - Drawback was that it needed three power supplies.

- Small computers (Microcomputers) were designed in mid 1970's
  Using 8080 as CPU.

- **<u>INTEL 8086/8088</u>**

  Year of introduction 1978 for 8086 and 1979 for 8088
  - 16-bit microprocessors
  - Data bus width of 8086 is 16 bit and 8 bit for 8088
  - 1 MB main memory
  - 400 nanoseconds clock cycle time
  - 6 byte instruction cache for 8086 and 4 byte for 8088
  - Other improvements included more registers and additional instructions
  - In 1981 IBM decided to use 8088 in its personal computer

- **<u>INTEL 80186</u>** (1982)

  - 16-bit microprocessor-upgraded version of 8086
  - 1 MB main memory
  - Contained special hardware like programmable counters, interrupt controller etc.
  - Never used in the PC
  - But was ideal for systems that required a minimum of hardware .
- **<u>INTEL 80286</u>** (1983)
  - 16-bit high performance microprocessor with memory management & protection
  - 16 MB main memory
  - Few additional instructions to handle extra 15 MB
  - Instruction execution time is as little as 250 ns
  - Concentrates on the features needed to implement MULTITASKING

- **Intel 80386** (1986)
- **Intel 80486** (1989)
- **Pentium** (1993)
- **Pentium pro**(1995)
- **Pentium ii** (1997)
- **Pentium iii** (1999)
- **Pentium iv** (2002)
- **Latest is Intel i9 processor**

General Architecture of Microprocessors

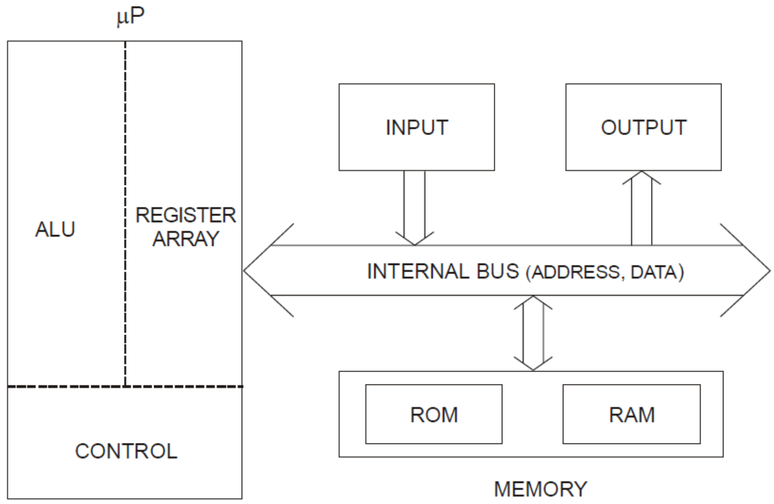


Figure 1.2 Architecture of Microprocessor

Buses

Figure shows busses interconnecting various blocks. These busses allow exchange of words between the blocks. A bus has a wire or line for each bit and thus allows exchange of all bits of a word in parallel. The processing of bits in the μP is also in parallel. The busses can thus be viewed as data highways. The width of a bus is the number of signal lines that constitute the bus.

Arithmetic-Logic Unit (ALU)

The arithmetic-logic unit is a combinational network that performs arithmetic and logical operations on the data.

Internal Registers

A number of registers are normally included in the microprocessor. These are used for temporary storage of data, instructions and addresses during execution of a program. Those in the Intel

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. Fig 1.4 shows register organization of 8086. We will categorize the register set into four groups as follows:

Fig.1.4 Register organization of 8086 Microprocessor

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 is as shown in fig 1.5. Each segment contains 64Kbyte of memory. There are four segment registers.

Code segment (CS) is a 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.

Fig1.5. Memory segmentation

Pointers and index registers.

The pointers contain within the particular segments. The pointers IP, BP, SP usually contain offsets within the code, data and stack segments respectively

Stack Pointer (SP) is a 16-bit register pointing to program stack in stack segment.

Base Pointer (BP) is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed or register indirect addressing.

Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data addresses in string manipulation instructions.

Destination Index (DI) is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.

Flag Register:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	1
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AC	X	PF	X	CY

X = Undefined

Fig1.6 . Flag Register of 8086

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. The 8086 flag register as shown in the fig 1.6. 8086 has 9 active flags and they are divided into two categories:

1. Conditional Flags
2. Control Flags

Conditional flags are as follows:

Carry Flag (CY): This flag indicates an overflow condition for unsigned integer arithmetic. It is also used in multiple-precision arithmetic.

Auxiliary Flag (AC): If an operation performed in ALU generates a carry/barrow from lower nibble (i.e. D0 – D3) to upper nibble (i.e. D4 – D7), the AC flag is set i.e. carry given by D3 bit to D4 is AC 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.

Control Flags

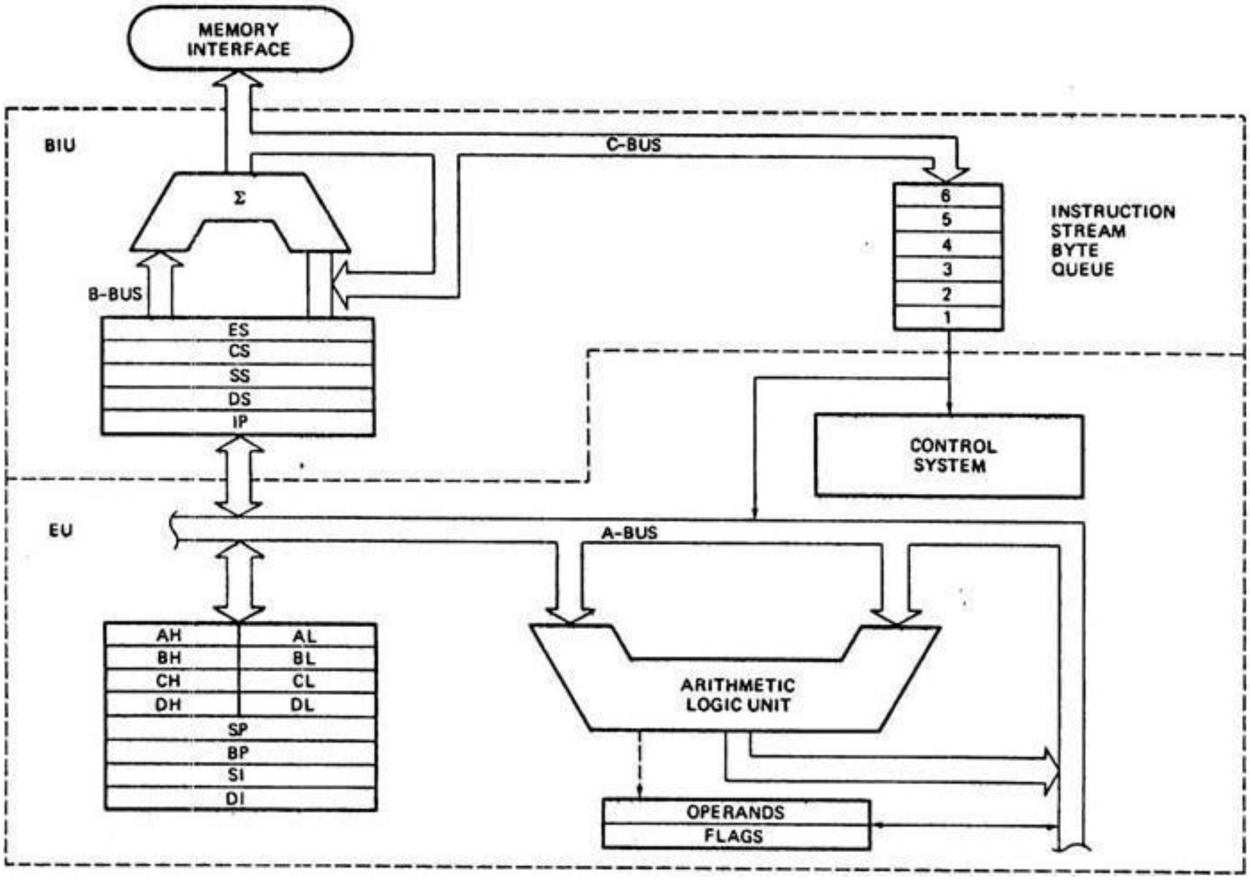
Control flags are set or reset deliberately to control the operations of the execution unit. Control flags are as follows:

Trap Flag (TF): It is used for single step control. It allows user to execute one instruction of a program at a time for debugging. When trap flag is set, program can be run in single step mode.

Interrupt Flag (IF): It is an interrupt enable/disable flag. If it is set, the maskable interrupt of 8086 is enabled and if it is reset, the interrupt is disabled. It can be set by executing instruction sit and can be cleared by executing CLI instruction.

Direction Flag (DF): It is used in string operation. 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.

8086 Architecture



The 8086 is mainly divided into mainly two blocks

1. Execution Unit (EU)

2.Bus interface Unit (BIU)

Dividing the work between these two will speedup the processing

1) EXECUTION UNIT(EU)

The Execution unit tells the BIU where to fetch instructions or data from

- decodes instructions and
- Executes instructions

The Execution unit contains:

1) Control circuitry
2) ALU
3) FLAGS
4) General purpose Registers
5) Pointer and Index Registers

Control Circuitry:

- It directs internal operations.

- A decoder in the EU translates instructions fetched from memory Into series of actions which the EU carries out

Arithmetic Logic Unit:

16 bit ALU

Used to carry the operations

- ADD
- SUBTRACT
- XOR
- INCREMENT
- DECREMENT
- COMPLEMENT
- SHIFT BINARY NUMBERS

FLAG REGISTERS:

- A flag is a flip flop that indicates some condition produced by execution of an instruction or controls certain operation of the EU.
- It is 16 bit
- It has nine active flags

Divided into two types

1. Conditional flags
2. Control flags

Conditional Flags

Carry Flag (CY): This flag indicates an overflow condition for unsigned integer arithmetic. It is also used in multiple-precision arithmetic.

Auxiliary Flag (AC): If an operation performed in ALU generates a carry/barrow from lower nibble (i.e. D0 – D3) to upper nibble (i.e. D4 – D7), the AC flag is set i.e. carry given by D3 bit to D4 is AC 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.

Control Flags

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

Trap Flag (TF): It is used for single step control. It allows user to execute one instruction of a program at a time for debugging. When trap flag is set, program can be run in single step mode.

Interrupt Flag (IF): It is an interrupt enable/disable flag. If it is set, the maskable interrupt of 8086 is enabled and if it is reset, the interrupt is disabled. It can be set by executing instruction sit and can be cleared by executing CLI instruction.

Direction Flag (DF): It is used in string operation. 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.

General Purpose Registers:
The 8086 general purpose registers are similar to those of earlier generations 8080 and 8085 .It was designed in such a way that many programs written for 8080 and 8085 could easily be translated to run on 8086.The advantage of using internal registers for the temporary storage of data is that since data already in the EU ., it can be accessed much more quickly than it could be accessed from external memory.
General Purpose 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.

2) BUS INTERFACE UNIT (BIU)

The BIU sends out

- Addresses
- Fetches instructions from memory
- Read data from ports and memory

Or

The BIU handles all transfer of data and addresses on the buses for the Execution Unit

The Bus interface unit contains

1) Instruction Queue
2) Instruction pointer
3) Segment registers
4) Address Generator

Instruction Queue:

BIU gets upto 6 bytes of next instructions and stores them in the instruction queue. When EU executes instructions and is ready for its next instruction, then it simply reads the instruction from this instruction queue resulting in increased execution speed. Fetching the next instruction while the current instruction executes is called **pipelining.**(based on FIFO) .This is much faster than sending out an addresses to the system memory and waiting for memory to send back the next instruction byte or bytes .Here the Queue will be dumped and then reloaded from the new Address.

Segment Register:

The 8086 20 bit addresses So it can address upto 2^{20} in memory (1 Mbyte) but at any instant it can address upto 4 64 KB segments. This four segments holds the upper 16 bits of the starting address of four memory segments that the 8086 is working with it at particular time .The BIU always inserts zeros for the lowest 4 bits of the 20 bit starting address

Example : If the code segment register contains 348AH then the code segment starts at 348A0H .In other words a 64Kbyte segment can be located anywhere within 1MByte address Space but the segment will always starts at an address with zeros in the lowest 4 bits

Stack: is a section of memory set aside to store addresses and data while subprogram executes is often called segment base . The stack segment register always holds the upper 16 bit starting address of program stack.

The extra segment register and data segment register is used to hold the upper 16 bit starting addresses of two memory segments that are used for data .

Instruction Pointer holds the 16 bit address or offset of the next code byte within the code segment. The value contained in the Instruction Pointer called as Offset because the value must be added to the segment base address in CS to produce the required 20 bit address.

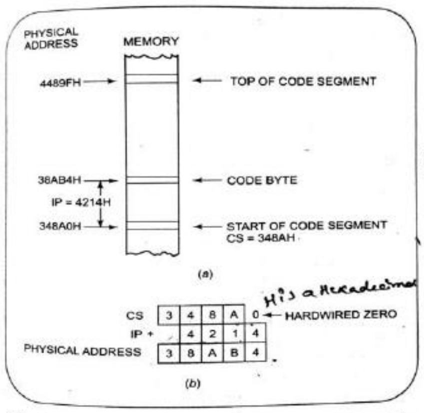


Fig. 2.10 *Addition of IP to CS to produce the physical address of the code byte. (a) Diagram. (b) Computation.*

CS register contains the Upper 16 bit of the starting address of the code segment in the 1 Mbyte address range the instruction pointer contains a 16 bit offset which tells wherein that 64 Kbyte code segment the next instruction byte has to be fetched from.

Stack Register and Stack Pointer:

Stack: is a section of memory set aside to store addresses and data while subprogram executes is often called segment base . The stack segment register always holds the upper 16 bit starting address of program stack. The Stack pointer (SP) holds the 16 bit offset from the starting of the segment to the memory location where a word was most recently stored .The memory location where the word is stored is called as top of the stack

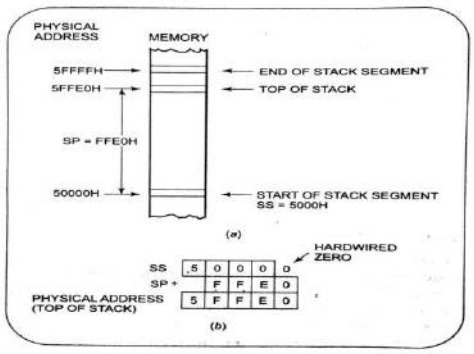


Fig. 2.11 *Addition of SS and SP to produce the physical address of the top of the stack. (a) Diagram. (b) Computation.*

Pointer and Index registers:
In addition to stack pointer register EU has
Base pointer Register (BP)
Source Pointer Register(SP)
Destination Pointer Register(DP)

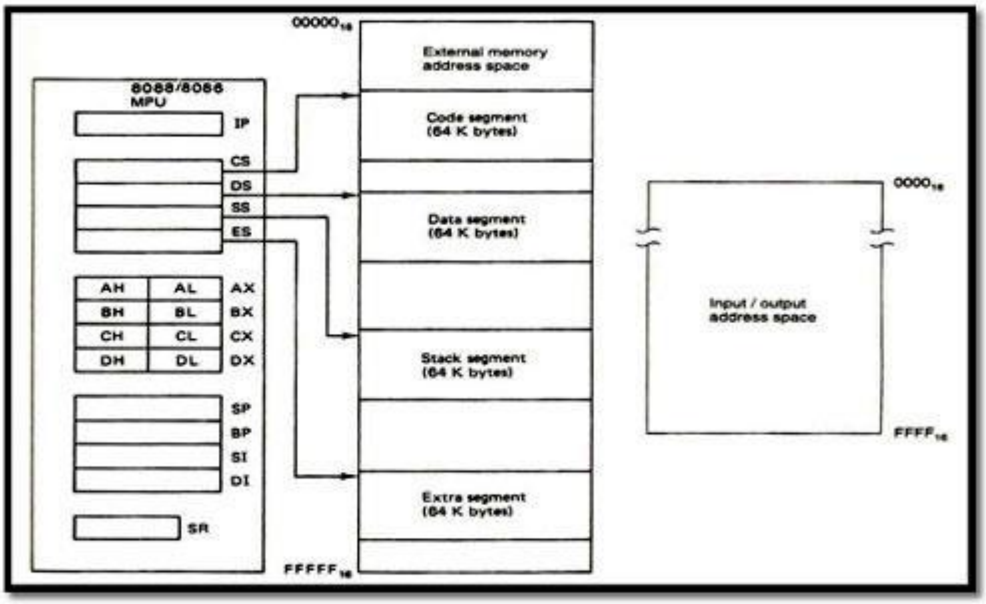
These three registers are used to store temporary storage of data like general purpose registers .They hold the 16 bit offset data of the data word in one of the segment

Programming model

How can a 20-bit address be obtained, if there are only 16-bit registers?
However, the largest register is only 16 bits (64k); so physical addresses have to be calculated. These calculations are done in hardware within the microprocessor.
The 16-bit contents of segment register gives the starting/ base address of particular segment. To address a specific memory location within a segment we need an offset address. The offset address is also 16-bit wide and it is provided by one of the associated pointer or index register.

To be able to program a microprocessor, one does not need to know all of its hardware architectural features. What is important to the programmer is being aware of the various registers within the device and to understand their purpose, functions, operating capabilities, and limitations.

The above figure illustrates the software architecture of the 8086 microprocessor. From this diagram, we see that it includes fourteenl6-bit internal registers: the instruction pointer (IP), four data registers (AX, BX, CX, and DX), two pointer registers (BP and SP), two index registers (SI and DI), four segment registers (CS, DS, SS, and ES) and status register (SR), with nine of its bits implemented as status and control flags.



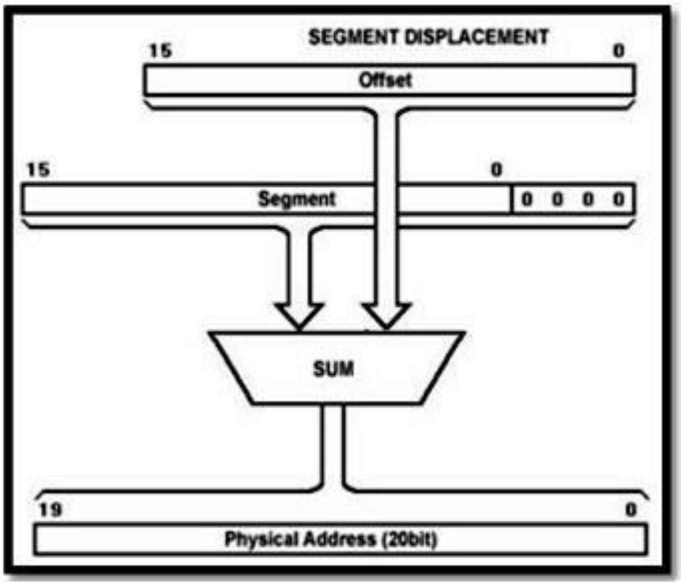
The point to note is that the beginning segment address must begin at an address divisible by 16.Also note that the four segments need not be defined separately. It is allowable for all four segments to completely overlap (CS = DS = ES = SS).

Logical and Physical Address

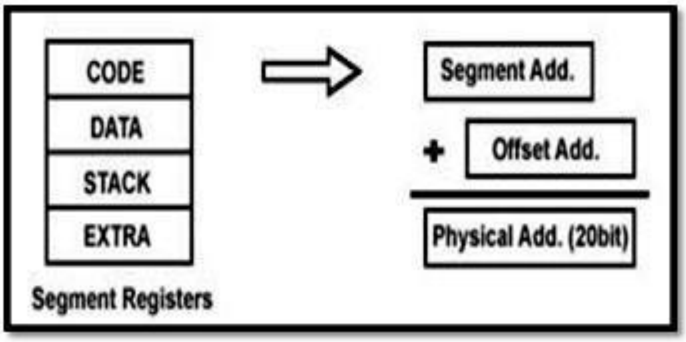
Addresses within a segment can range from address 00000h to address 0FFFFh. This corresponds to the 64K-bytelength of the segment. An address within a segment is
called an offset or logical address.

A logical address gives the displacement from the base address of the segment to the desired location within it, as opposed to its "real" address, which maps directly anywhere into the 1 MByte memory space. This "real" address is called the physical address.

What is the difference between the physical and the logical address? The physical address is 20 bits long and corresponds to the actual binary code output by the BIU on the address bus lines. The logical address is an offset from location 0 of a given segment.

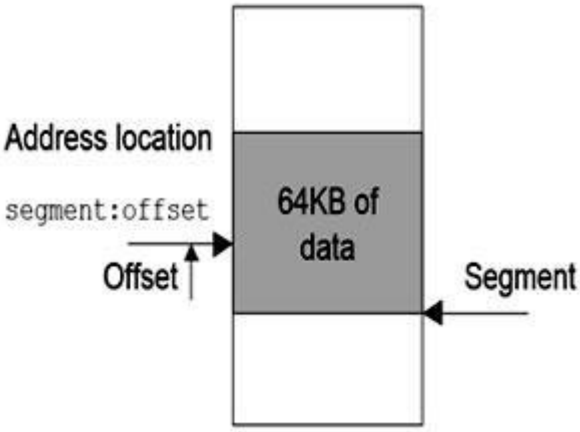


You should also be careful when writing addresses on paper to do so clearly. To specify the logical address XXXX in the stack segment, use the convention SS:XXXX, which is equal to [SS] * 16 + XXXX.



Logical address is in the form of: Base Address: Offset Offset is the displacement of the memory location from the starting location of the segment. To calculate the physical address of the memory, BIU uses the following formula:

Physical Address = Base Address of Segment * 16 + Offset



Example:

The value of Data Segment Register (DS) is 2222H.

To convert this 16-bit address into 20-bit, the BIU appends 0H to the LSB (by multiplying with 16) of the address. After appending, the starting address of the Data Segment becomes 22220H.

Data at any location has a logical address specified as:2222H: 0016H

Where 0016H is the offset, 2222 H is the value of DS Therefore the physical address:22220H + 0016H
: 22236 H

The following table describes the default offset values to the corresponding memory segments.

Segment	Offset Registers	Function
CS	IP	Address of the next instruction
DS	BX, DI, SI	Address of data
SS	SP, BP	Address in the stack
ES	BX, DI, SI	Address of destination data (for string operations)

Some of the advantages of memory segmentation in the 8086 are as follows:

- With the help of memory segmentation a user is able to work with registers having only 16-bits.
- The data and the user's code can be stored separately allowing for more flexibility.

- Also due to segmentation the logical address range is from 0000H to FFFFH the code can be loaded at any location in the memory.

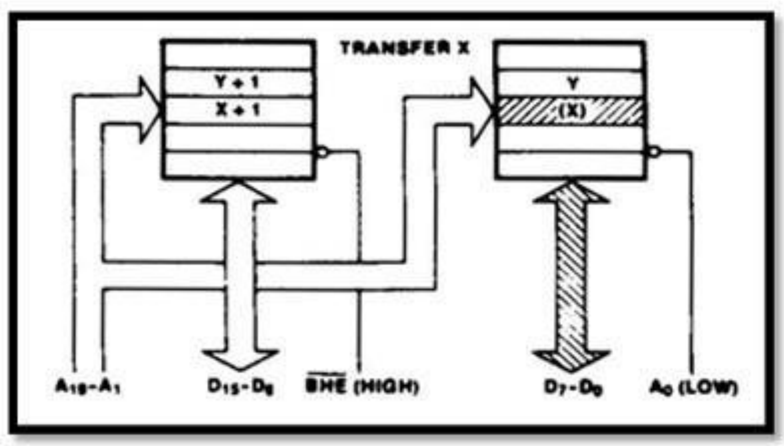
Physical memory organization:

The 8086's 1Mbyte memory address space is divided in to two independent 512Kbyte banks: the low (even) bank and the high (odd) bank. Data bytes associated with an even address (0000016, 0000216, etc.) reside in the low bank, and those with odd addresses (0000116, 0000316, etc.) reside in the high bank.

Address bits A1 through A19 select the storage location that is to be accessed. They are applied to both banks in parallel. A0and bank high enable (BHE) are used as bank-select signals.

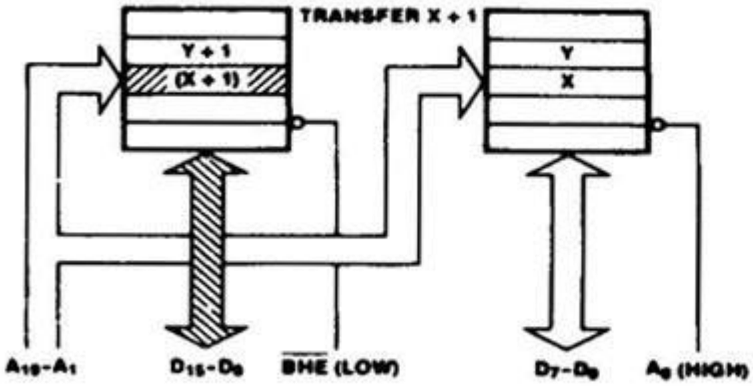
The four different cases that happen during accessing data:

Case 1: When a byte of data at an even address (such as X) is to be accessed:



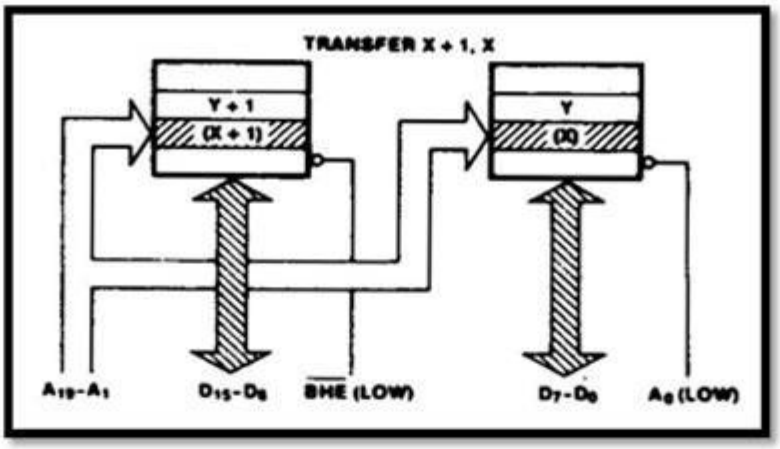
- A0 is set to logic 0 to enable the low bank of memory.
- BHE is set to logic 1 to disable the high bank.

Case 2: When a byte of data at an odd addresses (such as X+1) is to be accessed:



- A0is set to logic 1 to disable the low bank of memory.
- BHE is set to logic 0 to enable the high bank.

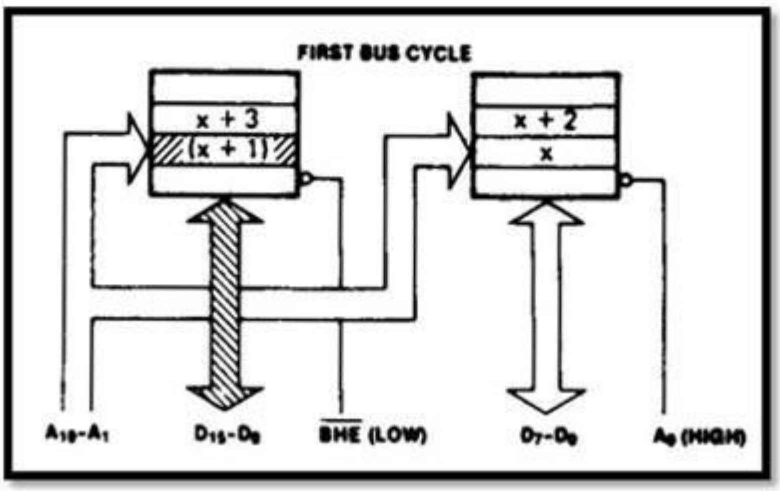
Case 3: When a word of data at an even address (aligned word) is to be accessed:



- A0 is set to logic 0 to enable the low bank of memory.
- BHE is set to logic 0 to enable the high bank.

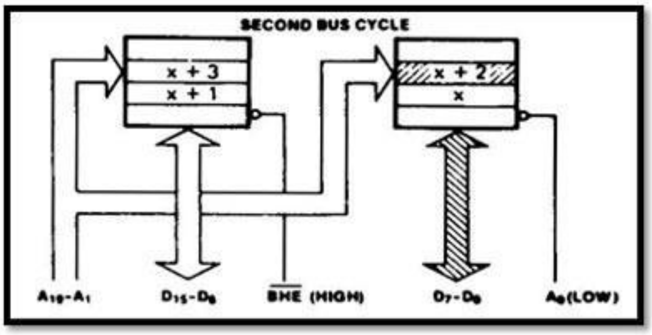
Case 4: When a word of data at an odd address (misaligned word) is to be accessed, then the 8086 need two bus cycles to access it:

a) During the first bus cycle, the odd byte of the word (in the high bank) is addressed



- A0 is set to logic 1 to disable the low bank of memory
- BHE is set to logic 0 to enable the high bank.

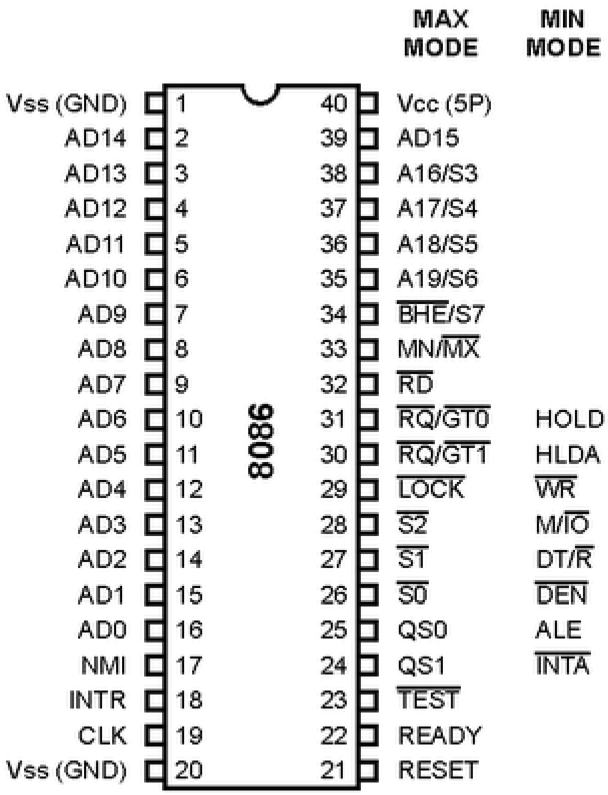
b) During the second bus cycle, the odd byte of the word (in the low bank) is addressed



- A0is set to logic 0 to enable the low bank of memory.
- BHE is set to logic 1 to disable the high bank.

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 is 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 combinedly indicate which segment register is presently being used for memory accesses as shown in Table 1.1.

These lines float to tri-state off (tristated) 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.

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

Table 1.1 Bus High Enable/Status

BHE/S7 (Active Low):

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 1.2. 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.

$\overline{BHE}$	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 interrrupt. 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".

$\overline{\text{INTA}}$-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 /$\overline{\text{R}}$-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'.

$\overline{\text{DEN}}$-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.

HOLD, HLDA-Hold/Hold Acknowledge:

When the HOLD line goes high, it indicates to the processor that another master is requesting the bus access. The processor, after receiving the HOLD request, issues the hold acknowledge signal on HLDA pin, in the middle of the next clock cycle after completing the current bus (instruction) cycle. At the same time, the processor floats the local bus and control lines. When the processor detects the HOLD line low, it lowers the HLDA signal. HOLD is an asynchronous input, and it should be externally synchronized.

S2, S1, S0 -Status Lines:

These are the status lines which reflect the type of operation, being carried out by the processor. These become active during T4 of the previous cycle and remain active during T1 and T2 of the current bus cycle. The status lines return to passive state during T3 of the current bus cycle so that they may again become active for the next bus cycle during T4. Any change in these lines during T3 indicates the starting of a new cycle, and return to passive state indicates end of the bus cycle. These status lines are encoded in table 1.3

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	**Indication**
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O Port
0	1	0	Write I/O Port
0	1	1	Halt
1	0	0	Code Access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

Table 1.3

$\overline{\text{LOCK}}$:

This output pin indicates that other system bus masters will be prevented from gaining the system bus, while the $\overline{\text{LOCK}}$ signal is low. The $\overline{\text{LOCK}}$ signal is activated by the 'LOCK' prefix instruction and remains active until the completion of the next instruction. This floats to tri-state off during "hold acknowledge". When the CPU is executing a critical instruction which requires the system bus, the LOCK prefix instruction ensures that other processors connected in the system will not gain the control of the bus. The 8086, while executing the prefixed instruction, asserts the bus lock signal output, which may be connected to an external bus controller.

QS1, QS0-Queue Status:

These lines give information about the status of the codeprefetch queue. These are active during the CLK cycle after which the queue operation is performed. These are encoded as shown in Table 1.4.

QS_1,	QS_0	Indication
0	0	No operation
0	1	First byte of opcode from the queue
1	0	Empty queue
1	1	Subsequent byte from the queue

Table 1.4

$\overline{RQ}/\overline{GT_0}$, $\overline{RQ}/\overline{GT_1}$-ReQuest/Grant:

These pins are used by other local bus masters, in maximum mode, to force the processor to release the local bus at the end of the processor's current bus cycle. Each of the pins is bidirectional with $\overline{RQ}/\overline{GT_0}$ having higher priority than $\overline{RQ}/\overline{GT_1}$, $\overline{RQ}/\overline{GT}$ pins have internal pull-up resistors and may be left unconnected. The request! Grant sequence is as follows:

1. A pulse one clock wide from another bus master requests the bus access to 8086.

2. During T4 (current) or T1 (next) clock cycle, a pulse one clock wide from 8086 to the requesting master, indicates that the 8086 has allowed the local bus to float and that it will enter the "hold acknowledge" state at next clock cycle. The CPU's bus interface unit is likely to be disconnected from the local bus of the system.

3. A one clock wide pulse from the another master indicates to 8086 that the 'hold' request is about to end and the 8086 may regain control of the local bus at the next clock cycle.

Minimum Mode 8086 System and Timings

In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX* pin to logic1. In this mode, all the control signals are given out by the microprocessor chip itself. There is a single microprocessor in the minimum mode system. The remaining components in the system are latches, transreceivers, clock generator, memory and I/O devices. Some type of chip selection logic may be required for selecting memory or I/O devices, depending upon the address map of the system.

Latches:

The latches are generally buffered output D-type flip-flops, like, 74LS373 or 8282. They are used for separating the valid address from the multiplexed address/data signals and are controlled by the ALE signal generated by 8086.

Transreceivers

Transreceivers are the bidirectional buffers and some times they are called as data amplifiers. They are required to separate the valid data from the time multiplexed address/data signal. They are controlled by two signals, namely, DEN* and DT/R*. The DEN* signal indicates that the valid data is available on the data bus, while DT/R indicates the direction of data, i.e. from or to the processor.

Memory:

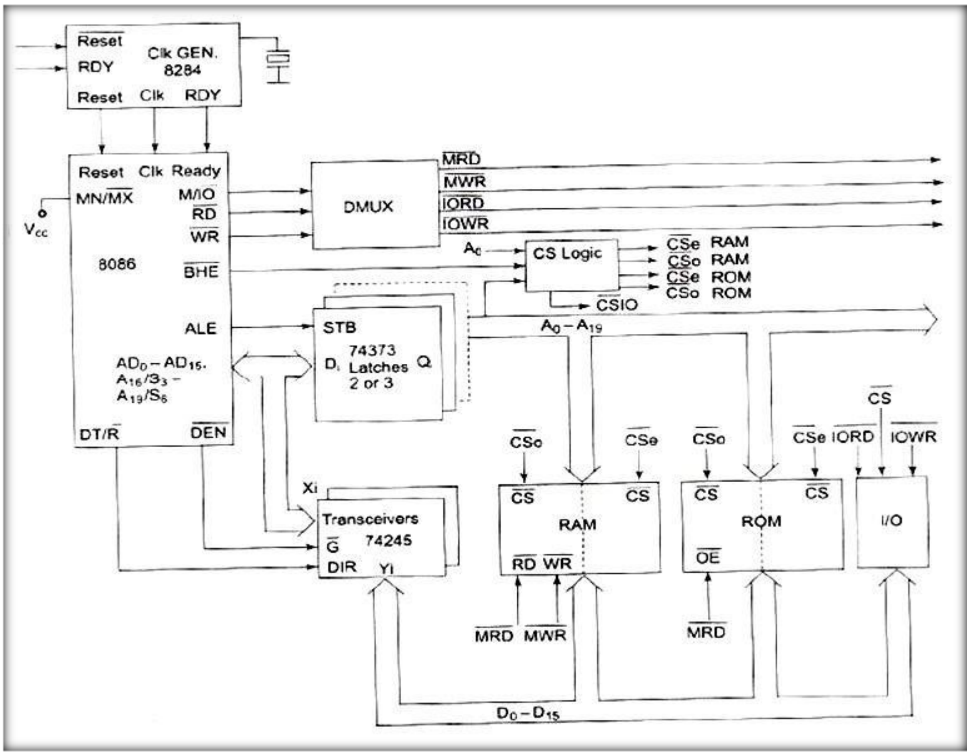
The system contains memory for the monitor and users program storage. Usually, EPROMS are used for monitor storage, while RAMs for users program storage.

IO Devices:

A system may contain I/O devices for communication with the processor as well as some special purpose I/O devices.

Clock Generator:

The clock generator generates the clock from the crystal oscillator and then shapes it and divides to make it more precise so that it can be used as an accurate timing reference for the system. The clock generator also synchronizes some external signals with the system clock.



The general system organization is shown in above fig .Since it has 20 address lines and 16 data lines, the 8086 CPU requires three octal address latches and two octal data buffers for the complete address and data separation.

The working of the minimum mode configuration system can be better described in terms of the timing diagrams rather than qualitatively describing the operations. The opcode fetch and read cycles are similar. Hence the timing diagram can be categorized in two parts.

1) Timing diagram for read cycle
2) Timing diagram for write cycle.

Timing diagram for Read cycle :

The read cycle begins in T1 with the assertion of the address latch enable (ALE) signal and also M/IO* signal. During the negative going edge of this signal, the valid address is latched on the local bus. The BHE* and A0 signals address low, high or both bytes. From Tl to T4, the M/IO* signal indicates a memory or I/O operation. At T2 the address is removed from the local bus and is sent to the output. The bus is then tristated. The read (RD*) control signal is also activated in T2 .

The read (RD) signal causes the addressed device to enable its data bus drivers. After RD* goes low, the valid data is available on the data bus.

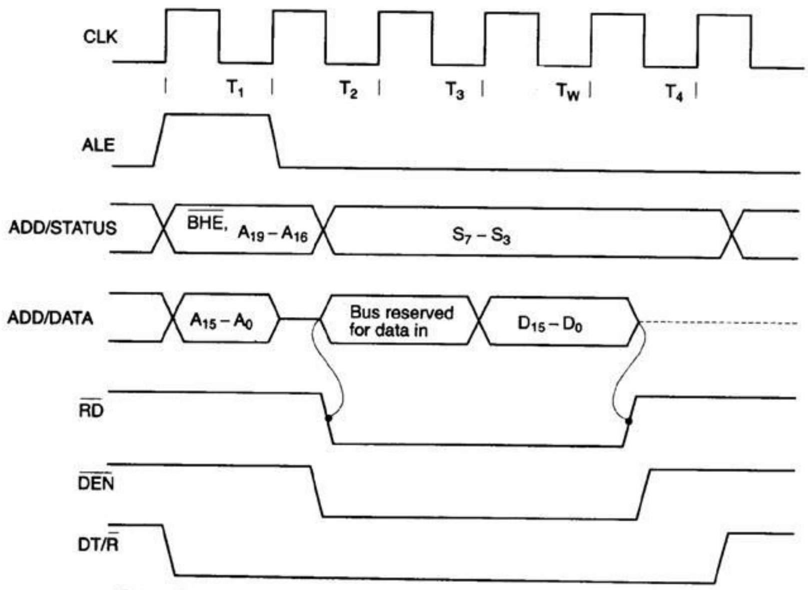
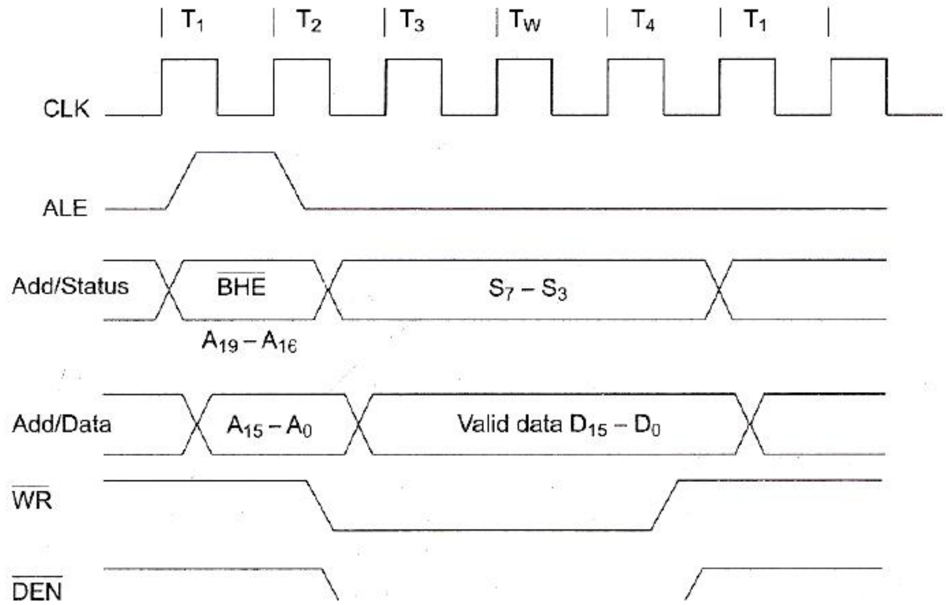


Fig.1.9(a) ***Read Cycle Timing Diagram for Minimum Mode***

The addressed device will drive the READY line high, when the processor returns the read signal to high level, the addressed device will again tristate its bus drivers.

Timing diagram for write cycle:

A write cycle also begins with the assertion of ALE and the emission of the address. The M/IO* signal is again asserted to indicate a memory or I/O operation. In T2 after sending the address in Tl the processor sends the data to be written to the addressed location. The data remains on the bus until middle of T4 state. The WR* becomes active at the beginning of T2.

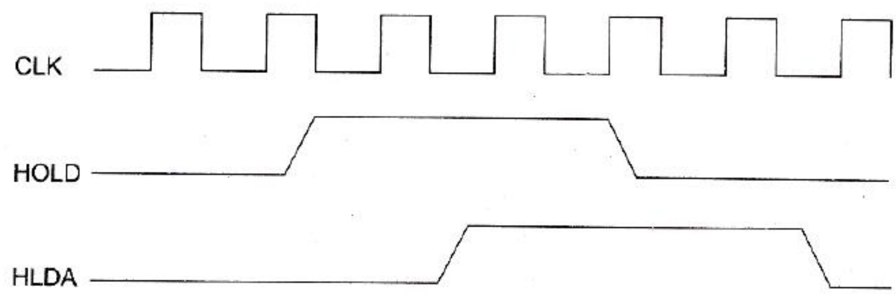


The BHE* and A0 signals are used to select the proper byte or bytes of memory or I/O word to be read or written. The M/IO*, RD* and WR* signals indicate the types of data transfer as specified in Table

M/IO	RD	WR	Transfer Type
0	0	1	I/O read
0	1	0	I/O write
1	0	1	Memory read
1	1	0	Memory write

HOLD Response Sequence

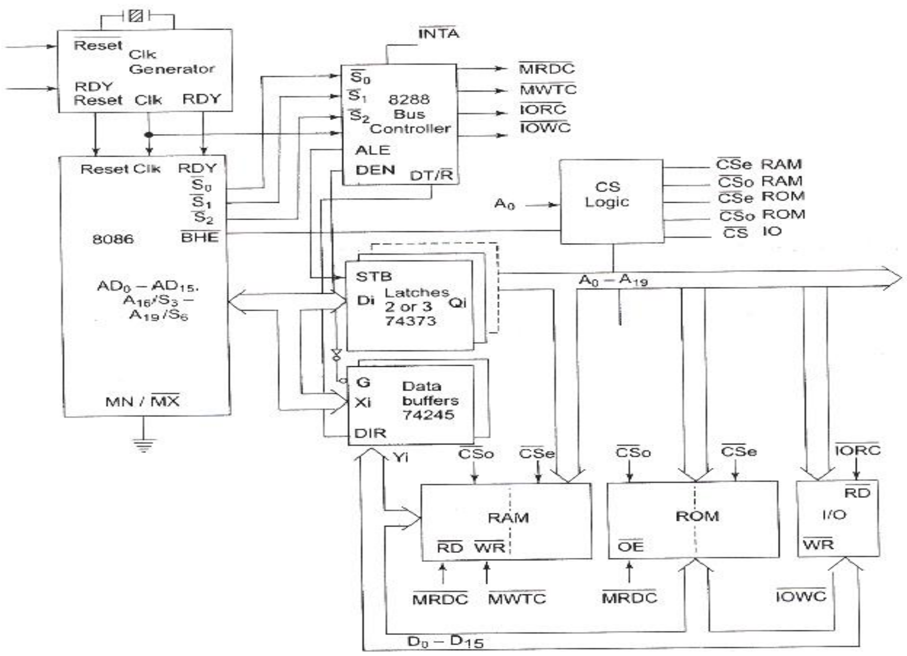
The HOLD pin is checked at the end of the each bus cycle. If it is received active by the processor before T4 of the previous cycle or during T1 state of the current cycle, the CPU activities HLDA in the next clock cycle and for the succeeding bus cycles, the bus will be given to another requesting master The control control of the bus is not regained by the processor until the requesting master does not drop the HOLD pin low. When the request is dropped by the requesting master, the HLDA is dropped by the processor at the trailing edge of the next clock as shown in fig



Maximum Mode 8086 System and Timings

In the maximum mode, the 8086 is operated by strapping the MN/MX* pin to ground. In this mode, the processor derives the status signals S2*, S1* and S0*. Another chip called bus controller derives the control signals using this status information. In the maximum mode, there may be more than one microprocessor in the system configuration. The other components in the system are the same as in the minimum mode system. The general system organization is as shown in the fig1.1

The basic functions of the bus controller chip IC8288, is to derive control signals like RD* and WR* (for memory and I/O devices), DEN*, DT/R*, ALE, etc. using the information made available by the processor on the status lines. The bus controller chip has input lines S2*, S1* and S0* and CLK. These inputs to 8288 are driven by the CPU. It derives the outputs ALE, DEN*, DT/R*, MWTC*, AMWC*, IORC*, IOWC* and AIOWC*. The AEN*, IOB and CEN pins are specially useful for multiprocessor systems. AEN* and IOB are generally grounded. CEN pin is usually tied to +5V.



INTA* pin is used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device.IORC*, IOWC* are I/O read command and I/O write command signals respectively. These signals enable an IO interface to read or write the data from or to the addressed port. The MRDC*, MWTC* are memory read command and memory write command signals respectively and may be used as memory read and write signals. All these command signals instruct the memory to accept or send data from or to the bus. For both of these write command signals, the advanced signals namely AIOWC* and AMWTC* are available. They also serve the same purpose, but are activated one clock cycle earlier than the IOWC* and MWTC* signals, respectively. The maximum mode system is shown in fig. 1.1.

The maximum mode system timing diagrams are also divided in two portions as read (input) and write (output) timing diagrams. The address/data and address/status timings are similar to the minimum mode. ALE is asserted in T1, just like minimum mode. The only difference lies in the status signals used and the available control and advanced command signals. The fig. 1.2 shows the maximum mode timings for the read operation while the fig. 1.3 shows the same for the write operation.

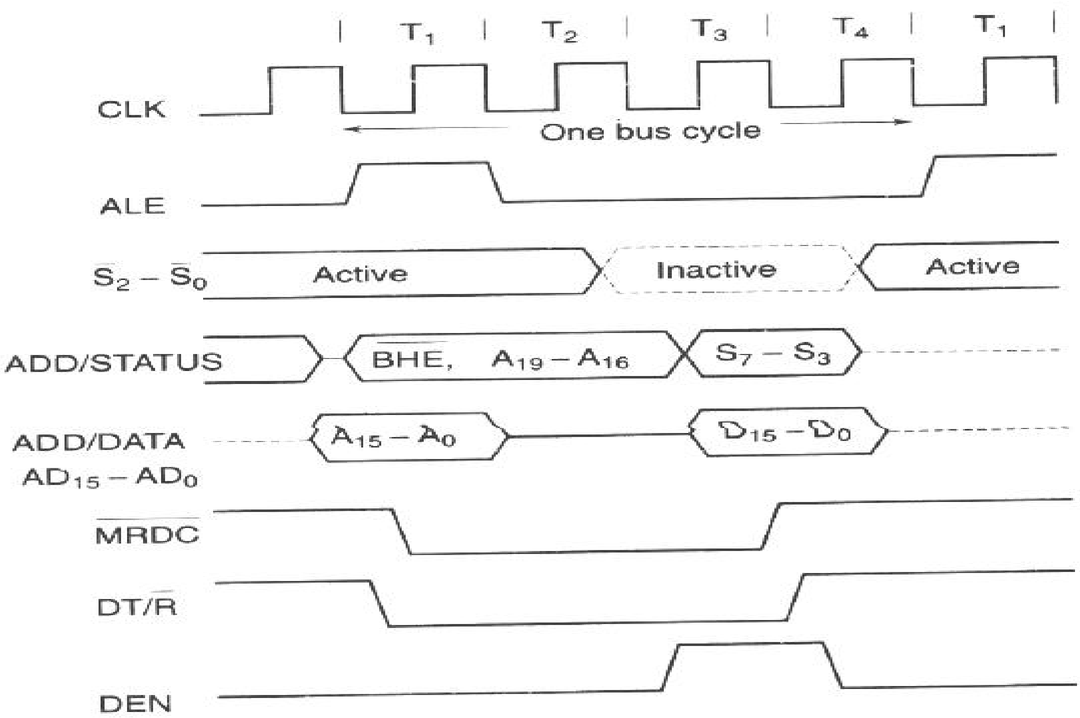


Fig. 1.2 Memory Read Timing in Maximum Mode

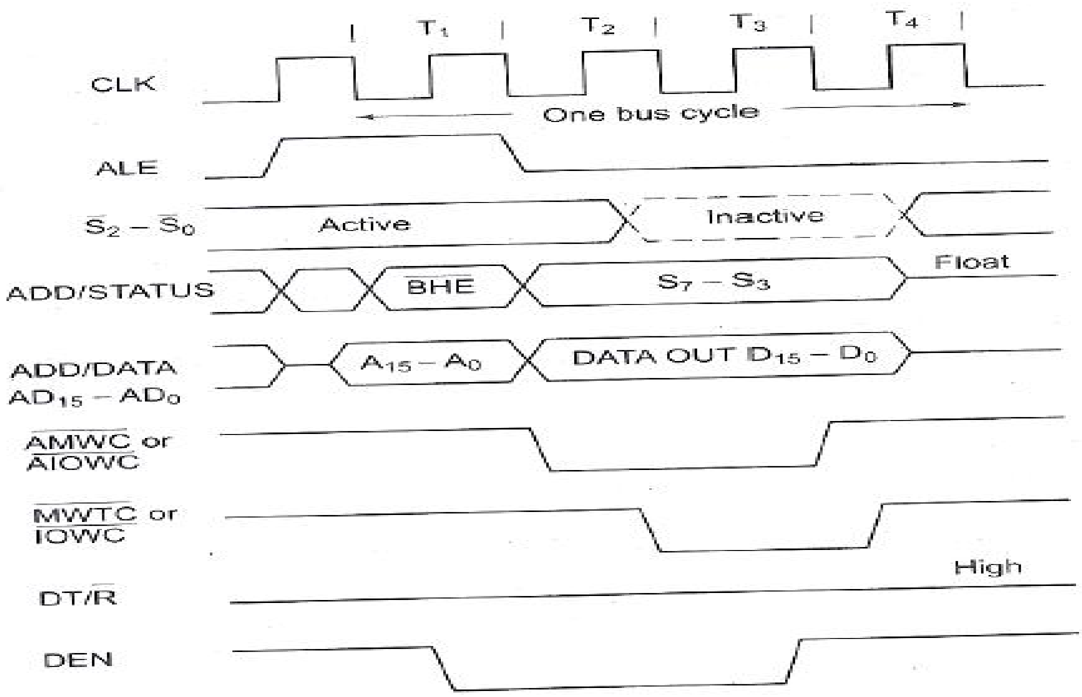


Fig. 1.3 Memory Write Timing in Maximum Mode

UNIT -II

- Instruction Set and Assembly Language Programming of 8086
- Instruction formats, Addressing modes,
- Instruction Set
- Assembler Directives,
- Procedures, Macros
- Simple Programs involving Logical
- Branch and Call Instructions
- Sorting Evaluating Arithmetic Expressions
- String Manipulations

UNIT-II

The instruction format contains two fields

- operation code / opcode
- Operand field

OPERATION CODE / OPCODE:

- It indicates the type of the operation to be performed by CPU
- Example : MOV , ADD …

OPERAND:

- The CPU executes the instruction using the information resides in these fields .

There are six general formats of instructions in 8086 instruction set.
The instruction of 8086 vary from 1to 6 bytes length

ONE BYTE INSTRUCTION:

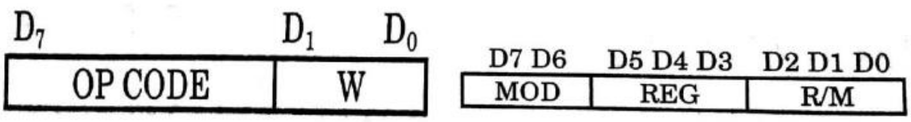
- It is only one byte long and may have implied data or register operands.
- The least three significant 3 bits of the opcode are used for specifying register operand if any otherwise all the 8 bits form an opcode and the operands are implied.

REGISTER TO REGISTER

- The format is 2 byte long
- The first byte of the code specifies the opcode and width
- The second byte of the code shows the register operand and R/M field
- The Register represented by REG is one of the operands . The R/M field specifies another register or memory location .ie the other operand

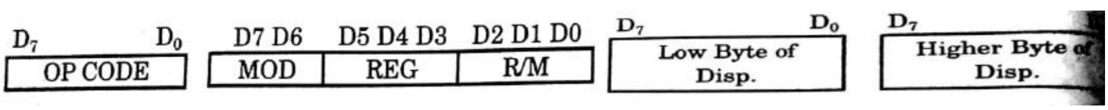
REGISTER TO/FROM MEMORY WITH NO DISPLACEMENT

- The format is 2 byte long
- This is similar to the register to register format except for the MOD field is shown.
- The MOD field shows the mode of addressing



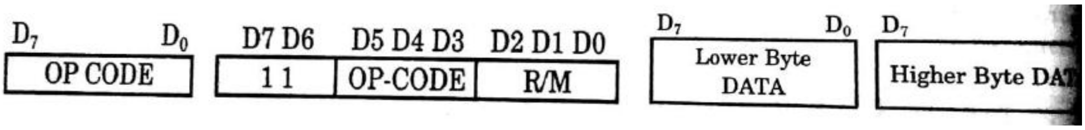
REGISTER TO/FROM MEMORY WITH DISPLACEMENT

- The format contains one or two additional bytes for displacement along with 2 bytes Register to/from memory with no displacement.



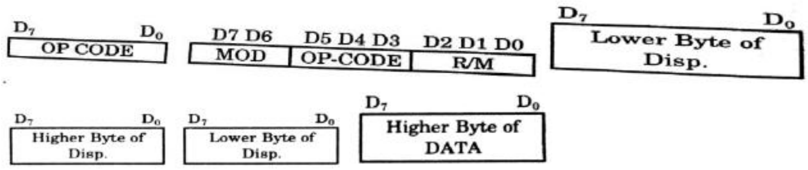
IMMEDIATE OPERAND TO REGISTER

- The first byte as well as the 3 bits from the second byte which are used for REG field in case of Register to register format or used for OPCODE.
- It also contains one are two bytes of data.



IMMEDIATE OPERAND TO MEMORY WITH 16 BIT DISPLACEMENTS

- It requires 5 to 6 bytes for coding
- The first two bytes contains the information regarding OPCODE,MOD and R/M fields
- The remaining 4 bytes contains 2 bytes of displacement and 2 bytes of data

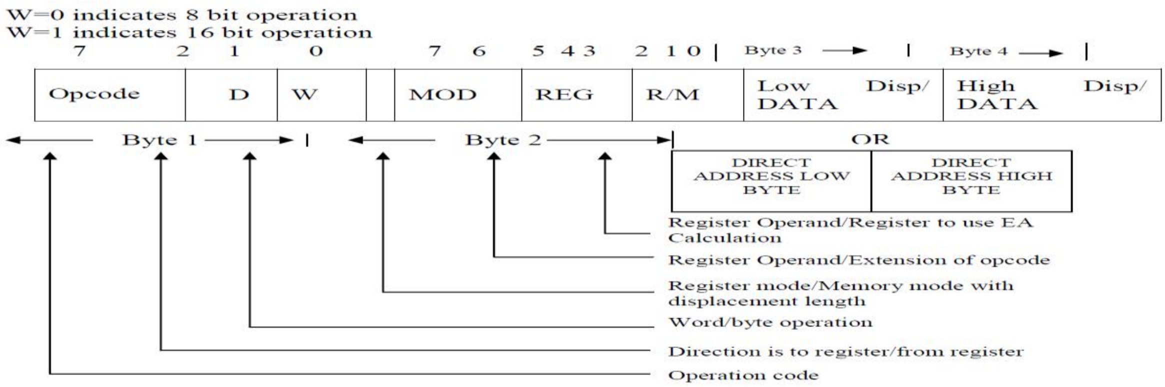


The 8086 instruction sizes vary from one to six bytes. Depending on the type of coding, an instruction may have more than one Hexcode, (not unique as in 8085)

The OP code field occupies 6-bits.It defines the operation to be carried out by the instruction.

Register Direct bit (D) occupies one bit. It defines whether the register operand in byte 2 is the source or destination operand.
D=1 Specifies that the register operand is the destination operand.
D=0 indicates that the register is a source operand.
Data size bit (W) defines whether the operation to be performed is an 8 bit or 16 bit data

W=0 indicates 8 bit operation
W=1 indicates 16 bit operation



This byte contains 3 fields. These are the mode (MOD) field, the register (REG) field and the Register/Memory (R/M) field.

MOD (2 bits)	Interpretation
00	Memory mode with no displacement follows except for 16 bit displacement when R/M=110
01	Memory mode with 8 bit displacement
10	Memory mode with 16 bit displacement
11	Register mode (no displacement)

Register field occupies 3 bits. It defines the register for the first operand which is specified as source or destination by the D bit.

REG	W=0	W=1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

The R/M field occupies 3 bits. The R/M field along with the MOD field defines the second operand as shown below.

MOD 11

R/M	W=0	W=1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Effective Address Calculation

R/M	MOD=00	MOD 01	MOD 10
000	(BX) + (SI)	(BX)+(SI)+D8	(BX)+(SI)+D16
001	(BX)+(DI)	(BX)+(DI)+D8	(BX)+(DI)+D16
010	(BP)+(SI)	(BP)+(SI)+D8	(BP)+(SI)+D16
011	(BP)+(DI)	(BP)+(DI)+D8	(BP)+(DI)+D10
100	(SI)	(SI) + D8	(SI) + D16
101	(DI)	(DI) + D8	(DI) + D16
110	Direct address	(BP) + D8	(BP) + D16
111	(BX)	(BX) + D8	(BX) + D16

In the above, encoding of the R/M field depends on how the mode field is set. If MOD=11 (register to register mode), this R/M identifies the second register operand.

MOD / R/M	Memory Mode (EA Calculation)			Register Mode	
	00	01	10	W=0	W=1
000	(BX)+(SI)	(BX)+(SI)+d8	(BX)+(SI)+d16	AL	AX
001	(BX) + (DI)	(BX)+(DI)+d8	(BX)+(DI)+d16	CL	CX
010	(BP)+(SI)	(BP)+(SI)+d8	(BP)+(SI)+d16	DL	DX
011	(BP)+(DI)	(BP)+(DI)+d8	(BP)+(DI)+d16	BL	BX
100	(SI)	(SI) + d8	(SI) + d16	AH	SP
101	(DI)	(DI) + d8	(DI) + d16	CH	BP
110	d16	(BP) + d8	(BP) + d16	DH	SI
111	(BX)	(BX) + d8	(BX) + d16	BH	DI

MOD selects memory mode, then R/M indicates how the effective address of the memory operand is to be calculated. Bytes 3 through 6 of an instruction are optional fields that

segment override prefix byte (SOP byte) to start with. The SOP byte is **001 xx 110**, where SR value is provided as per table shown below.

xx	**Segment register**
00	ES
01	CS
10	SS
11	DS

To specify DS register, the SOP byte would be 001 11 110 = 3E H. Thus the 5 byte code for this instruction would be 3E 89 96 45 23 H.

SOP	**Opcode**	**D**	**W**	**MOD**	**REG**	**R/M**	**LB disp.**	**HD disp.**
3EH	1000 10	0	1	10	010	110	45	23

Suppose we want to code MOV SS : 2345 (BP), DX. This generates only a 4 byte code, without SOP byte, as SS is already the default segment register in this case.

Example 5:
Give the instruction template and generate code for the instruction ADD 0FABE [BX], [DI], DX (code for ADD instruction is 000000)

ADD 0FABE [BX] [DI], DX
Here we have to specify DX using REG field. The bit D is 0, indicating that DX is the source register. The REG field must be 010 to indicate DX register. The w must be 1 to indicate it is a word operation. FABE (BX + DI) is specified using MOD value of 10 and R/M value of 001 (from the summary table). The 4 byte code for this instruction would be

Opcode	D	W	MOD	REG	R/M	16 bit disp.		=01 91 BE FAH
000000	0	1	10	010	001	BEH	FAH	

Example 6 :
Give the instruction template and generate the code for the instruction MOV AX, [BX]
(Code for MOV instruction is 100010)
AX destination register with D=1 and code for AX is 000 [BX] is specified using 00 Mode and R/M value 111. It is a word operation

Opcode	D	W	Mod	REG	R/M	=8B 07H
100010	1	1	00	000	111	

INPUT/OUTPUT INSTRUCTIONS:

IN acc, port : In transfers a byte or a word from input port to the AL register or the AX register respectively. The port number my be specified either with an immediate byte constant, allowing access to ports numbered 0 through 255 or with a number previously placed in the DX register allowing variable access (by changing the value in DX) to ports numbered from 0 through 65,535.

In Operands	**Example**
acc, immB	IN AL, 0E2H (OR) IN AX, PORT
acc, DX	IN AX, DX (OR) IN AL, DX

ADDRESSING MODES OF 8086

According two the flow of instructions may be categorized as

1. Sequential Control flow instructions
2. Control transfer instructions

Sequential control flow instructions are the instructions which after execution transfer control to the next instruction appearing immediately. The control transfer instructions transfer control to some predefined address or the address somehow specified in the instruction after their execution.

What is addressing mode?

The different ways in which a source operand is denoted in an instruction are known as addressing mode the addressing modes for sequential control flow instructions are

1. Immediate Addressing Mode
2. Direct Addressing mode
3. Register Addressing mode
4. Register Indirect Addressing mode
5. Indexed Addressing Mode
6. Register Relative addressing mode
7. Based indexed addressing mode
8. Relative based indexed Addressing mode

IMMEDIATE ADDRESSING MODE

The addressing mode in which the data operand is a part of the instruction itself is known as immediate addressing mode.

Example

MOV DL, 08H

The 8-bit data (08_H) given in the instruction is moved to DL

$(DL) \leftarrow 08_H$

MOV AX, 0A9FH

The 16-bit data (0A9FH) given in the instruction is moved to AX register

$(AX) \leftarrow$ 0A9FH

DIRECT ADDRESSING MODE

The addressing mode in which the effective address of the memory location at which the data operand is stored is given in the instruction. The effective address (Offset) is just a 16-bit number written directly in the instruction.

Example:MOV BX, [1354H]

MOV BL, [0400H]

The square brackets around the 1354H denote the contents of the memory location. When executed, this instruction will copy the contents of the memory location into BX register. This addressing mode is called direct because the displacement of the operand from the segment base is specified directly in the instruction.

REGISTER ADDRESSING MODE

The instruction will specify the name of the register which holds the data to be operated by the instruction. All registers except IP may be used in this mode

Example:

MOV CL, DH
The content of 8-bit register DH is moved to another 8-bit register CL
(CL) ← (DH)

REGISTER INDIRECT ADDRESSING MODE

This addressing mode allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI & SI.

Example

MOV AX, [BX]; suppose the register BX contains 4895H, then the contents
; 4895H are moved to AX
ADD CX, {BX}

INDEXED ADDRESSING MODE

In this addressing mode, the operands offset address is found by adding the contents of SI or DI register and 8-bit/16-bit displacements. DS and ES are the default segments for index registers SI and DI respectively. This is the special case of the of register indirect addressing mode.

Example

MOV BX, [SI+16], ADD AL, [DI+16]

REGISTER RELATIVE ADDRESSING MODE

In register relative Addressing, BX, BP, SI and DI is used to hold the base value for effective address and a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction. In case of 8-bit displacement, it is sign extended to 16-bit before adding to the base value. When BX holds the base value of EA, 20-bit physical address is calculated from BX and DS.When BP holds the base value of EA, BP and SS is used.

Example:

MOV AX, [BX + 08H]   MOV AX, 08H [BX]

BASED INDEXED ADDRESSING MODE

In this addressing mode, the offset address of the operand is computed by summing the base register to the contents of an Index register. The default segment registers may be ES or DS

Example:

MOV DX, [BX + SI]   MOV DX, [BX][SI]

RELATIVE BASED INDEXED ADDRESSING MODE

In this addressing mode, the operands offset is computed by adding the base register contents. An Index registers contents and 8 or 16-bit displacement.

Example

MOV AX, [BX+DI+08]
ADD CX, [BX+SI+16]

CONTROL TRANSFER INSTRUCTIONS ADDRESSING MODES /BRANCH ADDRESSING MODE

The control transfer instructions transfer control to some predefined address or the address somehow specified in the instruction after their execution

Examples : INT , CALL ,RET and JUMP instructions

The control transfer instruction the addressing modes depend upon whether destination location is within the same segment or a different one .It also depends on the method of passing the destination address to the processor

Basically there are two methods for passing control transfer instructions

1. Intersegment addressing mode
2. Intrasegment addressing mode

INTRASEGMENT ADDRESSING MODE

If the destination location is within the same segment the mode is called intrasegment addressing mode

There are two types

1. Intrasegement direct mode
2. Intrasegment indirect mode

INTRASEGMENT DIRECT MODE:

In this mode the address to which the control is to be transferred lies within the segment in which the control transfer instruction lies and appears directly in the instruction as an immediate displacement value .The displacement is computed relative to the content of the instruction pointer IP.

JMP SHORT LABEL;

is a control transfer instruction following intra segment direct mode. Here, SHORT LABEL represents a signed displacement.

INTRASEGMENT INDIRECT MODE :

In this mode the displacement to which the control is to be transferred is in the same segment in which the control transfer instruction lies but it is passed to the instruction indirectly Here the branch address is found as the content of a register or a memory location .

Example

JMP [AX]

INTERSEGMENT ADDRESSING MODE
If the destination location is in the different segment the mode is called intersegment addressing mode
There are two types
1. Intersegment direct mode
2. Intersegment indirect mode

INTERSEGMENT DIRECT MODE:
In this mode the address to which the control is to be transferred is in a different segment this addressing mode provides a means of branching from one code segment to another code segment. Here the CS and IP of the destination address are specified directly in the instruction.
Example
JMP 2000H: 3000H;
INTERSEGMENT INDIRECT MODE :
In this the address to which the control is to be transferred lies in a different segment and it is passed to the instruction indirectly .Content of memory block containing four bytes IP(LSB) ,IP(MSB),CS(LSB) and CS(MSB) sequentially The starting address of the memory block may be referred using any of the addressing mode except immediate mode .
Example
JMP [5000H];

INSTRUCTION SET OF 8086

The 8086 microprocessor supports 8 types of instructions −

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

1. 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.
- **PPUSH** − 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.
- **PUSHA** − Used to put all the registers into the stack.
- **POPA** − 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.

2. 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.

3. LOGICAL 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.

4. 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.

5. 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

6. 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.

7. 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

8. 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

ASSEMBLER DIRECTIVES

Assembler directives are the Instructions to the Assembler, linker and loader regarding the program being executed. also called ‘pseudo instructions. Control the generation of machine codes and organization of the program; but no machine codes are generated for assembler directives.

They are used to

› specify the start and end of a program
› attach value to variables
› allocate storage locations to input/ output data
› define start and end of segments, procedures, macros etc..

ASSUME

Used to tell the assembler the name of the logical segment it should use for a specified segment. You must tell the assembler that what to assume for any segment you use in the program.

Example

ASSUME: CODE

Tells the assembler that the instructions for the program are in segment named CODE.

DB – Defined Byte

Used to declare a byte type variable or to set aside one or more locations of type byte in memory.

Example

PRICES DB 49H, 98H, 29H:

Declare array of 3 bytes named PRICES and initialize 3 bytes as shown.

DD – Define Double Word

Used to declare a variable of type doubleword or to reserve a memory location which can be accessed as doubleword.

DQ – Define Quadword

Used to tell the assembler to declare the variable as 4 words of storage in memory.

DT – Define Ten Bytes

Used to tell the assembler to declare the variable which is 10 bytes in length or reserve 10 bytes of storage in memory.

DW – Define Word

Used to tell the assembler to define a variable type as word or reserve word in memory.

DUP: used to initialize several locations and to assign values to location

END – End the Program

To tell the assembler to stop fetching the instruction and end the program execution.

ENDP – it is used to end the procedure.

ENDS – used to end the segment.

EQU – EQUATE

Used to give name to some value or symbol.

EVEN – Align On Even Memory Address

Tells the assembler to increment the location counter to the next even address if it is not already at an even address.

EXTRN

Used to tell the assembler that the name or labels following the directive are in some other assembly module.

GLOBAL – Declares Symbols As Public Or Extrn

Used to make the symbol available to other modules.It can be used in place of EXTRN or PUBLIC keyword.

GROUP – Group related segment

Used to tell the assembler to group the logical segments named after the directive into one logical segment. This allows the content of all the segments to be accessed from the same group.

INCLUDE – include source code from file

Used to tell the assembler to insert a block of source code from the named file into the current source module. This shortens the source code.

LABEL

Used to give the name to the current value in the location counter. The LABEL directive must be followed by a term which specifies the type you want associated with that name.

LENGTH

Used to determine the number of items in some data such as string or array.

NAME

Used to give a specific name to a module when the programs consisting of several modules.

OFFSET

It is an operator which tells the assembler to determine the offset or displacement of named data item or procedure from the start of the segment which contains it.

ORG – Originate

Tells the assembler to set the location counter value.

Example, ORG 7000H sets the location counter value to point to 7000H location in memory.

$ is often used to symbolically represent the value of the location counter. It is used with ORG to tell the assembler to change the location according to the current value in the location counter. E.g. ORG $+100.

UNIT -III

I/O Interface

- 8255 PPI
- Various Modes of Operation and Interfacing to 8086
- D/A and A/D Converter
- Stepper motor
- Interfacing of DMA controller 8257

Interfacing with advanced devices

- Memory Interfacing to 8086
- Interrupt Structure of 8086
- Interrupt Vector Table, Interrupt Service Routine
- architecture of 8259.

Communication Interface

- Serial Communication Standards
- Serial Data Transfer Schemes
- 8251 USART Architecture and Interfacing.

UNIT-3

I/O Interface

Introduction:

Any application of a microprocessor based system requires the transfer of data between external circuitry to the microprocessor and microprocessor to the external circuitry. User can give information to the microprocessor based system using keyboard and user can see the result or output information from the microprocessor based system with the help of display device. The transfer of data between keyboard and microprocessor, and microprocessor and display device is called input/output data transfer or I/O data transfer. This data transfer is done with the help of I/O ports.

Input port:

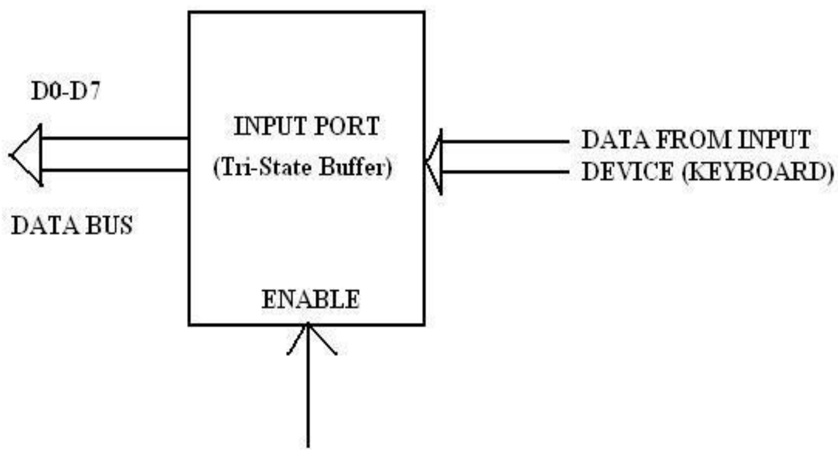


FIG.1 INPUT PORT

It is used to read data from the input device such as keyboard. The simplest form of input port is a buffer. The input device is connected to the microprocessor through buffer, as shown in the fig.1. This buffer is a tri-state buffer and its output is available only when enable signal is active. When microprocessor wants to read data from the input device (keyboard), the control signals from the microprocessor activates the buffer by asserting enable input of the buffer. Once the buffer is enabled, data from the input device is available on the data bus. Microprocessor reads this data by initiating read command.

Output port:

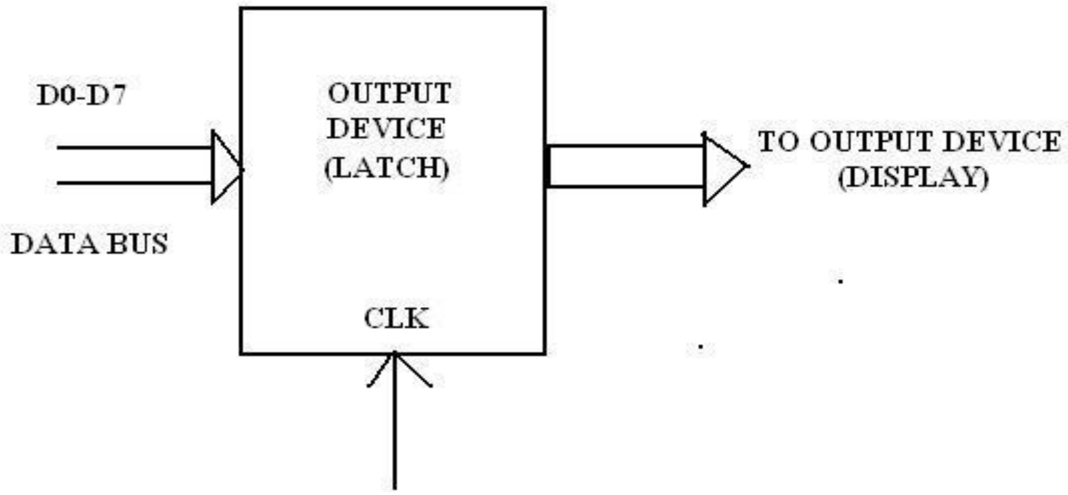


FIG.2 OUTPUT PORT

It is used to send data to the output device such as display from the microprocessor. The simplest form of output port is a latch. The output device is connected to the microprocessor through latch, as shown in the fig.2. When microprocessor wants to send data to the output device is puts the data on the data bus and activates the clock signal of the latch, latching the data from the data bus at the output of latch. It is then available at the output of latch for the output device.

Serial and Parallel Transmission:

In telecommunications, serial transmission is the sequential transmission of signal elements of a group representing a character or other entity of data. Digital serial transmissions are bits sent over a single wire, frequency or optical path sequentially. Because it requires less signal processing and less chance for error than parallel transmission, the transfer rate of each individual path may be faster. This can be used over longer distances as a check digit or parity bit can be sent along it easily.

In telecommunications, parallel transmission is the simultaneous transmission of the signal elements of a character or other entity of data. In digital communications, parallel transmission is the simultaneous transmission of related signal elements over two or more separate paths. Multiple electrical wires are used which can transmit multiple bits simultaneously, which allows for higher data transfer rates than can be achieved with serial transmission. This method is used internally within the computer, for example the internal buses, and sometimes externally for such things as printers, The major issue with this is "skewing" because the wires in parallel data transmission have slightly different properties (not intentionally) so some bits may arrive before others, which may corrupt the message. A parity bit can help to reduce this. However, electrical wire parallel data transmission is therefore less reliable for long distances because corrupt transmissions are far more likely.

Interrupt driven I/O:

In this technique, a CPU automatically executes one of a collection of special routines whenever certain condition exists within a program or a processor system. Example CPU gives response to devices such as keyboard, sensor and other components when they request for service. When the CPU is asked to communicate with devices, it services the devices. Example each time you type a character on a keyboard, a keyboard service routine is called. It transfers the character you typed from the keyboard I/O port into the processor and then to a data buffer in memory.

The interrupt driven I/O technique allows the CPU to execute its main program and only stop to service I/O device when it is told to do so by the I/O system as shown in fig.3. This method provides an external asynchronous input that would inform the processor that it should complete whatever instruction that is currently being executed and fetch a new routine that will service the requesting device. Once this servicing is completed, the processor would resume exactly where it left off.

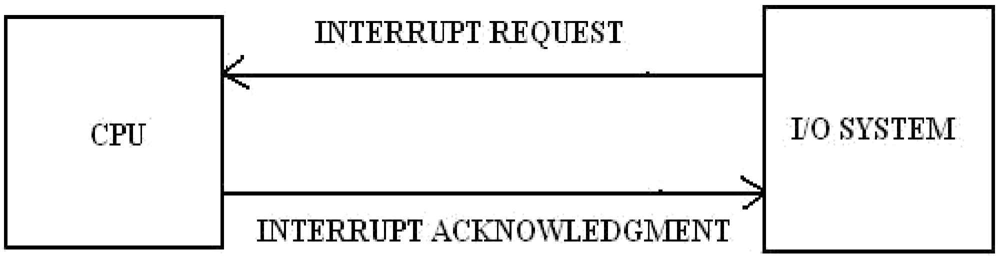


FIG.3 INTERRUPT DRIVEN I/O

An analogy to the interrupt concept is in the classroom, where the professor serves as CPU and the students as I/O ports. The classroom scenario for this interrupt analogy will be such that the professor is busy in writing on the blackboard and delivering his lecture.

The student raises his finger when he wants to ask a question (student requesting for service). The professor then completes his sentence and acknowledges student‟s request by saying “YES” (professor acknowledges the interrupt request). After acknowledgement from the professor, student asks the question and professor gives answer to the question (professor services the interrupt). After that professor continues its remaining lecture form where it was left.

PIO 8255:

The parallel input-output port chip 8255 is also called as programmable**peripheral input-output port.** The Intel‟s 8255 are designed for use with Intel‟s 8-bit, 16-bit and higher capability microprocessors. It has 24 input/output lineswhich may be individually programmed in two groups of twelve lines each, orthree groups of eight lines.

The two groups of I/O pins are named as Group A and Group B. Each of thesetwo groups contains a subgroup of eight I/O lines called as 8-bit port and anothersubgroup of four lines or a 4-bit port. Thus Group A contains an 8-bit port Aalong with a 4-bit port C upper.

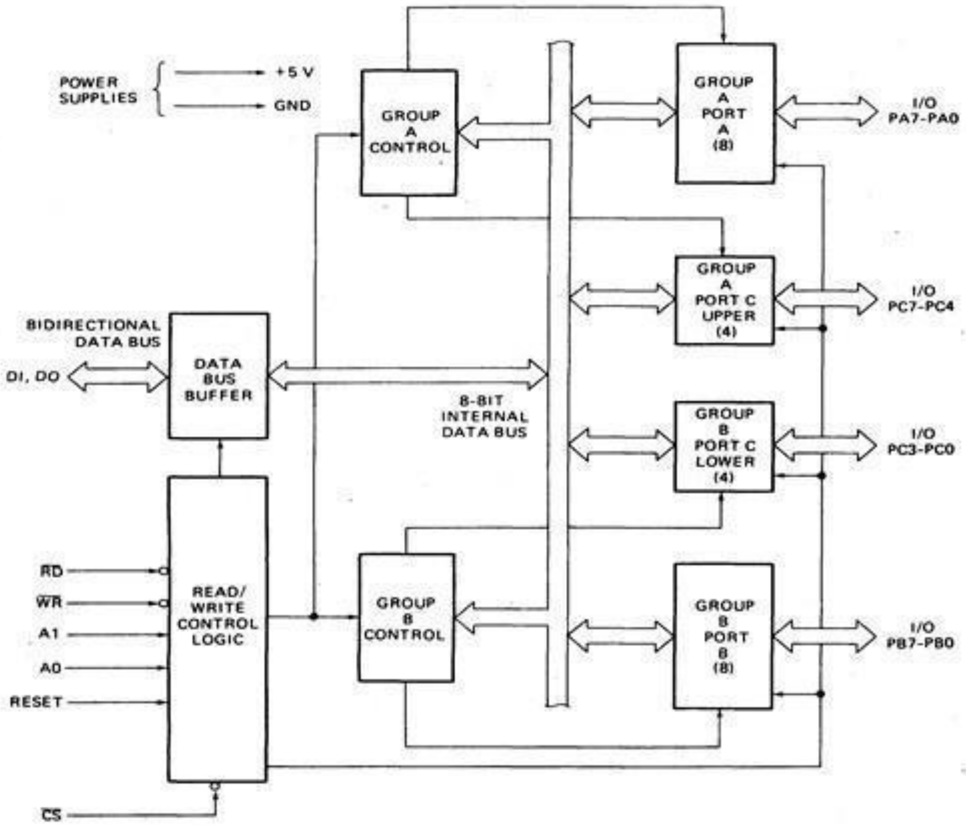
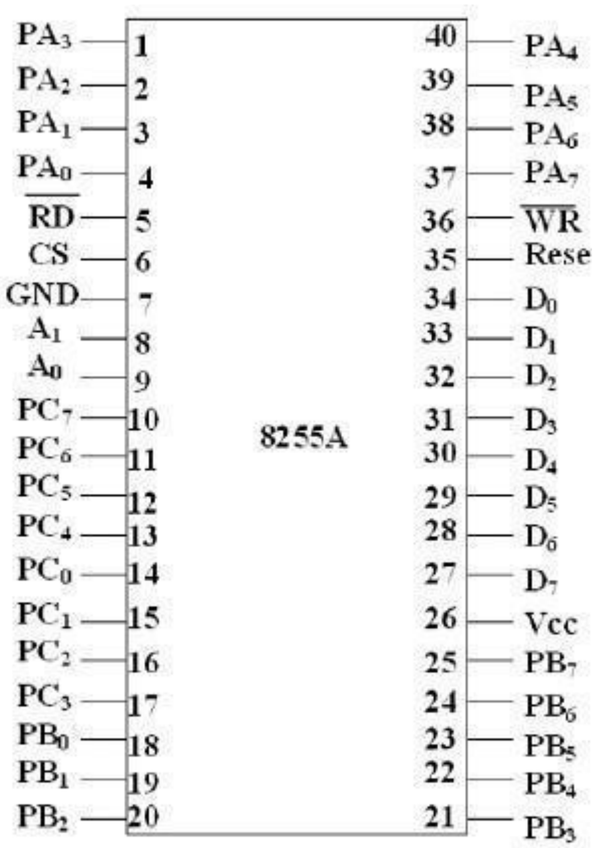


FIGURE Internal block diagram of 8255A programmable parallel port device. (*Intel Corporation*)

The port A lines are identified by symbols PA0-PA7 while the port C lines are identified as PC4-PC7 similarly. Group B contains an 8-bit port B, containing lines PB0-PB7 and a 4-bit port C with lower bits PC0-PC3. The port C upper and port C lower can be used in combination as an 8-bit port C. Both the port Cs is assigned the same address. Thus one may have either three 8-bit I/O ports or two 8-bit and two 4-bit I/O ports from 8255. All of these ports can function independently either as input or as output ports. This can be achieved by programming the bits of an internal register of 8255 called as control word register (CWR). The internal block diagram and the pin configuration of 8255 are shown in figs.

The 8-bit data bus buffer is controlled by the read/write control logic. The read/write control logic manages all of the internal and external transfer of both data and control words. RD, WR, A1, A0 and RESET are the inputs, provided by the microprocessor to READ/WRITE control logic of 8255. The 8-bit, 3-state bidirectional buffer is used to interface the 8255 internal data bus with the external system data bus. This buffer receives or transmits data upon the execution of input or output instructions by the microprocessor. The control words or status information is also transferred through the buffer.

Pin Diagram of 8255A



8255A Pin Configuration

The pin configuration of 8255 is shown in fig.

- The port A lines are identified by symbols PA0-PA7 while the port C lines are
- Identified as PC4-PC7. Similarly, Group B contains an 8-bit port B, containing lines PB0-PB7 and a 4-bit port C with lower bits PC0- PC3. The port C upper and port C lower can be used in combination as an 8-bit port C.
- Both the port C is assigned the same address. Thus one may have either three 8-bit I/O ports or two 8-bit and two 4-bit ports from 8255. All of these ports can function independently either as input or as output ports. This can be

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

The 8-bit data bus buffer is controlled by the read/write control logic. The read/write control logic manages all of the internal and external transfers of both data and control words.

RD,WR, A1, A0 and RESET are the inputs provided by the microprocessor to the READ/ WRITE control logic of 8255. The 8-bit, 3-state bidirectional buffer is used to interface the 8255 internal data bus with the external system data bus.

This buffer receives or transmits data upon the execution of input or output instructions by the microprocessor. The control words or status information is also transferred through the buffer.

The signal description of 8255 is briefly presented as follows:

PA7-PA0: These are eight port A lines that acts as either latched output or buffered input lines depending upon the control word loaded into the control word register.

PC7-PC4: Upper nibble of port C lines. They may act as either output latches or input buffers lines.

This port also can be used for generation of handshake lines in mode1 or mode2.

PC3-PC0: These are the lower port C lines; other details are the same as PC7-PC4 lines.

PB0-PB7: These are the eight port B lines which are used as latched output lines or buffered input lines in the same way as port A.

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

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

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

D0-D7: These are the data bus lines those carry data or control word to/from the microprocessor.

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

A1-A0: These are the address input lines and are driven by the microprocessor.

These lines A1-A0 with RD, WR and CS from the following operations for 8255. These address lines are used for addressing any one of the four registers, i.e. three ports and a control word register as given in table below.

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

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	Input (Read) cycle
0	1	0	0	0	Port A to Data bus
0	1	0	0	1	Port B to Data bus
0	1	0	1	0	Port C to Data bus
0	1	0	1	1	CWR to Data bus

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	Output (Write) cycle
1	0	0	0	0	Data bus to Port A
1	0	0	0	1	Data bus to Port B
1	0	0	1	0	Data bus to Port C
1	0	0	1	1	Data bus to CWR

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	Function
X	X	1	X	X	Data bus tristated
1	1	0	X	X	Data bus tristated

Control Word Register

Modes of Operation of 8255

These are two basic modes of operation of 8255. I/O mode and Bit Set-Reset mode (BSR).

In I/O mode, the 8255 ports work as programmable I/O ports, while in BSR mode only port C (PC0-PC7) can be used to set or reset its individual port bits.

Under the I/O mode of operation, further there are three modes of operation of 8255, so as to support different types of applications, mode 0, mode 1 and mode 2.

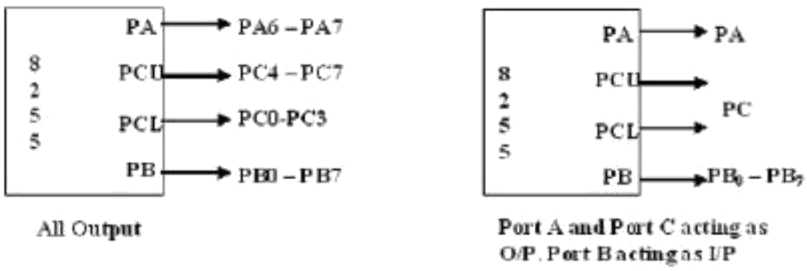
BSR Mode: In this mode any of the 8-bits of port C can be set or reset depending on D0 of the control word. The bit to be set or reset is selected by bit select flags D3, D2 and D1 of the CWR as given in table.

I/O Modes:

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

D_3	D_2	D_1	Selected bits of port C
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

BSR Mode : CWR Format



Mode 0

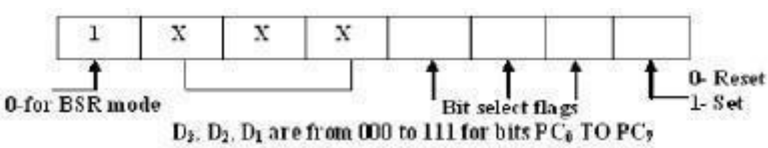
The salient features of this mode are as listed below:

1. Two 8-bit ports (port A and port B) and two 4-bit ports (port C upper and lower) are available. The two 4-bit ports can be combined used as a third 8-bit port.
2. Any port can be used as an input or output port.
3. Output ports are latched. Input ports are not latched.
4. A maximum of four ports are available so that overall 16 I/O configurations arepossible.

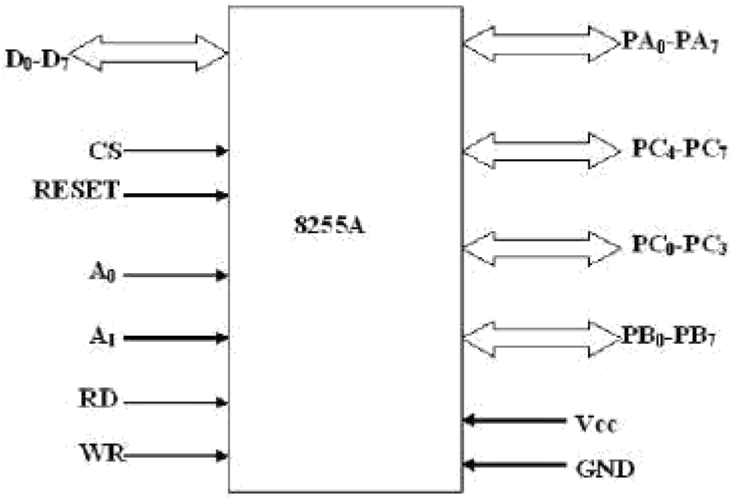
All these modes can be selected by programming a register internal to 8255known as CWR.

The control word register has two formats. The first format is valid for I/O modesof operation, i.e. modes 0, mode 1 and mode 2 while the second format is validfor bit set/reset (BSR) mode of operation.

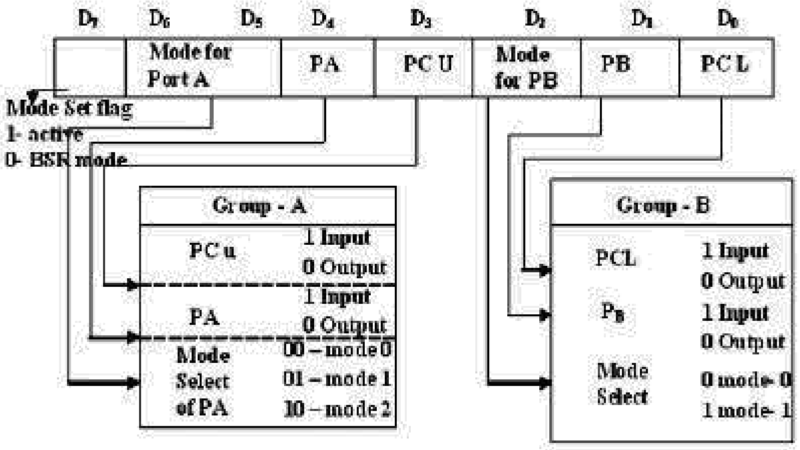
These formats are shown in followingfig.



I/O Mode Control Word Register Format and
BSR Mode Control Word Register Format



Signals of 8255



Control Word Format of 8255

b) Mode 1: (Strobed input/output mode) In this mode the handshaking control the input and output action of the specified port. Port C lines PC0-PC2, provide strobe orhandshake lines for port B. This group which includes port B and PC0-PC2 is called asgroup B for Strobed data input/output. Port C lines PC3-PC5 provides strobe lines for portA.This group including port A and PC3-PC5 from group A. Thus port C is utilized forgenerating handshake signals.

The salient features of mode 1 are listed as follows:

1. Two groups – group A and group B are available for strobed data transfer.
2. Each group contains one 8-bit data I/O port and one 4-bit control/data port.
3. The 8-bit data port can be either used as input and output port. The inputs andoutputs both are latched.
4. Out of 8-bit port C, PC0-PC2 are used to generate control signals for port B andPC3-PC5 are used to generate control signals for port A. the lines PC6, PC7 may be used as independent data lines.

The control signals for both the groups in input and output modes areexplained as follows:

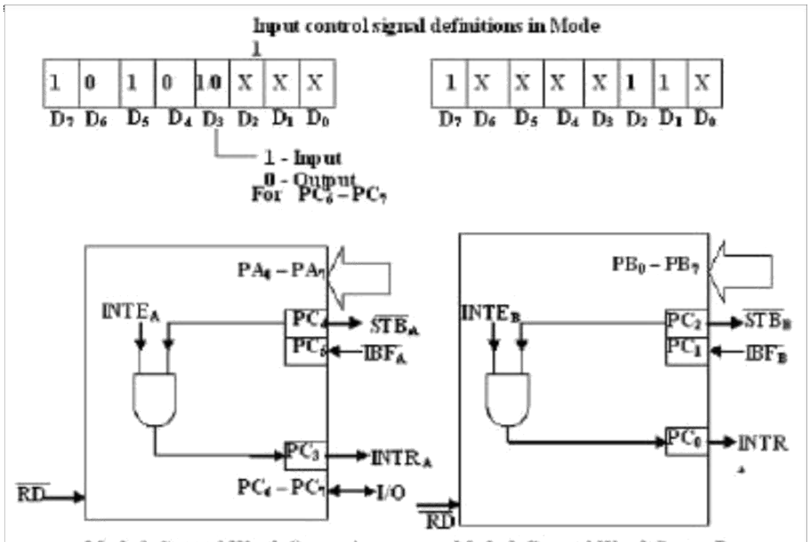
Input control signal definitions (mode 1):

- **STB** (Strobeinput) – If this lines falls to logic low level, the data available at 8-bit input port is loaded into input latches.
- **IBF** (Input buffer full) – If this signal rises to logic 1, it indicates that data hasbeen loaded into latches, i.e. it works as an acknowledgement. IBF is set by a lowon STB and is reset by the rising edge of RD input.
- **INTR** (Interruptrequest) – This active high output signal can be used tointerrupt the CPU whenever an input device requests the service. INTR is set by ahigh STBpin and a high at IBF pin. INTE is an internal flag that can be controlledby the bit set/reset mode of either PC4 (INTEA) or PC2 (INTEB) as shown in fig.
- INTR is reset by a falling edge of RD input. Thus an external input device can berequest the service of the processor by putting the data on the bus and sending thestrobe signal.

Output control signal definitions (mode 1):

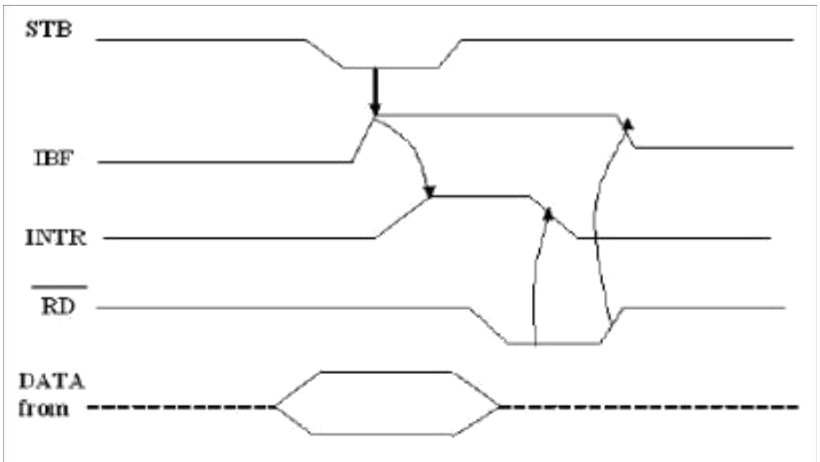
- **OBF** (Output buffer full) – This status signal, whenever falls to low, indicatesthat CPU has written data to the specified output port. The OBF flip-flop will beset by a rising edge of WR signal and reset by a low going edge at the ACKinput.
- ACK (Acknowledgeinput) – ACK signal acts as an acknowledgement to begiven by an output device. ACK signal, whenever low, informs the CPU that thedata transferred by the CPU to the output device through the port is received bythe output device.
- **INTR** (Interruptrequest) – Thus an output signal that can be used to interruptthe CPU when an output device acknowledges the data received from the CPU.INTR is set when ACK, OBF and INTE are 1. It is reset by a

fallingedge on WRinput. The INTEA and INTEB flags are controlled by the bit set-reset mode ofPC6 and PC2 respectively.

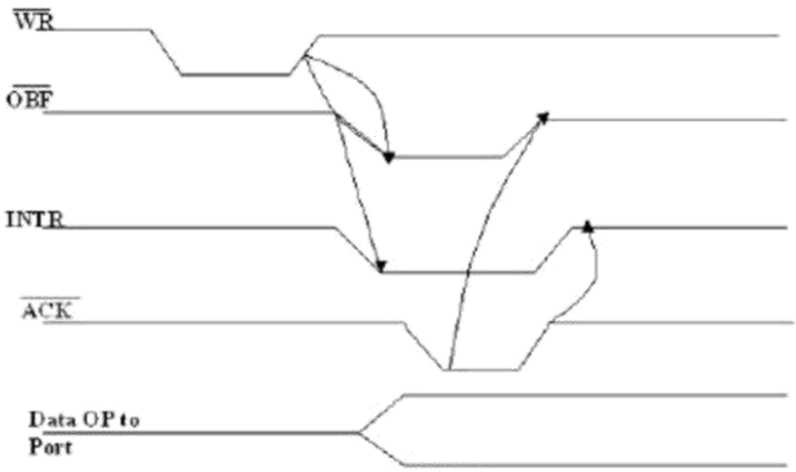


Mode 1 Control Word Group A I/P

Mode 1 Control Word Group B I/P

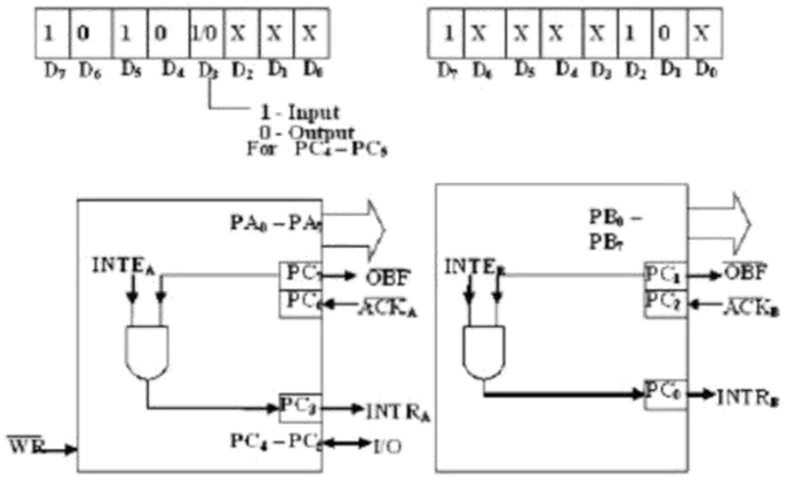


Mode 1 Strobed Input Data Transfer



Mode 1 Strobed Data Output

Output control signal definitions Mode 1



Mode 1 Control Word Group A Mode 1 Control Word Group B

c) Mode 2 (Strobed bidirectional I/O): This mode of operation of 8255 is alsocalled as strobed bidirectional I/O. This mode of operation provides 8255 with additional features for communicating with a peripheral device on an 8-bit databus. Handshaking signals are provided to maintain proper data flow andsynchronization between the data transmitter and receiver. The interruptgeneration and other functions are similar to mode 1.

In this mode, 8255 is a bidirectional 8-bit port with handshake signals. The Rdand WR signals decide whether the 8255 is going to operate as an input port oroutput port.

The Salient features of Mode 2 of 8255 are listed as follows:

1. The single 8-bit port in group A is available.
2. The 8-bit port is bidirectional and additionally a 5-bit control port is available.
3. Three I/O lines are available at port C.(PC2 – PC0)
4. Inputs and outputs are both latched.

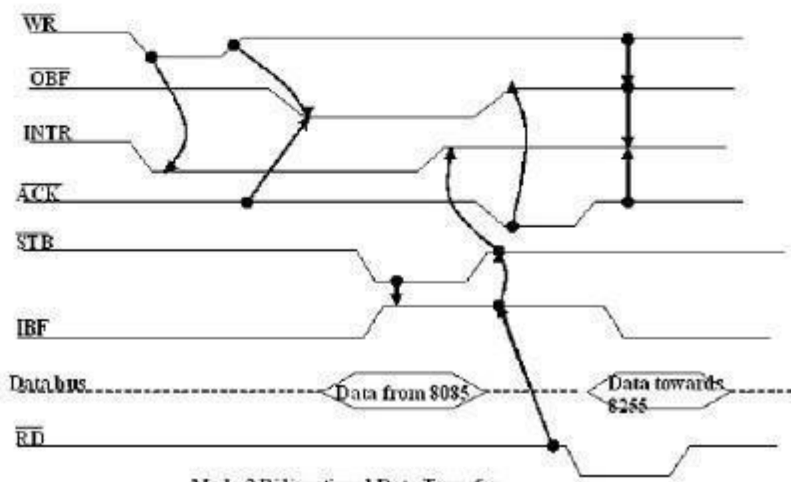
5. The 5-bit control port C (PC3-PC7) is used for generating / accepting handshakesignals for the 8-bit data transfer on port A.

Control signal definitions in mode 2:

- **INTR** – (Interrupt request) As in mode 1, this control signal is active high and isused to interrupt the microprocessor to ask for transfer of the next data byteto/from it. This signal is used for input (read) as well as output (write) operations.
- **Control Signals for Output operations**:
- OBF (Output buffer full) – This signal, when falls to low level, indicates that theCPU has written data to port A.
- ACK (Acknowledge) This control input, when falls to logic low level, Acknowledges that the previous data byte is received by the destination and nextbyte may be sent by the processor. This signal enables the internal tristate buffersto send the next data byte on port A.
- **INTE1** (A flag associated with OBF) This can be controlled by bit set/resetmode with PC6.

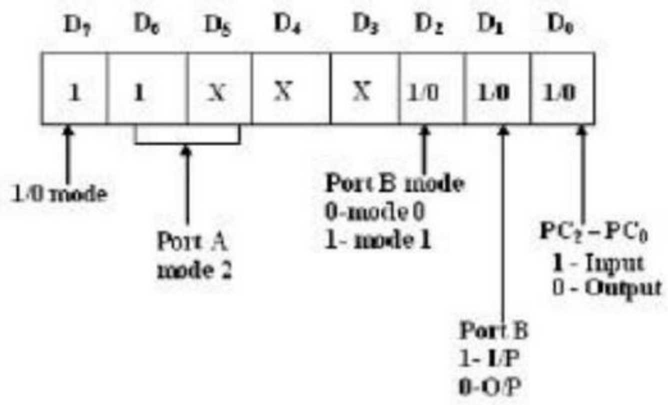
Control signals for input operations:

- STB (Strobe input)a low on this line is used to strobe in the data into the inputLatches of 8255.
- **IBF** (Input buffer full) when the data is loaded into input buffer, this signal risesto logic „1". This can be used as an acknowledge that the data has been receivedby the receiver.
- The waveforms in fig show the operation in Mode 2 for output as well as inputport.
- Note: WR must occur before ACK and STB must be activated before RD.

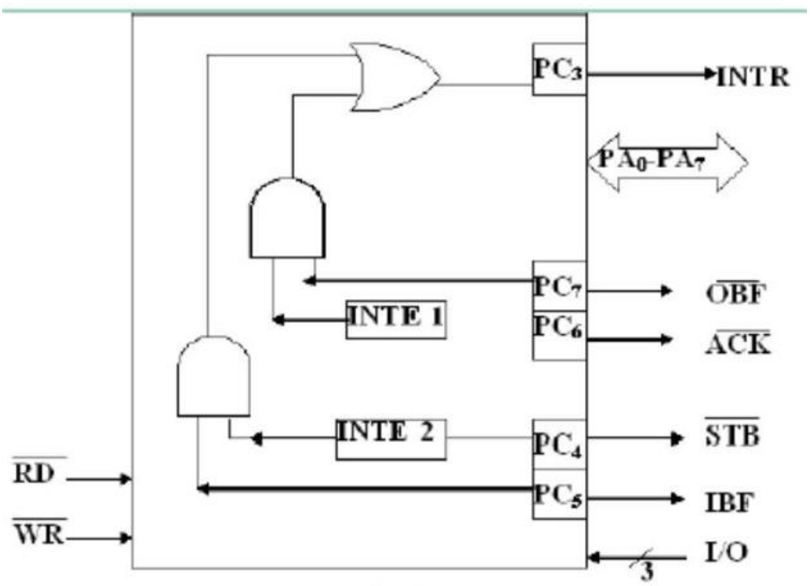


Mode 2 Bidirectional Data Transfer

- The following fig shows a schematic diagram containing an 8-bit bidirectionalport, 5-bit control port and the relation of INTR with the control pins. Port B caneither be set to Mode 0 or 1 with port A(Group A) is in Mode 2.
- Mode 2 is not available for port B. The following fig shows the control word.
- The INTR goes high only if IBF, INTE2, STB and RD go high or OBF,
- INTE1, ACK and WR go high. The port C can be read to know the status of theperipheral device, in terms of the control signals, using the normal I/Oinstructions.



Mode 2 control word



Mode 2 pins

Interfacing Analog to Digital Data Converters:

- In most of the cases, the PIO 8255 is used for interfacing the analog to digital converters with microprocessor.
- We have already studied 8255 interfacing with 8086 as an I/O port, in previous section. This section we will only emphasize the interfacing techniques of analog to digital converters with 8255.
- The analog to digital converters is treated as an input device by the microprocessor that sends an initializing signal to the ADC to start the analogy to digital data conversation process. The start of conversation signal is a pulse of a specific duration.
- The process of analog to digital conversion is a slow
- Process and the microprocessor have to wait for the digitaldata till the conversion is over. After the conversion isover, the ADC sends end of conversion EOC signal toinform themicroprocessor that the conversion is over andthe result is ready at the output buffer of the ADC. Thesetasks of issuing an SOC pulse to ADC, reading EOC signalfrom the ADC and reading the digital output of the ADCare carried out by the CPU using 8255 I/O ports.
- The time taken by the ADC from the active edge of SOCpulse till the active edge of EOC signal is called as theconversion delay of the ADC.
- It may range anywhere from a few microseconds in caseof fast ADC to even a few hundred milliseconds in case ofslow ADCs.
- The available ADC in the market use different conversiontechniques for conversion of analog signal to digitals.Successive approximation techniques and dual slopeintegration techniques are the most popular techniquesused in the integrated ADC chip.
- General algorithm for ADC interfacing contains thefollowing steps:
- Ensure the stability of analog input, applied to the ADC.
- Issue start of conversion pulse to ADC
- Read end of conversion signal to mark the end ofconversion processes.
- Read digital data output of the ADC as equivalent digitaloutput.
- Analog input voltage must be constant at the input of theADC right from the start of conversion till the end of theconversion to get correct results. This may be ensured by asample and hold circuit which samples the analog signaland holds it constant for specific time duration. Themicroprocessor may issue a hold signal to the sample andhold circuit.
- If the applied input changes before the completeconversion process is over, the digital equivalent of theanalog input calculated by the ADC may not be correct.

ADC 0808/0809:

- The analog to digital converter chips 0808 and 0809 are 8-bit CMOS, successive approximation converters. This technique is one of the fast techniques for analog to digital conversion. The conversion delay is 100µs at a clock frequency of 640 KHz, which is quite low as compared to other converters. These converters do not need any external zero or full scale adjustments as they are already taken care of by internal circuits.
- These converters internally have a 3:8 analog multiplexer so that at a time eight different analog conversion by using address lines - ADD A, ADD B, ADD C, as shown. Using these address inputs, multichannel data acquisition system can be designed using a single ADC. The CPU may drive these lines using output port lines in case of multichannel applications. In case of single input applications, these may be hardwired to select the proper input.
- There are unipolar analog to digital converters, i.e. they are able to convert only positive analog input voltage to their digital equivalent. These chips do not contain any internal sample and hold circuit.
- If one needs a sample and hold circuit for the conversion of fast signal into equivalent digital quantities, it has to be externally connected at each of the analog inputs.

Fig (1) and Fig (2) show the block diagrams and pin diagrams for ADC 0808/0809.

Table.1

Analog I/P selected	**Address lines**		
	C	**B**	**A**
I/P 0	0	0	0
I/P 1	0	0	1
I/P 2	0	1	0
I/P 3	0	1	1
I/P 4	1	0	0
I/P 5	1	0	1
I/P 6	1	1	0
I/P 7	1	1	1

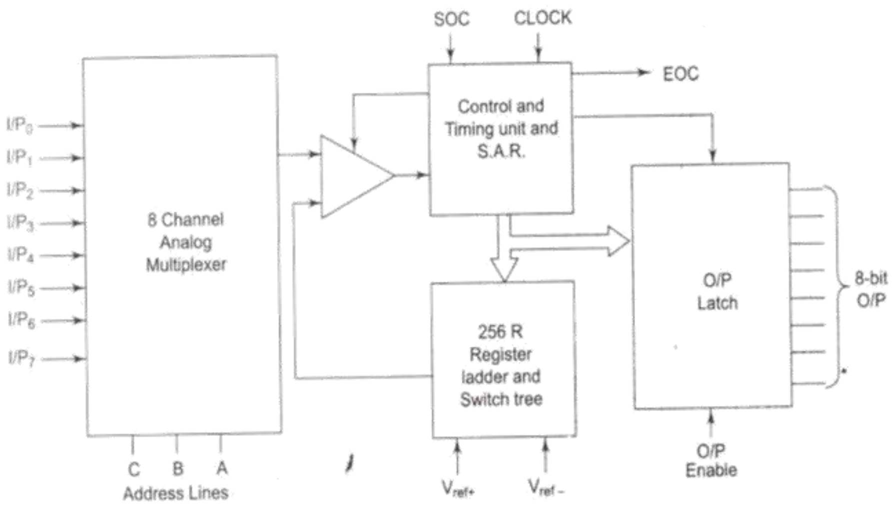
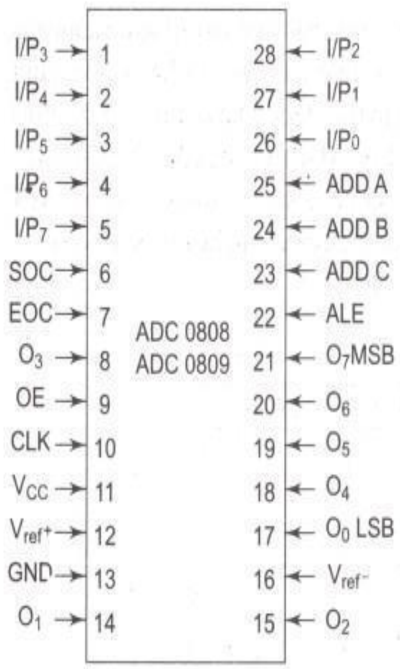


Fig.1 Block Diagram of ADC 0808/0809



I/P_0 - I/P_7	Analog inputs
ADD A, B, C	Address lines for selecting analog inputs
O_7 - O_0	Digital 8-bit output with O_7 MSB and O_0 LSB
SOC	Start of conversion signal pin
EOC	End of conversion signal pin
OE	Output latch enable pin, if high enable output
CLK	Clock input for ADC
V_{CC}, GND	Supply pins +5V and GND
V_{ref+} and V_{ref-}	Reference voltage positive (+5 Volts maximum) and Reference voltage negative (0V minimum)

Fig.2 Pin Diagram of ADC 0808/0809

Some Electrical Specifications Of The ADC 0808/0809 Are Given In Table.2.

Table.2

Minimum SOC pulse width	100 ns
Minimum ALE pulse width	100 ns
Clock frequency	10 to 1280 kHz
Conversion time	100 ms at 640 kHz
Resolution	8-bit
Error	+/–1 LSB
V_{ref+}	Not more than +5V
V_{ref-}	Not less than GND
+ V_{cc} supply	+ 5 V DC
Logical 1 i/p voltage	minimum V_{cc} –1.5 V
Logical 0 i/p voltage	maximum 1.5 V
Logical 1 o/p voltage	minimum V_{cc}–0.4 V
Logical 0 o/p voltage	maximum 0.45 V

The Timing Diagram Of Different Signals Of Adc0808 Is Shown In Fig.3

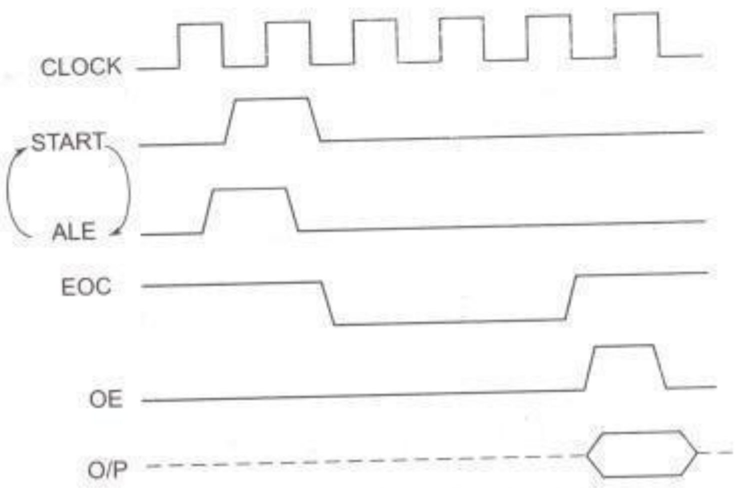
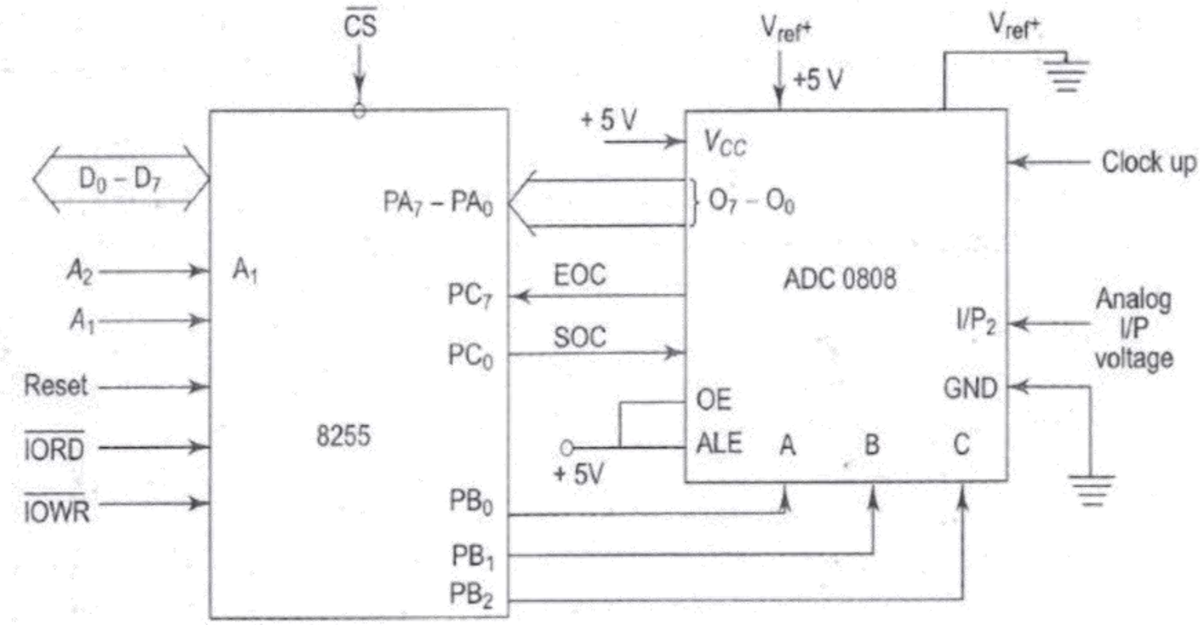


Fig.3 Timing Diagram Of ADC 0808.



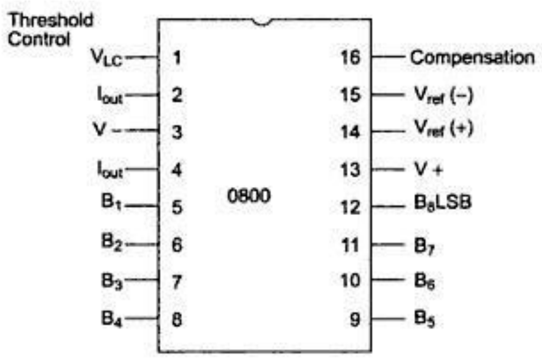
Interfacing ADC0808 with 8086

Interfacing Digital To Analog Converters:

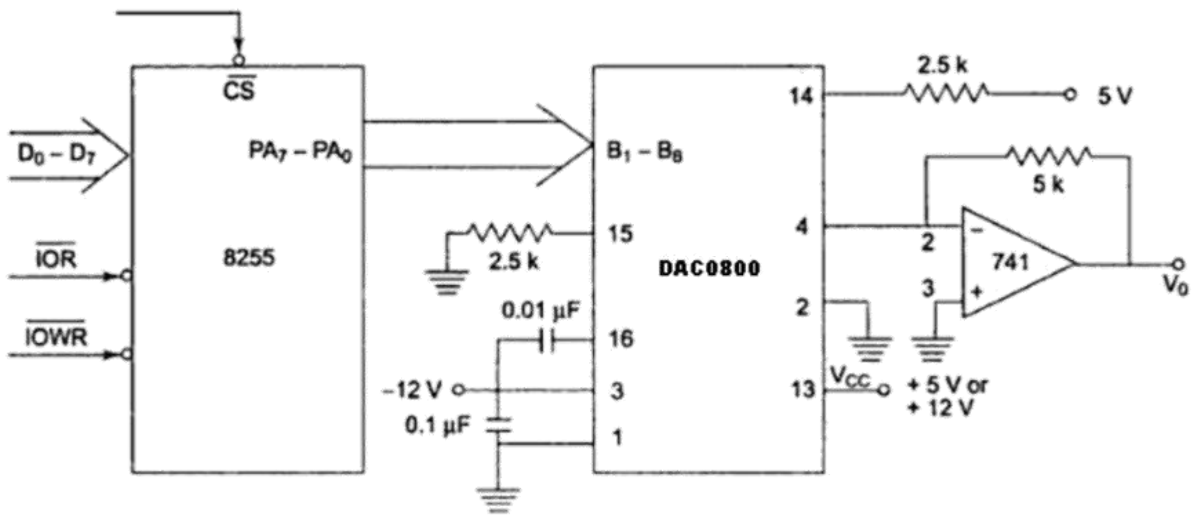
The digital to analog converters convert binary numbers into their analog equivalent voltages. The DAC find applications in areas like digitally controlled gains, motor speed controls, programmable gain amplifiers, etc.

DAC0800 8-bit Digital to Analog Converter

- The DAC 0800 is a monolithic 8-bit DAC manufactured by National Semiconductor.
- It has settling time around 100ms and can operate on a range of power supply voltages i.e. from 4.5V to +18V.
- Usually the supply V+ is 5V or +12V.
- The V-pin can be kept at a minimum of -12V.



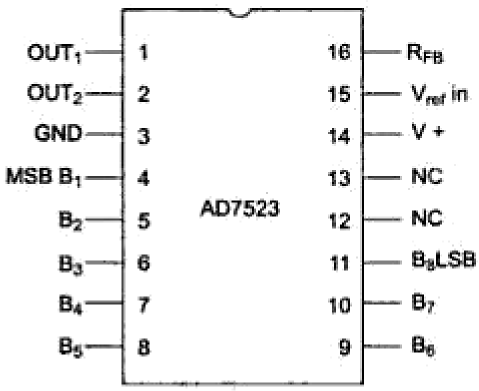
Pin Diagram of DAC 0800



Interfacing DAC0800 with 8086

Ad 7523 8-Bit Multiplying DAC:

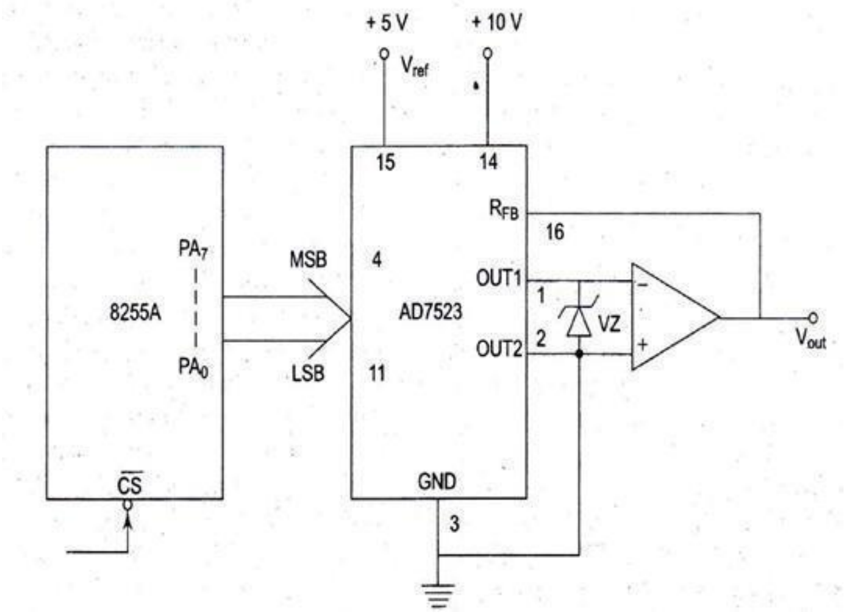
- Intersil‟s AD 7523 is a 16 pin DIP, multiplying digital to analog converter, containing R-2R ladder(R=10KΩ) for digital to analog conversion along with single pole double through NMOS switches to connect the digital inputs to the ladder.



Pin Diagram of AD7523

- The supply range extends from +5V to +15V , while Vref may be anywhere between -10V to +10V. The maximum analog output voltage will be +10V, when all the digital inputs are at logic high state. Usually a Zener is connected between OUT1 and OUT2 to save the DAC from negative transients.
- An operational amplifier is used as a current to voltage converter at the output of AD 7523 to convert the current output of AD7523 to a proportional output voltage.

- It also offers additional drive capability to the DAC output. An external feedback resistor acts to control the gain. One may not connect any external feedback resistor, if no gain control is required.



Interfacing AD7523 with 8086

Stepper Motor Interfacing:

- A stepper motor is a device used to obtain an accurate position control of rotating shafts. It employs rotation of its shaft in terms of steps, rather than continuous rotation as in case of AC or DC motors. To rotate the shaft of the stepper motor, a sequence of pulses is needed to be applied to the windings of the stepper motor, in a proper sequence.
- The number of pulses required for one complete rotation of the shaft of the stepper motor is equal to its number of internal teeth on its rotor. The stator teeth and the rotor teeth lock with each other to fix a position of the shaft.
- With a pulse applied to the winding input, the rotor rotates by one teeth position or an angle x. The angle x may be calculated as:

$$X=360^0/\text{no. of rotor teeth}$$

- After the rotation of the shaft through angel x, the rotor locks itself with the next tooth in the sequence on the internal surface of stator.
- The internal schematic of a typical stepper motor with four windings is shown in fig.1.
- The stepper motors have been designed to work with digital circuits. Binary level pulses of 0-5V are required at its winding inputs to obtain the rotation of shafts. The sequence of the pulses can be decided, depending upon the required motion of the shaft.

- Fig.2 shows a typical winding arrangement of the stepper motor.
- Fig.3 shows conceptual positioning of the rotor teeth on the surface of rotor, for a six teeth rotor.

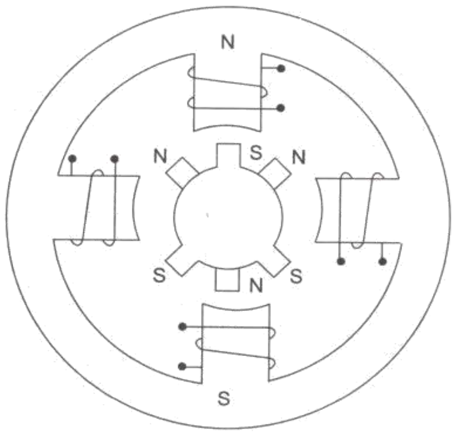


Fig.1 Internal schematic of a four winding stepper motor

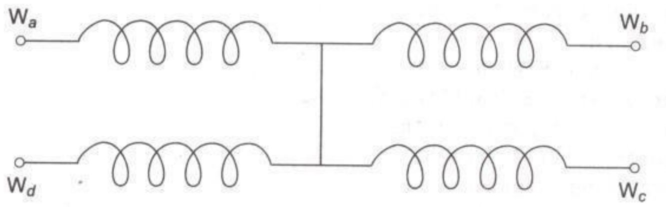


Fig.2 Winding arrangement of a stepper motor.

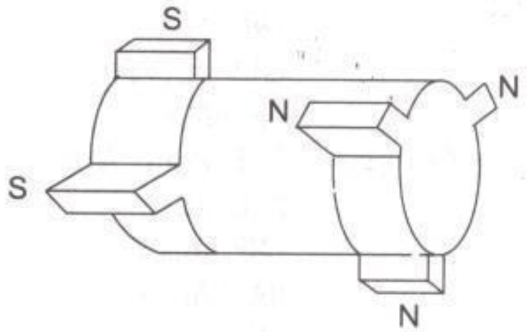


Fig.3 Stepper motor rotor

- The circuit for interfacing a winding Wn with an I/O port is given in fig.4. Each of the windings of a stepper motor needs this circuit for its interfacing with the output port. A typical stepper motor may have parameters like torque 3 Kg-cm, operating voltage 12V, current rating 0.2 A and a step angle 1.8^0 i.e. 200 steps/revolution (number of rotor teeth).

- A simple schematic for rotating the shaft of a stepper motor is called a wave scheme. In this scheme, the windings Wa, Wb, Wc and Wd are applied with the required voltages pulses, in a cyclic fashion. By reversing the sequence of excitation, the direction of rotation of the stepper motor shaft may be reversed.
- Table.1 shows the excitation sequences for clockwise and anticlockwise rotations. Another popular scheme for rotation of a stepper motor shaft applies pulses to two successive windings at a time but these are shifted only by one position at a time. This scheme for rotation of stepper motor shaft is shown in table2.

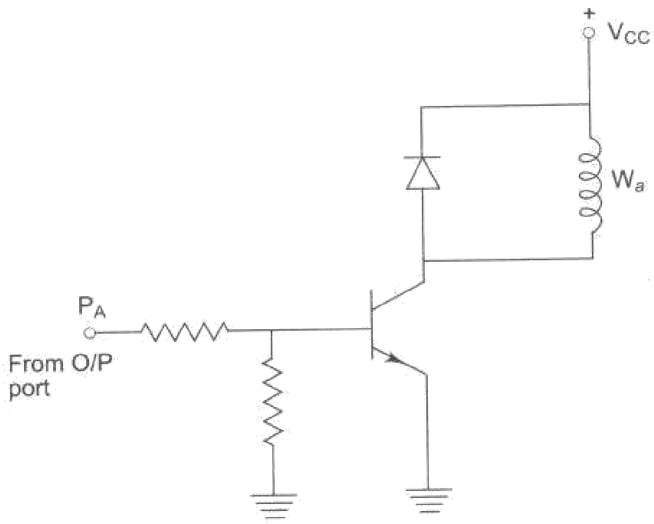


Fig.4 interfacing stepper motor winding.

Table.1 Excitation sequence of a stepper motor using wave switching scheme.

Motion	step	A	B	C	D
Clock wise	1	1	0	0	0
	2	0	1	0	0
	3	0	0	1	0
	4	0	0	0	1
	5	1	0	0	0
Anticlock wise	1	1	0	0	0
	2	0	0	0	1
	3	0	0	1	0
	4	0	1	0	0
	5	1	0	0	0

Table.2 An alternative scheme for rotating stepper motor shaft

Motion	step	A	B	C	D
Clock wise	1	0	0	1	1
	2	0	1	1	0
	3	1	1	0	0
	4	1	0	0	1
	5	0	0	1	1
Anticlock wise	1	0	0	1	1
	2	1	0	0	1
	3	1	1	0	0
	4	0	1	1	0
	5	0	0	0	0

Keyboard Interfacing

- In most keyboards, the key switches are connected in a matrix of Rows and Columns.
- Getting meaningful data from a keyboard requires three major tasks:

1. D e t e c t a k e y p r e s s
2. D e b o u n c e t h e k e y p r e s s .
3. Encode the keypress (produce a standard code for the pressed key).

Logic „0‟ is read by the microprocessor when the key is pressed.

Key Debounce:

Whenever a mechanical push-bottom is pressed or released once,the mechanical components of the key do not change the positionsmoothly; rather it generates a transient response. These may be interpreted as the multiple pressures and responded accordingly.

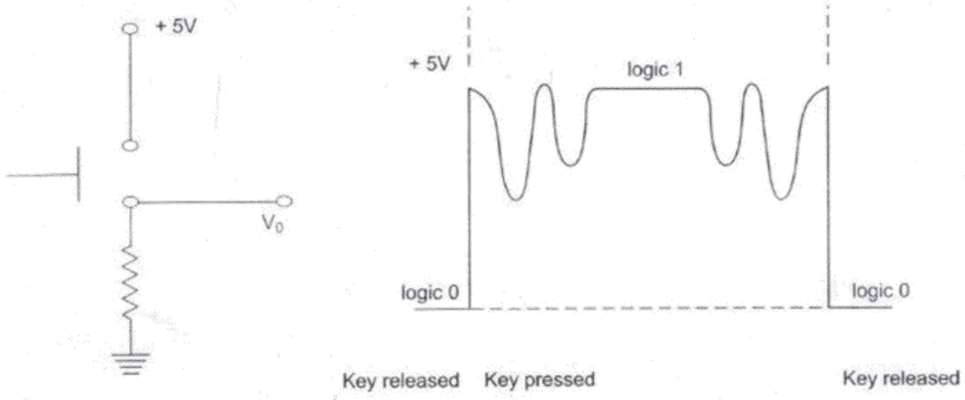


Fig. 5.23 *A Mechanical Key and Its Response*

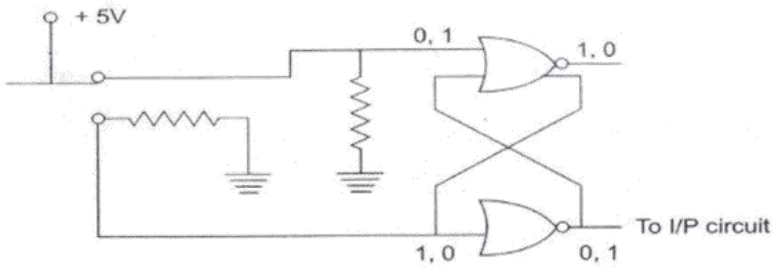


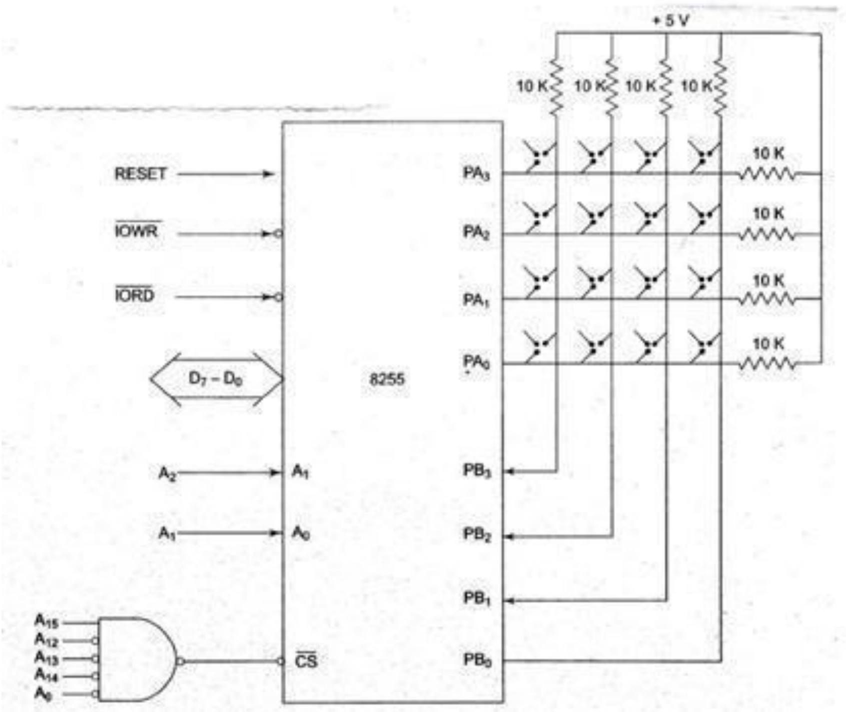
Fig. 5.24 *Hardware Debouncing Circuit*

- The rows of the matrix are connected to four output Port lines, &columns are connected to four input Port lines.
- When no keys are pressed, the column lines are held high by the pull-up resistors connected to +5v.
- Pressing a key connects a row & a column.
- To detect if any key is pressed is to output 0‟s to all rows & then check columns to see it a pressed key has connected a low (zero) to a column.
- Once the columns are found to be all high, the program enters another loop, which waits until a low appears on one of the columns i.e indicating a key press.
- A simple 20/10 msec delay is executed to debounce task.
- After the debounce time, another check is made to see if the key is still pressed. If the columns are now all high, then no key is pressed & the initial detection was caused by a noise pulse.
- To avoid this problem, two schemes are suggested:
    1. Use of Bistablemultivibrator at the output of the key to debounce it.
    2. The microprocessor has to wait for the transient period (at least for 10 ms), so that the transient response settles down and reaches a steady state.

- If any of the columns are low now, then the assumption is made that it was a valid key press.

- The final task is to determine the row & column of the pressed key &convert this information to Hex-code for the pressed key.

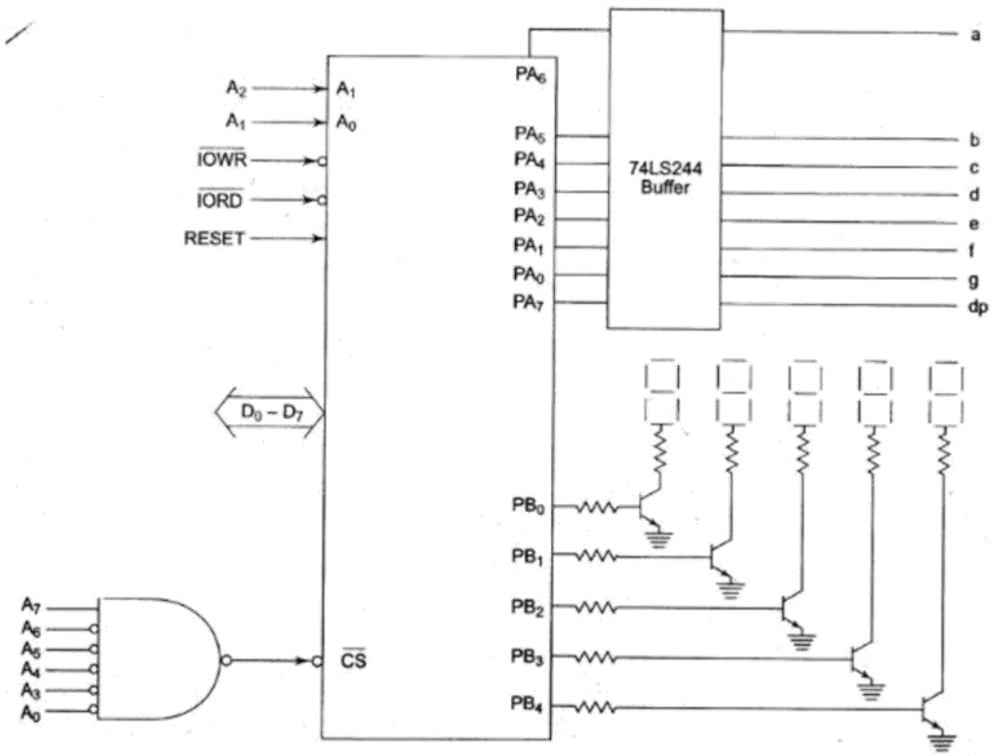
- The 4-bit code from I/P port & the 4-bit code from O/P port (row &column) are converted to Hex-code.



Interfacing 4x4 keyboard

Display Interface

Number to be displayed	PA7 dp	PA6 a	PA5 b	PA4 c	PA3 d	PA2 e	PA1 f	PA0 g	Code
1	1	1	0	0	1	1	1	1	CF
2	1	0	0	1	0	0	1	0	92
3	1	0	0	0	0	1	1	0	86
4	1	1	0	0	1	1	0	0	CC
5	1	0	1	0	0	1	0	0	A4



Interfacing multiplexed 7-segment display

Interfacing with Advanced devices

4.1 MEMORY AND I/O INTERFACING

(Ref: Interfacing through Microprocessors by K. Subba Rao, Hi-tech publishers, P. 163-166)

4.1.1 I/O Interface

Any application of a microprocessor system requires the transfer of data between microprocessor and external environment and also with in the microprocessor. This is known as Input/Output. There are three different ways that the data transfer can take place. They are

(1) Program controlled I/O
(2) Interrupt Program Controlled I/O
(3) Hardware controlled I/O

In program controlled I/O data transfer scheme the transfer of data is completely under the control of the microprocessor program. In this case an I/O operation takes place only when an I/O transfer instruction is executed.

In an interrupt program controlled I/O an external device indicates directly to the microprocessor its readiness to transfer data by a signal at an interrupt input of the microprocessor. When microprocessor receives this signal the control is transferred to ISS (Interrupt service subroutine) which performs the data transfer.

Hardware controlled I/O is also known as direct memory access DMA. In this case the data transfer takes place directly between an I/O device and memory but not through microprocessors. Microprocessor only initializes the process of data transfer by indicating the starting address and the number of words to be transferred.

The instruction .set of any microprocessor contains instructions that transfer information to an I/O device and to read information from an I/O device. In 8086 we have IN, OUT instructions for this purpose. OUT instruction transfers information to an I/O device where as IN instruction is used to read information from an I/O device. Both the instructions perform the data transfer using accumulator AL or AX. The I/O address is stored in register DX.

The port number is specified along with IN or OUT instruction. The external I/O interface decodes to find the address of the I/O device. The 8 bit fixed port number appears on address bus A_0 - A_7 with A_8 - A_{15} all zeros. The address connections above A_{15} are undefined for an I/O instruction. The 16 bit variable port number appears on address connections A_0 - A_{15}. The above notation indicates that first 256 I/O port addresses 00 to FF are accessed by both the fixed and variable I/O instructions. The I/O addresses from 0000 to FFFF are accessed by the variable I/O address.

I/O devices can be interfaced to the microprocessors using two methods. They are I/O mapped I/O and memory mapped I/O. The I/O mapped I/O is also known as isolated I/O or direct I/O. In I/O mapped I/O the IN and OUT instructions transfer data between the accumulator or memory and I/O device. In memory mapped I/O the instruction that refers memory can perform the data transfer.

I/O mapped I/O is the most commonly used I/O transfer technique. In this method I/O locations are placed separately from memory. The addresses for isolated I/O devices are separate from memory. Using this method user can use the entire memory. This method allows data transfer only by using instructions IN, OUT. The pins $M/\overline{IO}$ and W/R are used to indicate I/O read or an I/O write operations. The signals on these lines indicate that the address on the address bus is for I/O devices.

Memory mapped I/O does not use the IN, OUT instruction it uses only the instruction that transfers data between microprocessor and memory. A memory mapped I/O device is treated as memory location. The disadvantage in this system is the overall memory is reduced. The advantage of this system is that any memory transfer instruction can be used for data transfer and control signals like I/O read and I/O write are not necessary which simplify the hardware.

4.1.2 Memory interfacing

Memory is an integral part of a microcomputer system. There are two main types of memory.

- **(i)** **Read only memory (ROM):** As the name indicates this memory is available only for reading purpose. The various types available under this category are PROM, EPROM, EEPROM which contain system software and permanent system data.
- **(ii)** **Random Access memory (RAM):** This is also known as Read Write Memory. It is a volatile memory. RAM contains temporary data and software programs generally for different applications.

While executing particular task it is necessary to access memory to get instruction codes and data stored in memory. Microprocessor initiates the necessary signals when read or write operation is to be performed. Memory device also requires some signals to perform read and write operations using various registers. To do the above job it is necessary to have a device and a circuit, which performs this task is known as interfacing device and as this is involved with memory it-is known as memory interfacing device. The basic concepts of memory interfacing involve three different tasks. The microprocessor should be able to read from or write into the specified register. To do this it must be able to select the required chip, identify the required register and it must enable the appropriate buffers.

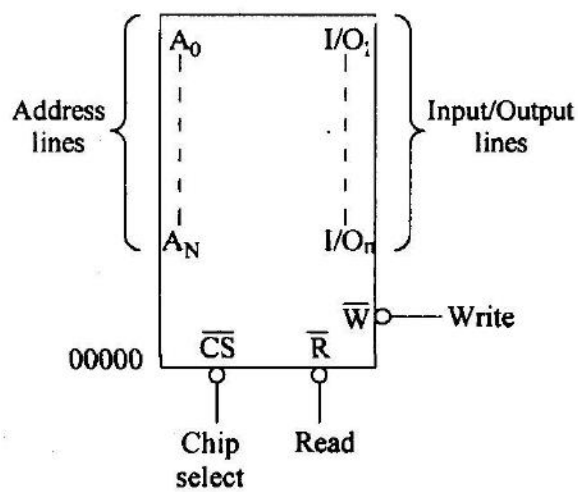


Fig. 3.13 Simple memory device

Any memory device must contain address lines and Input, output lines, selection input, control input to perform read or write operation. All memory devices have address inputs that select memory location with in the memory device. These lines are labeled as A_O ...... A_N. The number of address lines indicates the total memory capacity of the memory device. A 1K memory requires 10 address lines A_0-A_9. Similarly a 1MB requires 20 lines A_0-A_{19} (in the case of 8086). The memory devices may have separate I/O lines or a common set of bidirectional I/O lines. Using these lines data can be transferred in either direction. Whenever output buffer is activated the operation is read whenever input buffers are activated the operation is write. These lines are labelled as I/O,......... I/O_n or D_0 ....D_n. The size of a memory location is dependent upon the number of data bits. If the number of data lines are eight D_0 - D_7 then 8 bits or 1 byte of data can be stored in each location. Similarly if numbers of data bits are 16 (D_0 - D_{15}) then the memory size is 2 bytes. For example 2K x 8 indicates there are 2048 memory locations and each memory location can store 8 bits of data.

Memory devices may contain one or more inputs which are used to select the memory device or to enable the memory device. This pin is denoted by $\overline{CS}$ (Chip select) or $\overline{CE}$ (Chip enable). When this pin is at logic '0' then only the memory device performs a read or a write operation. If this pin is at logic '1' the memory chip is disabled. If there are more than one $\overline{CS}$ input then all these pins must be activated to perform read or write operation.

All memory devices will have one or more control inputs. When ROM is used we find $\overline{OE}$ output enable pin which allows data to flow out of the output data pins. To perform this task both $\overline{CS}$ and $\overline{OE}$ must be active. A RAM contains one or two control inputs. They are R / $\overline{W}$ or $\overline{RD}$ and $\overline{WR}$. If there is only one input R/ $\overline{W}$ then it performs read operation when R/ $\overline{W}$ pin is at logic 1. If it is at logic 0 it performs write operation. Note that this is possible only when $\overline{CS}$ is also active.

4.4 Memory Interface using RAMS, EPROMS and EEPROMS

(Ref: Advanced Microprocessors and Peripherals by A.K. Ray & K.M. Bhurchandi, McGraw-Hill, 2nd Edition.P.158-164)

Semiconductor Memory Interfacing:

Semiconductor memories are of two types, viz. RAM (Random Access Memory) and ROM (Read Only Memory).

Static RAM Interfacing:

The semiconductor RAMs are of broadly two types-static RAM and dynamic RAM. The semiconductor memories are organised as two dimensional arrays of memory locations. For example, 4K x 8 or 4K byte memory contains 4096 locations, where each location contains 8-bit data and only one of the 4096 locations can be selected at a time. Obviously, for addressing 4K bytes of memory, twelve address lines are required. In general, to address a memory location out of N memory locations , we will require at least n bits of address, i.e. n address lines where n = Log_2 N. Thus if the microprocessor has n address lines, then it is able to address at the most N locations of memory, where 2^n = N. However, if out of N locations only P memory locations are to be interfaced, then the least significant p address lines out of the available n lines can be directly connected from the microprocessor to the memory chip while the remaining $(n-p)$ higher order address lines may be used for address decoding (as inputs to the chip selection logic). The memory address depends upon the hardware circuit used for decoding the chip select ($\overline{CS}$). The output of the decoding circuit is connected with the $\overline{CS}$ pin of the memory chip. The general procedure of static memory interfacing with 8086 is briefly described as follows:

1. Arrange the available memory chips so as to obtain 16-bit data bus width. The upper 8-bit bank is called 'odd address memory bank' and the lower 8-bit bank is called 'even address memory bank'.

2. Connect available memory address lines of memory chips with those of the microprocessor and also connect the memory $\overline{RD}$ and $\overline{WR}$ inputs to the corresponding processor control signals. Connect the 16-bit data bus of the memory bank with that of the microprocessor 8086.

3. The remaining address lines of the microprocessor, $\overline{BHE}$ and A_0 are used for decoding the required chip select signals for the odd and even memory banks. $\overline{CS}$ of memory is derived from the O/P of the decoding circuit.

As a good and efficient interfacing practice, the address map of the system should be continuous as far as possible, i.e. there should be no windows in the map. A memory location should have a single address corresponding to it, i.e. absolute decoding should be preferred, and minimum hardware should be used for decoding. In a number of cases, linear decoding may be used to minimise the required hardware.Let us now consider a few example problems on memory interfacing with 8086.

Problem 5.1

Interface two 4K × 8 EPROMS and two 4K × 8 RAM chips with 8086. Select suitable maps.

Solution We know that, after reset, the IP and CS are initialised to form address FFFF0H. Hence, this address must lie in the EPROM. The address of RAM may be selected any where in the 1MB address space of 8086, but we will select the RAM address such that the address map of the system is continuous, as shown in Table 5.1.

Table 5.1 *Memory Map for Problem 5.1*

Address	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
							EPROM						8K × 8							
FE000H	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
FDFFFH	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
							RAM					8K × 8								
FC000H	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Total 8K bytes of EPROM need 13 address lines $A_0 - A_{12}$ (since $2^{13} = 8K$). Address lines $A_{13} - A_{19}$ are used for decoding to generate the chip select. The $\overline{BHE}$ signal goes low when a transfer is at odd address or higher byte of data is to be accessed. Let us assume that the latched address, $\overline{BHE}$ and demultiplexed data lines are readily available for interfacing. Figure 5.1 shows the interfacing diagram for the memory system.

The memory system in this example contains in total four 4K × 8 memory chips.

The two 4K × 8 chips of RAM and ROM are arranged in parallel to obtain 16-bit data bus width. If A_0 is 0, i.e. the address is even and is in RAM, then the lower RAM chip is selected indicating 8-bit transfer at an even address. If A_0 is 1, i.e. the address is odd and is in RAM, the $\overline{BHE}$ goes low, the upper RAM chip is selected, further indicating that the 8-bit transfer is at an odd address. If the selected addresses are in ROM, the respective ROM chips are selected. If at a time A_0 and $\overline{BHE}$ both are 0, both the RAM or ROM chips are selected, i.e. the data transfer is of 16 bits. The selection of chips here takes place as shown in Table 5.2.

Table 5.2 *Memory Chip Selection for Problem 5.1*

Decoder I/P → Address/$\overline{BHE}$ →	A_2 A_{13}	A_1 A_0	A_0 $\overline{BHE}$	Selection/ Comment
Word transfer on $D_0 - D_{15}$	0	0	0	Even and odd addresses in RAM
Byte transfer on $D_7 - D_0$	0	0	1	Only even address in RAM
Byte transfer on $D_8 - D_{15}$	0	1	0	Only odd address in RAM
Word transfer on $D_0 - D_{15}$	1	0	0	Even and odd addresses in ROM
Byte transfer on $D_0 - D_7$	1	0	1	Only even address in ROM
Byte transfer on $D_8 - D_{15}$	1	1	0	Only odd address in ROM

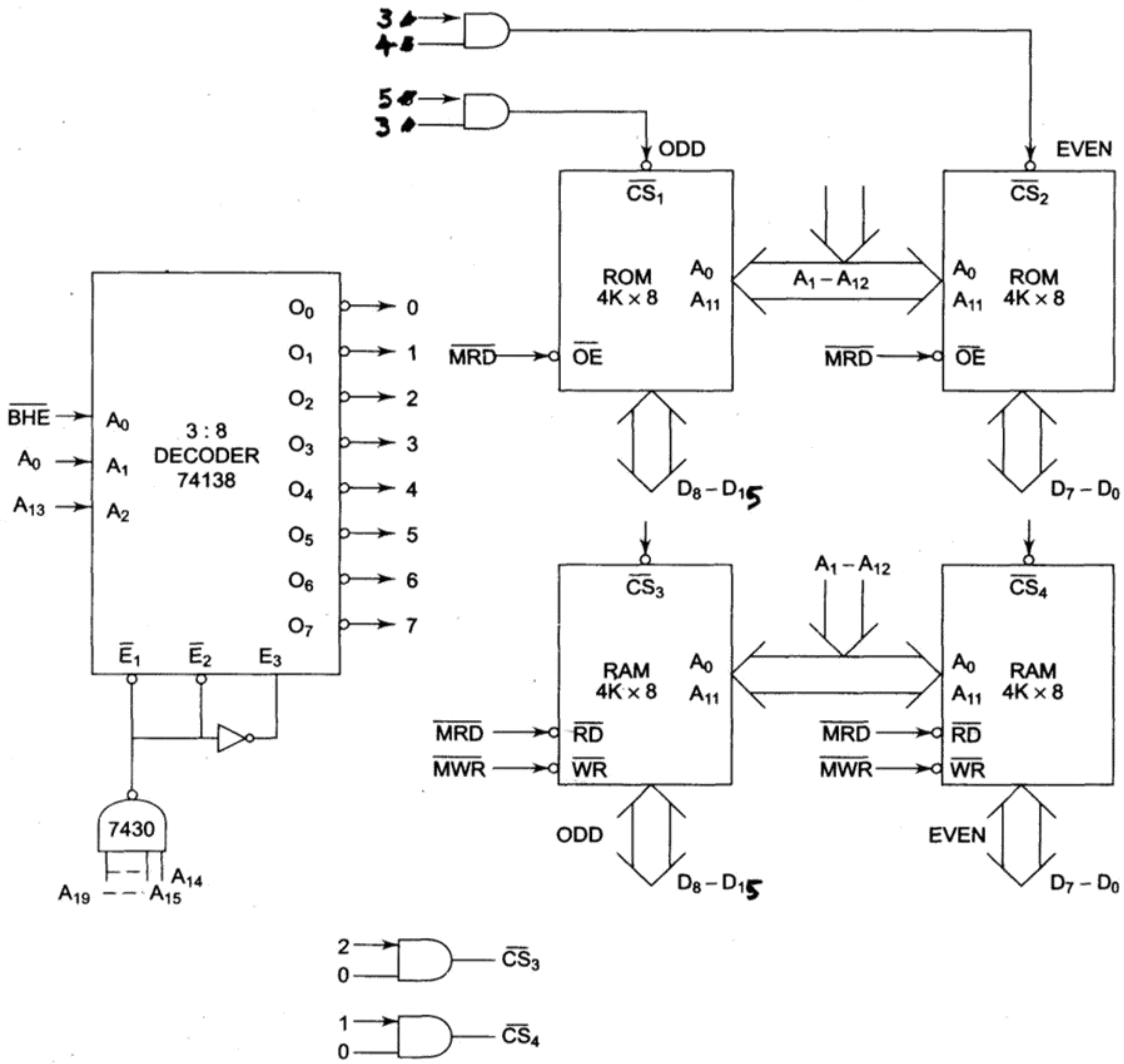


Fig. 5.1 *Interfacing Problem 5.1*

Problem 5.2

Design an interface between 8086 CPU and two chips of 16K × 8 EPROM and two chips of 32K × 8 RAM. Select the starting address of EPROM suitably. The RAM address must start at 00000H.

Solution: The last address in the map of 8086 is FFFFFH. After resetting, the processor starts from FFFF0H. Hence this address must lie in the address range of EPROM. Figure 5.2 shows the interfacing diagram, and Table 5.3 shows complete map of the system.

Table 5.3 *Address Map for Problem 5.2*

Addresses	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
							32KB			EPROM										
F8000H	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0FFFFH	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
						64KB RAM														
00000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

It is better not to use a decoder to implement the above map because it is not continuous, i.e. there is some unused address space between the last RAM address (0FFFFH) and the first EPROM address (F8000H). Hence the logic is implemented using logic gates, as shown in Fig. 5.2.

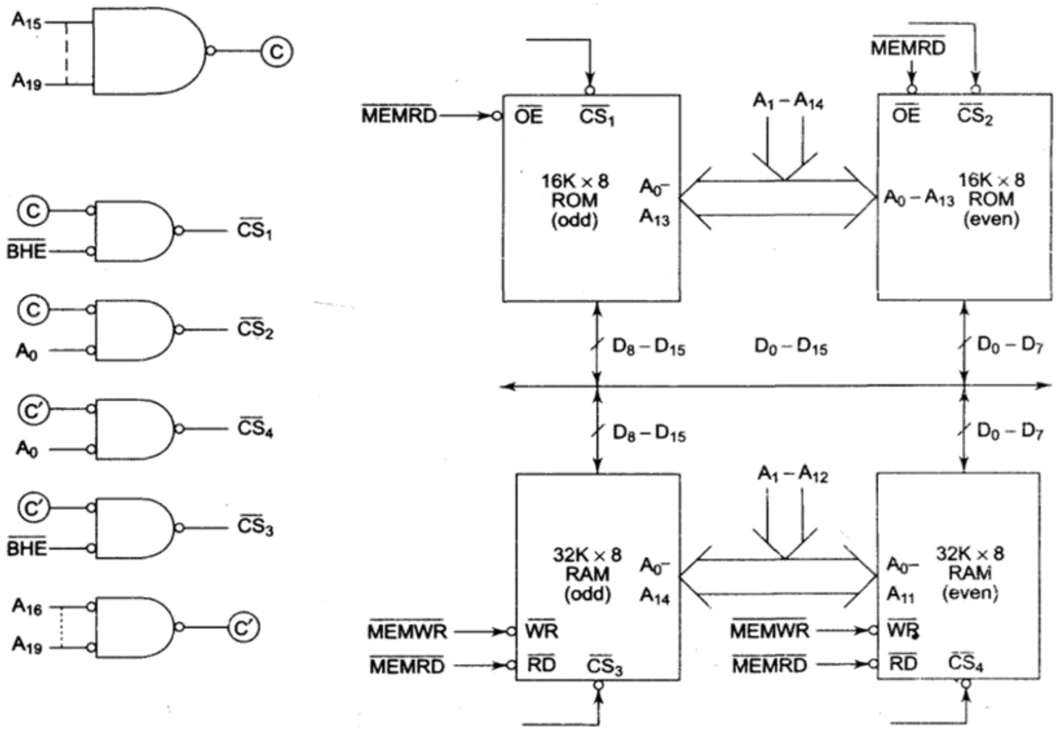


Fig. 5.2 *Interfacing Problem 5.2*

Problem 5.3

It is required to interface two chips of 32K × 8 ROM and four chips of 32K × 8 RAM with 8086, according to the following map.

```
ROM 1 and 2 F0000H - FFFFFH, RAM 1 and 2 D0000H - DFFFFH
RAM 3 and 4 E0000H - EFFFFH
```

Show the implementation of this memory system.

Solution Let us write the memory map of the system as shown in Table 5.6.

The implementation of the above map is shown in Fig. 5.3 using the same technique as in Problem 5.1 and Problem 5.2. All the address, data and control signals are assumed to be readily available.

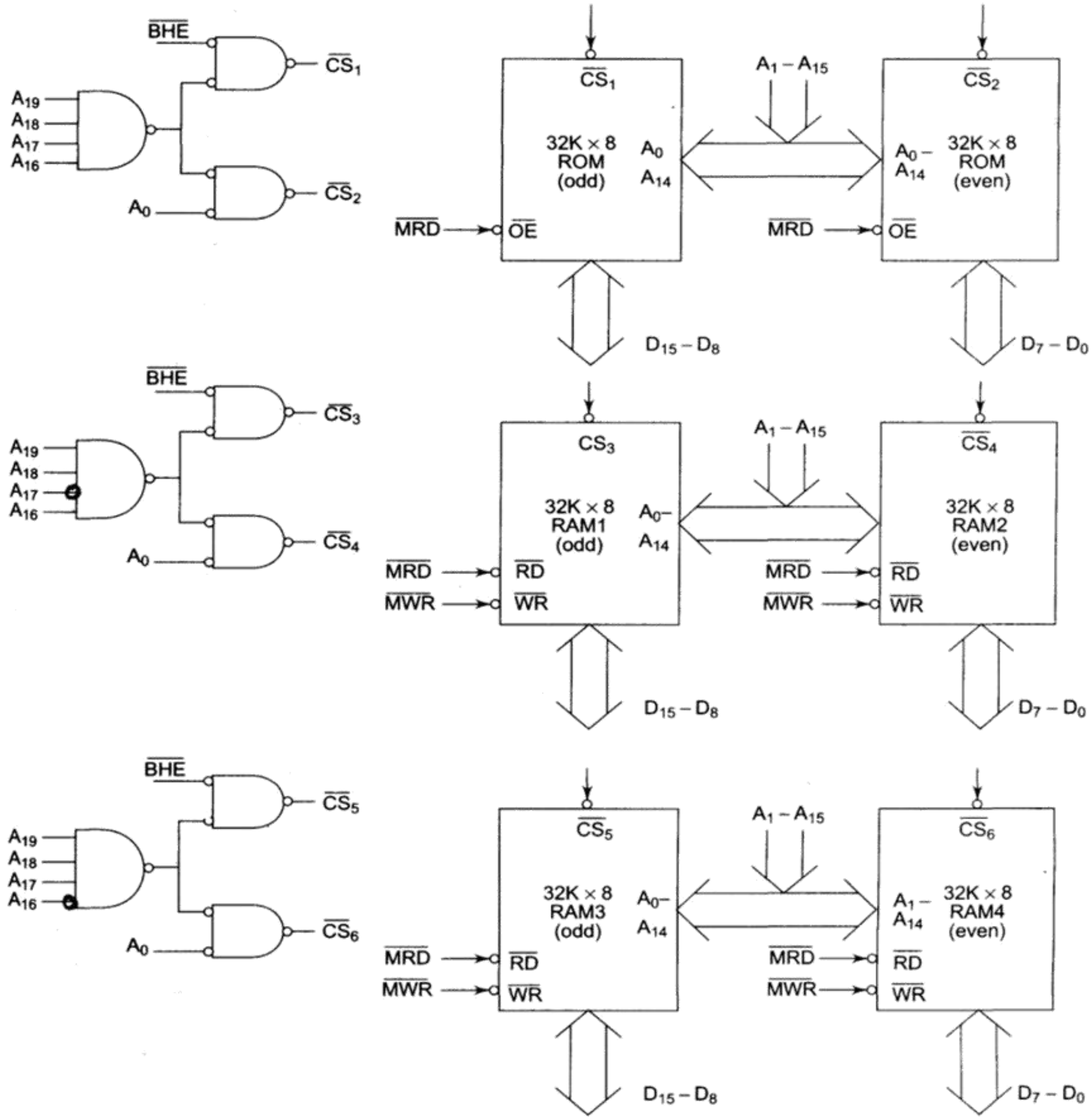


Fig. 5.3 *Interfacing Problem 5.3*

SERIAL COMMUNICATION STANDARDS

(Ref: Interfacing through Microprocessors by K. Subba Rao, Hi-tech publishers, P. 250-260)

Most of devices are parallel in nature. These devices transfer data simultaneously on data lines. But parallel data transfer process is very complicated and expensive. Hence in some situations the serial I/O mode is used where one bit is transferred over a single line at a time. In this type of transmission parallel word is converted into a stream of serial bits which is known as parallel to serial conversion. The rate of transmission in serial mode is BAUD, i.e., bits per second. The serial data transmission involves starting, end of transmission, error verification bits along with the data. Any serial I/O involves the following concepts.

(a) Interfacing requirements (b) Alphanumeric codes (c) Transmission format (d) Error checks in data communication (e) Data communication over lines (f) Standards in serial I/O

The microprocessor has to identify the port address to perform read or write operation. Serial I/O uses only one data line, chip select, read, write control signals.

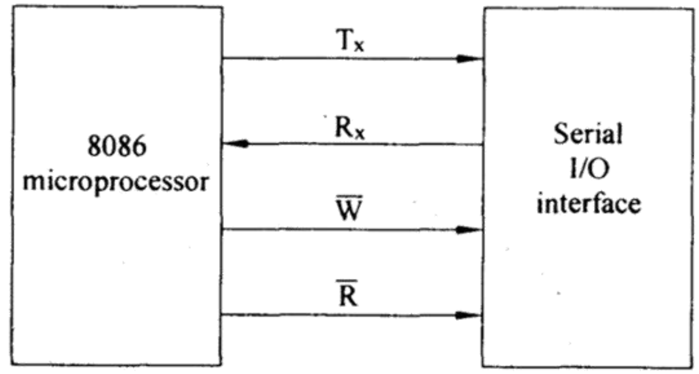
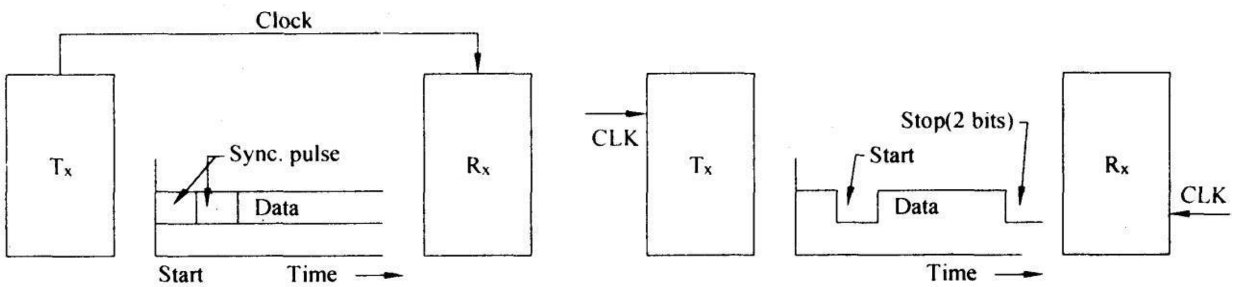


Fig. 5.1 Block diagram of serial I/O interfacing

Data transfer takes place using ASCII code (American standard code for Information Interchange) which is 7 bit code with 128 combinations. The data can be transmitted by taking various parameters into consideration such as synchronization or asynchronization, direction of data flow speed, errors, medium of data transmission etc. In synchronous transmission both transmitter and receiver operate, in synchronous to each other.

Synchronization used for high speed operations. In asynchronous data transmission data is transmitted between Start and Stop bits with logic 1 as mark logic 0 as space. In asynchronous we get around 11 bits for data transmission one start, 8 bits of data, 2 stop bits. A synchronous data transmission is used for less than 20 Kbits /second transmission.



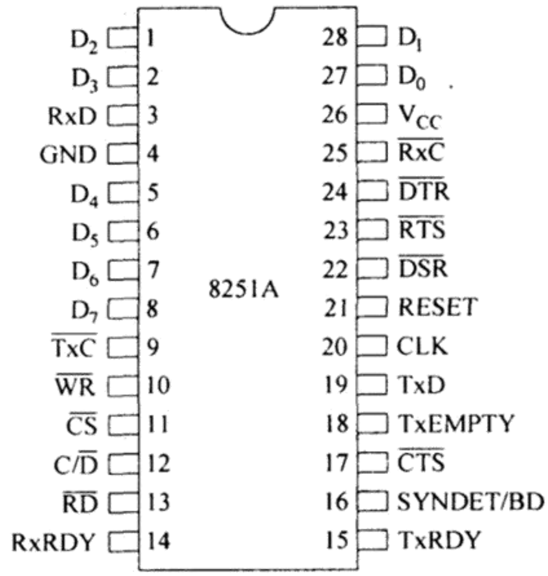
(a) Synchronous data transmission (b) Asynchronous data transmission

Fig. 5.2 Data communication

DIFFERENCE BETWEEN SYNCHRONOUS AND ASYNCHRONOUS TRANSMISSION

S.No.	Synchronous	Asynchronous
1.	Same clock pulse is applied to both Tx & Rx simultaneously	Different clock pulses are applied to Tx & Rx sepeartely
2.	Only hardware is required to implement this.	Both hardware and software are requied for this.
3.	Group or a set of characters can be transmitted at a time.	Only one character is transmitted at a time.
4.	Synchronous pulses are required	Synchronous pulses are not required but uses start and stop bits.
5.	Uses for high speed Tx.	Used for low speed Tx.

5.2 UNIVERSAL SYNCHRONOUS/ASYNCHRONOUS RECEIVER/TRANSMITTER (USART)



Pin diagram of 8521

The 8251A is Universal Synchronous/Asynchronous Receiver/Transmitter (USART) designed for the data communication with Intel's family of microprocessor such as 8085, 8086 and 8088. Like other I/O devices in a microcomputer system, its functional configuration is programmed by the system's software for maximum flexibility. The USART accepts data characters from the CPU in the parallel format and converts them into continuous serial data stream for transmission. Simultaneously, it can receive serial data streams arid convert them into parallel data characters for the CPU. The CPU can read the complete status of USART at any time, these includes data transmission errors, control signals etc.

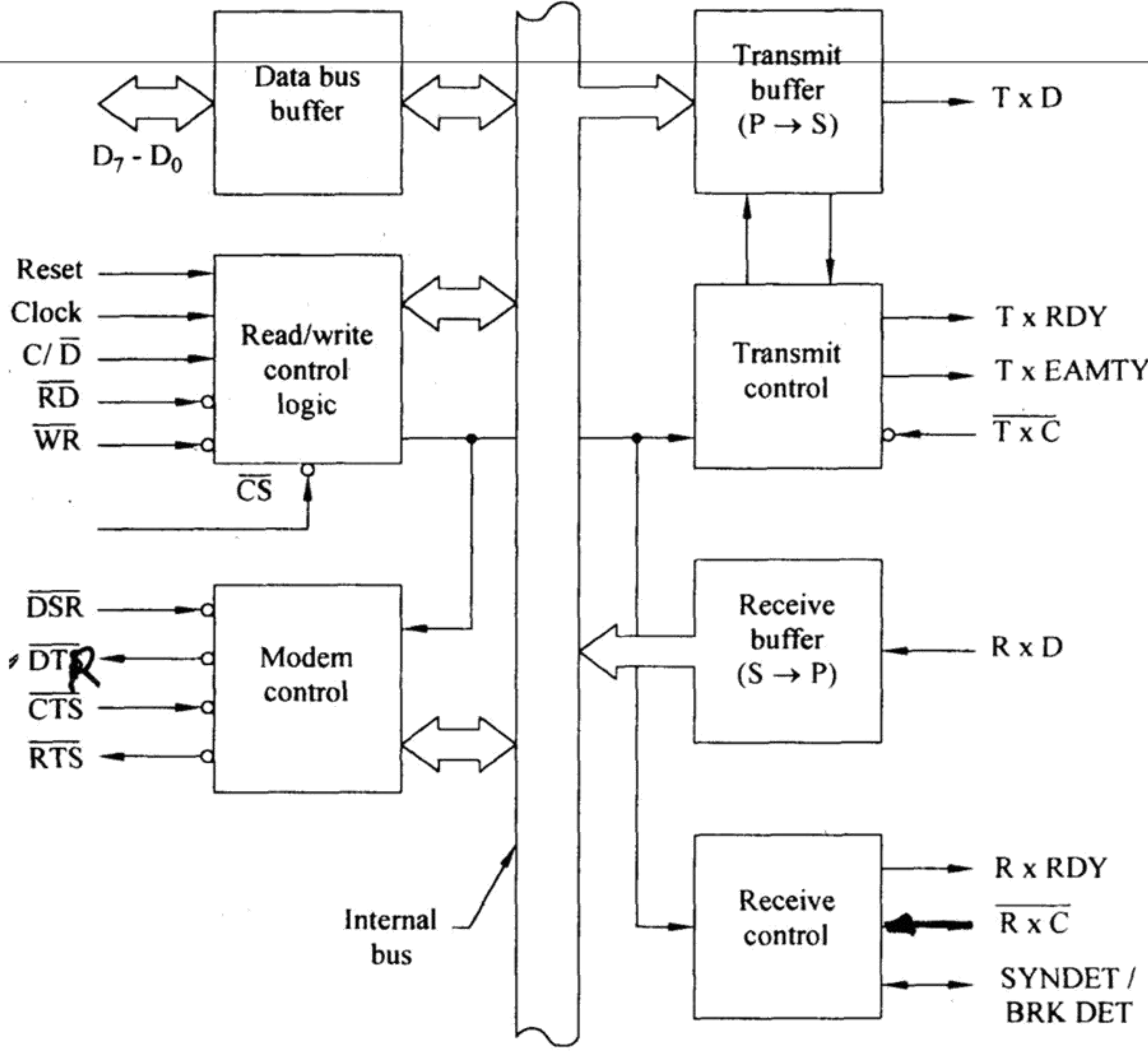


Fig. 5.7 Block diagram of 8251

Fig. 5.7 shows the block diagram of 8251 A. The block diagram shows all the elements of a programmable chip; it includes the interfacing signals, the control register and the status register. The functions of various blocks are described below:

(A) Data bus buffer: This 3-state, bidirectional buffer is used to interface the 8251A to the system data bus. Data is transmitted or received by the buffer upon execution of input and output instruction of the CPU Command words and status information are also transferred through the data bus buffer. The command, status and data in and data out are separate 8-bit registers to provide double buffering.

The functional block accepts inputs form the control bus and generates control signals for overall device operation. It contains the control word register and command word register that store the various control formats for the device functional definition.

(B) Read/Write logic and Registers:

This section includes R/W control logic, six input signals, control logic, and three buffer registers; data register, control register and status register. The input signals to control logic are as follows:

RESET: A high on this input forces the 8251A into an idle mode. The device will remain at idle until a new set of control words is written into the 8251A to program its functional definition.

A command reset operation also puts the device into the idle state.

CLK (Clock): The CLK input is used to generate internal device timing and is normally connected to the phase 2 (TTL) output of the Clock Generator. No external inputs or outputs are referenced to CLK but the frequency of CLK must be greater than 30 times the Receiver or Transmitter data bit rates.

$\overline{WR}$ (Write): A "low" on this input informs the 8251A that the CPU is writing data or control words to the 8251A.

$\overline{RD}$ (Read): A "low" on this input informs the 8251A that the CPU is reading data or status information from the 8251A.

$C/\overline{D}$	RD	WR	CS	
0	0	1	0	8251 data-data bus
0	1	0	0	Data bus-8251A data
1	0	1	0	Status-data bus
1	1	0	0	Data bus-control
×	1	1	0	Data bus-3 state
×	×	×	1	Data bus 3 state

$C/\overline{D}$ (Control/Data): This input, in conjuction with the $\overline{WR}$ and $\overline{RD}$ inputs informs the 8251A that the word on the Data bus is either a data character, control word or status information.

1 = CONTROL/STATUS; 0 = DATA

$\overline{CS}$ (Chip Select) : A "low" on this input selects the 8251A. No reading or writing will occur unless the device is selected. When $\overline{CS}$ is high, the Data Bus is in the float state and $\overline{RD}$ and $\overline{WR}$ have no effect on the chip.

(C) Modem Control:

The 8251A has a set of control inputs and outputs that can be used to simplify the interface to almost any modem. The modem control signals are general purpose in nature and can be used for functions other than modem control, if necessary.

$\overline{DSR}$ (Data Set Ready) : The $\overline{DSR}$ input signal is a general-purpose, 1-bit inverting input port. Its condition can be tested by the CPU using a Status Read operation. The DSR input is normally used to test modem condition such as Data Set Ready.

$\overline{DTR}$ (Data Terminal Ready): The $\overline{DTR}$ output signal is a general-purpose, 1-bit inverting output port. It can be set "low" by programming the appropriate bit in the Command instruction word. The $\overline{DTR}$ output signal is normally used for modem control such as Data Terminal Ready.

$\overline{RTS}$ (Request to Send): The $\overline{RTS}$ output signal is a general-purpose, 1-bit inverting output port. It can be set "low" by programming the appropriate bit in the Command instruction word. The $\overline{RTS}$ output signal is normally used for modem control such as Request to send.

$\overline{CTS}$ (Clear to Send): A "low" on this input enables the 8251A to transmit serial data if the Tx Enable bit in the Command byte is set to a "one", if either a T× Enable off or $\overline{CTS}$ off condition occurs while the T× is in operation, the T× will transmit all the data in the USART, written prior to T× Disable command before shutting down.

(D) Transmitter Buffer:

The transmitter Buffer accepts parallel data from the Data Bus Buffer, converts it to a serial bit stream, inserts the appropriate characters or bits (based on the communication technique) and outputs a composite serial stream of data on the TxD output pin on the falling edge of $\overline{T\,x\,C}$. The transmitter will begin transmission upon being enabled, if $\overline{CTS}$ = 0. The $\overline{T\,x\,C}$ line will be held in the marking state immediately upon a master Reset or when Tx Enable or $\overline{CTS}$ is off or the transmitter is empty.

(E) Transmitter Control:

The Transmitter Control manages all activities associated with the transmission of serial data. It accepts and issues signals both externally and internally to accomplish this function.

T x RDY (Transmitter Ready): This output signals the CPU that the transmitter is ready to accept a data character. The T x RDY output pin can be used as an interrupt to the system, since it is masked by T x Enable; or, for Polled operation, the CPU can check T x RDY using a Status Read operation. T x RDY is automatically reset by the leading edge of $\overline{WR}$ when a data character is loaded from the CPU.

T x E (Transmitter Empty): When the 8251A has no character to send, the T x empty will go high. It resets upon receiving a character from CPU if the transmitter is enabled, Tx empty remains high when the transmitter is disabled. T x Empty can be used to indicate the end of transmission node, so that the CPU knows when to turn around in the half-duplex operation mode.

In the synchronous mode, a high on this output indicates that a character has not been loaded and the SYNC character or characters are about to be are being transmitted as filters. T x Empty does not go low when the Sync characters are being shifted out.

T x C (Transmitter Clock): The Transmitter Clock control the rate at which the character is to be transmitted. In the Synchronous transmission mode, the Baud Rate (1x) is equal to the T x C frequency. In Asynchronous transmission mode, the baud rate is a fraction of the actual T x C frequency. A portion of the mode instruction selects this factor it can be 1,1/16 or 1/64 the T x C.

For example

If Baud rate equals 220 Baud

$\overline{TXC}$ equals 220 Hz in the 1x mode.

$\overline{TXC}$ equals 3.52 KHz in the 16x mode.

$\overline{TXC}$ equals 14.08 KHz in the 64x mode.

The falling edge of $\overline{TXC}$ shifts the serial data out of the 8251A.

(F) Receiver Buffer:

The Receiver accepts serial data, converts this serial input to parallel format checks for bits or characters that are unique to the communication technique and sends an "assembled" character to the CPU. Serial data is input to R × D pin, and is clocked in on this rising edge of $\overline{R \times C}$.

(G) Receiver Control:

This functional block shown in Fig. manages all receiver - related activities which consists of the following features.

The R × D initialization circuits prevents the 8251A from mistaking and unused input line for an active low data line in the break condition. Before starting to receive serial characters on the R × D line, a valid '1' must first be detected after a chip master reset. Once this has been determined, a search for a valid low bit (start bit) is enabled. This feature is only active in the asynchronous mode, and is only done once for each master reset.

The false start bit detection circuit prevents false starts due to a transient noise spike by first detecting the falling edge and then strobing the nominal center of the start (R × D = low).

Parity error detection sets the corresponding status bit.

The framing error status bit is set if the stop bit is absent at the end of the data byte (asynchronous mode).

R × RDY (Receiver Ready) : This output indicates the the 8251A contains a character that is ready to be input to the CPU. R × RDY can be connected to the interrupt structure of the CPU or, for polled operation, the CPU can check the condition of R × RDY using a status read operation.

R × Enable, when off, holds R × RDY in the reset condition. For asynchronous mode, to set R × RDY, the receiver must be enabled to sense a start bit and a complete character must be assembled and transferred to the data output register.

Failure to read the received character from the R × Data output register prior to the assembly of the next R × data character will set overrun condition error and the previous character will be written over and lost. If the R × data is being read by the CPU when the internal transfer is occurring, overrun error will be set and the old character will be lost.

$\overline{R \times C}$ **(Receiver Clock) :** The receiver clock controls the rate at which the character is to be received. In synchronous mode, the baud rate (1×) is equal to the actual frequency of $\overline{R \times C}$. In asynchronous mode, the baud rate is a fraction of the actual $\overline{R \times C}$ frequency. A portion of the mode instruction selects this factor: 1, 1/16 of 1/64 the $\overline{R \times C}$.

For Example:

Baud rate equals 200 baud, if

$\overline{R \times C}$ equals 200 Hz in the 1 × mode.

$\overline{R \times C}$ equals 3200 Hz in the 16 × mode.

$\overline{R \times C}$ equals 128 KHz in the 64 × mode.

SYNDET (SYNC Detect/BRKDET Break Detect): This is used in Synchronous Mode for SYNDET and may be used as either input or output, programmable through the Control Word. It is reset to output mode low upon RESET. When used as an output (internal Sync mode), the SYNDET pin will go "high" to indicate that the 8251A has located the SYNC character in the Receive mode. If the 8251A is programmed to use double Sync characters (bi-sync), then SYNDET will go "high" in the middle of the last bit of the second Sync character SYNDET is automatically reset upon a status Read operation.

When used as an input (external SYNC detect mode), a positive going signal will cause the 8251A to start assembling data characters on the rising edge of the next $\overline{R \times C}$. Once in SYNC, the "high" input signal can be removed. When External SYNC Detect is programmed, internal SYNC Detect is disabled.

BREAK (Async Mode Only): This output will go high whenever the receiver remains low through two consecutive stop bit sequences (including the start bits, data bits, and parity bits); Break Detect may also be read as a Status bit. It is reset only upon a master chip Reset of R × Data returning to a "one" state.

INTERFACING STANDARDS

(Ref: Interfacing through Microprocessors by K. Subba Rao, Hi-tech publishers, P. 266)

Serial I/O is used to interface various devices or for connecting various equipment to the system. Common understanding is necessary among various manufacturers such that a standard notation is followed for interfacing these components. These standards may be provided by IEEE or by any standard professional organisation. The serial I/O standards must specify clearly voltage levels, speed of data transfer, length of cables etc. In serial I/O data can be transmitted as either current or voltage 20 mA or 60 mA current loops are used if data is transmitted using current. Current flow takes place when the system is at logic 1. The current flow is stopped when the system is at logic 0. In the current loop method the signals are relatively noise-free and they are best suited for long distance transmission.

RS-232 is developed long before which is used for communication between terminals and modems. Using RS-232C data can be transmitted as voltage. The data terminals equipment and data communication equipment are used to communicate using RS-232C cable. RS-232C is not compatible with TTL logic and cannot be used for long distance transmission.

RS-232C Serial Data Standard

(Ref: Microprocessors and interfacing by Douglas V. Hall, 2nd edition, TMH, P.494-495)

OVERVIEW

Modems were developed so that terminals could use phone lines to communicate with distant computers. As we stated earlier, modems and other devices used to send serial data are often referred to as *data communication equipment* or DCE. The terminals or computers that are sending or receiving the data are referred to as *data terminal equipment* or DTE. In response to the need for signal and handshake standards between DTE and DCE, the Electronic Industries Association (EIA) developed EIA standard RS-232C. This standard describes the function of 25 signal and handshake pins for serial-data transfer. It also describes the voltage levels, impedance levels, rise and fall times, maximum bit rate, and maximum capacitance for these signal lines.

RS-232C specifies 25 signal pins, and it specifies that the DTE connector should be a male and the DCE connector should be a female. A specific connector is not given, but the most commonly used connectors are the DB-25P male shown in Figure 14-7a. For systems where many of the 25 pins are not needed, a 9-pin DIN connector such as the DE-9P male connector shown in Figure 14-7b is used.

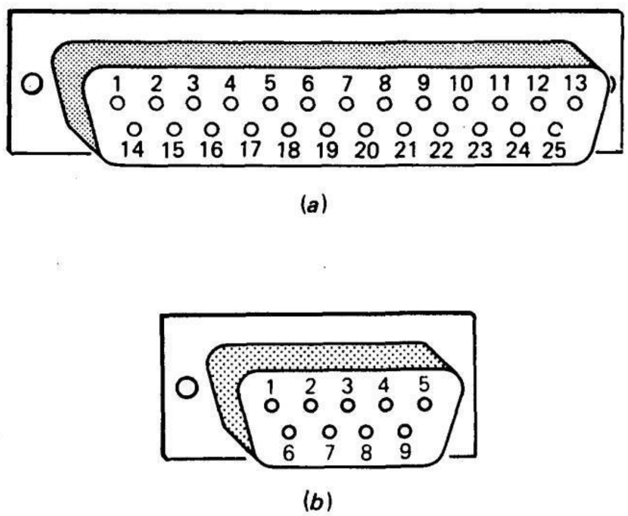


FIGURE 14-7 Connectors often used for RS-232C connections. (*a*) DB-25P 25-pin male. (*b*) DE-9P 9-pin male DIN connector.

The voltage levels for all RS-232C signals are as follows. A logic high, or mark, is a voltage between -3V and -15 V under load (-25 V no load). A logic low or space is a voltage between +3 V and +15 V under load (+ 25 V no load). Voltages such as ±12 V are commonly used.

RS-232C to TTL INTERFACING

Obviously a USART such as the 8251A is not directly compatible with RS-232C signal levels. The standard way to interface between RS-232C and TTL levels is with MCI488 quad TTL-to-RS-232C drivers and MCI489 quad RS-232C-to-TTL receivers shown in Figure 14-8.

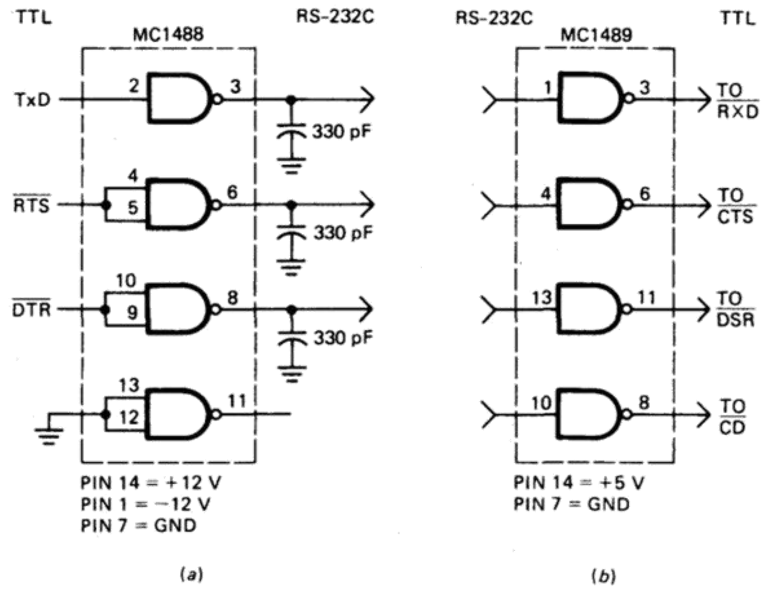


FIGURE 14-8 TTL to RS-232C to TTL signal conversion. (*a*) MC1488 used to convert TTL to RS-232C. (*b*) MC1489 used to convert RS-232C to TTL.

The MCI488s require + and - supplies, but the MCI489s require only+ 5 V. Note the capacitor to ground on the outputs of the MCI488 drivers is to reduce cross talk between adjacent wires, the rise and fall times for RS-232C signals are limited to 30 V/µs.

RS-232C SIGNAL DEFINITIONS

PIN NUMBERS FOR 9 PINS	PIN NUMBERS FOR 25 PINS	COMMON NAME	RS-232C NAME	DESCRIPTION	SIGNAL DIRECTION ON DCE
	1		AA	PROTECTIVE GROUND	–
3	2	TXD	BA	TRANSMITTED DATA	IN
2	3	RXD	BB	RECEIVED DATA	OUT
7	4	$\overline{RTS}$	CA	REQUEST TO SEND	IN
8	5	$\overline{CTS}$	CB	CLEAR TO SEND	OUT
6	6	$\overline{DSR}$	CC	DATA SET READY	OUT
5	7	GND	AB	SIGNAL GROUND (COMMON RETURN)	–
1	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		SBB	SECONDARY RECEIVED DATA	OUT
	17		DD	RECEIVER SIGNAL ELEMENT TIMING (DCE SOURCE)	OUT
	18			UNASSIGNED	–
	19		SCA	SECONDARY REQUEST TO SEND	IN
4	20	$\overline{DTR}$	CD	DATA TERMINAL READY	IN
	21		CG	SIGNAL QUALITY DETECTOR	OUT
9	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	–

FIGURE 14-9 RS-232C pin names and signal directions.

Figure 14-9 shows the signal names, signal direction, and a brief description for each of the 25 pins denned for RS-232C. For most applications only a few of these pins are used.

Note that the signal direction is specified with respect to the DGE, this convention is part of the standard. Note that there is both a chassis ground (pin 1) and a signal ground (pin 7). To prevent large ac-induced ground currents in the signal ground, these two should be connected together only at the power supply in the terminal or the computer.

The TxD, RxD, and handshake signals shown with common names in Figure 14-9 are the ones most often used for simple systems. These signals control what is called the *primary or forward* communications channel of the modem. Some modems allow communication over a secondary or *backward* channel, which operates in the reverse direction from the forward channel and at a much lower baud rate. Pins 12, 13, 14, 16, and 19 are the data and handshake lines for this backward channel. Pins 15, 17, 21, and 24 are used for synchronous data communication.

UNIT -IV

Introduction to Microcontrollers:

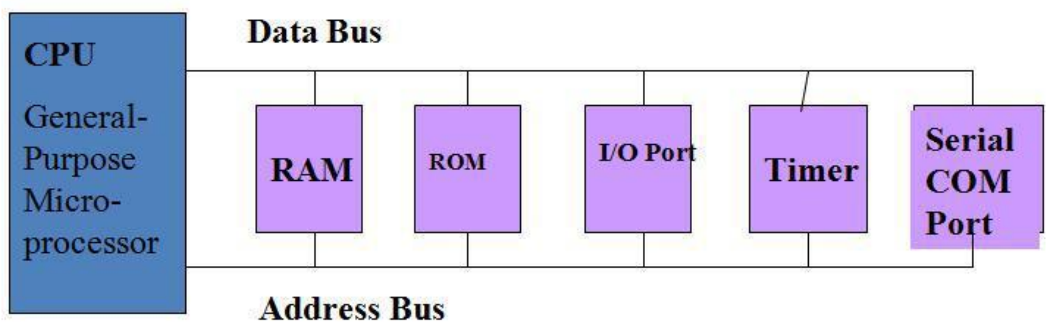
- **Overview of 8051 Microcontroller**
- **Architecture**
- **I/O Ports**
- **Memory Organization**
- **Addressing Modes and Instruction set of 8051**
- **Simple Programs**
- **memory interfacing to 8051**

The necessary tools for a microprocessor/controller:

- CPU: Central Processing Unit
- I/O: Input /Output
- Bus: Address bus & Data bus
- Memory: RAM & ROM
- Timer
- Interrupt
- Serial Port
- Parallel Port
-

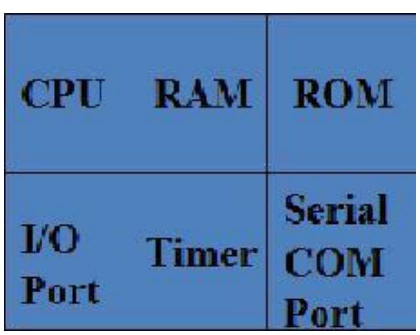
Microprocessors:
General-purpose microprocessor :

- CPU for Computers
- No RAM, ROM, I/O on CPU chip itself
- Example：Intel's x86, Motorola's 680x0



Microcontroller :

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



Microprocessor vs. Microcontroller :

Microprocessor

- CPU is stand-alone, RAM, ROM, I/O, timer are separate
- designer can decide on the amount of ROM, RAM and I/O ports.
- expansive
- versatility
- general-purpose

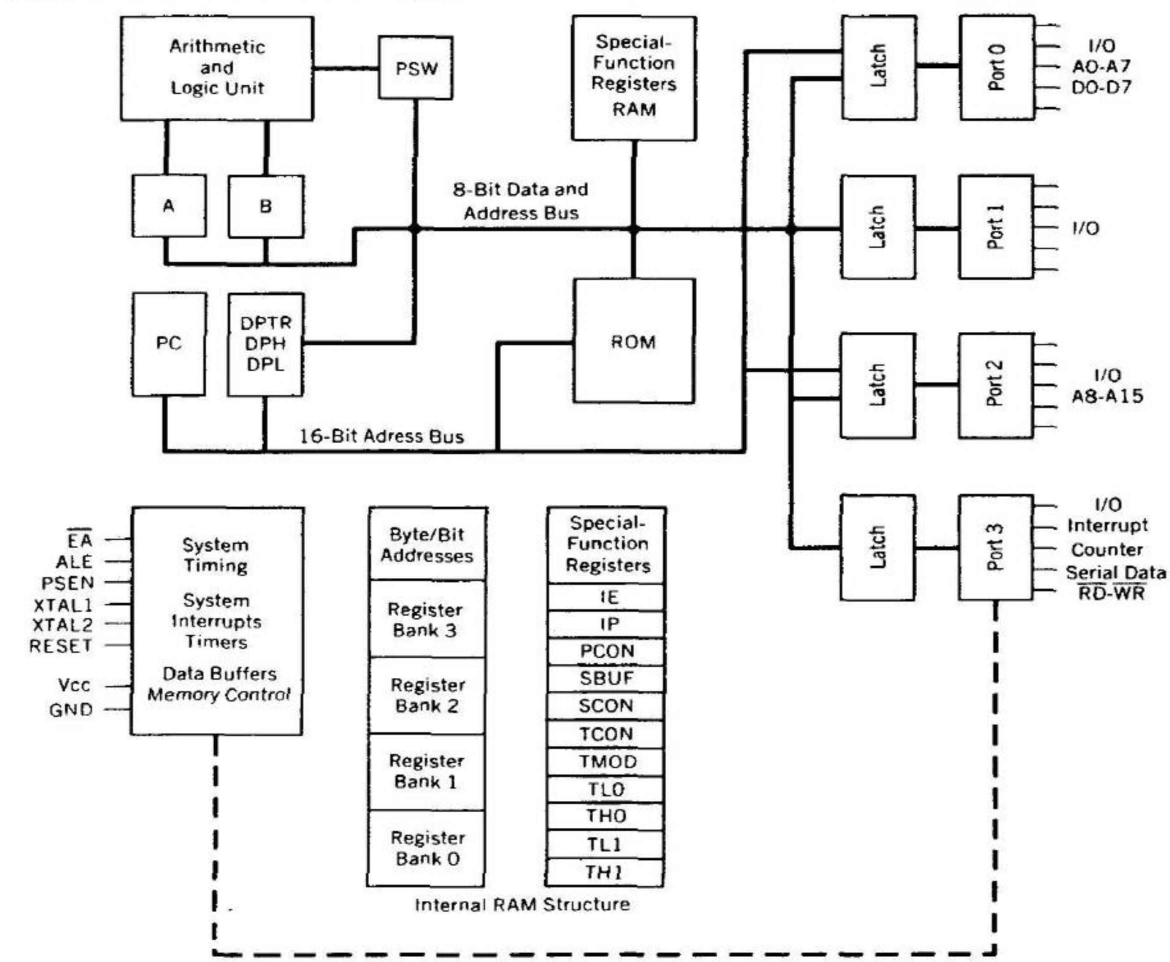
Microcontroller

- CPU, RAM, ROM, I/O and timer are all on a single chip
- fix amount of on-chip ROM, RAM, I/O ports
- for applications in which cost, power and space are critical
- single-purpose

8051 Microcontroller Hardware:

The 8051 microcontroller actually includes a whole family of microcontrollers that have numbers ranging from 8031 to 8751 and are available in N-Channel Metal Oxide Silicon (NMOS) and Complementary Metal Oxide Silicon (CMOS) construction in a variety of

FIGURE 2.1a 8051 Block Diagram



housed in a 40-pin DIP, and direct the investigation of a particular type to the data books.

The block diagram of the 8051 in Figure 2. la shows all of the features unique to microcontrollers:

1. Internal ROM and RAM
2. I/O ports with programmable pins
3. Timers and counters
4. Serial data communication

The figure also shows the usual CPU components: program counter, ALU, working registers, and clock circuits.'

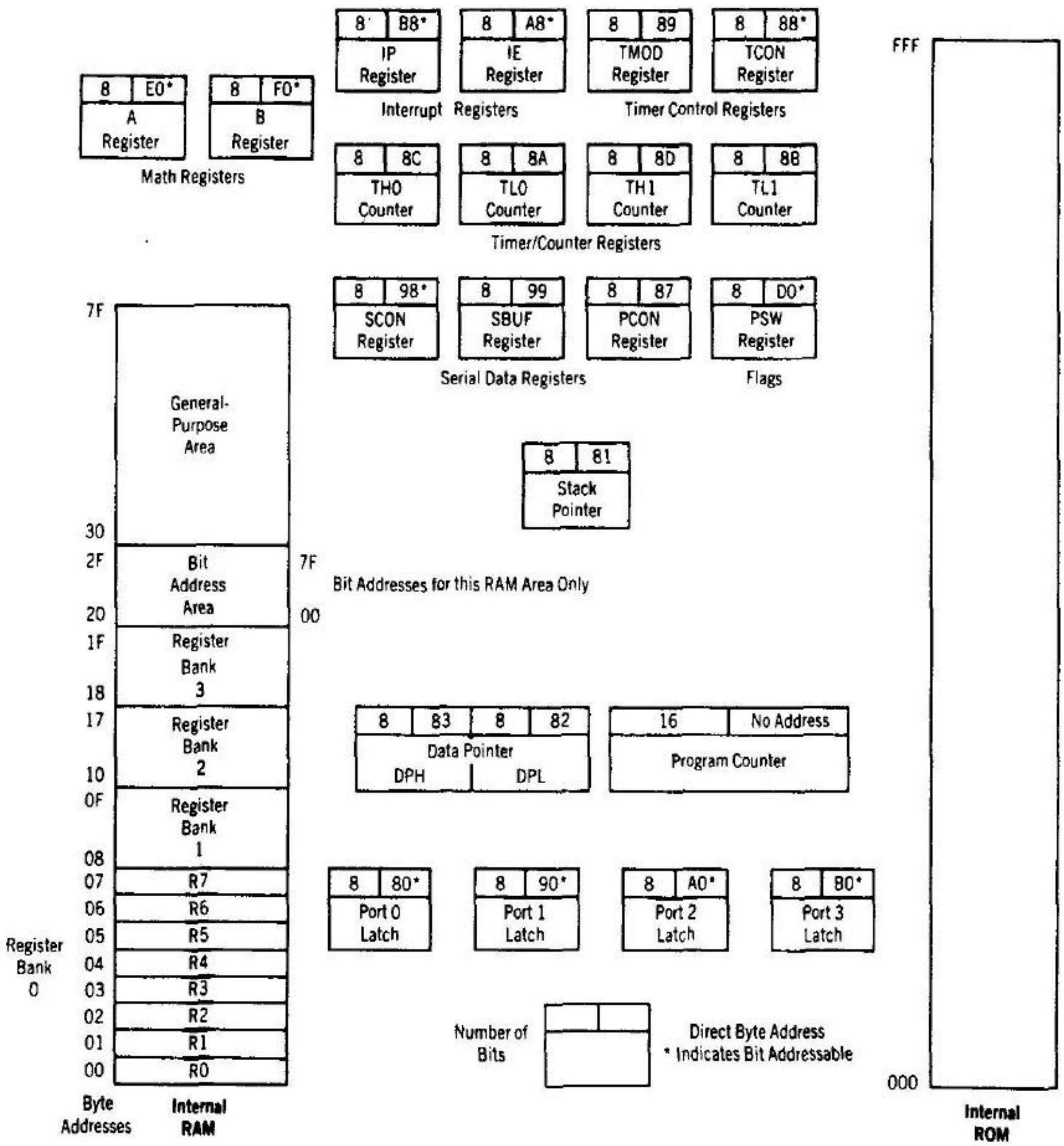
The 8051 architecture consists of these specific features:

- Eight-bit CPU with registers A (the accumulator) and B
- Sixteen-bit program counter (PC) and data pointer (DPTR)
- Eight-bit program status word (PSW)
- Eight-bit stack pointer (SP)
- Internal ROM or EPROM (8751) of 0 (8031) to 4K (8051)
- Internal RAM of 128 bytes:
- Four register banks, each containing eight registers
- Sixteen bytes, which may be addressed at the bit level

- Eighty bytes of general-purpose data memory
- Thirty-two input/output pins arranged as four 8-bit ports: PO-P3
- Two 16-bit timer/counters: TO and Tl
- Full duplex serial data receiver/transmitter: SBUF

- Control registers: TCON, TMOD, SCON, PCON, IP, and IE
- Two external and three internal interrupt sources
- Oscillator and clock circuits

FIGURE 2.1b 8051 Programming Model



The programming model of the 8051 in Figure 2. Ib shows the 8051 as a collection of 8- and 16-bit registers and 8-bit memory locations. These registers and memory locations can be made to operate using the software instructions that are incorporated as part of the design. The program instructions have to do with the control of the registers and digital data paths that are physically contained inside the 8051, as well as memory locations that are physically located outside the 8051.

The model is complicated by the number of special-purpose registers that must be present to make a microcomputer a microcontroller. A cursory inspection of the model is recommended for the first-time viewer; return to the model as needed while progressing through the remainder of the text.

Most of the registers have a specific function; those that do occupy an individual block with a symbolic name, such as A or THO or PC. Others, which are generally indistinguishable from each other, are grouped in a larger block, such as internal ROM or RAM memory.

Each register, with the exception of the program counter, has an internal 1-byte address assigned to it. Some registers (marked with an asterisk * in Figure 2.1b) are both byte and bit addressable. That is, the entire byte of data at such register addresses may be read or altered, or individual bits may be read or altered. Software instructions are generally able to specify a register by its address, its symbolic name, or both. A pinout of the 8051 packaged in a 40-pin DIP is shown in Figure 2.2 with the full and abbreviated names of the signals for each pin. It is important to note that many of the pins are used for more than one function (the alternate functions are shown in parentheses in Figure 2.2). Not all of the possible 8051 features may be used *at the same time.*

Programming instructions or physical pin connections determine the use of any multifunction pins. For example, port 3 bit 0 (abbreviated P3.0) may be used as a general purpose I/O pin, or as an input (RXD) to SBUF, the serial data receiver register. The system designer decides which of these two functions is to be used and designs the hardware and software affecting that pin accordingly.

Program Counter and Data Pointer

The 8051 contains two 16-bit registers: the program counter (PC) and the data pointer (DPTR). Each is used to hold the address of a byte in memory.

Program instruction bytes are fetched from locations in memory that are addressed by the PC. Program ROM may be on the chip at addresses OOOOh to OFFFh, external to the chip for addresses that exceed OFFFh, or totally external for all addresses from OOOOh to FFFFh. The PC is automatically incremented after every instruction byte is fetched and may also be altered by certain instructions. The PC is the only register that does not have an internal address.

The DPTR register is made up of two 8-bit registers, named DPH and DPL, that are

used to furnish memory addresses for internal and external code access and external data access. The DPTR is under the control of program instructions and can be specified by its 16-bit name, DPTR, or by each individual byte name, DPH and DPL. DPTR does not have a single internal address; DPH and DPL are each assigned an address.

A and B CPU Registers

The 8051 contains 34 general-purpose, or working, registers. Two of these, registers A and B, comprise the mathematical core of the 8051 central processing unit (CPU). The other 32 are arranged as part of internal RAM in four banks, BO-B3, of eight registers each, named RO to R7.

The A (accumulator) register is the most versatile of the two CPU registers and is used for many operations, including addition, subtraction, integer multiplication and division, and Boolean bit manipulations. The A register is also used for all data transfers between the 8051 and any external memory. The B register is used with the A register for multiplication and division operations and has no other function other than as a location where data may be stored.

Flags and the Program Status Word (PSW):

Flags are I -bit registers provided to store the results of certain program instructions. Other instructions can test the condition of the flags and make decisions based upon the flag states. In order that the flags may be conveniently addressed, they are grouped inside the program status word (PSW) and the power control (PCON) registers.

The 8051 has four math flags that respond automatically to the outcomes of math operations and three general-purpose user flags that can be set to I or cleared to 0 by the programmer as desired. The math flags include carry (C), auxiliary carry (AC), overflow (OV), and parity (P). User flags are named FO, GFO, and GF1; they are general-purpose flags that may be used by the programmer to record some event in the program. Note that all of the flags can be set and cleared by the programmer at will. The math flags, however, are also affected by math operations.

The program status word is shown in Figure 2.4. The PSW contains the math flags,

user program flag FO, and the register select bits that identify which of the four generalpurpose register banks is currently in use by the program. The remaining two user flags, GFO and GFl, are stored in PCON, which is shown in Figure 2.13.

PSW Program Status Word Register

7	6	5	4	3	2	1	0
CY	AC	F0	RS1	RS0	OV	—	P

THE PROGRAM STATUS WORD (PSW) SPECIAL FUNCTION REGISTER

Bit	Symbol	Function
7	CY	Carry flag; used in arithmetic, JUMP, ROTATE, and BOOLEAN instructions
6	AC	Auxilliary carry flag; used for BCD arithmetic
5	F0	User flag 0
4	RS1	Register bank select bit 1
3	RS0	Register bank select bit 0

RS1	RS0	
0	0	Select register bank 0
0	1	Select register bank 1
1	0	Select register bank 2
1	1	Select register bank 3

Bit	Symbol	Function
2	OV	Overflow flag; used in arithmetic instructions
1	—	Reserved for future use
0	P	Parity flag; shows parity of register A: 1 = Odd Parity

Bit addressable as PSW.0 to PSW.7

Detailed descriptions of the math flag operations will be discussed in chapters that
cover the opcodes that affect the flags. The user flags can be set or cleared using data move instructions covered in Chapter 3.

Internal Memory:

A functioning computer must have memory for program code bytes, commonly in ROM, and RAM memory for variable data that can be altered as the program runs. The 8051 has internal RAM and ROM memory for these functions. Additional memory can be added externally using suitable circuits.

Unlike microcontrollers with Von Neumann architectures, which can use a *single* memory address for either program code or data, *but not for both,* the 8051 has a Harvard architecture, which uses the *same address,* in *different* memories, for code and data. Internal circuitry accesses the correct memory based upon the nature of the operation in progress.

Internal RAM:

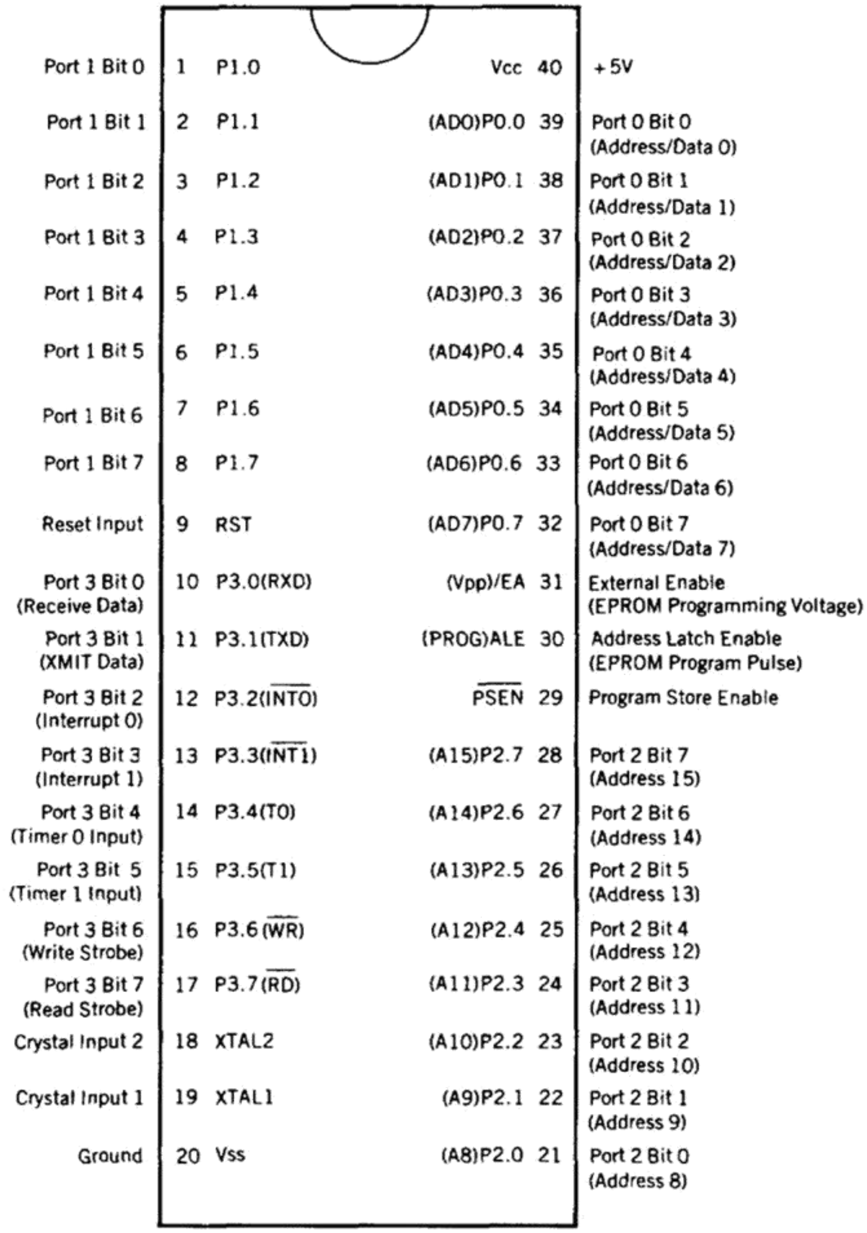
The 128-byte internal RAM, which is shown generally in Figure 2.1 and in detail in Figure 2.5, is organized into three distinct areas:

1. Thirty-two bytes from address OOh to I Fh that make up 32 working registers organized as four banks of eight registers each. The four register banks are numbered 0 to 3 and are made up of eight registers named RO to R7. Each register can be addressed by name (when its bank is selected) or by its RAM address. Thus RO of bank 3 is RO (if bank 3 is currently selected) or address 18h (whether bank 3 is selected or not). Bits RSO and RSI in the PSW determine which bank of registers is currently in use at any time when the program is running. Register banks not selected can be used as general-purpose RAM. Bank 0 is selected upon reset.

2. A Wf-addressable area of 16 bytes occupies RAM *byte* addresses 20h to 2Fh, forming a total of 128 addressable bits. An addressable bit may be specified by its *bit* address of OOh to 7Fh, or 8 bits may form any *byte* address from 20h to 2Fh. Thus, for example, bit address 4Fh is also bit 7 of byte address 29h. Addressable bits are useful when the program need only remember a binary event (switch on, light off, etc.). Internal RAM is in short supply as it is, so why use a byte when a bit will do?

3. A general-purpose RAM area above the bit area, from 30h to 7Fh, addressable as bytes.

FIGURE 2.2 8051 DIP Pin Assignments



Note: Alternate functions are shown below the port name (in parentheses). Pin numbers and pin names are shown inside the DIP package.

PortO:

Port 0 pins may serve as inputs, outputs, or, when used together, as a bi-directional loworder address and data bus for external memory. For example, when a pin is to be used as an input, a 1 *must be* written to the corresponding port 0 latch by the program, thus turning both of the output transistors off, which in turn causes the pin to "float" in a highimpedance state, and the pin is essentially connected to the input buffer.

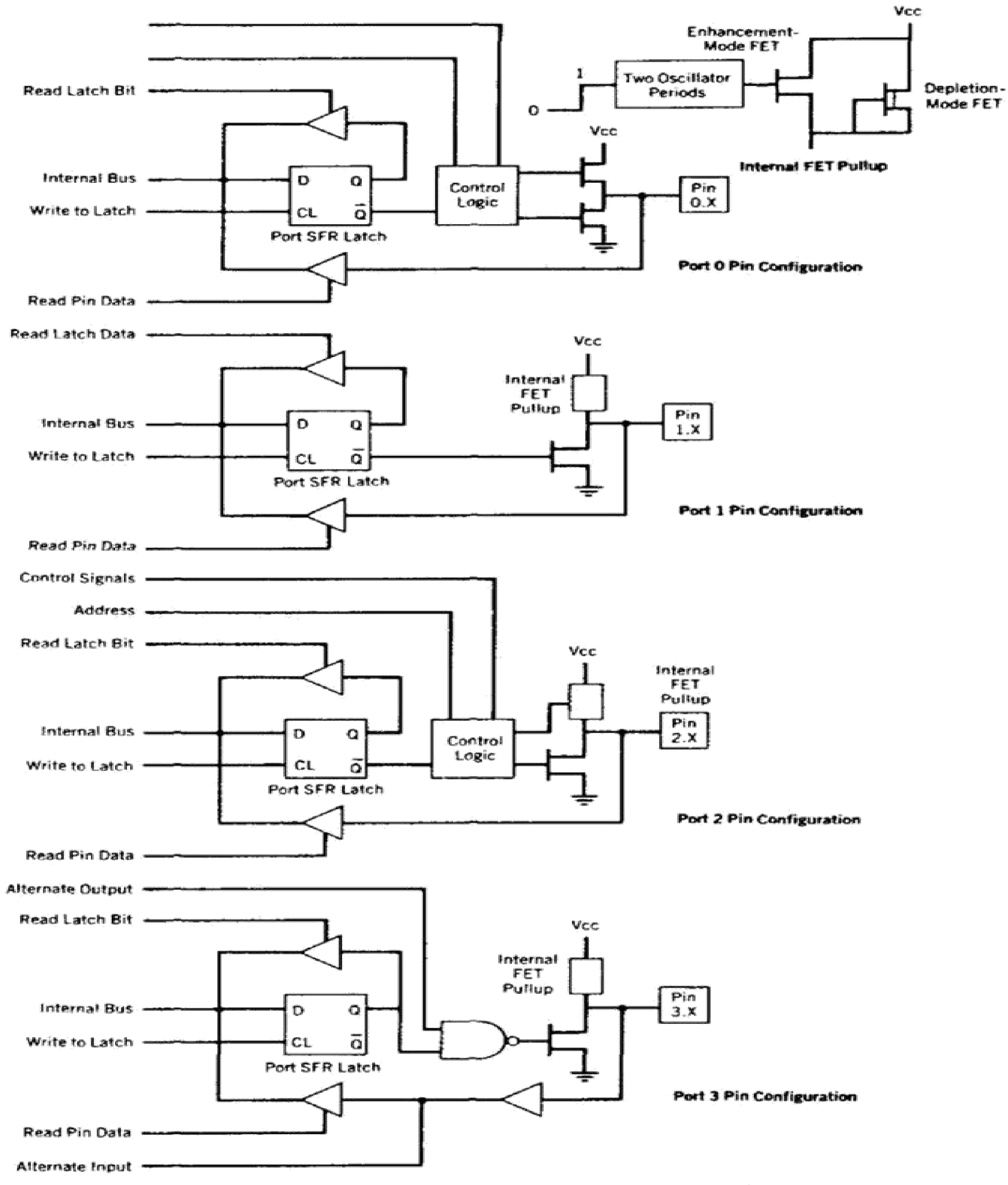
When used as an output, the pin latches that are programmed to a 0 will turn on the

lower FET, grounding the pin. All latches that are programmed to a 1 still float; thus,external pullup resistors will be needed to supply a logic high when using port 0 as an output.

When port 0 is used as an address bus to external memory, internal control signals

switch the address lines to the gates of the Field Effect Transistories (FETs). A logic I on an address bit will turn the upper FET on and the lower FET off to provide a logic high at the pin. When the address bit is a zero, the lower FET is on and the upper FET off to

Port Pin Circuits



provide a logic low at the pin. After the address has been formed and latched into external circuits by the Address Latch Enable (ALE) pulse, the bus is turned around to become a data bus. Port 0 now reads data from the external memory and must be configured as an input, so a logic 1 is automatically written by internal control logic to all port 0 latches.

Port l

Port 1 pins have no dual functions. Therefore, the output latch is connected directly to the gate of the lower FET, which has an FET circuit labeled "Internal FET Pull up" as an active pull up load.

Used as an input, a 1 is written to the latch, turning the lower FET off; the pin and the input to the pin buffer are pulled high by the FET load. An external circuit can overcome the high impedance pull up and drive the pin low to input a 0 or leave the input high for a 1.

If used as an output, the latches containing a I can drive the input of an external circuit high through the pull up. If a 0 is written to the latch, the lower FET is on, the pull up is off, and the pin can drive the input of the external circuit low.

To aid in speeding up switching times when the pin is used as an output, the internal FET pull up has another FET in parallel with it. The second FET is turned on for two oscillator time periods during a low-to-high transition on the pin, as shown in Figure 2.7.

This arrangement provides a low impedance path to the positive voltage supply to help reduce rise times in charging any parasitic capacitances in the external circuitry.

Port 2

Port 2 may be used as an input/output port similar in operation to port 1. The alternate use of port 2 is to supply a high-order address byte in conjunction with the port 0 low-order byte to address external memory.

Port 2 pins are momentarily changed by the address control signals when supplying the high byte of a 16-bit address. Port 2 latches remain stable when external memory is addressed, as they do not have to be turned around (set to 1) for data input as is the case for port 0.

Port3

Port 3 is an input/output port similar to port I. The input and output functions can be programmed under the control of the P3 latches or under the control of various other special function registers. The port 3 alternate uses are shown in the following table:-

PIN	ALTERNATE USE	SFR
P3.0–RXD	Serial data input	SBUF
P3.1–TXD	Serial data output	SBUF
P3.2–$\overline{\text{INT0}}$	External interrupt 0	TCON.1
P3.3–$\overline{\text{INT1}}$	External interrupt 1	TCON.3
P3.4–T0	External timer 0 input	TMOD
P3.5–T1	External timer 1 input	TMOD
P3.6–$\overline{\text{WR}}$	External memory write pulse	—
P3.7–$\overline{\text{RD}}$	External memory read pulse	—

Unlike ports 0 and 2, which can have external addressing functions and change all

eight port bits when in alternate use, each pin of port 3 may be individually programmed to be used either as I/O or as one of the alternate functions.

External Memory

The system designer is not limited by the amount of internal RAM and ROM available on chip. Two separate external memory spaces are made available by the 16-bit PC and DPTR and by different control pins for enabling external ROM and RAM chips. Internal control circuitry accesses the correct physical memory, depending upon the machine cycle state and the op code being executed.

There are several reasons for adding external memory, particularly program memory, when applying the 8051 in a system. When the project is in the prototype stage, the expense—in time and money—of having a masked internal ROM made for each program "try" is prohibitive.

To alleviate this problem, the manufacturers make available an EPROM version, the 8751, which has 4K of on-chip EPROM that may be programmed and erased as needed as the program is developed. The resulting circuit board layout will be identical to one that uses a factory-programmed 8051. The only drawbacks to the 8751 are the specialized EPROM programmers that must be used to program the non-standard 40-pin part, and the limit of "only" 4096 bytes of program code. The 8751 solution works well if the program will fit into 4K bytes. Unfortunately, many times, particularly if the program is written in a high-level language, the program size exceeds 4K bytes, and an external program memory is needed. Again, the manufacturers provide a version for the job, the ROMIess 8031. The EA pin is grounded when using the 8031, and all program code is contained in an external EPROM that may be as large as 64K bytes and that can be programmed using standard EPROM programmers.

External RAM, which is accessed by the DPTR, may also be needed when 128 bytes of internal data storage is not sufficient. External RAM, up to 64K bytes, may also be added to any chip in the 8051 family.

Connecting External Memory

Figure 2.8 shows the connections between an 8031 and an external memory configuration consisting of I6K bytes of EPROM and 8K bytes of static RAM. The 8051 accesses external RAM whenever certain program instructions are executed. External ROM is accessed whenever the EA (external access) pin is connected to ground or when the PC contains an address higher than the last address in the internal 4K bytes ROM (OFFFh). 8051 designs can thus use internal and external ROM automatically; the 8031, having no internal ROM, must have EA grounded.

Figure 2.9 shows the timing associated with an external memory access cycle. During any memory access cycle, port 0 is time multiplexed. That is, it first provides the lower byte of the 16-bit memory address, then acts as a bidirectional data bus to write or read a byte of memory data. Port 2 provides the high byte of the memory address during the entire memory read/write cycle.

The lower address byte from port 0 must be latched into an external register to save
the byte. Address byte save is accomplished by the ALE clock pulse that provides the correct timing for the '373 type data latch. The port 0 pins then become free to serve as a data bus.

If the memory access is for a byte of program code in the ROM, the PSEN (program store enable) pin will go low to enable the ROM to place a byte of program code on the data bus. If the access is for a RAM byte, the WR (write) or RD (read) pins will go low, enabling data to flow between the RAM and the data bus.

The ROM may be expanded to 64K by using a 27512 type EPROM and connecting the remaining port 2 upper address lines AI4-A15 to the chip.

At this time the largest static RAMs available are 32K in size; RAM can be expanded to 64K by using two 32K RAMs that are connected through address A14 of port 2. The

FIGURE 2.8 External Memory Connections

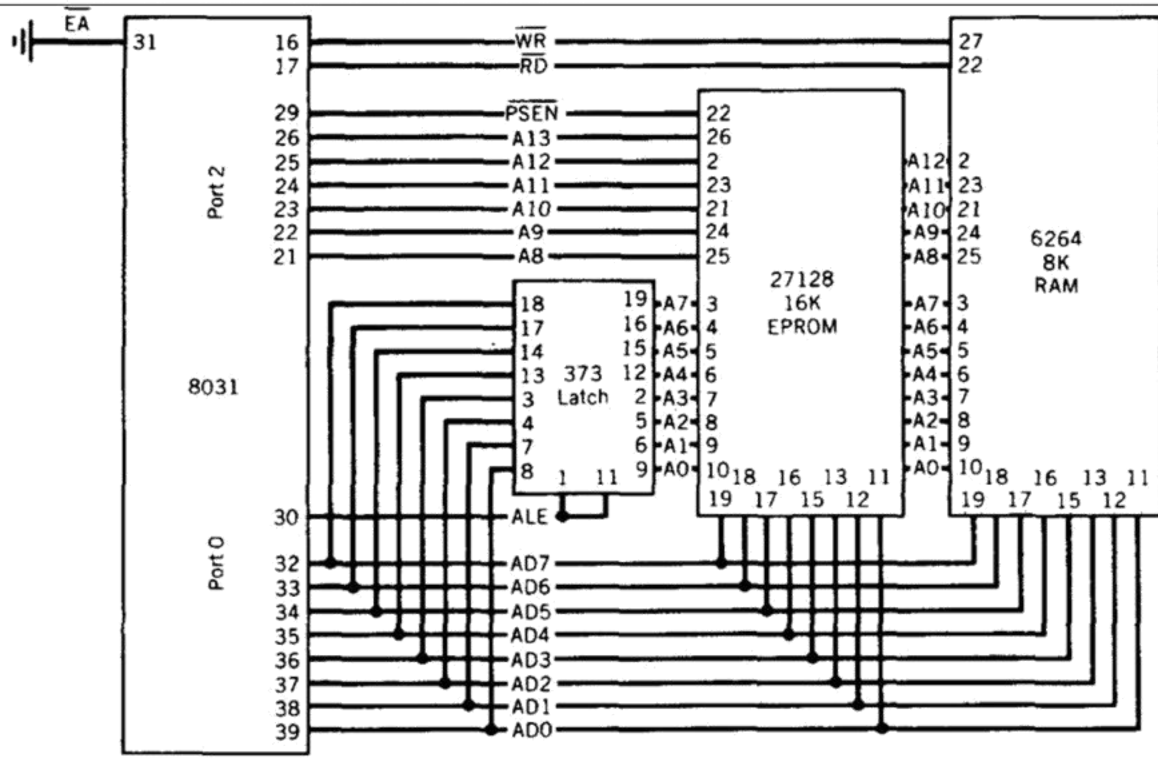
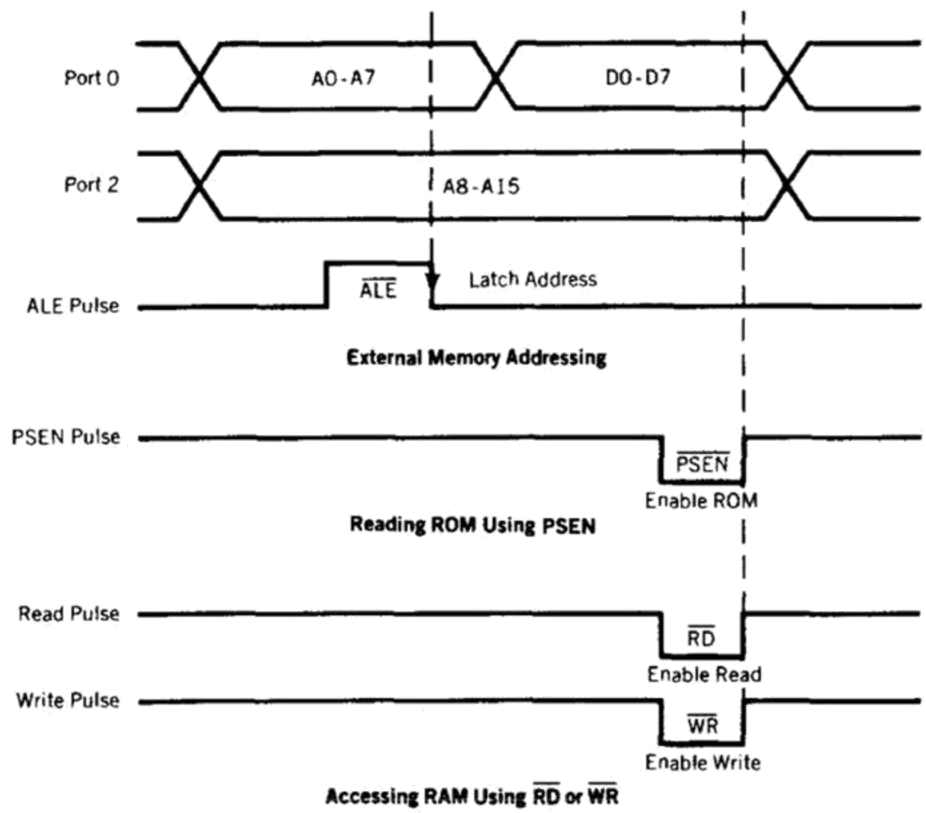


FIGURE 2.9 External Memory Timing



first 32K RAM (OOOOh-7FFFh) can then be enabled when AI5 of port 2 is low, and the second 32K RAM (SOOOh-FFFFh) when A15 is high, by using an inverter.

Note that the WR and RD signals are alternate uses for port 3 pins 16 and 17. Also,

port 0 is used for the lower address byte and data; port 2 is used for upper address bits. The use of external memory consumes many of the port pins, leaving only port 1 and parts of port 3 for general I/O.

8051 INSTRUCTION SET

8051 has about 111 instructions. These can be grouped into the following categories

- Arithmetic Instructions
- Logical Instructions
- Data Transfer instructions
- Boolean Variable Instructions
- Program Branching Instructions

The following nomenclatures for register, data, address and variables are used while write instructions

- A: Accumulator
- B: "B" register
- C: Carry bit
- Rn: Register R0 - R7 of the currently selected register bank
- Direct: 8-bit internal direct address for data. The data could be in lower 128bytes of RAM (00 - 7FH) or it could be in the special function register (80 - FFH).
- @Ri: 8-bit external or internal RAM address available in register R0 or R1. This is used for indirect addressing mode.
- #data8: Immediate 8-bit data available in the instruction.
- #data16: Immediate 16-bit data available in the instruction.
- Addr11: 11-bit destination address for short absolute jump. Used by instructions AJMP & ACALL. Jump range is 2 kbyte (one page).
- Addr16: 16-bit destination address for long call or long jump.
- Rel: 2's complement 8-bit offset (one - byte) used for short jump (SJMP) and all conditional jumps.
- bit: Directly addressed bit in internal RAM or SFR

Some Simple Instructions:

```
MOV dest,source              ; dest = source
    MOV A,#72H               ;A=72H
    MOV R4,#62H ;R4=62H
    MOV   B,0F9H             ;B=the content of F9'th byte of RAM
    MOV   DPTR,#7634H
    MOV   DPL,#34H
    MOV   DPH,#76H
    MOV   P1,A               ;mov A to port 1
```

Note 1:

MOV A,#72H ≠ MOV A,72H

After instruction "MOV A,72H " the content of 72'th byte of RAM will replace in Accumulator.

Note 2:

MOV A,R3 ≡ MOV A,3

```
ADD   A, Source              ;A=A+SOURCE
    ADD   A,#6               ;A=A+6
    ADD   A,R6               ;A=A+R6

    ADD   A,6                ;A=A+[6] or A=A+R6
    ADD         A,0F3H       ;A=A+[0F3H]
SUBB  A, Source              ;A=A-SOURCE-C
    SUBB  A,#6               ;A=A-6
    SUBB  A,R6               ;A=A+R6
```

MUl & Div:

- MUL AB ;B|A = A*B

```
    MOV   A,#25H
    MOV   B,#65H
    MUL   AB                 ;25H*65H=0E99
                             ;B=0EH, A=99H
```

- DIV AB ;A = A/B, B = A mod B

```
MOV   A,#25
      MOV   B,#10
      DIV   AB          ;A=2, B=5
SETB  bit         ; bit=1
CLR   bit         ; bit=0
SETB  C           ; CY=1
SETB  P0.0        ;bit 0 from port 0 =1
SETB  P3.7        ;bit 7 from port 3 =1
SETB  ACC.2       ;bit 2 from ACCUMULATOR =1
SETB  05          ;set high D5 of RAM loc. 20h
```

Note:

CLR instruction is as same as
SETB i.e.:

```
      CLR         C     ;CY=0
```

But following instruction is only for CLR:

```
      CLR         A     ;A=0
DEC  byte         ;byte=byte-1
INC         byte        ;byte=byte+1
INC         R7
DEC         A
DEC         40H         ; [40]=[40]-1
```

RR – RL – RRC – RLC A

EXAMPLE:
RR A

RR:

RRC:

RL:

RLC:

ANL - ORL – XRL

Bitwise Logical Operations:

AND, OR, XOR

EXAMPLE:

```
      MOV   R5,#89H
      ANL   R5,#08H
```

```
CPL         A           ;1's complement
Example:
            MOV         A,#55H ;A=01010101 B
L01:  CPL         A
            MOV         P1,A
            ACALL       DELAY
            SJMP        L01
```

UNIT-V

8051 Real Time Control:

- Programming Timer Interrupts
- Programming External Hardware
- Interrupts
- Programming the Serial Communication Interrupts
- Programming 8051 Timers and Counters

ARM Processor:

- Fundamentals
- Registers
- current program status register,
- pipeline
- Interrupt and the vector table

Interrupts:

1. Enabling and Disabling Interrupts
2. Interrupt Priority
3. Writing the ISR (Interrupt Service Routine)

Interrupt Enable (IE) Register :

- EA : Global enable/disable.
- --- : Undefined.
- ET2 :Enable Timer 2 interrupt.
- ES :Enable Serial port interrupt.
- ET1 :Enable Timer 1 interrupt.
- EX1 :Enable External 1 interrupt.
- ET0 : Enable Timer 0 interrupt.
- EX0 : Enable External 0 interrupt.

EA	—	ET2	ES	ET1	EX1	ET0	EX0

Interrupt Vectors:

Interrupt	Vector Address
System Reset	0000H
External 0	0003H
Timer 0	000BH
External 1	0013H
Timer 1	001BH
Serial Port	0023H
Timer 2	002BH

Peripheral Control Registers

PCON (Power Control)

The PCON or Power Control register, as the name suggests is used to control the 8051 Microcontroller’s Power Modes and is located at 87H of the SFR Memory Space. Using two bits in the PCON Register, the microcontroller can be set to Idle Mode and Power Down Mode.

During Idle Mode, the Microcontroller will stop the Clock Signal to the ALU (CPU) but it is given to other peripherals like Timer, Serial, Interrupts, etc. In order to terminate the Idle Mode, you have to use an Interrupt or Hardware Reset.

In the Power Down Mode, the oscillator will be stopped and the power will be reduced to 2V. To terminate the Power Down Mode, you have to use the Hardware Reset.

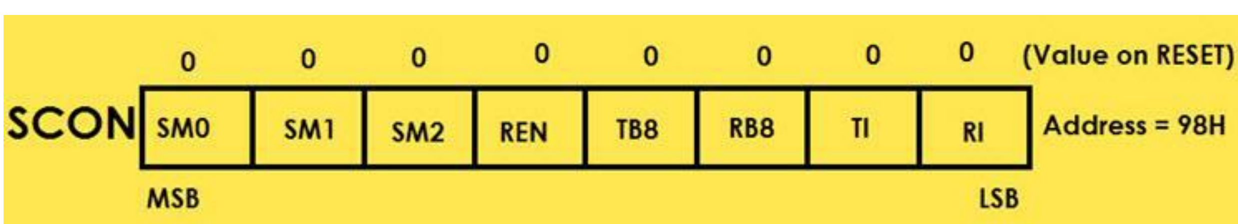
Apart from these two, the PCON Register can also be used for few additional purposes. The SMOD Bit in the PCON Register is used to control the Baud Rate of the Serial Port.

There are two general purpose Flag Bits in the PCON Register, which can be used by the programmer during execution.

SCON (Serial Control)

The Serial Control or SCON SFR is used to control the 8051 Microcontroller’s Serial Port. It is located as an address of 98H. Using SCON, you can control the Operation Modes of the Serial Port, Baud Rate of the Serial Port and Send or Receive Data using Serial Port.

SCON Register also consists of bits that are automatically SET when a byte of data is transmitted or received.

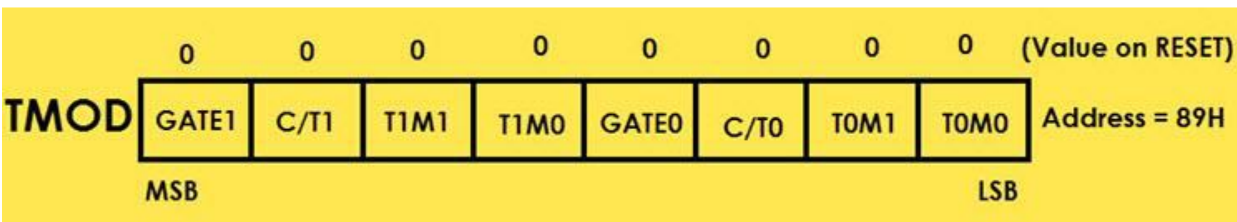


TCON (Timer Control)

Timer Control or TCON Register is used to start or stop the Timers of 8051 Microcontroller. It also contains bits to indicate if the Timers has overflowed. The TCON SFR also consists of Interrupt related bits.

TMOD (Timer Mode)

The TMOD or Timer Mode register or SFR is used to set the Operating Modes of the Timers T0 and T1. The lower four bits are used to configure Timer0 and the higher four bits are used to configure Timer1.



The Gatex bit is used to operate the Timerx with respect to the INTx pin or regardless of the INTx pin.

GATE1 = 1 ==> Timer1 is operated only if INT1 is SET.

GATE1 = 0 ==> Timer1 is operates irrespective of INT1 pin.

GATE0 = 1 ==> Timer0 is operated only if INT0 is SET.

GATE0 = 0 ==> Timer0 is operates irrespective of INT0 pin.

The C/Tx bit is used selects the source of pulses for the Timer to count.

C/T1 = 1 ==> Timer1 counts pulses from Pin T1 (P3.5) (Counter Mode)

C/T1 = 0 ==> Timer1 counts pulses from internal oscillator (Timer Mode)

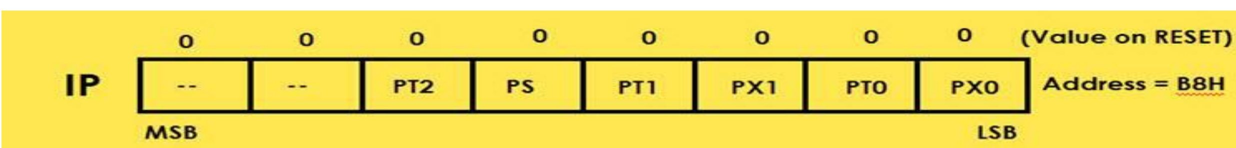
C/T0 = 1 ==> Timer0 counts pulses from Pin T0 (P3.4) (Counter Mode)

C/T0 = 0 ==> Timer0 counts pulses from internal oscillator (Timer Mode)

TxM0	**TxM1**	**Mode**	**Description**
0	0	0	13-bit Timer Mode (THx – 8-bit and TLx – 5-bit)
0	1	1	16-bit Timer Mode
1	0	2	8-bit Auto Reload Timer Mode
1	1	3	Two 8-bit Timer Mode or Split Timer Mode

IP (Interrupt Priority)

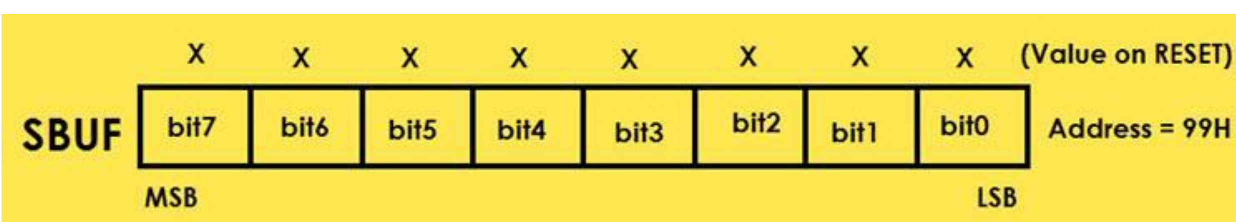
The IP or Interrupt Priority Register is used to set the priority of the interrupt as High or Low. If a bit is CLEARED, the corresponding interrupt is assigned low priority and if the bit is SET, the interrupt is assigned high priority.



Peripheral Data Registers

SBUF (Serial Data Buffer)

The Serial Buffer or SBUF register is used to hold the serial data while transmission or reception.



ARM processor

An ARM processor is one of a family of CPUs based on the RISC (reduced instruction set computer) architecture developed by Advanced RISC Machines (ARM).

ARM makes 32-bit and 64-bit RISC multi-core processors. RISC processors are designed to perform a smaller number of types of computer instructions so that they can operate at a higher speed, performing more millions of instructions per second (MIPS). By stripping out unneeded instructions and optimizing pathways, RISC processors provide outstanding performance at a fraction of the power demand of CISC (complex instruction set computing) devices.

ARM processors are extensively used in consumer electronic devices such as smartphones, tablets, multimedia players and other mobile devices, such as wearables. Because of their reduced instruction set, they require fewer transistors, which enables a smaller die size for the integrated circuitry (IC). The ARM processor's smaller size, reduced complexity and lower power consumption makes them suitable for increasingly miniaturized devices.

ARM processor features include:

- Load/store architecture.
- An orthogonal instruction set.
- Mostly single-cycle execution.
- Enhanced power-saving design.
- 64 and 32-bit execution states for scalable high performance.
- Hardware virtualization support.

The simplified design of ARM processors enables more efficient multi-core processing and easier coding for developers. While they don't have the same raw compute throughput as the products of x86 market leader Intel, ARM processors sometimes exceed the performance of Intel processors for applications that exist on both architectures.

The head-to-head competition between the vendors is increasing as ARM is finding its way into full size notebooks. Microsoft, for example, offers ARM-based versions of Surface computers. The cleaner code base of Windows RT versus x86 versions may be also partially responsible -- Windows RT is more streamlined because it doesn't have to support a number of legacy hardwares.

ARM is also moving into the server market, a move that represents a large change in direction and a hedging of bets on performance-per-watt over raw compute power. AMD offers 8-core versions of ARM processors for its Opteron series of processors. ARM serversrepresent an important shift in server-based computing. A traditional x86-class server with 12, 16, 24 or more cores increases performance by scaling up the speed and sophistication of each processor, using brute force speed and power to handle demanding computing workloads.

In comparison, an ARM server uses perhaps hundreds of smaller, less sophisticated, low-power processors that share processing tasks among that large number instead of just a few higher-capacity processors. This approach is sometimes referred to as "scaling out," in contrast with the "scaling up" of x86-based servers.

ARM registers

ARM processors provide general-purpose and special-purpose registers. Some additional registers are available in privileged execution modes.

In all ARM processors, the following registers are available and accessible in any processor mode:

- 13 general-purpose registers R0-R12.
- One *Stack Pointer* (SP).
- One *Link Register* (LR).
- One *Program Counter* (PC).
- One *Application Program Status Register* (APSR).

ARM processors, with the exception of ARMv6-M and ARMv7-M based processors, have a total of 37 registers, with 3 additional registers if the Security Extensions are implemented, and in ARMv7-A only, 3 more if the Virtualization Extensions are implemented. The registers are arranged in partially overlapping banks. There is a different register bank for each processor mode. The banked registers give rapid context switching for dealing with processor exceptions and privileged operations.

The additional registers that are available in privileged software execution, with the exception of ARMv6-M and ARMv7-M, are:

- Two Supervisor mode registers for banked SP and LR.
- Two Abort mode registers for banked SP and LR.
- Two Undefined mode registers for banked SP and LR.
- Two Interrupt mode registers for banked SP and LR.
- Seven FIQ mode registers for banked R8-R12, SP and LR.
- Two Monitor mode registers for banked SP and LR. These are only present if the Security Extensions are implemented.
- Two Hyp mode registers for banked SP, and to hold the return address from Hyp mode. These are only present if the Virtualization Extensions are implemented.
- One *Saved Program Status Register* (SPSR) for each exception mode.

Current Program Status Register

The *Current Program Status Register* (CPSR) holds the same program status flags as the APSR, and some additional information.

The CPSR holds:

- The APSR flags.
- The processor mode.
- The interrupt disable flags.
- The instruction set state (ARM, Thumb, ThumbEE, or Jazelle®).
- The endianness state (on ARMv4T and later).
- The execution state bits for the IT block (on ARMv6T2 and later).
-

The execution state bits control conditional execution in the IT block.
Only the APSR flags are accessible in all modes. ARM deprecates using an MSR instruction to change the endianness bit (E) of the CPSR, in any mode. SETEND is the preferred instruction to write to the E bit.

The execution state bits for the IT block (IT[1:0]), Jazelle bit (J), and Thumb bit (T) can be accessed by MRSonly in Debug state.

The instruction pipeline

The ARM uses a pipeline to increase the speed of the flow of instructions to the processor. This allows several operations to take place simultaneously, and the processing, and memory systems to operate continuously.

A three-stage pipeline is used, so instructions are executed in three stages:

- Fetch
- Decode
- Execute.

During normal operation, while one instruction is being executed, its successor is being decoded, and a third instruction is being fetched from memory.