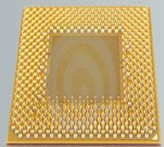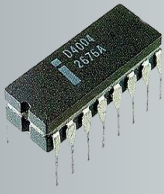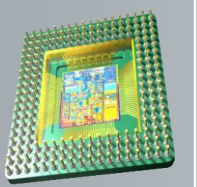# CSC 153

MICROPROCESSOR

Second Semester
BSc.CSIT

Presented by:

1. Introduction

1.1 History of microprocessor

One of the greatest inventions that have changed man's life is the computer. History reveals that the evolution of computer started with the development of calculator. Firstly there were mechanical computers which have now evolved to electronic ones. The mechanical computers which cannot be put aside are 'difference engine' and 'analytical engine'. Both of them were developed by Charles Babbage, **the father of the computer**.

The 'difference engine' was a mechanical device that could add and subtract, but could only run a single algorithm. Its output system was incompatible to write on medium such as punched cards and early optical disks. The 'analytical engine' was a improvement on 'difference engine' and provided features such as: the store (memory), the mill (computation unit), input section (punched card reader) and output section (punched and printed output). The store consisted of 1000 words of 50 decimal digits used to hold variables and results. The mill could accept operands from the store, add, subtract, multiply or divide them, and return a result to the store. The great advantage of the analytical engine was that it was general purpose. It read instructions from punched cards and carried them, out. By punching a different program on the input cards, it was possible to have the analytical engine perform different computations.

The evolution of the vacuum tubes led the development of computers into a new era. The world's first general purpose electronic digital computer was ENIAC (Electronic Numerical Integrator and calculator). It was designed and constructed under the supervision of John Mauchly and John Presper Eckert at the University of Pennsylvania. The ENIAC built by using vacuum tubes was enormous in size and consumed very high power. However it was faster than mechanical computers.

The ENIAC was a decimal machine, in which the numbers were in decimal number system and the arithmetic was also performed in the same number system. Its memory consisted of 20 'accumulators' each capable of holding a 10 digit decimal numbers. Each digit was represented by the 'ON' state of the vacuum tubes. The main drawback of the ENIAC was that it had to be programmed manually by setting switches and plugging and unplug cables.

1.2 Calculator and Stored Program Computer

A calculator is a data processing device that carries out logic and arithmetic operations, but has limited programming capability for the user. The calculator accepts data from a small keyboard, one digit at a time, performs the required arithmetic and logical calculation and show the result on visual display via LCD or LED. The calculator's programs are stored in ROM (Read only Memory) while the data from users are stored in RAM (Random access Memory).

1.3 Von Neumann and Harvard architecture

As we know the task of entering and altering the programs for the ENIAC was extremely tedious. The programming process could be facilitated if the program could be represented in a form suitable for storing in memory alongside the data. Then a computer could get its instructions by reading them from the memory and a program could be set or altered by

setting the values of a portion of memory. This approach is known as '**stored program concept**' and was first adopted by **John von Neumann**. Hence, this architecture is also called **Von-Newman's architecture**. The general structure of this architecture is shown below.
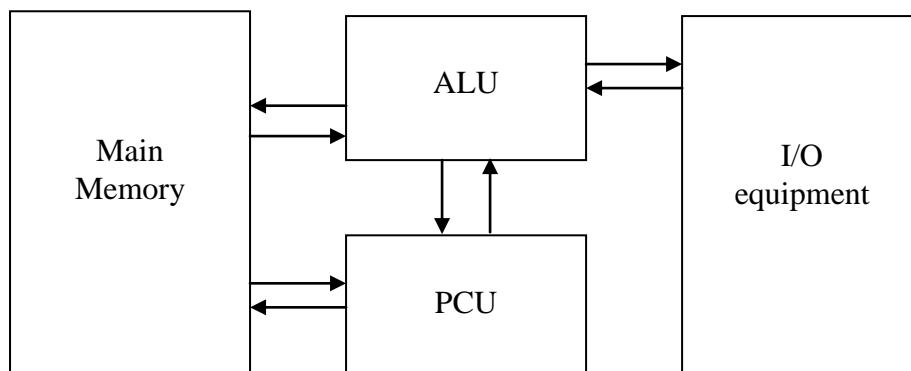

Figure: Von Neumann architecture

As shown in figure 1.1, the main memory is used to store both data and instructions. The ALU is capable of performing arithmetic and logical operations on binary data. The program control unit interprets the instructions in memory and causes them to be executed. The I/O unit gets operated from the control unit.

The Von-Neumann's Architecture is the fundamental basis for the architecture of today's digital computers. Thus it is important to have an idea of the internal structures of the central processing unit (CPU), program control unit of Von-Neumann's machine.

The memory of the Neumann's machine consists of 1000 storage locations, called words of 40 binary digits (bits). Both data and instructions are stored in it. The control unit operates the computer by fetching instructions from memory and executing them one at a time. The storage locations of the control unit and ALU are called registers. The various registers of this model are:

- MBR (Memory Buffer Register): consists of a word to be stored in memory or is used to receive a word from memory.
- MAR (Memory Address Register): contains the address in memory of the word to be written from or read into the MBR.
- IR (Instruction Register): contains the 8-bit op code instruction being executed.
- IBR (Instruction Buffer Register): used to temporarily hold the instruction from a word in memory.
- PC (Program Counter): contains the address of the next instruction to be fetched from memory.
- The Accumulator (AC) and the Multiplier Quotient (MQ) are employed to temporarily hold the operands and results of ALU operations.

In Von-Neumann's architecture, the same memory is used for storing instructions and data. Similarly, a single bus called data bus or address bus is used for reading data and instructions from or writing data and instructions to the memory. Later, it was revealed that this feature of Von-Neumann's architecture limited the processing speed of the computer. So in order to improve the processing speed of the computer, Harvard architecture was evolved.

The Harvard architecture consists of separate memory location for the programs and data. Each memory space has its own address and data buses. As a result of this both instruction and data can be fetched from memory concurrently. Thus a significant processing speed improvement is observed over Von-Neumann type architecture. The Harvard architecture is shown below:
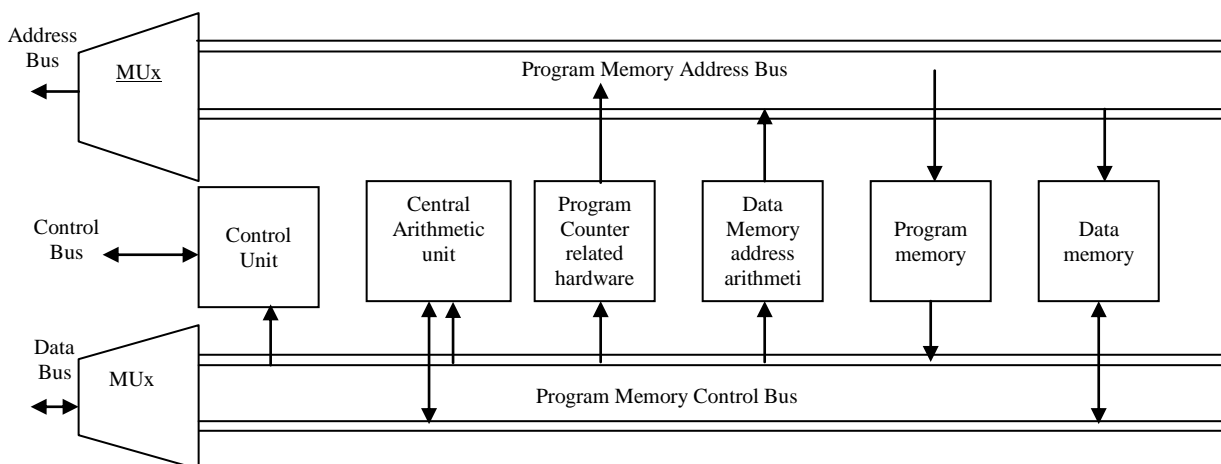

Fig: Block diagram of a Harvard architecture based microprocessor

From the figure, we see there are two data and two addresses buses, for the program and data memory spaces separately. The program memory data bus and data memory data bus are multiplexed to form single data bus where as program memory address bus and data memory address bus are multiplexed to form a single address bus. Hence there are two blocks of RAM chips, one for program memory and other for data memory space. The data memory address arithmetic unit generates data memory addresses. The data memory address bus carries the memory address of the data where as the program memory address bus carries the memory address of the instruction. There is a central arithmetic logic unit (ALU), which consists of the ALU, the multiplier, accumulator, and scaling chief register. The program counter is used to address program memory. The PC contents are updated following each instruction decode operation. The control unit controls the sequence of operations to be executed. The data and control bus are bi directional where as address bus is unidirectional.

1.4 Simple stored program computer architecture

See Von Neumann architecture

1.5 Microprocessor Architecture (8 bit)

The 8085A is an 8 bit general purpose microprocessor capable of addressing 64K of memory. The main components of the 8085 include the ALU, timing and control unit, Instruction register and decoder, Register array. These are linked by an internal data bus.

The ALU

The arithmetic/logic unit performs the computing functions; it includes the accumulator the temporary register, the arithmetic and logic circuits, and five flags. The temporary register is used to hold the data during an arithmetic/logic operation. The result of the operation is stored in the accumulator and flags are set or reset according to the result of operation. Flags generally reflect data conditions in the accumulator.
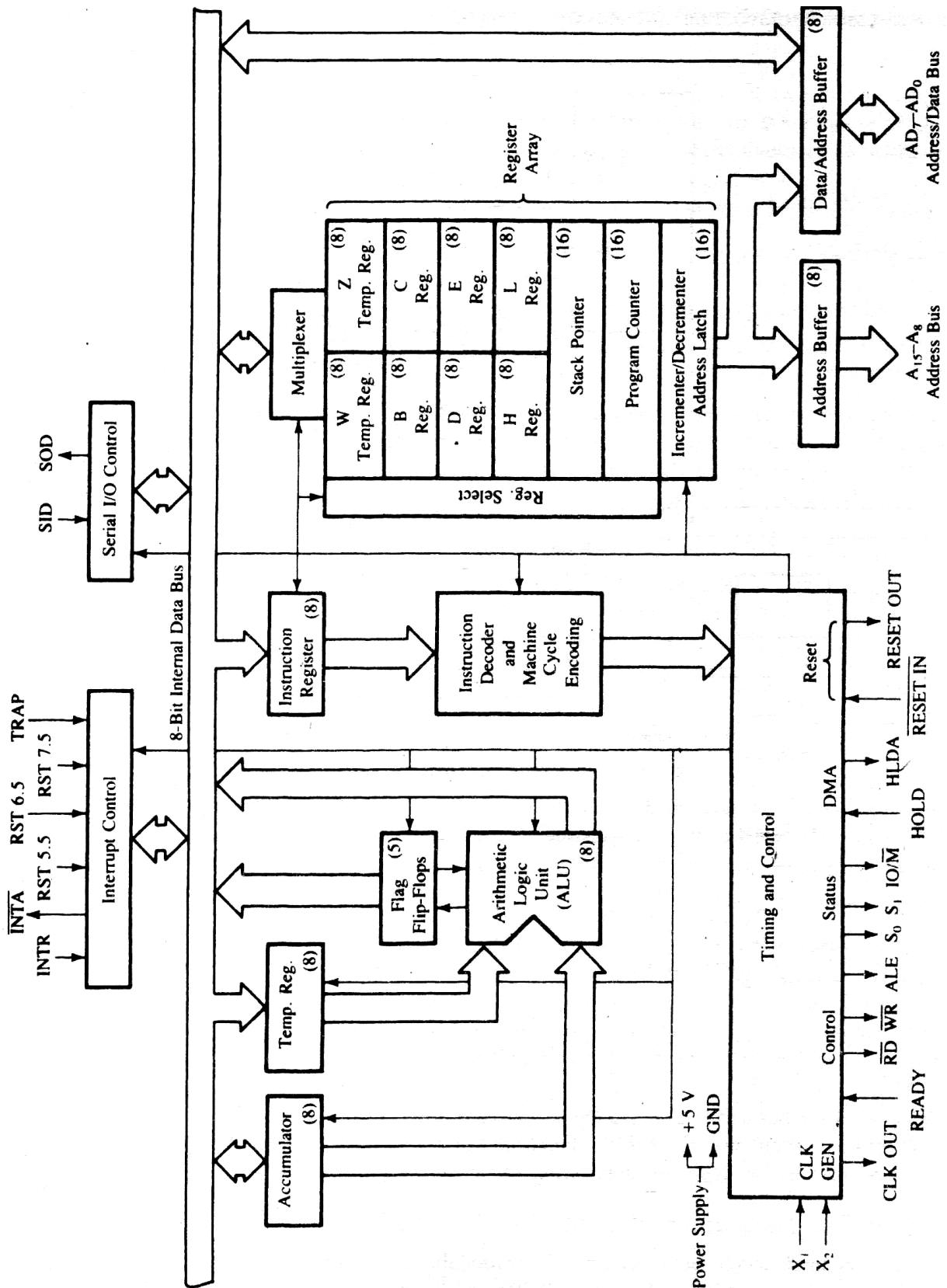
Fig: Functional Block Diagram of 8085A

Timing and Control Unit

This unit synchronizes all the microprocessor operations with the clock and generates the control signals necessary for communication between the microprocessor and peripherals.

Instruction Register and Decoder

The instruction register and the decoder are part of the ALU. When an instruction is fetched from memory, it is loaded in the instruction register. The decoder decodes the instruction and establishes the sequences of events to follow (what operation is to be performed).

Accumulator

The accumulator (Register A) is an 8 bit register that is part of the arithmetic/logic unit (ALU). This register is accessible to user and holds one of the operands and the result itself. 8085 is accumulator based microprocessor because the operation of 8085 depends on the accumulator. Almost all the arithmetic and logic functions are performed on the data are in the accumulator

Registers B, C, D, E

These four 8 bit registers are accessible to the programmer. These can be used individually as 8 bit individual registers or in pairs as BC and DE as 16 bit registers.

Register H & L

Register H & L can be used as 8 bit registers or as pairs. However, these can also be used for indirect addressing.

Program Counter (PC) and Stack Pointer (SP)

These are 16 bit registers used to hold addresses. The size of these registers is 16 bits because the memory addresses are 16 bits.

The computer program consists of the sequence of coded instructions. These are stored in memory. The microprocessor uses the PC register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. PC is incremented by one when the machine code is fetched.

A stack is an area of memory set aside for the purpose of storing data. The location of the memory is defined by the 16 bit register called SP. When a data is stored its value is decreased by 1 while it is increased by 1 when the data is retrieved.

Flags

It consists of a register with five flip flops, each holding a specific status of the output of the operation of ALU. The register is known as Flag Register and each flip flop is called flag. The states of the flags indicate the result of arithmetic and logic operation, which in turn is

used for decision making processes. There are no decision instructions for AC flag. The different flags are:

- S – Sign Flag: Is set if the MSB of the result of last operation is 1. (Also called negative, since 1 in the MSB position of two's complement number)
- Z – Zero Flag: Is set if the ALU operation results in 0, and flag is reset if the result is not 0. Operation of the flag is affected by accumulator as well as that of other registers.
- AC – Auxiliary Carry Flag: If the arithmetic operation generates a carry from lower nibble to upper nibble then this flag is set. It is useful while performing BCD arithmetic.
- P – Parity Flag: Is set if the result has an even numbers of 1s while reset if the number of 1 is odd.
- CY – Carry Flag: Is set if the arithmetic operation results in a carry and is reset if otherwise.

The bit positions reserved for these flags in the flag register are as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z		AC		P		CY

1.6 Applications

The applications of microprocessors are not bound. They can be used virtually anywhere and in any field. However, the applications are sorted as follows:

- Test Instruments
- Communications
- Computer
- Industries

Test Instruments

Microprocessors are widely used in devices such as signal generators, oscilloscopes, counters, digital multi-meters, x-ray analyzers, blood group analyzers, baby incubator, frequency synthesizers, data acquisition systems, spectrum analyzers etc. For example fluke 6010A synthesized signal generator uses 4004 microprocessor.

Communications

Communication today requires tens of thousands of circuits to be managed. Data should be received, checked for errors and further analysis should also be performed. The speed at which the microprocessor can take decisions and compute errors is truly substantial.

Computer

The microprocessor is a central processing unit (CPU) of the microcomputers. It can perform arithmetic and logic functions as well as control function. The control unit of microprocessor

sends signals to input, output units, memory, ALU and arrange the sequence of their controlling operation.

Industries

The microprocessor is widely used in data monitoring systems, smart cameras for quality control, automatic weighing, batching systems, assembly machine control, torque certification systems, machine tool controller etc.

2. Microprocessor Instructions (8 bit Microprocessor)

2.1 Resister transfer language (RTL)

During the execution of an instruction different actions are exercised. The preliminary action: op-code fetch is same for all the instructions however the latter action depends on the size of the instruction. A microprocessor can handle only a fixed number of instructions in case of 8085 it can handle a maximum of 2^8 i.e. 256 instructions. The different instructions and their equivalent op-codes are available in the 8085 instruction sheet.

Depending on the byte size the instruction set is classified into the following three groups.

- One byte Instructions: A 1byte instruction includes the op-code and the operand in the same byte. The operand will be default for that instruction. These instructions require one memory locations. E.g.: MOV C,A , ADD B, CMA etc.

Op-code	Operand
MOV	C, A
ADD	B
CMA	

- Two byte Instructions: In a 2byte instruction, the first byte specifies the operation code and the second byte specifies the operand. Thus, two memory locations are required. E.g.: MVI A,32H , ADI F2H etc.

Op-code	Operand
MVI A	32H
ADI	F2H

- Three byte Instruction: In three byte instruction first byte will be op-code where as second and third byte will be the operands. Operands may be data or address depending upon the instruction. E.g.: LXI B, 4455H , LDA 2050H, JMP 2085H etc.

Op-code	Operand
LXI B	4455H
LDA	4050H
JMP	2085H

Now, depending upon the instruction they are executed accordingly. These operations are expressed using a language. Such type of language, which is basically used to express the transfer of data among the registers, is called Register Transfer Language (RTL). If a data is transferred from register A to register B the in RTL

register A ← register B

During the execution of an instruction we have fetch cycle and execute cycle. The operations performed during fetch cycle are:

- The PC contains the address of the next instruction to be executed. As first operation of fetch cycle, the contents of program counter will be transferred to the memory address register (MAR). The memory address register then uses the address bus to transmit its contents that specifies the address of the memory location from where the instruction code is to be fetched. Let t1 be period of this operation.

  MAR ← PC

- Now as soon as the control unit issues the memory read signal, the contents of the addressed memory location specified by MAR will be transferred to the memory buffer register (MBR). Let t2 be the period of this operation.

    MBR ← Memory

- Now the contents of MBR will be transferred to the instruction register and the program counter (PC) gets incremented to fetch another instruction and the fetching cycle is thus complete. Here two operations take place within a single time unit t3. Let t3 be the time required by the CPU for this operation.

    IR ← (MBR)
    PC ← PC+1

The control unit of the computer maintains the timing sequence of the all of the above operations that constitute the fetch cycle. The operations are sequenced on the basis of the single time period called the period of the clock. The RTL for fetch cycle is:

      t1:   MAR ← PC
      t2:   MBR ← Memory
      t3:   IR ← (MBR)
                   PC ← PC +1

All three time units are of equal duration. A time unit is defined by a regularly spaced clock pulses. The operations performed within this single unit of time are called micro-operations. Since each micro-operation specifies the transfer of data into or out of a register, such type of language is called Register Transfer Language.

After the fetch of the instruction is complete, the execution cycle starts. Different instructions are executed in different fashions. Let us consider execution of a simple instruction MOV A,B. The first step needed is to obtain the data from the location B. For this the address field of instruction register indicating the address of memory location will be transferred to address bus through the Memory Address Register (MAR).

When the control unit issues a memory read signal, the contents of location B will be output to the MBR. Now the address of memory location A is loaded in MAR. Finally, after the memory write signal from the control unit is issued the data is copied to memory location A. Using RTL language and defining above operations in the form of micro operation the execute cycle is represented in following time units.

      t1:   MAR ← (IR (Address of B))
      t2:   MBR ← (B)
      t3:   MAR ← (IR (Address of A))
      t4:   A ← MBR

2.2 Instruction and Machine cycle

Consider a simple instruction: MOV A, B which states that data from register B is to copied to register A. The instructions are all stored in the memory. The computer spends certain period of time on Fetching, Decoding and Executing this instruction.

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

Machine cycle: It is defined as the time required to complete one operation of accessing memory, I/O, or acknowledging an external request.

T-state: It is defined as one subdivision of the operation performed in one clock period.
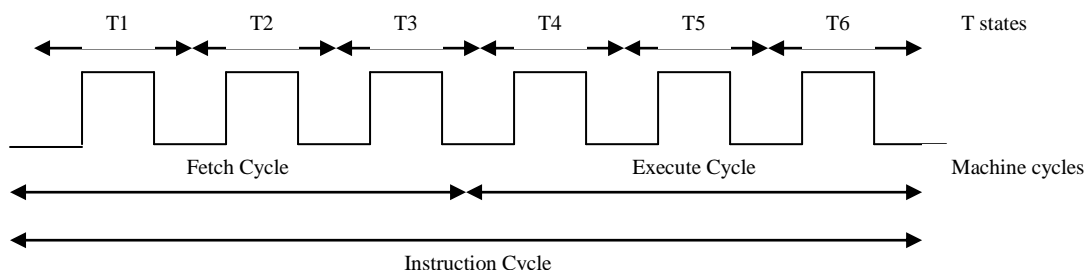

Fig 2.1 Instruction cycle

2.3 Addressing Modes

Instructions are commands to the microprocessor to perform a certain task. The instructions consist of Op-code and data called operand. The way operands are chosen during program execution is dependent on the addressing mode. Addressing mode specifies the rule for interpreting and modifying the address of the instruction before the operand is actually referenced. The 8085 instruction set has six different addressing modes.
- Direct Addressing
- Register Indirect Addressing
- Register Direct Addressing
- Implied Addressing
- Immediate Addressing

Direct Addressing:

This addressing mode is called direct because the effective address of the operand is specified directly in the instruction. Instructions using this mode may contain 2 or 3 bytes, with first byte as the Op-code followed by 1 or 2 bytes of address of data. Loading and storing in memory use 2 bytes of address while IN and OUT have one byte address. For example:
LDA 2035H
IN FCH

Register Indirect Addressing:

Here the address part of the instruction specifies the memory location whose content's is the address of the operand. So in this type of addressing mode, it is the address of the address rather than the address itself is part of the instruction. In 8085, wherever the instruction uses HL pointer the address is called indirect addressing. For example:
MOV A, M

Register Direct Addressing:

Register direct addressing mode means that a register is the source of an operand for an instruction. It is similar to direct addressing. For example:
MOV A, B

ADD B

Implied or Inherent Addressing:

The instructions of this mode do not have operands. For example:
> EI (Enable Interrupt),
> STC (set the carry flag),
> NOP (No operation)

Immediate Addressing:

This is the simplest for of addressing. When it executes the instruction will operate on immediate hexadecimal number. The operand is present in instruction in this mode. This mode is used to define and use constants of set initial values of variables. The operand may be 8 bit data or 16 bit data. For example:
> MVI B, 05H
> LXI B, 7A21H

2.4 RTL description of data transfer, arithmetic, logical, branch, miscellaneous instructions

An 'instruction' is a binary pattern designed to perform a specific function. The list of entire instructions is called the instruction set. The instruction set determines what function the microprocessor can perform. The 8085 instruction set can be classified into the following five categories:
- Data transfer group
- Arithmetic group
- Logical group
- Branching group
- Miscellaneous group

Data transfer group instructions:

The data transfer group is the longest group of instructions in 8085 and consists of 88 different op-codes. This group of instructions copy data from a source location to the destination location, without modifying the contents of the source. The transfer of data may be between the registers or between register and memory or between an I/O device and accumulator. None of these instructions modify the flags.

- Load register (immediate instruction): loads the designated register with the operand. Some of the instructions are:

MVI A, data	MVI B, data	MVI C, data	MVI D, data
MVI E, data	MVI H, data	MVI L, data	

Instruction	MVI A, data
RTL (execute cycle)	t1: MAR ← PC
	t2: MBR ← (MAR) , PC++
	t3: register A ← MBR

- Load memory (immediate): loads the memory location whose address is contained in registers H and L with the operand

Instruction	MVI M, data
RTL	t1: MAR ← PC
	t2: MBR ← (MAR), PC++
	t3: MAR ← registers H&L
	t4: (MAR) ← MBR

- Load register pairs (Immediate): loads the register pair with 2 bytes of data. The register pairs are B>B&C, D>D&E, H>H&L. The instructions are:

  LXI B, data   LXI D, data   LXI H, data

Instruction	LXI B, data
RTL	t1: MAR ← PC
	t2: MBR ← (MAR), PC++
	t3: register C ← MBR
	t4: MAR ← PC
	t5: MBR ← (MAR), PC++
	t6: register B ← MBR

- Load Accumulator (direct): loads the accumulator with the contents of memory location specified by the 2 byte address. The instruction is LDA 2050H.

Instructions	LDA, address
RTL	t1: MAR ← PC
	t2: MBR ← (MAR++ MAR)
	t3: MAR ← MBR
	t4: MBR ← (MAR)
	t5: register A ← MBR

- Load Accumulator (indirect): load the accumulator with the contents of memory location whose address is contained in the designated register pair (only register pair B & D are used). The instructions are LDAX B & LDAX D.

Instruction	LDAX B
RTL	t1: MBR ← register B & register C
	t2: MAR ← MBR
	t3: MBR ← (MAR)
	t4: register A ← MBR

- Move register: this instruction loads the first specified register with the contents of the second specified register. The instructions are MOV A, B , MOV C, D.

Instruction	MOV A, B
RTL	t1:	MAR ← (Address of B)
	t2:	MBR ← (B)
	t3:	MAR ← (Address of A)
	t4:	A ← MBR

- Move to memory from designated registers: load the memory location whose address is contained in register H-L with the contents of the register. The instructions are MOV M,A , MOV M,D.

Instruction	MOV M, reg
RTL	t1: MBR ← register H & register L

t2: MAR ← MBR
t3: MBR ← reg
t4: (MAR) ← MBR

- Move to designated registers from memory: loads the designated register with the contents of the memory location whose address is specified by the contents of register H&L. The instructions are MOV A,M , MOV D,M.

- Store Accumulator contents (indirect address): loads the memory location whose address is contained in the designated register pair (only register pair B & D are used) with the contents of the accumulator. The instructions are STAX B, STAX D.

- Store Accumulator Contents (direct): loads the memory location whose address is specified by the operand. The instruction is STA operand.

- Exchange contents of H&L with D&E: This instruction is used to exchange the contents of register H with the contents of register D and the contents of register E with the contents of register L. The instruction is XCHG.

- Load H & L directly: This instruction loads the registers H&L with the contents of the memory location whose address is the operand of the instruction. (Register L contains the value at the address and Register H contains the value at the address+1)

- Store H&L directly: This instruction is similar to load instruction and stores the contents of H & L at the designated memory address.

Arithmetic group instructions:

There are different instructions for arithmetic operations in 8085. The arithmetic operations performed by 8085 are addition, subtraction, increment and decrement. All the addition and subtraction operations are performed with accumulator. However, increment and decrement can be performed in any register and affect the flags except carry flag.

For addition and subtraction the following features exist:
- The instructions assume implicitly that the accumulator is one of the operands.
- The flags get modified according to the data conditions of the result.
- The result is placed in the accumulator.
- The execution of the instruction does not affect the operand register.

The different instructions are:

ADD register: Add the contents of a register with the accumulator
ADI data: Add immediate 8 bit data with the accumulator
ADC register: Add the contents of a register with the accumulator and the carry
ADI data: Add immediate 8 bit data with the accumulator and the carry
SUB register: Subtract the contents of register from accumulator
SBB register: Subtract the contents of register and borrow from accumulator
SUI data: Subtract the immediate data from the accumulator
SBI data: Subtract the immediate data and borrow from accumulator
INR register: Increment the contents of register by 1

DCR register: Decreases the contents of register by 1

Logical group instructions:

A microprocessor is programmable logic device. All the operations of the microprocessor is carried out via the hardwired logic circuits. The logical functions that 8085 microprocessor can perform are AND, OR, Ex-OR and NOT. The different instructions are:

ANA register: Logical AND the contents of register with accumulator
ANI data: Logical AND the immediate data with accumulator
ORA register: Logical OR the contents of register with accumulator
ORI data: Logical OR the immediate data with accumulator
XRA register: Logical exclusive OR the contents of register with accumulator
XRI data: Logical exclusive OR the data with accumulator
CMA: Complements the contents of accumulator

The following features hold true for all logic instructions:
- The accumulator is one of the operands
- Except the instruction CMA, all instruction reset carry flag
- They modify the Z, P, A, S flags according to the data conditions of result
- They place the result in accumulator
- The contents of operand registers are not affected.

Branching group instructions:

The branch instructions allow the microprocessor to change the sequence of a program, either on certain conditions or unconditionally. The microprocessor is a sequential machine i.e. the instructions are executed from memory location sequentially. Branch instructions cause the microprocessor to execute from another memory locations. The branch instructions can be categorized as:
- Jump Instruction
- Call and Return Instruction
- Restart Instruction

Jump Instruction:

The jump instructions specify the memory location explicitly. These are 3 byte instructions where the first byte is op-code and other bytes represent the 16 bit address. The jump may be conditional or unconditional.

An unconditional jump enables the programmer to set up continuous loops without depending on any type of conditions. The instruction is JMP data. The jump location can also be specified using a label if the exact memory location which a program sequence should be directed, is not known. However, it should be considered that a single label cannot be used for different memory locations.

A conditional jump instruction allows the microprocessor to make decision based on certain conditions indicated by the flags. After a arithmetic or logical operation, flags are set or reset to reflect the data conditions. These instructions check flag conditions and make decision to change the sequence of program. The flags tested for conditional jump are zero, carry, sign and parity. The different instructions are:

JC: Jump on carry
JNC: Jump on if no carry
JZ: Jump on zero
JNZ: Jump on no zero
JP: Jump on plus
JM: Jump on minus
JPE: Jump on even parity
JPO: Jump on odd parity

Call and return Instructions:

Call and return instructions are used for using subroutine. A subroutine is a group of instruction that performs a subtask of repeated occurrence. The subroutine is written as a separate unit, apart from the main program and the control from program is transferred to subroutine when the call instruction is encountered. After the completion of the subroutine the control is returned back to the main program.

The call instruction and return instructions can also be set conditionally or unconditionally. The different call and return instructions are:

CALL: Unconditional call instruction
RET: Unconditional return instruction
CC, CNC, CZ, CNZ, CM, CP, CPE, CPO: Conditional Call
RC, RNC, RZ, RNZ, RM, RP, RPE, RPO: Conditional Return

Restart Instruction:

The 8085 instruction set includes eight RST (Restart instructions). These are 1 byte call instructions and transfer the program execution to a specific location as listed in table below:

Restart Instructions	Hex. Code	Call Location in Hex
RST0	C7	0000H
RST1	CF	0008H
RST2	D7	0010H
RST3	DF	0018H
RST4	E7	0020H
RST5	EF	0028H
RST6	F7	0030H
RST7	FF	0038H

Actually these restart instructions are inserted through additional hardware and for this purpose the INTA signal of 8085A is used. These instructions are part of interrupt process.

Miscellaneous group instruction:

All other instructions in the instruction set fall on this group. These instructions include stack operation instructions, input/ output operations and interrupt operations of 8085 microprocessor. The different instructions are.
PUSH, POP, DI, EI, HLT, etc….

2.5 Fetch and execution cycle, fetch execution overlap

The processor does the actual work by executing instructions specified in the program. In its simplest form instruction processing consists of two steps: The processor reads instructions from memory one at a time and executes each instruction. Program execution consists of repetitive fetch and instruction execution.

The processing time required for a single instruction is called an instruction cycle. The instruction cycle is composed of: fetch cycle and execute cycle. During fetch the instruction is read from its memory location into the processor. The time required for this memory read operation is called fetch cycle. After the instruction is fetched it ought to be executed. The time required for execution is the execution cycle.

The fetch and execute stages of instruction execution are quite independent. The fetch cycle involves accessing the main memory for instruction. But during execution the memory need not be referenced. So, during this duration the next instruction can be fetched. This is called instruction prefetch. This prefetching data before it is asked for in the anticipation that it will be used is called pipelining. The process of pipelining enhances the speed of processor greatly. However, 8085 doesn't support pipelining. Thus there is no overlap between fetch cycle and execution cycle.

2.6 Timing diagram for register move, indirect read, indirect write and out instructions

The 8085 microprocessor is designed to execute 74 different instruction types. Each instruction has two parts: operation code, known as op-code, and operand. To execute an instruction, the 8085 needs to perform various operations such as Memory Read/Write and I/O Read/ Write. The total operations of a microprocessor can be classified into the following operations.
- Op-code Fetch
- Memory Read and Write
- I/O Read and Write
- Request Acknowledge

Op-code Fetch Cycle:

The first operation in any instruction is Op-code Fetch. The microprocessor needs to get the machine code from the memory register where it is stored before the microprocessor can begin to execute any instruction.

Let us consider the timing for execution of the instruction MVI A,32H. The op-code of MVI is 3EH. Since the op-code fetch cycle is analogous for all the instructions all op-code fetch cycle consists of four machine cycles.

The following figure shows the timing of how a data is transferred from memory to MPU; it has five different groups of signals in relation to system clock. The address bus and data bus are shown as two parallel lines. Other control signals are shown using single lines representing logic levels. The crossover of the lines indicates a new byte is placed on the bus. The dashed straight line indicates the high impedance state.
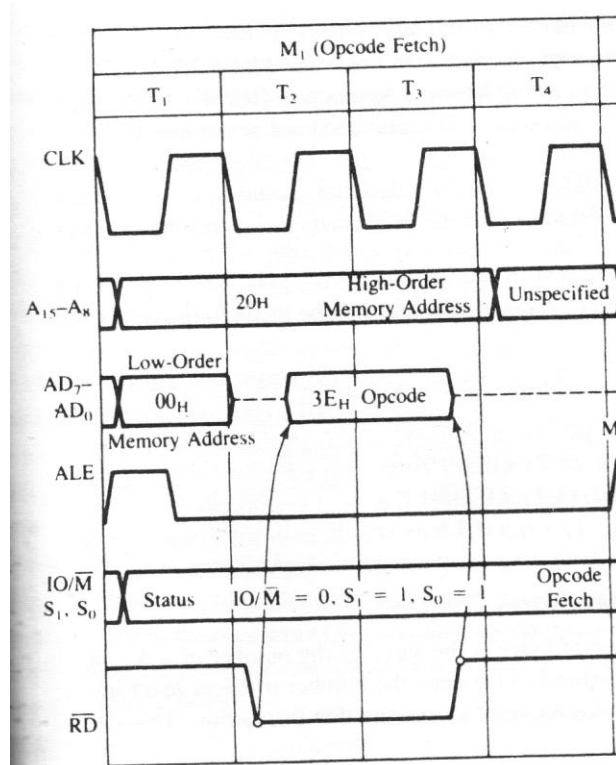
Fig: Timing Diagram of Op-code Fetch Cycle

The following steps occur during Op-code Fetch Cycle

1. The Program counter (PC) places the 16 bit memory address on the address bus. At T1 high order address is placed at A_8-A_{15} and lower order address is placed at AD_0-AD_7 and the ALE signal goes high, IO/M goes low, and both s0 and s1 goes high; which identifies the op-code fetch cycle.
2. The control unit sends the control signal RD to enable the memory chip and remains active till two clock periods.
3. Now the op-code from memory location is placed <u>on</u> the multiplexed data bus. In this case 3E is placed on $AD_7 - AD_0$. The transition of RD indicates the end of this step.
4. The op-code byte is now placed on instruction decoder and the execute cycle is carried out

Memory Read cycle:

The Op-code fetch cycle is a memory read cycle. Thus, other memory read cycles are similar to the op-code fetch cycle. Let us take the same example as above. The instruction cycle of MVI 32H is shown below:

The total cycle consist of 7 T states and 2 machine cycles: op-code fetch and memory read. At the end of op-code fetch the PC is incremented thus the address is now 2001h and instruction decoder has 3Eh. Now the operand is to be read from the memory to register A. The second machine cycle is the memory read and consists of 3 T states. The signal content of this cycle are similar to the first three T states of op-code fetch except the status signal, In this case s0=0 and s1=1.

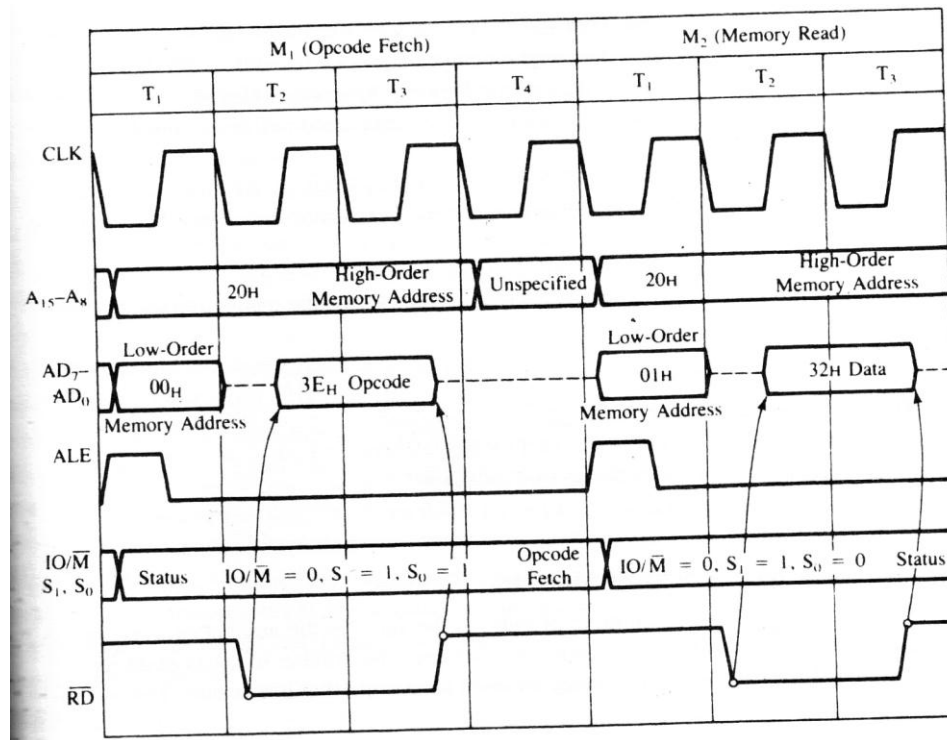The byte read in T3 of memory read cycle will be copied into the accumulator in the same cycle.

Fig: Op-code fetch and memory read cycle

I/O Write Cycle:

Let us consider the instruction OUT 01h stored at memory location 2050h. The op-code of this instruction is D3h and the complete cycle requires 10 T states. The write cycle executes as shown below:
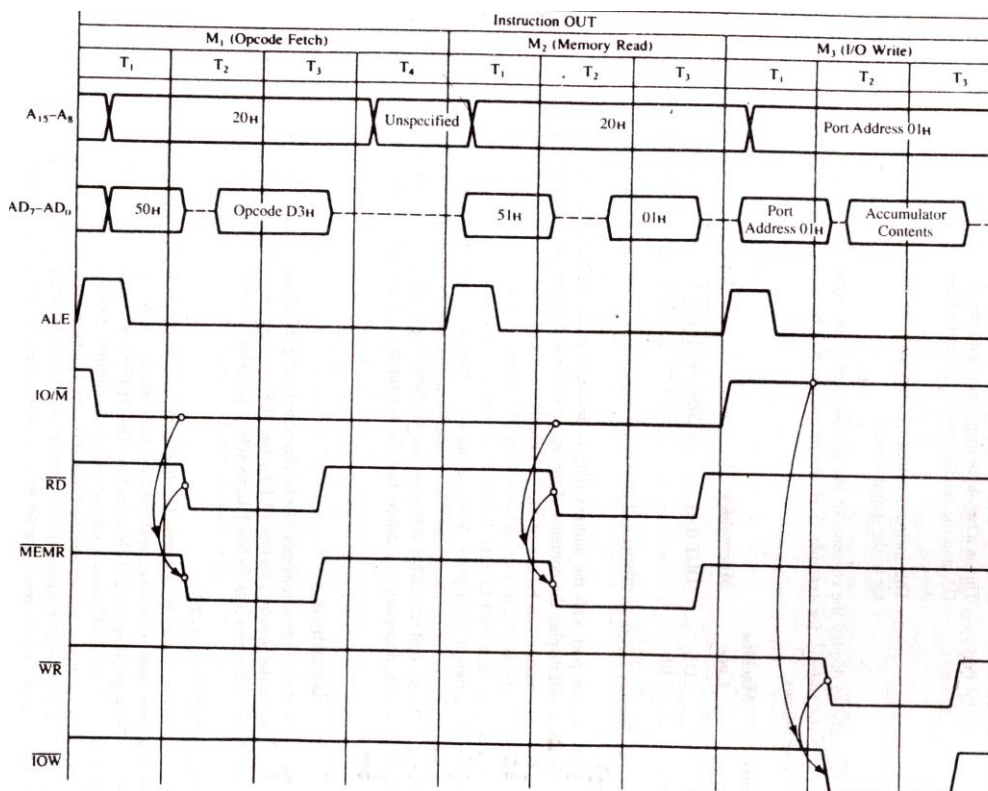

Fig: Timing diagram of I/O write cycle

The instruction is executed under three machine cycles: op-code fetch, memory read and I/O write. The op-code fetch and memory read cycles exactly match the previous ones except for the contents of the buses. At the end of memory read cycle the PC points at 2051h.

Now instruction decoder contains D3h and on the next machine cycle the port address 01h is placed on higher and lower port address. Unlike previous cases now IO/M signal goes high to indicate a I/O operation. At T2 the accumulator contents are placed on the data bus, followed by the control signal WR. The signal IOW is obtained by anding the IO/M and WR signals to enable the output device

I/O Read Cycle:

The I/O read cycle is exactly same as I/O write cycle. The only difference is on the third machine cycle where the signal IOR is generated and the content of the port is read to the accumulator instead of otherwise. The IOR signal is generated by anding the IO/M and RD signals to enable the input device.
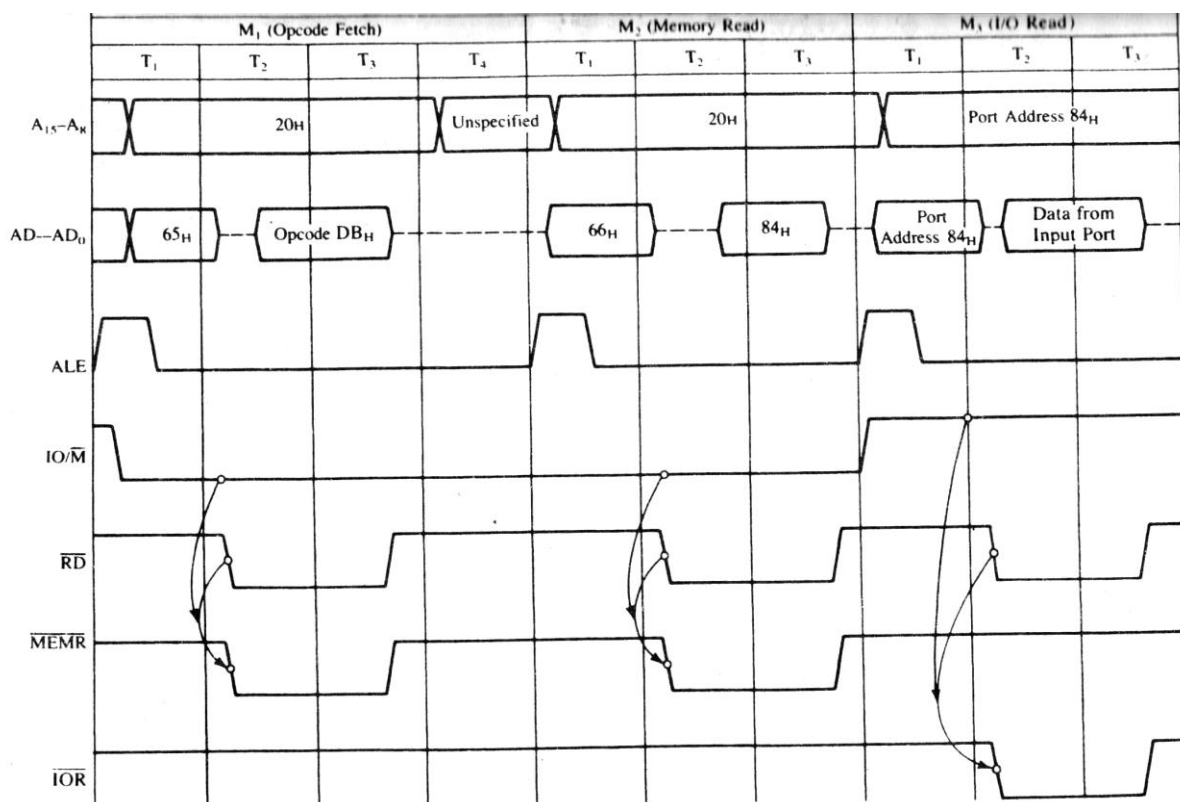

Fig: Timing diagram of I/O read cycle

3. Assembly Language Programming (16 bit Microprocessor)

Introduction to 16 bit Microprocessor Architecture

The 8086 is a 16-bit microprocessor. The term 16 bit implies that its arithmetic logic unit, its internal registers, and most of its instructions are intended to work with 16 bit binary data. The 8086 has a 16 bit data bus, so it can read data from or write data to memory and ports either 16 bits or 8 bits at a time. The 8086 has a 20 bit address bus, so it can address any one of 2^{20}, or 1,048,576 memory locations.

8086 CPU is divided into 2 independent functional parts to speed up the processing namely **BIU** (Bus interface unit) & **EU** (execution unit).

8080 Microprocessor Internal Architecture
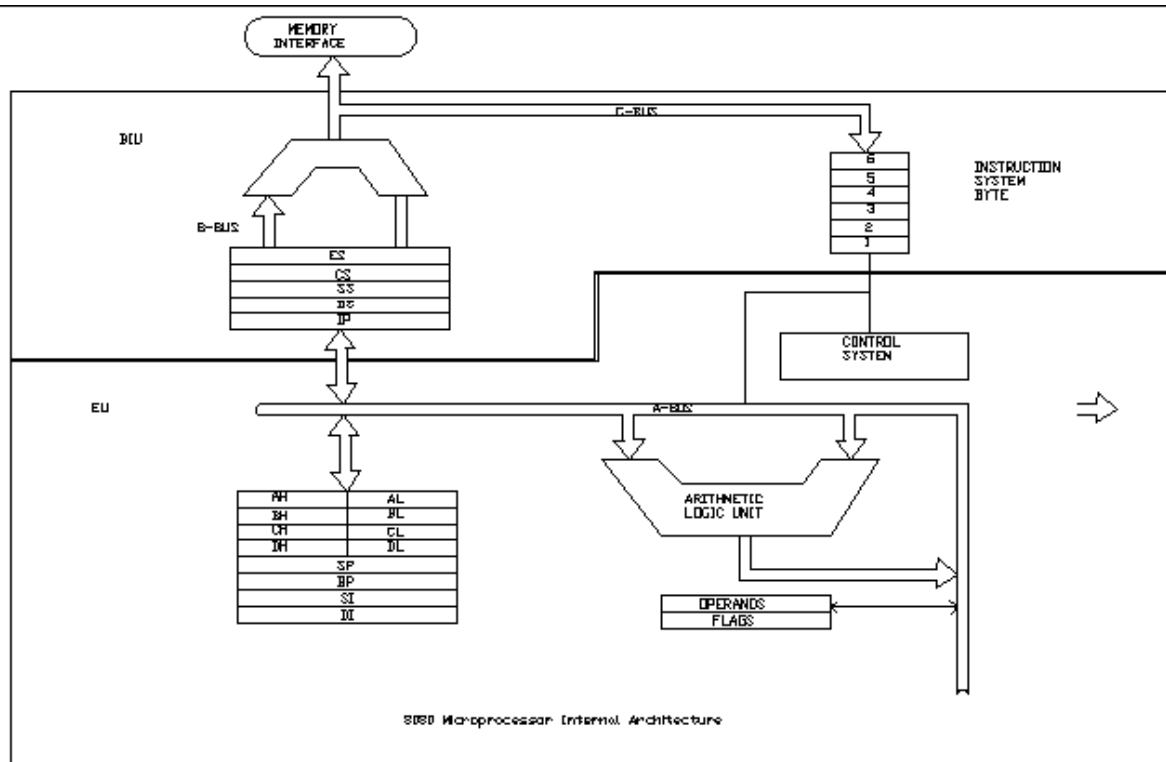
BIU: It handles all transfers of data and addresses on the buses for the execution unit.
- Sends out addresses
- Fetches instructions from memory
- Read / write data from/to ports and memory i.e handles all transfers of data and addresses on the busses

EU
- Tells BIU where to fetch instructions or data from
- Decodes instructions
- Executes instructions

Execution Unit

Instruction Decoder & ALU:

Decoder in the EU translates instructions fetched from the memory into a series of actions which the EU carries out.16-bit ALU in the EU performs actions such as AND, OR, XOR, increment, decrement etc.

FLAG Register:

It is a 16-bit register. 9-bit are used as different flags, remaining bits unused

U	U	U	U	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

Out of 9-flags, 6 are conditional flags and three are control flags

Conditional flags:

These are set or reset by the EU on the basis of the results of some arithmetic or logic operation. 8086 instructions check these flags to determine which of two alternative actions should be done in executing the instructions.
- OF (Overflow flag): is set if there is an arithmetic overflow, i.e. the size of the result exceeds the capacity of the destination location.
- SF (Sign flag): is set if the MSB of the result is 1
- ZF (Zero flag): if the result is zero
- AF (Aux carry flag): is set is there is carry from lower nibble to upper nibble or from lower byte to upper byte
- PF (Parity flag): is set if the result has even parity
- CF (Carry flag): is set if there is carry from addition or borrow from subtraction

Control flags:

They are set using certain instructions. They are used to control certain operations of the processor.
- TF (Trap flag): for single stepping through the program
- IF (Interrupt flag): to allow or prohibit the interruption of a program
- DF (Direction flag): Used with string instructions

General purpose Registers:

- 8 GPRs AH, AL (Accumulator), BH, BL, CH, CL, DH,DL are used to store 8 bit data.
- AL register is also called the accumulator
- Used individually for the temporary storage of data
- GPRs can be used together (as register pair) to store 16-bit data words. Acceptable register pairs are:

AH-AL pair AX register	BH-BL pair BX register
CH-CL pair CX register	DH-DL pair DX register

Pointer and Index registers: SP (Stack Pointer), BP (Base pointer), SI (Source Index), DI (Destination index)

The two pointer registers, SP and BP are used to access data in the stack segment. The SP is used as offset from current Stack Segment during execution of instruction that involve stack. SP is automatically updated. BP contains offset address and is utilized in based addressing mode.

Index Registers:

EU also contains a 16 bit source index (SI) register and 16 bit destination index (DI) register. These three registers can be used for temporary storage of data similarly as the general purpose registers. However they are specially to hold the 16-bit offset of the data word.

Bus Interface Unit

The QUEUE:

When EU is decoding or executing an instruction, bus will be free at that time. BIU prefetches up to 6-instructions bytes to be executed and places them in QUEUE. This improves the overall speed because in each time of execution of new instruction, instead of sending address of next instruction to be executed to the system memory and waiting fro the memory to send back the instruction byte, EU just picks up the fetched instruction byte from the QUEUE.

The BIU stores these pre-fetched bytes in a first-in-first-out (FIFO) register set called a queue. Fetching the next instruction while the current instruction executes is called pipelining.

Segment Registers:

The BIU contains a dedicated address, which is used to produce the 20 bit address. The bus control logic of the BIU generates all the bus control signals, such as the READ and WRITE signals, for memory and I/O. The BIU also has four 16 bit segments registers namely:

- Code segment: holds the upper 16-bits of the starting addresses of the segment from which BIU is currently fetching instruction code bytes.
- Stack segment: store addresses and data while subprogram executes
- Extra segment: store upper 16-bit s of starting addresses of two memory segments that are used for data.
- Data segment: store upper 16-bit s of starting addresses of two memory segments that are used for data.

Code Segment Register (CS) and Instruction Pointer (IP)

All program instructions located in memory are pointed using 16 bits of segment register Cs and 16 bits offset contained in the 16 bit instruction pointer (IP). The BIU computes the 20 bit physical address internally using the logical address that is the contents of CS and IP. 16 bit contents of CS will be shifted 4 bits to the left and then adding the 16 bit contents of IP. Thus all instructions of the program are relative contents of IP. Simply stated, CS contains the base or start of the current code segment, and IP contains the distance or offset from this address to the next instruction byte to be fetched.
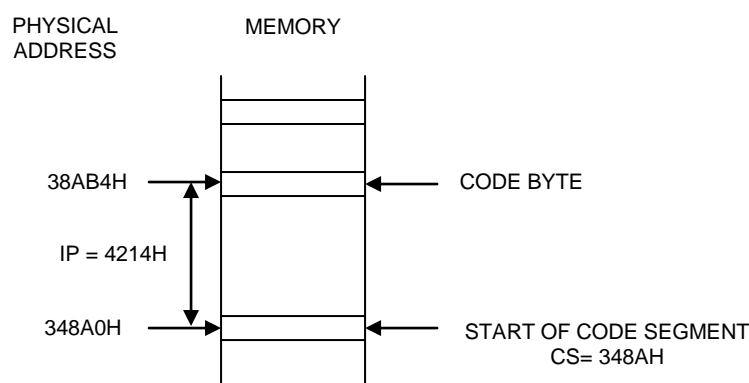
Graphically,

Fig: Diagram showing addition of IP to CS to produce the physical address of code byte

Stack Segment Register (SS) and Stack Pointer (SP)

A stack is a section of memory to store addresses and data while a subprogram is in progress. The stack segment register points to the current stack. The 20 bit physical stack address is calculated from the SS and SP. The programmer can also use Base Pointer (BP) instead of SP for addressing. In this case the 20 bit physical address is calculated using SS and BP.

Addressing Modes

- Immediate Addressing mode: Operand (one of the operand say source) is immediate data in the instruction
  Example
  	MOV CX,234AH
  	ADI 23H

- Register Addressing mode: Transfers a copy of byte or word from source register or memory location to destination register or memory location.
  Example
  	MOV CX, DX
  	transfers the content of DX to CX register pair

- Direct Addressing mode: One operand should be memory location i.e moves byte or word between memory location and the register.
  Example
  	MOV BX, [4371H]
  	Copies 16-bit word into a memory pointed by the displacement address 4371H to the BX register. (In little endian format: lowest byre->lowest address)

- Register Indirect Addressing: Transfers a word or byte between register and memory addressed by index or base register. Index or base register are BP, BX, DI, SI.
  Example
  	MOV AX,[BX]
  	Copies the word-sized data from data segment offset address indexed by BX into AX

- Register Relative Addressing Mode: Transfers a byte or word between register and memory location addressed by an index or base register plus displacement.
  Example

  MOV AX, [BX+4]

- Base-Plus-Index Addressing Mode: Transfers a word or byte between register and memory location addressed by base register (BP or BX) plus index register (DI or SI).
  Example

  MOV [BX+DI], CL

  Used for locating array data type.

3.1 Assembler instructions format: Op-codes, memonics and operands

Machine Language:

There is only one programming language that any computer can actually understand and execute: its own native binary machine code. This is the lowest possible level of language in which it is possible to write a computer program. However, binary code is not native to humans and it is very easy for error to occur in the program. These bugs are not easy to determine in a pool of binary digits. Also for CPU like 8086 it is tedious and virtually impossible to memorize thousands of binary instructions codes.

Example

Program memory address	Content (binary)	Content (Hex)
00100H	0000 0100	04h

Assembly Language:

Programs in assembly language are represented by certain words representing the operation of instruction. Thus programming gets easier. These words (usually two-to-four letter) is used to represent each instruction are called Mnemonics. Assembly language statements are generally written in a standard form that has four fields.

- Label field
- Instruction, Mnemonic or Op-code field
- Operation field
- Comment field

Let us consider a simple example to add two numbers

Label	Mnemonic	Operand	Comment
Start:	MVI	A, 10h	Move 10h into accumulator
	MVI	B, 20h	Move 20h into register B
	ADD	B	Add the contents of register B with accumulator

Thus, we see the ease with the assembly language rather than machine language.

However, it must be remembered that for execution these codes are converted to machine codes. This is done by assembler. An assembler translates a program written in assembly language into machine language program (object code). Assembly language program are called 'source codes'. Machine language programs are known as object codes. A translator

converts source codes to object codes and then into executable formats. The process of converting source code into object code is called compilation and assembler does it. The process of converting object codes into executable formats is called linking and linker does it.

There are two ways of converting an assembly language program into machine language: manual assembly and using assembler. In manual assembly, the programmer is the assembler and he converts the entire mnemonic into its numerical machine language representation by looking up a table of microprocessor's instruction set. This method is only convenient for short programs.

An assembler reads each instruction of a program as ASCII characters and translates them to respective binary op-codes. An advantage of an assembler is address computation. Most programs used addresses within the program as data or as labels for jumps and calls. Assembler automatically takes care of addresses. Manual calculation of addresses for these instructions is a time consuming task.

3.2 Types of Assemblers

One Pass Assembler:

This assembler reads the assembly language program once and translates the assembly language program to machine code. This assembler has the program of defining forward references. This means that a branching instruction using an address that appears later in the program must be defined by the programmer after the program is assembled.

Two Pass Assembler:

This assembler scans the assembly language program twice. In the first pass, the assembler generates the table of the symbols. A symbol table consists of labels with addresses assigned to them. In this way labels can be used for JUMP statements, and no address calculation has to be done by the user. On the second pass, the assembler translates the assembly language program into the machine code. The two pass assembler is thus easier to use.

3.3 Macro assemblers, linking assemblers and assembler directives

Macro assembler is an assembly language that allows macros to be defined and used. The **Microsoft Macro Assembler** (MASM) is an x86 assembler that uses the Intel syntax for MS-DOS and Microsoft Windows.

Assembler can be linked with two or more than two assembler called linking assemblers.

Assembly directives, also called pseudo opcodes, pseudo-operations or pseudo-ops, are instructions that are executed by an assembler at assembly time, not by a CPU at run time. They can make the assembly of the program dependent on parameters input by a programmer, so that one program can be assembled different ways, perhaps for different applications. They also can be used to manipulate presentation of a program to make it easier to read and maintain.

4. Bus Structure and Memory Devices

4.1 Bus Structure, synchronous and asynchronous data bus, address bus, bus timing

Bus Structure:

A microprocessor unit performs basically four operations: memory read, memory write, I/O read, I/O write. These operations are part of communication between MPU & peripheral devices.

A communication includes identifying peripheral or memory location, transfer of data & control functions. These are carried out using address bus, data bus and control bus respectively. All the buses together is called the system bus.

In case of 8085 MPU we have

   8 unidirectional address pins
   8 bi directional multiplexed address/ data pins
   11 control output pins
   11 control input pins

Data bus:

The data bus provides a path for data flow between the system modules. It consists of a number of separate lines, generally 8, 16, 32 or 64. The number of lines is referred to as width of the data bus. A single line can only carry one bit at a time, the number of lines determine how many bits can be transmitted at a time. The width also determines the overall system performance.

An 8 bit data bus would require twice the time required by 16 bit data bus to transmit 16 bit data

   8085 – 8 bit data bus
   8086 – 16 bit data bus

Asynchronous bus:

In an asynchronous data bus the timing is maintained in such a way that occurrence of one event on the bus follows and depends on the occurrence of previous event. Let us consider a simple memory read:
- The CPU places the memory read and address signals on the bus.
- After allowing for these two signals to stabilize, it issues Master synchronous signal (MSYNC) to indicate the presence of valid address and control signals on the bus.
- The addressed memory module responds with the data and the slave synchronous (SSYNC) signal.

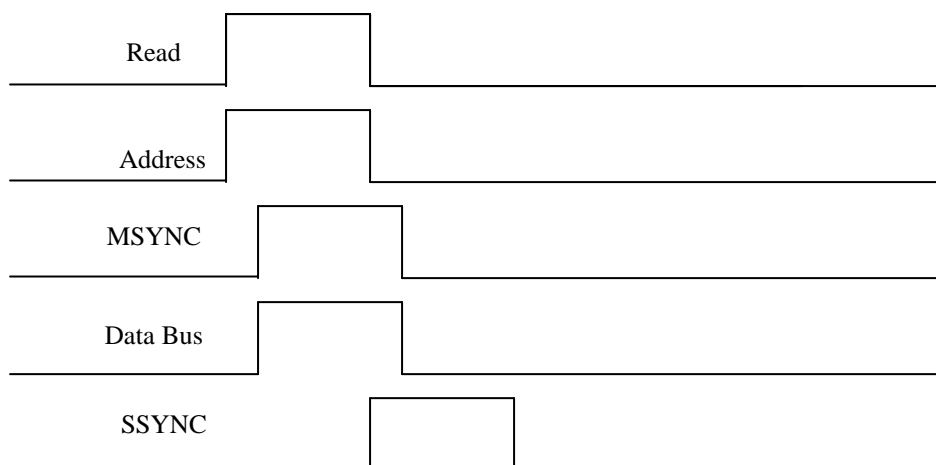Thus at a certain time it cannot be identified when there is data or an address on the bus.

Fig: Asynchronous Bus

Synchronous Bus:

A synchronous bus has its events tied by a clock. The clock transmits a regular sequence of a 0's and 1's of equal duration. A single 1-0 transmission is called clock cycle or bus cycle. All other devices work with the clock cycle. The events start at the beginning of the clock cycle. A memory read in case of a synchronous bus progresses as:

- The CPU issues a start signal to indicate the presence of address and control information on the bus.
- Then it issues the memory read signal and places the memory address on the address bus.
- The addressed memory module recognizes the address and after a delay or one clock cycle it places the data and acknowledgement signals on the buses.

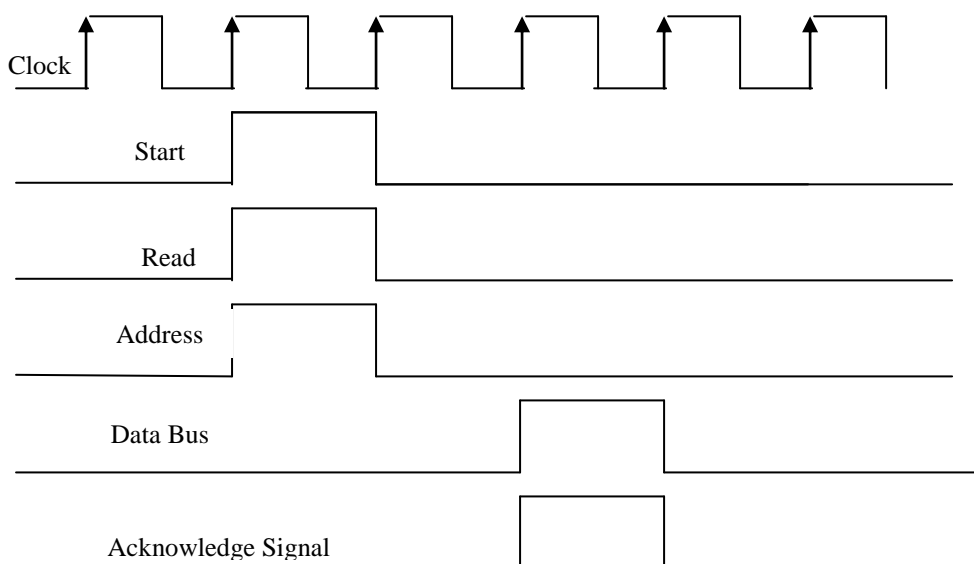In a synchronous bus, all the devices are tied to a fixed rate.

Fig: Synchronous Bus

Address bus:

The address bus is used to designate the source and destination of data in data bus. In a computer system, each peripheral or memory location is identified by a binary number called an address.

The width of the address bus determines the maximum possible memory capacity of the system. The address bus is also used to address I/O ports. Usually higher order bits are used to select particular modules and lower order bit select a memory location or I/O port within a module.

8085 – 16 bit address bus
Thus, maximum amount of memory locations it can address
2^{16} = 65, 536 or 64 Kb

Control bus:

These are group of lines used to control the data and address bus. Since this bus is shared by all the component of the microcomputer system; there must be come control mechanism to distinguish between data and address. The timing signals indicate the validity of data and address information; while command signal specify operations to be performed. Some of the control signals are:

- Memory write
- Memory Read
- I/O write
- I/O read
- ALE
- Interrupt Request
- Interrupt Acknowledge

4.2 Static and Dynamic RAM, ROM

Memory is like the pages of a notebook with space for a fixed number of binary numbers on each line. Memory is an essential component of microcomputer system; storing binary instructions and data for the microprocessor. Memory locations are identified by an address. The address bus of the microprocessor defines the maximum amount of memory a microprocessor can address.

Microcomputer memory can be grouped as

- Processor Memory
- Primary or Main memory
- Secondary Memory

Processor Memory:

Processor memory is a group of register along with processor. They temporarily hold data during computation since processor and registers are fabricated with same technology they have same speed. Although the use of such registers increase speed of the processor; the cost factor allows only few registers to be included.

Now, a separate memory is kept alongside the processor to keep the most frequently needed information. The performance could be improved if this new memory could be improved if this new memory could be included within the processor. This newly added memory is known as 'cache memory'.

Primary Memory:

This is the memory the microprocessor uses in executing & storing programs. Usually the size of this memory is larger and slower than processor memory.

Primary Memory can be grouped as:
- Random Access Memory (RAM)
- Read Only Memory (ROM)

RAM

It is used primarily for information that is likely to be altered, such as writing programs or receiving data. This memory is volatile. Two types of RAM are available.

Static RAM (SRAM)

This memory is made up of flip-flops, and it stores the bit as a voltage. Each memory cell requires six transistors; they have low packing density but high speed and consume more power.

Dynamic RAM (DRAM)

DRAM stores data in capacitors; so it can hold data for a few milliseconds. Hence DRAM must be refreshed from time to time. In DRAM greater number of bits can be stored in small chips but the interfacing circuit of DRAM is complicated because of refreshing. They are cheaper than SRAM.

ROM (Read Only Memory)

The ROM is a non-volatile memory. This memory is used for programs and data that need not be altered. As the name suggests, the information is read only, which means once a bit pattern is stored it is permanent or at least semi permanent. The different types of ROM are:

4.3 PROM, UVEPROM, EEPROM, PROM programmer and erasure

Masked ROM:

They are permanent ROM recorded by masking. Generally manufacturers use this process to produce ROM in large numbers.

PROM:
These are un-programmed ROM. The fuses on the ROM are not burned. A programmer can program this ROM according to his needs. The process is known as 'burning the PROM', and the information stored is permanent.

EPROM:

These ROM can be reprogrammed and erased. Two types of such EPROM are available.

UV-EPROM

The memory of such ROM can be erased by exposing the chip via a lid or window on the chip to ultraviolet light. The erase time generally varies between 10 to 30 minutes. The EPROM can be programmed by inserting the chip into a socket of the PROM programmer and providing proper addresses. The programming time varies from 1 to 2 minutes.

EEPROM

These are similar to UVEPROMs, except that the memory is altered by electrical signals at register level.

Secondary Memory

Secondary memory are storage devices. These devices have high data holding capacity. They store programs that are not frequently used by the processor. They are slow and have larger size. Some of the examples are: Hard disk, Floppy disk, Magnetic Tapes, CD, DVD etc.

4.4 Address decoding, memory interface (8, 16, 32, 64 bit)

A microcomputer has microprocessor, memory & I/O devices attached. A processor communicates with all parts interconnected in the system through common address and data bus. Hence only one device can transmit data through the bus at a time & others can only receive data. Now, to ensure that the proper device gets addressed at proper time, the method of address decoding is used

In this process memory blocks & I/O units are assigned a specific address. The address is determined by the way the device is connected to processor. The signal used to drive the device is called chip select. Since only one chip select is activated at a time only one block of memory or I/O device is activated. The purpose of address decoding circuit is to ensure that the Chip select signals to other devices are not activated. There are two methods for mapping address of these devices. They are:
- I/O mapped I/O
- Memory mapped I/O

In I/O mapped I/O method the I/O devices are addressed with 8-bit address. It means that the Chip Select signals for the devices are derived by using the 8 address lines. In 8085 microprocessor, the general procedure involves the use of lower 8 bit address lines for deriving the Chip Select signal. Thus only 256 devices or addresses can be mapped. The higher order address bits are don't care conditions. Instruction used for input and output are IN and OUT.

In memory mapped memory I/O mode the I/O devices are addressed with 16 bit data. It means that in this mode the Chip Select signals for the devices are derived using all 16 address lines from the processor. Thus a total of 64 KB can be addressed.

Usually in microcomputers based on 8085 processor the memories like RAM, ROM, EPROM etc uses memory mapping whereas the devices like 8255A, 8251A, input/output latches etc use I/O mapping. Now depending on the addresses that are allocated to the device the address decoding can be categorized as

1. Unique address decoding:

If all of the address lines available on that mapping mode are used for address decoding, then that decoding is called unique address decoding. Thus, in memory mapped memory I/O all 16 lines are used and in I/O mapped I/O method all 8 lines are used for address decoding.

2. Non-Unique address decoding:

If all of the address lines available on that mode are not used address decoding, then that decoding is called non-unique address decoding. Such practice of address decoding is simple but rather conflicting as the device is activated for a wide range of addresses.

Memory Interface

Using the proper address and data bus, a control bus must also be used to control the operation of the memory circuitry. The operations to be considered for memory interface are:
- Read data from memory (Access memory)
- Write data from memory (Access memory)
- Do no access memory

5 Input/Output Interfaces

5.1 Serial Communication

Serial Communication involves the transmission of data from one place to another using serial device. In serial communication the data is transmitted bit by bit on a single line. This minimizes the interconnecting wires, thereby also reducing the number of line drivers and receivers. Thus, only single bit can be transmitted at a time reducing the data transfer rate as the time required to transmit increases. However, since only one wire is used for data transfer the amount of cross-talk (interference between different lines) decreases. This in turn allows us to increase data rate which was the limiting factor in parallel communication.

5.1.1 Asynchronous interface: ASCII code, baud rate, start bit, stop bit, parity bit

ASCII code

The acronym ASCII stands for the American Standard Code for Information Interchange. It is an 8-bit code commonly used with microprocessors (including those of computers) for representing alphanumeric codes. Out of the 8 bits first 7 are used to represent the character while the eighth bit is used to test for errors and is referred to as parity bit. The 7 bit code provide with 128 (2^7) combinations. The parity bit used can be that of even parity or odd parity.

Baud rate

The term baud rate is used to indicate the rate at which serial data is being transferred. Bad rate is defined as 1/(bit interval). The rate is usually expressed as Bd or bits/second.

Start bit & Stop bit

In serial communication the data are sent serially so to differentiate between proper data and garbage data the data is enclosed in start bit and stop bit. The start bit indicates the beginning of a data character and indicated by the line going low for 1 bit time. The stop bit is indicated by line going continuously high after the data is over.

Parity bit

The parity bit follows the final data bit. Depending on the type of parity its value may be 0 or 1. This bit is used to check errors in received data.

5.1.2 Synchronous interface

Synchronous Interface is a widely used interface standard for industrial applications between a master (e.g. controller) and a slave (e.g. sensor). Synchronous Serial Interface (SSI) is based on RS422.

5.1.3 8255 Programmable Peripheral Interface (Block diagram and Modes only)

The 8255A is a widely used programmable parallel I/O device. It can be programmed to transfer data under various conditions from simple I/O to interrupt I/O. It is a general purpose

I/O device that can be used with almost any microprocessor. The PPI can work in two modes depending on the bit D7 on the control word. These are:

- BSR (Bit Set Reset) mode
- I/O (input/output) mode

24 I/O ports available in 8255 are divided into two groups (Group A and Group B). The 24 I/O ports form three 8-bit parallel ports (Port A, Port B and Port C). The eight bits of port C can be used as individual bits or be grouped in two four bit ports. The functions of these ports are defined by writing a control word in the control register. The different modes available in 8255 are shown below.
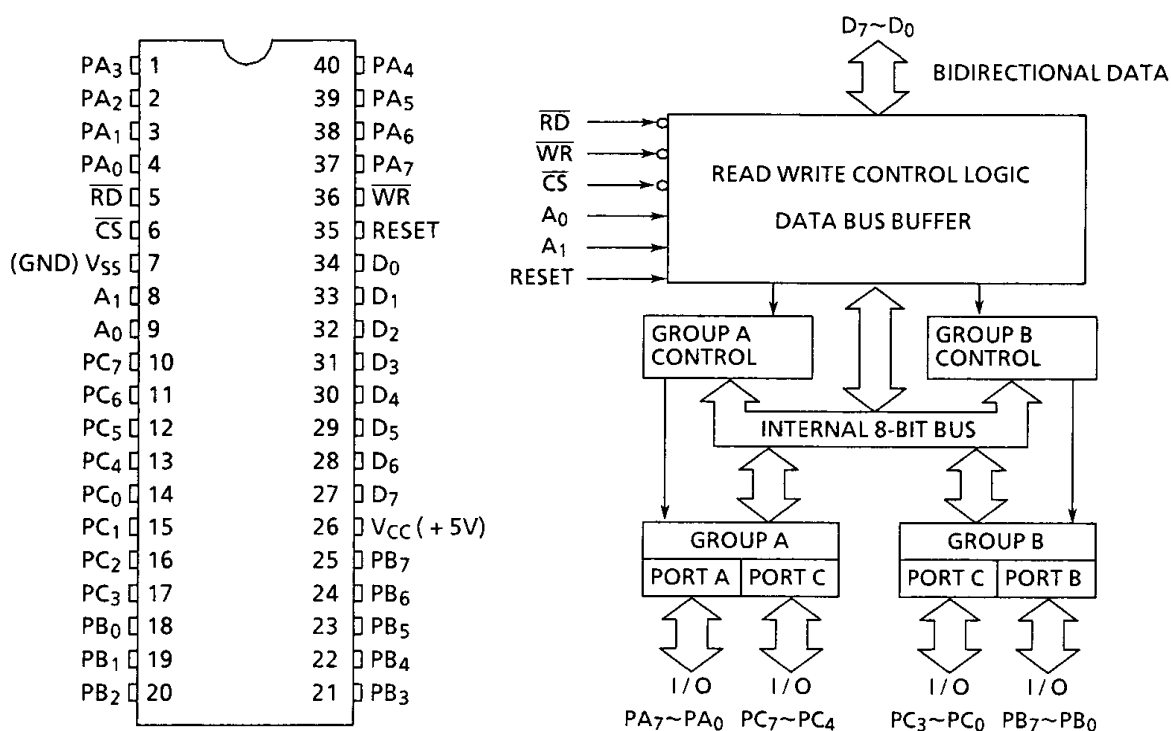
Block Diagram of 8255

Fig: Block diagram of 8255 PPI

The figure above shows the pin configuration and the block diagram of 8255 PPI. We see that there are 24 I/O lines. Port A and Port B can be used as 8 bit input or output port. Port C is a special supplementary port which can be used as an 8 bit I/O port, two 4 bit ports, as individual lines, or to produce handshake signals for port A and B.

We also see that there are eight data lines which allow you to write data bytes to a port or the control register and read bytes from the port or the status register under the control of the RD' and WR' lines. The address inputs, A0 and A1, allow you to selectively access one of the three ports or the control register. The internal addresses for the device are: port A, 00; port B, 01; port C, 10; control register, 11. Selecting the CS' input will enable 8255 for reading or writing. It is connected to address decoder circuitry to select the device when addressed. The RESET input of the 8255 is connected to the system reset line. When reset the device is programmed as input.

8255 Operational Modes and Initialization

The ports in 8255 can attain three modes of operation that can be selected by control words.

Mode 0- Basic I/O	(Group A, Group B)
Mode 1- Strobe input/Strobe output	(Group A, Group B)
Mode 2- Two way bus	(Port A only)

MODE 0
When we need a port for simple input or output without handshaking, we initialize the port in mode 0. If both port A and port B are initialized in mode 0, then the two halves of port C can be used together as an additional 8 bit port, or they can be used individually as two 4 bit ports. When used as outputs, the port C lines can be individually set or reset by sending a special control word to control register address (Bit set/reset mode). Also the two halves of port C are independent so one half can be initialized as input and the other half initialized as output.

* Output are latched
* Inputs are not latched
* Ports don't have handshake or interrupt capability

MODE 1
When we need to use strobed input or output then we can initialize port A and port B in mode 1. In this mode, some of the pins of port C function as handshake lines. Pins PC0, PC1, and PC2 function as handshake lines for port B if it is initialized in mode 1. If port A is initialized as a input handshake port, then pins PC3, PC4, and PC4 function as handshake signals. Pins PC6 and PC7 are available for used as input or output lines. If port A is initialized as a handshake output port, then port C pins PC3, PC6 and PC7 function as handshake signals, Port C pins PC4 and PC5 are available for use as input or output lines.

* Input and output data are latched
* Interrupt logic is supported

MODE 2
Only port A can be initialized in mode 2. In mode 2, port A can be used for bidirectional handshake data transfer. This means that data can be output or input on the same eight lines. If port A is initialized in mode 2, then pins PC3 through PC7 are used as handshake lines for port A.

Precaution for use in Mode 1 and 2
*When used in Mode 1 and 2, bits which are not used as control or status in Port C can be used as follows:*
* *If programmed as the input, they are accessed by normal Port C read.*
* *If programmed as the output, high order bits of Port C (PC7-PC4) are accessed using the bit set/reset function. As to low order bits of Port C (PC3-PC0), in addition to access by the bit set/reset function, 3 bits only can be accessed by normal writing.*

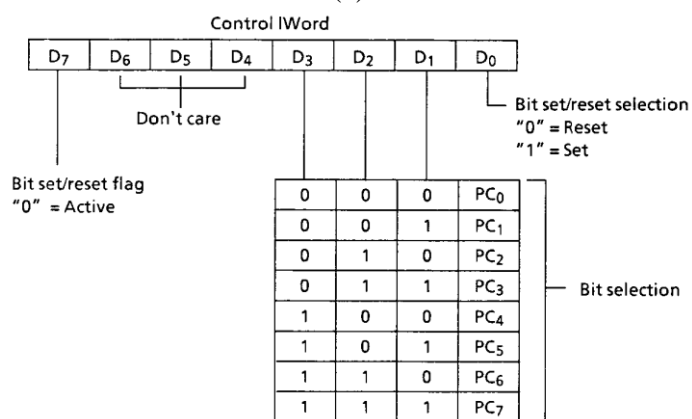**Constructing 8255 Control Word**

The following figure shows the formats for the two 8255 control words. The MSB of the control word distinguishes between the two control words. The *mode definition control word* format in the figure tells the device what modes you want the ports to operate in. We use the

bit set/reset control word format in the latter figure when we want to set or reset the output on a pin of port C or when you want to enable the interrupt signals for handshake data transfers.

Both the control word are sent to the control register of 8255
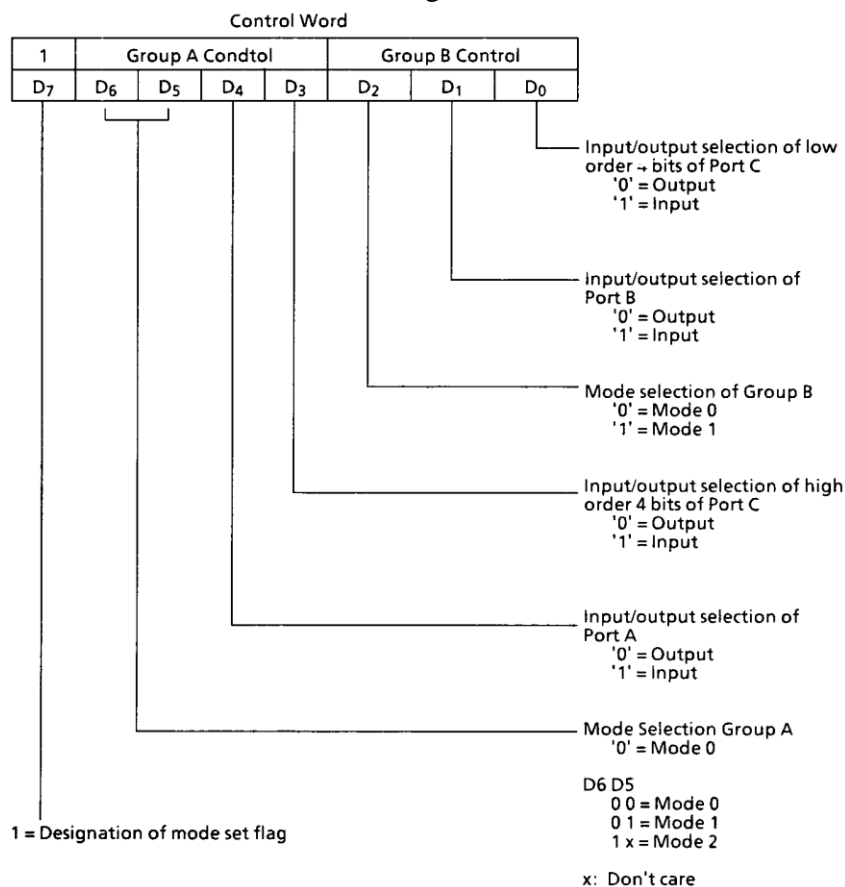
(a)

(b)

Fig: 8255 control word formats. (a) Mode set control word
(b) Port C bit set/reset control word.

Example 1:
Construct a control word to configure Ports A and port C_U as output ports and port B and port C_L as input ports.

D7	D6	D5	D4	D3	D2	D1	D0	Control Word
1	0	0	0	0	0	1	1	83H
I/O function	Port A in mode 0		Port A as output	Port C_U as output	Port B in mode 0	Port B as input	Port C_L as input	

Example2:
Write Control word to set and reset bits PC7 and PC3.

	D7	D6	D5	D4	D3	D2	D1	D0	Control Word
To set bit PC7	0	0	0	0	1	1	1	1	0FH
To reset bit PC7	0	0	0	0	1	1	1	0	OEH
To set bit PC3	0	0	0	0	0	1	1	1	O7H
To reset bit PC3	0	0	0	0	0	1	1	0	06H

5.1.4 8251 Programmable Communication Interface (Block diagram and Modes only)

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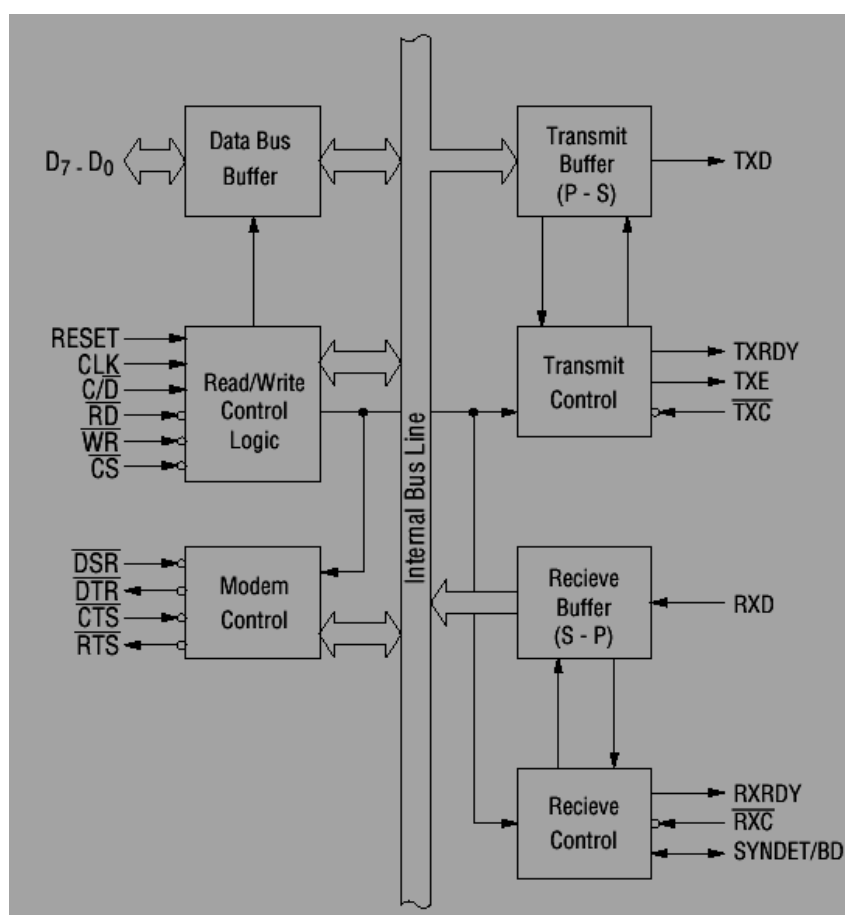

Fig: Block Diagram of Intel 8251

The internal block diagram of 8251 is shown above and the different pin functions of 8251 are listed on table below:

Pin Name	Pin Function
D7-D0	Data bus (8 bit)
C/D'	Control or data is to be written or read
RD'	Read data command
WR'	Write data or control command
CS'	Chip select
CLK	Clock pulse (TTL0
REST	Reset
TxC'	Transmitter clock
TxD	Transmitter data
RxC'	Receiver clock
RxD	Receiver data
RxRDY	Receiver ready (has character for CPU)
TxRDY	Transmitter ready (ready for char from CPU)
DSR'	Data set ready
DTR'	Data terminal ready
SYNDET/BD	Sync detect/ break detect
RTS'	Request to send data
CTS'	Clear to send data
TxEMPTY	Transmitter empty
Vcc	+5 V supply
GND	Ground

Fig: Pin Description of 8251

The eight parallel lines, D7-D0, connect to the system data bus so that data words and control/status words can be transferred to and from the device. The chip select (CS') input is connected to an address decoder so the device is enabled when addressed. The 8251 has two internal addresses, a control address which is selected when C/D is high, and a data address which is selected when C/D input is low. The RESET, RD', and WR' lines are connected to the system signals with the same names. The clock input of the 8251 is usually connected to a signal derived from the system clock to synchronize internal operations of the USART with the processor timing.

The signal labeled TxD on the upper right corner of the 8251 block diagram is the actual serial data output. The pin labeled RxD is the serial data input. The additional circuitry is needed to convert the TTL logic levels from 8251A to current loop or RS232C signals and opposite to receive the data. We will discuss it on section 5.4.

The shift registers in the USART require clocks to shift the serial data in and out. TxC is the transmit shift register clock input, and RxC' is the receive shift register clock input. Usually these two inputs are tied together so they are driven at same clock frequency. The frequency chosen must be 1, 16, or 64 times the transmit and receive baud rate depending upon the mode in which the 8251 is initialized. Using a clock frequency higher than the baud rate allows the receive shift register to be clocked at the centre of a bit time rather than at a transition. This reduces the chance of noise at a transition causing a read error.

The 8251A is double buffered. This means that one character can be loaded into a holding buffer while another character is being shifted out of the actual transmit shift register. The TxRDY output from the 8251 will go high when the holding buffer is empty and another character can be sent from CPU. The TxEMPTY pin on the 8251 will go high when both the holding buffer and the transmit shift register are empty. The RxRDY pin of the 8251A will go high when a character has been shifted into the receive buffer and is ready to be readout by the CPU. Incidentally, if a character is not read out before another character is shifted in, the first character will be over-written and lost.

The sync-detect/break-detect (SYNBDET/BD) pin has two uses. When the device is operating in asynchronous mode, this pin will go high if the3 serial data input line, RxD, stays low for more than two character times. This signal then indicates an intentional break in data transmission, or a break in the signal line. When programmed for synchronous data transmission this pin will go high, when the 8251 finds a specified Sync character(s) in the incoming string of data bits.

Initialization

To initialize an 8251A you must send first a mode word and then a command word to the control register address for the device. The following figure shows the formats for these words and for the 8251A status word which is read from the same address.

In the mode word Baud rate factor is specified by the two least significant bits of the mode word, is the ratio between the clock signal applied to the TxC'-RxC' inputs and the desired baud rate. For example, if you want to use TxC' of 19200 Hz and transmit data at 1200Bd, the baud rate factor is 19200/1200 or 16x. If bits D0 and D1 are both made 0's, the 8251A is programmed for synchronous data transfer. In this case the baud rate will be the same as the applied TxC' and RxC'. The other three combinations for these 2 bits represent asynchronous transfer. A baud rate factor of 1 can be used for asynchronous transfer only if the transmitting system and the receiving system both use the same TxC' and RxC'. The character length is specified by D2 and D3 in the mode word includes only the actual data bits not the start bit, parity bit or stop bit(s). If parity (D4) is disabled then no parity bit is inserted in the transmitted bit string. The bit D5 represents the type of parity used for error correction. The last two bits signifies the number of stop bits used after the data is sent.

After the mode word is send, you must then send it a command word.

Bit D0=1 enables the transmitter section and TxRDY output. When enabled, the 8251A TxRDY output will be asserted high if the CTS input has been asserted low, and the transmitter holding buffer is ready for another character from CPU. When a character is written to the 8251A data address, the TxRDY signal will go low and remain low until the holding buffer is again ready for another character.

Bit D1=1 will cause the DTR output of 8251 to be asserted low. This signal is used to tell a modem that a terminal or computer is operational.

Bit D2=1 enables the RxRDY output pin of 8251A. If enabled, the RxRDY pin will go high when the 8251 has a character in its receiver buffer ready to be send. This signal can be connected to an interrupt input so that characters can be read in on an interrupt basis. The RxRDY output is reset when a character is read from the 8251A.

Bit D3=1 causes the 8251 to output a character of all 0's which is called a break character. A break character is sometimes used to indicate the end of a block of transmitted data.

Bit D4=1 causes 8251 to reset the parity, overrun and framing error flags in the 8251 status register.

Bit D5=1 causes 8251 to assert its RTS output low. This signal is sent to a modem to ask whether the modem and the receiving system are ready for a data character to be sent.

Bit D6=1 causes the 8251 to internally reset when the command word is sent. After the software reset command is sent in this way, a new mode word must be sent

Bit D7=1 tells the 8251 to look for specified sync characters in a system of bits being shifted in. If 8251 finds specified sync characters, it will assert its SYNDET/BD pin high.

5.2 Parallel Communication

In all preceding chapters, we have used port devices to input and output parallel data to and from the microprocessor and other devices. Although serial communication is also popular parallel data transfer takes place between any two devices like microprocessor and memory, microprocessor and I/O devices and memory and I/O devices.

Methods of Parallel Communication
- Simple Input and Output:
  When you need to get digital data from a simple switch such as a thermostat, into a microprocessor, all you have to do is connect the switch to an input port line and read the port. The thermostat data is always present and ready, so you can read it at any time. Similarly, to output data to a simple device such as LED, all you have to do is connect the input of the LED buffer on an output port in and output the logic level required to turn on the light. The LED is always there and ready, so you can send data to it at any time. The timing waveform below shows the data transfer sequence.
- Simple Strobe I/O
  In many applications, valid data is present on an external device only at a certain time, so it must be read in at that time. An example of this is the ASCII-encoded keyboard. When a key is pressed circuitry on the keyboard sends out the ASCII code for the pressed key on eight parallel data lines, and then sends out a strobe signal on another line to indicate that valid data is present on the eight data lines as shown in the figure. You can connect this strobe line to an input port line and poll it to determine when you can input valid data from the keyboard. Another alternative is to connect the strobe line to an interrupt input on the processor and have an interrupt service procedure read in the data when the processor receives an interrupt. The scheme behind this arrangement is that the transfer is time dependent. Thus you can only read data when a strobe pulse tells you that the data is valid.
- Single handshake I/O
  The figure shows an example for a handshake data transfer from a peripheral device to a microprocessor. The peripheral outputs some parallel data and sends an STB signal to the microprocessor. The microprocessor detects the asserted STB

signal on a polled or interrupt basis and reads in the byte of data. Then the microprocessor sends an Acknowledge signal (ACK) to the peripheral to indicate that the data has been read and that the peripheral can send the next byte of data. Such transfer is strobed input from microprocessor's point of view.

Similarly, for an output the microprocessor outputs a character to the printer and raises the STB signal to the printer to tell "Here is a character for you." When the printer is ready, it answers back with the ACK signal to tell "I got that one; send me another".

- Double Handshake Data Transfer

  Such mode of data transfer is used where even more coordination is required between the sending systems and the receiving system. Such mode of data transfer can be much easily understood as a conversation between two people. In the waveforms each signal edge has meaning. The sending device asserts its STB line to ask, "Are you ready?" The receiving system raises its ACK line high to say, "I'm ready." Then the peripheral device sends the byte of data and raises its STB' line high to say. "Here is some valid data for You." After it has read in the data the receiving system drops its ACK line low to say. "I have the data. Thank you. and I await your request to send the next byte of data".

5.3 Data transfer using wait interface

To understand wait interface, let us consider a simple keyboard consisting of 8 switches, connected to a microprocessor through a parallel interface circuit. In this case, each switch is of dip switches. The microprocessor should be able to detect that a key has been activated. This can be done by observing that all bits are equal to 1 when none of the key is pressed. The processor should repeatedly read the state of the input section until it finds that one of the eight bits is equal to 0.

The microprocessor should ensure that when a key is activated, it is read only once. This can be done in the software. When one of the keys is activated and the input byte is read, the software program could e repeatedly check the input byte until it reaches to all 1s indicating that the pressed key has been released

The other problem occurs from the nature of the operation of the mechanical switches. Almost all mechanical contacts are subject to vibrations or bouncing, while the switch is being opened or closed. The time required for a switch to settle o fully open or a fully closed position is about 2ms. This may not seem much as it is $1/500^{th}$ of a second, however, a microprocessor can execute several hundred of instruction in this time. Thus there must be a mechanism to ignore the keyboard unit till the keyboard denounce is stopped. This is done by checking the input until it is the same before accepting it as valid. This can be visualized in the following flowchart.
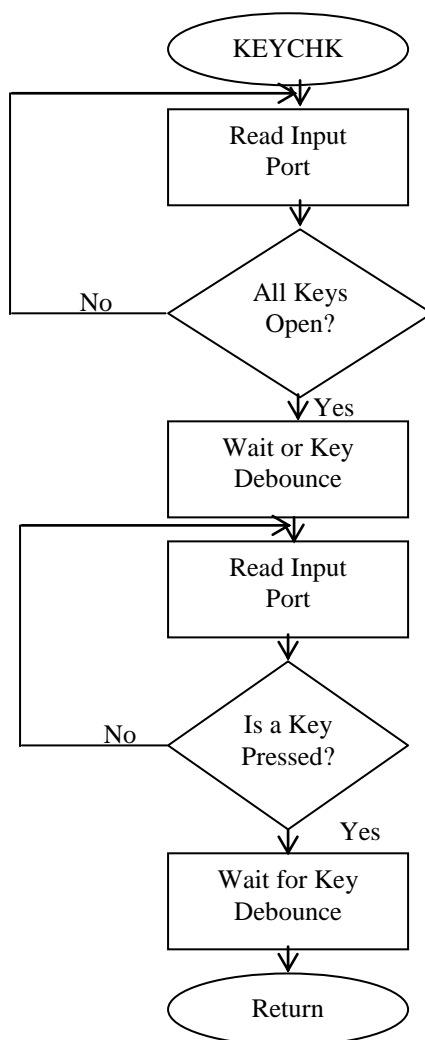
Fig: flowchart of Switch Interface

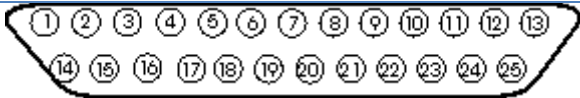## 5.4 RS-232 and IEEE 488-1978 general purpose interface standard

RS-232C Serial Data Standard

In 8251(USART), we discussed how serial communication takes place. The TTL signals output by a USART, however, are not suitable for transmission over long distances, so these signals are converted to some other form to be transmitted. In this section we discuss device used to send serial data signals over long distances.

The 1960s saw widespread use of timeshare computer terminals. However, to install new communication lines was a Herculean task. Thus modems were developed so that the terminals could use phone lines to communicate, which were already in existence. Modems 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 standard RS-232C. This standard describes the function of 25 signals 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-232 stands for Recommend Standard number 232 and C is the latest revision of the standard. RS-232 is a 25 pin standard but for systems where many of 25 pins are not needed, a 9 pin connecter is also used. The voltage levels for all RS-232 signals are as follows. A logic high, or mark, is a voltage between -3V and -15V. A logic low or space is a voltage between +3V and +15V. Voltages such as ±12V are commonly used.

Male RS232 DB25	① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰ ⑱ ⑲ ⑳ ㉑ ㉒ ㉓ ㉔ ㉕
Pin Number	**Direction of signal:**
1	Protective Ground
2	Transmitted Data (TD) Outgoing Data (from a DTE to a DCE)
3	Received Data (RD) Incoming Data (from a DCE to a DTE)
4	Request To Send (RTS) Outgoing flow control signal controlled by DTE
5	Clear To Send (CTS) Incoming flow control signal controlled by DCE
6	Data Set Ready (DSR) Incoming handshaking signal controlled by DCE
7	Signal Ground Common reference voltage
8	Carrier Detect (CD) Incoming signal from a modem
20	Data Terminal Ready (DTR) Outgoing handshaking signal controlled by DTE
22	Ring Indicator (RI) Incoming signal from a modem

Male RS232 DB9	① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨
Pin Number	**Direction of signal:**
1	Carrier Detect (CD) (from DCE) Incoming signal from a modem
2	Received Data (RD) Incoming Data from a DCE
3	Transmitted Data (TD) Outgoing Data to a DCE
4	Data Terminal Ready (DTR) Outgoing handshaking signal
5	Signal Ground Common reference voltage
6	Data Set Ready (DSR) Incoming handshaking signal
7	Request To Send (RTS) Outgoing flow control signal
8	Clear To Send (CTS) Incoming flow control signal
9	Ring Indicator (RI) (from DCE) Incoming signal from a modem

Fig: 25 pin and 9 pin connectors used for RS 232 standards on DTE device

IEEE 488 standard

The IEEE-488 bus was developed to connect and control programmable instruments, and to provide a standard interface for communication between instruments from different sources. Hewlett-Packard originally developed the interfacing technique, and called it HP-IB (Hewlett-Packard Instrument Bus). The interface quickly gained popularity in the computer industry. Because the interface was so versatile, the IEEE committee renamed it GPIB (General Purpose interface Bus).

IEEE-488 allows up to 15 devices to share a single 8-bit parallel electrical bus by daisy chaining connections. The slowest device participates in control and data transfer handshakes to determine the speed of the transaction. The maximum data rate is about one Mbyte/sec in the original standard, and about 8 Mbyte/sec with IEEE-488.1-2003 (HS-488).

The IEEE-488 bus employs 16 signal lines eight bi-directional used for data transfer, three for handshake, and five for bus management plus eight ground return lines. The main features of IEEE-488 standard are as follows.

- Parallel 8 – bit data path for data read and write
- Three flow control lines (handshake signals)
- Five special lines for control of bus and interrupts
- Data speed of 1M bytes/sec limited by the speed of the slowest listener
- Special stackable connectors.
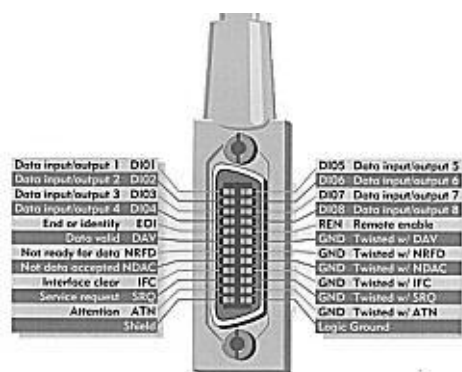- A maximum of 15 instruments connected to a common bus
- Up to 20m maximum cable length

Fig: The IEEE-488 Connector

[SAP1 and SAP2 Coming Soon]

 http://www.facebook.com/bsccsit.com