Lecture Notes
On
Microprocessor and Microcomputer

(For 4th Semester CSE/IT)

Prepared by:

Miss Barsha Subudhi Ray

Guest faculty (CSE/IT)

UCP Engineering School Berhampur.

Unit -1

Microprocessor and Microcomputer:

Microprocessor:

The processor on a single chip is called a Microprocessor which can process micro-instructions. Instructions in the form of 0sand 1s are called micro-instructions. The microprocessor is the CPU part of a microcomputer, and it is also available as a single integrated circuit. Thus as main components, the microprocessor will have theControl Unit (CU) and the Arithmetic Logic Unit (ALU) of a microcomputer. An example is Intel 8085 microprocessor. In addition to the microprocessor features, a microcomputer will have the following additional features:

- ROM/PROM/EPROM/EEPROM for storing program;
- RAM for storing data, intermediate results, and final results;
- I/O devices for communication with the outside world;
- I/O ports for communication with the I/O devices.

Microcomputer:

A microcomputer can be defined as a small sized, inexpensive, and limited capability computer. It has the same architectural block structure that is present on a computer. Present-day microcomputers are having smaller sizes. Nowadays, they are of the size of a notebook. But in the coming days also their sizes will get more reduced as well. Due to their lower costs, individuals can possess them as their personal computers. Because of mass production, they are becoming still cheaper. Initially, in the earlier days, they were not very much powerful. Their internal operations and instructions were very much limited and restricted. But at present days, microcomputers have not only multiplied and divide instructions on unsigned and signed numbers but are also capable of performing floating point arithmetic operations

Microprocessor	Micro computer
• it is a standalone device • microprocessor doesn’t have memory inside it to use memory it is connected by address and data bus	• Microcomputer is a miniature computer whis has microprocessor as well as other peripheral devices. • Microcomputer has rom and ram inside itself.

Evolution of Microprocessor:

1. **4-bit Microprocessor:** The first microprocessor (Intel 4004) was invented in 1971. It was a 4-bit calculation device with a speed of 108 kHz. Since then, microprocessor power has grown exponentially. It has 3200 PMOS transistors. It is a 4-bit device used in calculator.

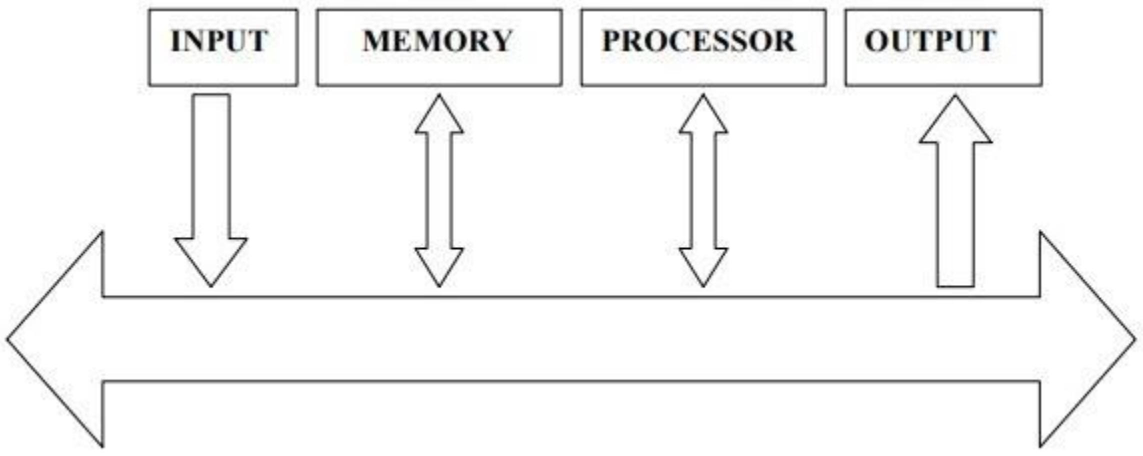
2. **8-Bit microprocessor:**
   In 1972, Intel came out with the 8008 which is 8-bit. In 1974, Intel announced the 8080 followed by 8085 i.e a 8-bit processor 8085 processor has 8 bit ALU (Arithmetic Logic unit). Similarly 8086 processor has 16 bit ALU. This had a larger instruction set then 8080. used NMOS transistors, so it operated much faster than the 8008. The 8080 is referred to as a "Second generation Microprocessor".

3. **Third Generation (16 - bit Microprocessor):**
   The third generation microprocessors, introduced in 1978 were represented by Intel's 8086, Zilog Z800 and 80286, which were 16 - bit processors with a performance like minicomputers.
4. **Fourth Generation (32 - bit Microprocessors**):
   Several different companies introduced the 32-bit microprocessors, but the most popular one is the Intel 80386. It has around 275000 transitors.

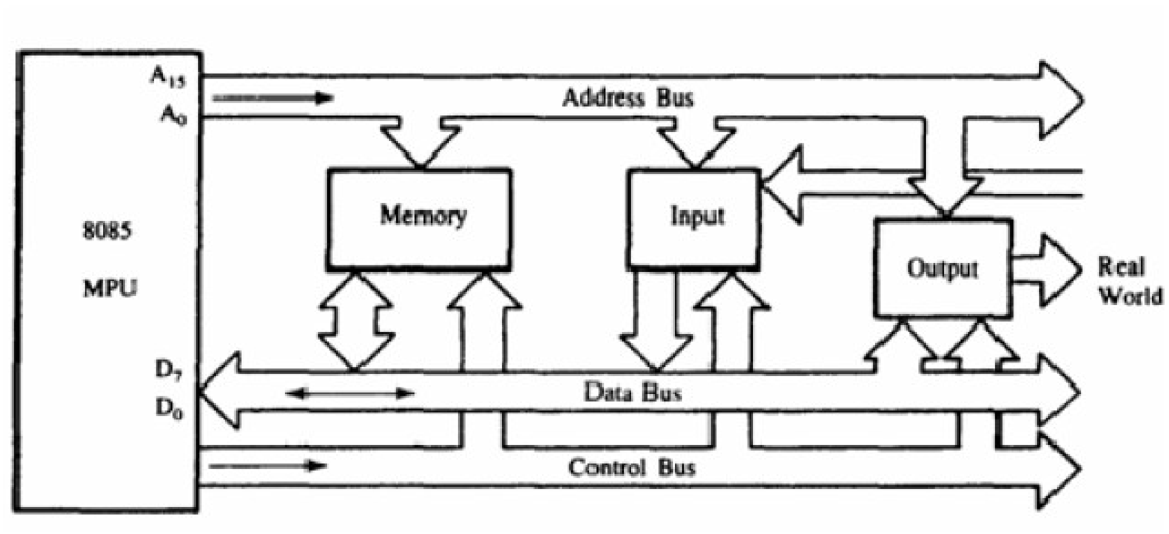
5. **Fifth Generation (64 - bit Microprocessors):**
   From 1995 to now we are in the fifth generation. After 80856, Intel came out with a new processor namely Pentium processor followed by **Pentium Pro CPU**, which allows multiple CPUs in a single system to achieve multiprocessing. Other improved 64-bit processors are **Celeron, Dual, Quad, Octa Core processors**.

Bus structure:

The simplest and most common way of interconnecting various parts of the computer. To achieve a reasonable speed of operation, a computer must be organized so that all its units can handle one full word of data at a given time.A group of lines that serve as a connecting port for several devices is called a bus. In addition to the lines that carry the data, the bus must have lines for address and control purpose. Simplest way to interconnect is to use the single bus as shown



Since the bus can be used for only one transfer at a time, only two units can actively use the bus at any given time. Bus control lines are used to arbitrate multiple requests for use of one bus.



Address Bus:

- The address bus is a group of 16 lines generally identified as A0 to A15.
- The address bus is unidirectional: bits flow in one direction-from the MPU to peripheral devices.
- The MPU uses the address bus to perform the first function: identifying a peripheral or a memory location.

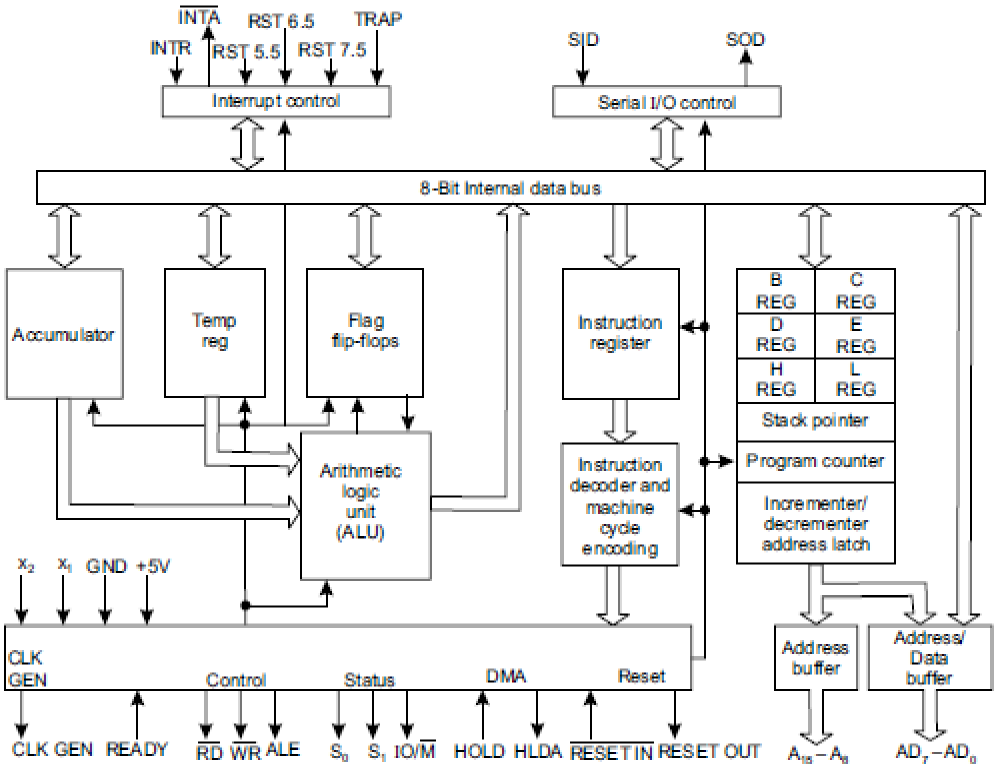
Data bus:

- The data bus is a group of eight lines used for data flow.
- These lines are bi-directional - data flow in both directions between the MPU and memory and peripheral devices.
- The MPU uses the data bus to perform the second function: transferring binary information.
- The eight data lines enable the MPU to manipulate 8-bit data ranging from 00H to FFH (2^8 = 256 numbers).
- The largest number that can appear on the data bus is 11111111 i.e 0FFH.

Control Bus:

- The control bus carries synchronization signals and providing timing signals.
- The MPU generates specific control signals for every operation it performs. These signals are used to identify a device type with which the MPU wants to communicate.

Architecture of 8085 (8 bit) Microprocessor



The 8085 microprocessor is an 8-bit processor available as a 40-pin IC package and uses +5 V for power. It can run at a maximum frequency of 3 MHz. Its data bus width is 8-bit and address bus width is 16-bit, thus it can address 216 = 64 KB of memory. The internal architecture of 8085 is shown in above figure.

Arithmetic and Logic Unit:

The ALU performs the actual numerical and logical operations such as Addition (ADD), Subtraction (SUB), AND, OR etc. It uses data from memory and from Accumulator to perform operations. The results of the arithmetic and logical operations are stored in the accumulator.

Registers :

The 8085 includes six registers, one accumulator and one flag register, as shown in Fig. 3. In addition, it has two 16-bit registers: stack pointer and program counter. They are briefly described as follows. The 8085 has six general-purpose registers to store 8-bit data; these are identified as B, C, D, E, H and L. they can be combined as register pairs - BC, DE and HL to perform some 16-bit operations. The programmer can use these registers to store or copy data into the register by using data copy instructions.

S. No.	Name of the Register	Quantity	Capacity
1.	Accumulator (or) Register A	1	8-bit
2.	Temporary register	1	8-bit
3.	General purpose registers (B, C, D, E, H and L)	6	8-bit each
4.	Stack pointer (SP)	1	16-bit
5.	Program counter (PC)	1	16-bit
6.	Instruction register	1	8-bit
7.	Incrementer/Decrementer address latch	1	16-bit
8.	Status flags register	1	8-bit

Accumulator:

The accumulator is an 8-bit register that is a part of ALU. This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.

Flag register:

The ALU includes five flip-flops, which are set or reset after an operation according to data condition of the result in the accumulator and other registers. They are called Zero (Z), Carry (CY), Sign (S), Parity (P) and Auxiliary Carry (AC) flags. Their bit positions in the flag register are shown in Fig. 4. The microprocessor uses these flags to test data conditions.

D7	D6	D5	D4	D3	D2	D1	D0
S	Z	X	AC	X	P	X	CY

Sign (S) flag: – If the MSB of the result of an operation is 1, this flag is set, otherwise it is reset.

Zero (Z) flag:– If the result of an instruction is zero, this flag is set, otherwise reset.

Auxiliary Carry (AC) flag:– If there is a carry out of bit 3 and into bit 4 resulting from the execution of an arithmetic operation, it is set otherwise reset. This flag is used for BCD operation and is not available to the programmer to change the sequence of an instruction.

Carry (CY) flag:– If an instruction results in a carry (for addition operation) or borrow (for subtraction or comparison) out of bit D7, then this flag is set, otherwise reset.

Parity (P) flag:– This flag is set when the result of an operation contains an even number of 1's and is reset otherwise.

Program Counter (PC):

This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer. The microprocessor uses this 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. When a byte is being fetched, the program counter is automatically incremented by one to point to the next memory location.

Stack Pointer (SP) :

The Stack pointer is also a 16-bit register, used as a memory pointer. It points to a memory location in R/W memory, called stack. The beginning of the stack is defined by loading 16-bit address in the stack pointer.

Instruction Register/Decoder:

It is an 8-bit register that temporarily stores the current instruction of a program. Latest instruction sent here from memory prior to execution. Decoder then takes instruction and decodes or interprets the instruction. Decoded instruction then passed to next stage.

Control Unit :

Control Generates signals on data bus, address bus and control bus within microprocessor to carry out the instruction, which has been decoded.

Data Bus: Data bus carries data in binary form between microprocessor and other external units such as memory. It is used to transmit data i.e. information, results of arithmetic etc between memory and the microprocessor. Data bus is bidirectional in nature. The data bus width of 8085 microprocessor is 8-bit i.e. 28 combination of binary digits and are typically identified as D0 – D7. Thus size of the data bus determines what arithmetic can be done. If only 8-bit wide then largest number is 11111111 (255 in decimal). Therefore, larger numbers have to be broken down into chunks of 255. This slows microprocessor.

Address Bus: The address bus carries addresses and is one way bus from microprocessor to the memory or other devices. 8085 microprocessor contain 16-bit address bus and are generally identified as A0 - A15. The higher order address lines (A8 – A15) are unidirectional and the lower order lines (A0 – A7) are multiplexed (time-shared) with the eight data bits (D0 – D7) and hence, they are bidirectional.

Control Bus: Control bus are various lines which have specific functions for coordinating and controlling microprocessor operations. The control bus carries control signals partly unidirectional and partly bidirectional. The following control and status signals are used by 8085 processor:

ALE (output): Address Latch Enable is a pulse that is provided when an address appears on the AD0 – AD7 lines, after which it becomes 0.

$R\acute{}D$ **(active low output):** The Read signal indicates that data are being read from the selected I/O or memory device and that they are available on the data bus.

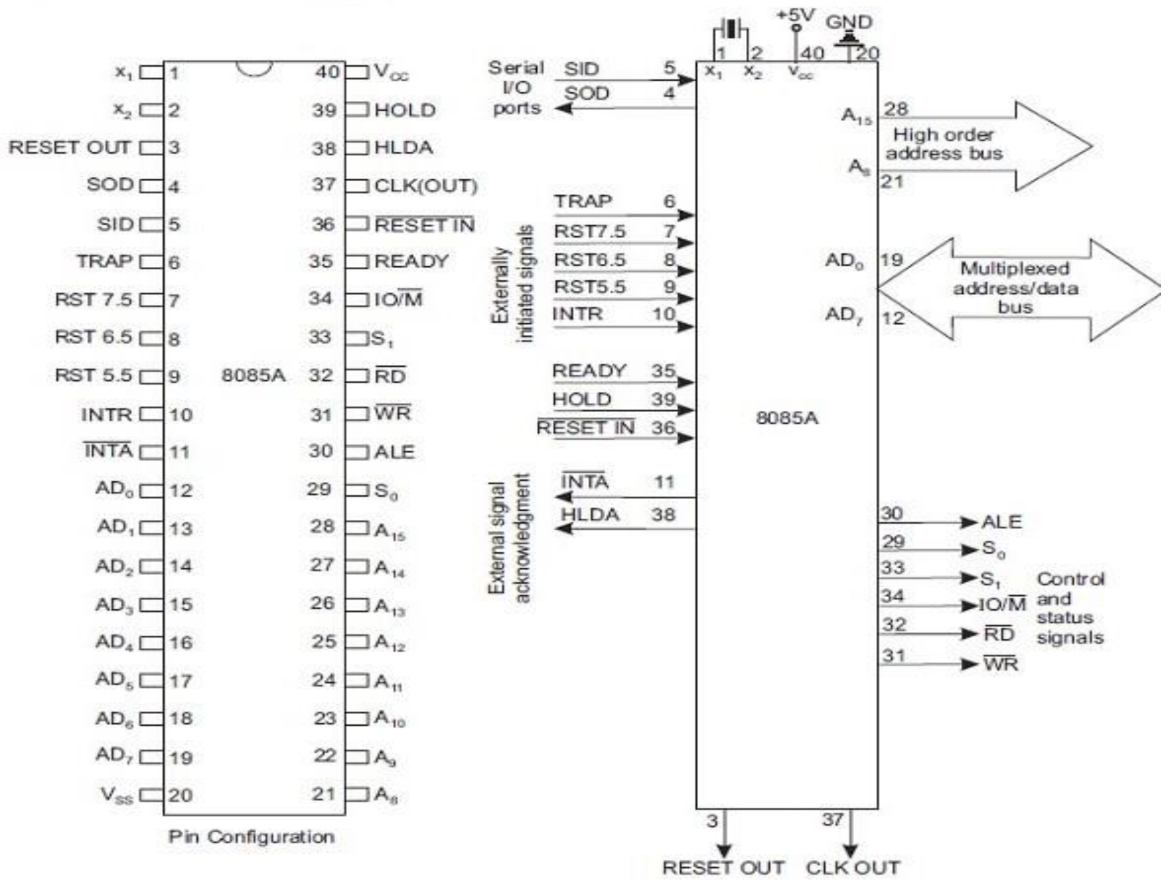
$\acute{W}R$ **(active low output):** The Write signal indicates that data on the data bus are to be written into a selected memory or I/O location.

$IO/\acute{M}$**(output)**: It is a signal that distinguished between a memory operation and an I/O operation. When $IO/\acute{M}$ = 0 it is a memory operation and $IO/\acute{M}$ = 1 it is an I/O operation.

S1 and S0 (output): These are status signals used to specify the type of operation being performed; they are listed in Table ,

S1	S0	States
0	0	Halt
0	1	Write
1	0	Read
1	1	Fetch

8085 pin diagram



Pin Configuration
8085A
RESET OUT
CLK OUT
High order address bus
Multiplexed address/data bus
Control and status signals
Serial I/O ports
Externally initiated signals
External signal acknowledgment

Properties:

- It is a 8-bit microprocessor

- Manufactured with N-MOS technology
- 40 pin IC package
- It has 16-bit address bus and thus has 216 = 64 KB addressing capability.
- Operate with 3 MHz single-phase clock
- +5 V single power supply

The logic pin layout and signal groups of the 8085nmicroprocessor are shown in Fig. 6. All the signals are classified into six groups:

- Address bus
- Data bus
- Control & status signals
- Power supply and frequency signals
- Externally initiated signals
- Serial I/O signals

Address and Data Buses:

- A8 – A15 (output, 3-state): Most significant eight bits of memory addresses and the eight bits of the I/O addresses. These lines enter into tri-state high impedance state during HOLD and HALT modes.
- AD0 – AD7 (input/output, 3-state): Lower significant bits of memory addresses and the eight bits of the I/O addresses during first clock cycle. Behaves as data bus during third and fourth clock cycle. These lines enter into tri-state high impedance state during HOLD and HALT modes.

Control & Status Signals:

- ALE: Address latch enable
- $\acute{RD}$: Read control signal.
- $\acute{WR}$: Write control signal.
- $IO/\acute{M}$, S1 and S0 : Status signals.

Power Supply & Clock Frequency:

- Vcc: +5 V power supply
- Vss: Ground reference
- X1, X2: A crystal having frequency of 6 MHz is connected at these two pins.
- CLK: Clock output

Externally Initiated and Interrupt Signals:

- RESET IN: When the signal on this pin is low, the PC is set to 0, the buses are tristated and the processor is reset.
- RESET OUT: This signal indicates that the processor is being reset. The signal can be used to reset other devices.
- READY: When this signal is low, the processor waits for an integral number of

clock cycles until it goes high.

- HOLD: This signal indicates that a peripheral like DMA (direct memory access) controller is requesting the use of address and data bus.
- HLDA: This signal acknowledges the HOLD request.
- INTR: Interrupt request is a general-purpose interrupt.
- INTA : This is used to acknowledge an interrupt.
- RST 7.5, RST 6.5, RST 5,5 – restart interrupt: These are vectored interrupts and have highest priority than INTR interrupt.
- TRAP: This is a non-maskable interrupt and has the highest priority.

Serial I/O Signals:
SID: Serial input signal. Bit on this line is loaded to D7 bit of register A using RIM instruction.
SOD: Serial output signal. Output SOD is set or reset by using SIM instruction.

Difference between SPR & GPR:

GPR	SPR
8085 has 6 general purpose registers .	All the registers other than gpr are known as special purpose registers.
General purpose registers can be used either data and address register.	All specif purpose registers has some special functions

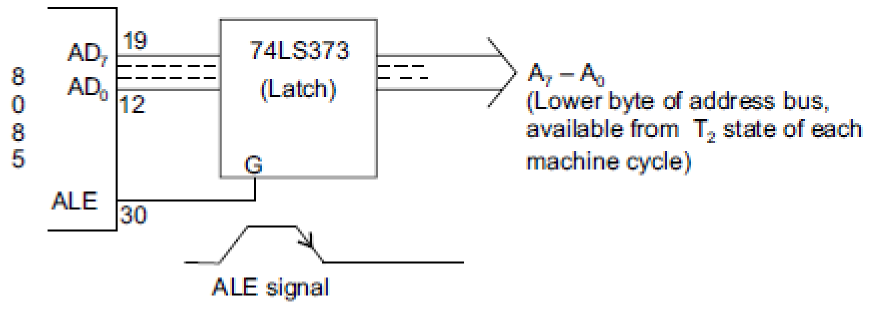
Timing and control unit:

The T&C section is a part of CPU and generates timing and control signals for execution of instructions. This section includes Clock signals, Control signals, Status signals, DMA signals as also the Reset section. This section controls fetching and decoding operations. It also generates appropriate control signals for instruction execution as also the signals required to interface external devices.

- The control and status signals are ALE, $R\acute{}D$, $W\acute{}R$, IO/$\acute{M}$, S0, S1 and READY.

- The interrupt signals are TRAP, RST 7.5, RST 6.5, RST 5.5, INTR. $IN\acute{}TA$ is an interrupt acknowledgement signal indicating that the processor has acknowledged an INTR interrupt.
- Serial I/O signals are SID and SOD
- DMA signals are HOLD and HLDA
- Reset signals are $RESE\acute{}T \in$¿¿ and RESET OUT.

ALE:
Pin 30 of 8085 is the ALE pin which stands for 'Address Latch Enable'. ALE signal is used to demultiplex the lower order address bus (AD0 – AD7). Pins 12 to 19 of 8085 are AD0 – AD7 which is the multiplexed address-data bus. Multiplexing is done to reduce the number of pins of 8085. Lower byte of address (A0 – A7) are available from AD0 – AD7 (pins 12 to 19) during T1 of machine cycle. But the lower byte of address (A0 – A7), along with the upper byte A8 – A15 (pins 21 to 28) must be available during T2 and rest of the machine cycle to access memory location or I/O ports. Now ALE signal goes high at the beginning of T1 of each machine cycle and goes low at the end of T1 and remains low during the rest of the machine cycle. This high to low transition of ALE signal at the end of T1 is used to latch the lower order address byte (A0 – A7) by the latch IC 74LS373, so that the lower byte A0 – A7 is continued to be available till the end of the machine cycle. The situation is explained in the following figure:


8085
AD_7
AD_0
19
12
74LS373
(Latch)
G
ALE
30
$A_7 – A_0$
(Lower byte of address bus, available from T_2 state of each machine cycle)
ALE signal

DMA:
DMA mode of data transfer is fastest and pins 39 and 38 (HOLD and HLDA) become active only in this mode.
When DMA is required, the DMA controller IC (8257) sends a 1 to pin 39 of 8085. At the end of the current instruction cycle of the microprocessor it issues a1 to pin 38 of the controller. After this the bus control is totally taken over by the controller.
When 8085 is active and 8257 is idle, then the former is MASTER and the latter is SLAVE, while the roles of 8085 and 8257 are reversed when 8085 is idle and 8257 becomes active. mode of data transfer is fastest and pins 39 and 38 (HOLD and HLDA) become active only in this mode.
When DMA is required, the DMA controller IC (8257) sends a 1 to pin 39 of 8085. At the end of the current instruction cycle of the microprocessor it issues a1 to pin 38 of the controller. After this the bus control is totally taken over by the controller.
When 8085 is active and 8257 is idle, then the former is MASTER and the latter is SLAVE, while the roles of 8085 and 8257 are reversed when 8085 is idle and 8257 becomes

active.

IO/ $\acute{M}$ signal indicates whether I/O or memory operation is being carried out. A high on this signal indicates I/O operation while a low indicates memory operation. S0 and S1 indicate the type of machine cycle in progress.

$RES\acute{E}T$ ∈¿¿ It is an input signal which is active when its status is low. When this pin is low, the following occurs:

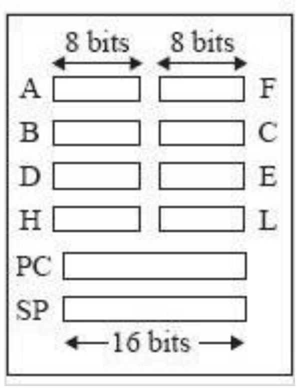
- The program counter is set to zero (0000H).
- Interrupt enable and HLDA F/Fs are resetted.
- All the buses are tri-stated.
- Internal registers of 8085 are affected in a random manner.

RESET OUT:
When this signal is high, the processor is being reset. This signal is synchronised to the processor clock and is used to reset other devices which need resetting

STACK:

Stack is a reserved of memory used to keep track of a program's internal operations, including functions, return addresses, passed parameters ed parameters, etc. A stack is usually maintained as a "last in, first out" (LIFO) data s) data structure, so that the last item added to the structure is the first item used. Sometimes it is useful to have region of memory for temporary storage, which does not have to be allocated named variables. When we use subroutines and interrupts it will be essential to have such a storage region. Such region is called a Stack

The Stack Pointer register will hold the address of the top location of the stack. And the program counter is a register always it will hold the address of the memory location from where the next instruction for execution will have to be fetched. The complete programmer's view of 8085 is shown in the following figure



SP is a special purpose 16-bit register. It contains a memory address. Suppose SP contents are FC78H, then the 8085 interprets it as follows

Memory locations FC78H, FC79H, ..., FFFFH are having useful information. In other words, these locations are treated as filled locations. Memory locations FC77H, FC76H, ..., 0000H are not having any useful information. In other words, these locations are treated as empty locations.

On a stack, we can perform two operations. PUSH and POP. In case of PUSH operation, the SP register gets decreased by 1 and new data item used to insert on to the top of the stack. On the other hand, in case of POP operation, the data item will have to be deleted from the top of the stack and the SP register will get increased by the value of 1.

Thus, the contents of SP specify the top most useful location in the stack. This is known as **stacktop** .In other words, it indicates the memory location with the smallest address having useful information. This is pictorially represented in the following figure –

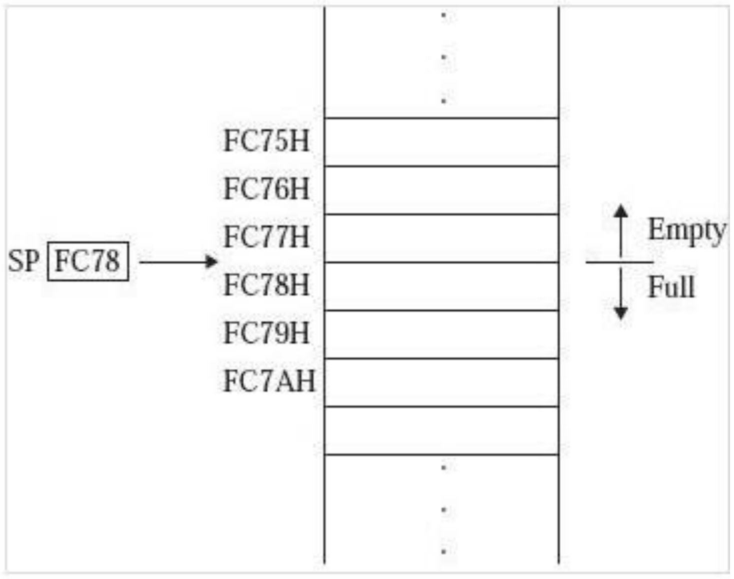


Fig. Interpretation of SP contents

8085 Interrupts:

Six signals are associated with interrupt logic. There are five interrupt request pins through which the 8085 may be interrupted. These are TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR. The first four interrupts (TRAP, RST 7.5, RST 6.5, RST 5.5) are supported by hardware implemented vectoring, i.e. the locations of Interrupt Service Routines(ISR) in memory for these interrupts are fixed and automatic vectoring to the location takes place whenever any of these four interrupts is recognized. The interrupt service routine locations for these interrupts are as follows:

Interrupt	**ISR location**
TRAP	24H
RST 7.5	3CH
RST 6.5	34H
RST 5.5	2CH

The last interrupt INTR is a general-purpose interrupt The interrupt request on this line is acknowledged by the 8085 on the output line. On receiving the acknowledgement, the interrupting device places on the data bus the address of Interrupt Service Routine (ISR) in the memory. The microprocessor executes the ISR to service the interrupt. There are two ways by which the interrupting device can send the ISR address. It can input one byte code of one of the "RST n" instructions (RST 0 to RST 7). **These are other wise known as software interrrupts** . The microprocessor will execute this instruction to branch to ISR at 8 bit address location. The RST n is a single byte subroutine call instruction. Another way is to input a 3-byte code for "CALL addr16" instruction. On receiving this 3-byte code, the microprocessor will execute this instruction to branch to ISR at addr16.

The vector address for a software interrupt is calculated as follows:
Vector address = interrupt number × 8
For example, the vector address for RST 5 is calculated as
$5 \times 8 = (40)_{10} = (28)_{H}$
Vector address for RST 5 is 0028H.

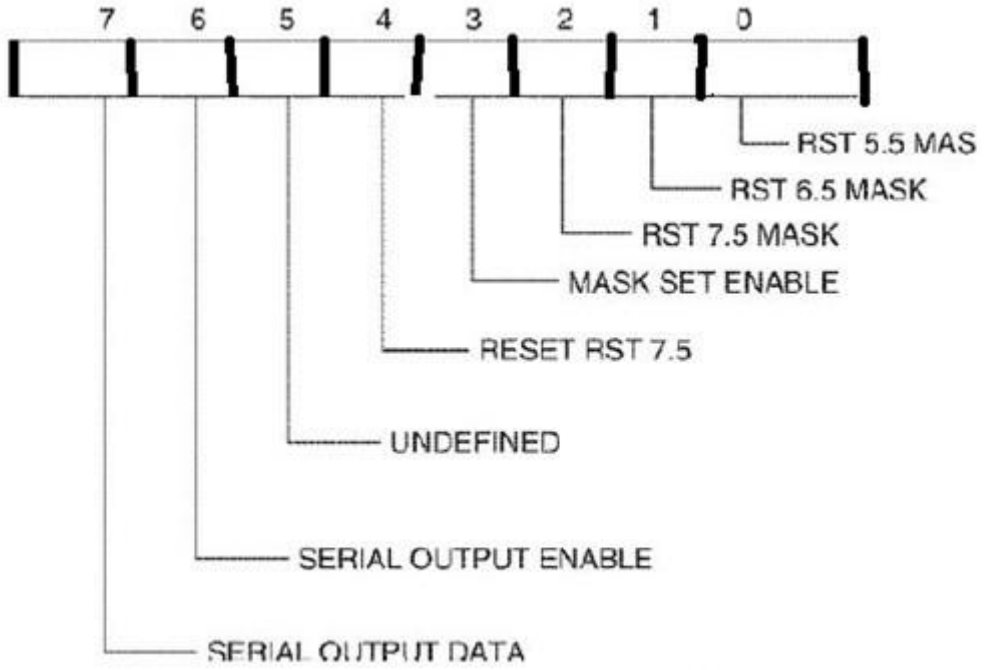
Instruction	Corresponding HEX code	Vector addresses
RST 0	C7	0000H
RST 1	CF	0008H
RST 2	D7	0010H
RST 3	DF	0018H
RST 4	E7	0020H
RST 5	EF	0028H
RST 6	F7	0030H
RST 7	FF	0038H

TRAP interrupt is the non-maskable interrupt for 8085. It means that if an interrupt comes via TRAP, 8085 will have to recognise the interrupt.

Interrupts INTR, RST 5.5 and RST 6.5 are level-triggered, i.e. a high signal should be present on these lines when the interrupt is recognized. In other words, if a device sends an interrupt pulse of short duration on any of these lines, it is possible that this interrupt request is ignored totally owing to the reason that the interrupt pulse may not be present when these lines are scanned by the microprocessor for the interrupt request. RST 7.5 is edge-triggered and TRAP is level- as well as edge-triggered. Both are associated with internal latches which store the interrupt requests as and when the interrupt signal on these lines goes from low to high. TRAP is normally reserved for power failure or some other highest priority interrupt. Thus, it will be desirable to protect this pin from spurious signals. It is because of this reason that TRAP is level as well as edge-triggered.

The interrupts of 8085 have their priorities fixed—TRAP interrupt has the highest priority, followed by RST 7.5, RST 6.5, RST 5.5 and lastly INTR.

Interrupt masking:
An interrupt which can be disabled by software means, is called a maskable interrupt. Thus an interrupt which cannot be masked is an unmaskable interrupt**.** Various interrupts can be enabled or disabled selectively by masking. There is an interrupt mask register in the interrupt control system Various bits of this register are used for enabling or disabling of interrupts. The first 3 bits—bit 0, bit 1, and bit 2—are devoted to interrupt masks for RST 5.5, RST 6.5 and RST 7.5 interrupts respectively. If any of these bits is set to 1, the particular interrupt is disabled, otherwise it is enabled. Bit 3 is the overriding bit for bit 0, bit 1, and bit 2. This bit is called the Mask Set Enable (MSE). Only when this bit is set to 1, the mask bits for interrupts RST 5.5, RST 6.5, and RST 7.5 are taken into consideration. Bit 4 is used to reset the RST 7.5 latch. When this bit is set to 1, the latch associated. with the RST 7.5 interrupt is reset regardless of whether RST 7.5 is masked or not. Bit 5 is not used. Bit 6 and bit 7 are devoted to serial output; these bits are discussed under Section 3.2.7 on Serial Input– Output. All the interrupts can also be reset/disabled through the signal at $RES\acute{E}T \in ¿¿$ the pin.

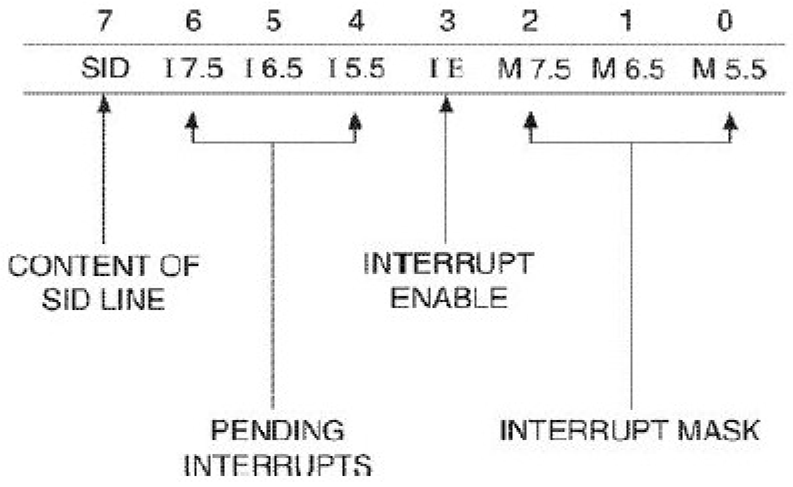


Interrupt mask register.

To mask any interrupt, the user will set the bit pattern in accumulator (ACC) according to his need and execute the SIM (Set Interrupt Mask) instruction. As an example, if we want to enable RST 7.5 and RST 5.5 and disable RST 6.5, the bit pattern will be

0 0 0 1 1 0 1 0 = 1AH

The user will transfer 1AH to ACC and execute the SIM instruction. The user will often wish to know about the status of the various interrupts, i.e. whether they are enabled and whether any interrupt is pending on these lines. This is facilitated by the RIM (Read Interrupt Mask) instruction. This instruction when executed, loads the accumulatorwith a byte—the structure of which is shown inbelow Figure. Bit 0, bit 1, and bit 2 of this byte indicate interrupt masks set for interrupts RST 5.5, RST 6.5, and RST 7.5 respectively. Bit 3 shows whether the interrupt system is enabled or disabled. The next three bits indicate whether any interrupt is pending on RST 5.5, RST 6.5, and RST 7.5 respectively. Bit7 is reserved for serial input data. The user will examine the status of various interrupts and then take appropriate decision to mask or unmask any interrupt.



Microprocessor and Microcontroller

LECTURE NOTES

Department of Electronics and Communication Engineering

BY: ANURAG SETHY (LECTURER)

UGMIT RAYAGADA

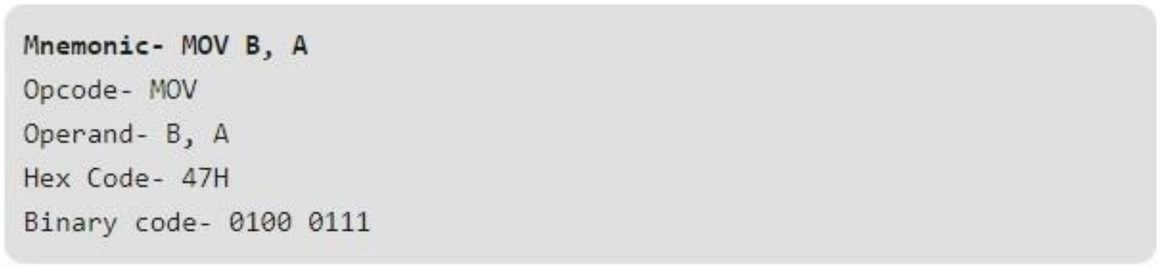
UNIT-2

Instruction Word Size in Microprocessor

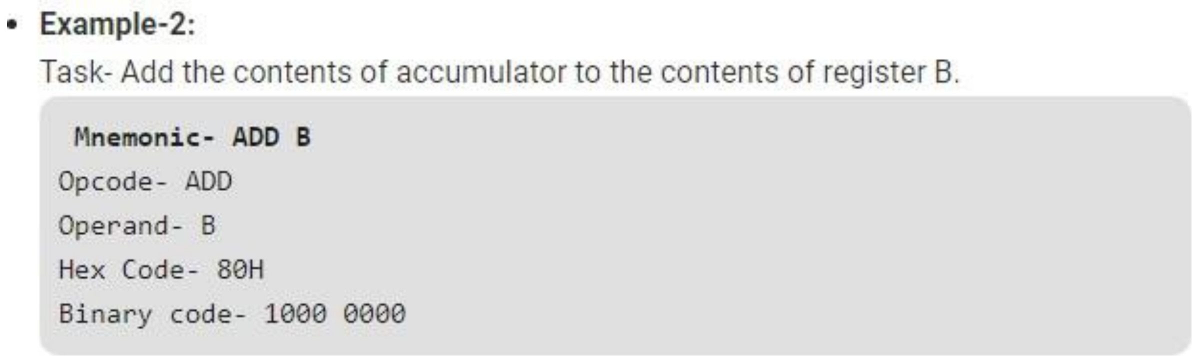
The 8085 instruction set is classified into 3 categories by considering the length of the instructions. In 8085, the length is measured in terms of “byte” rather then “word” because 8085 microprocessor has 8-bit data bus. Three types of instruction are: 1-byte instruction, 2-byte instruction, and 3-byte instruction.

1. **One-byte instructions:**
   In 1-byte instruction, the opcode and the operand of an instruction are represented in one byte.

   Example-1



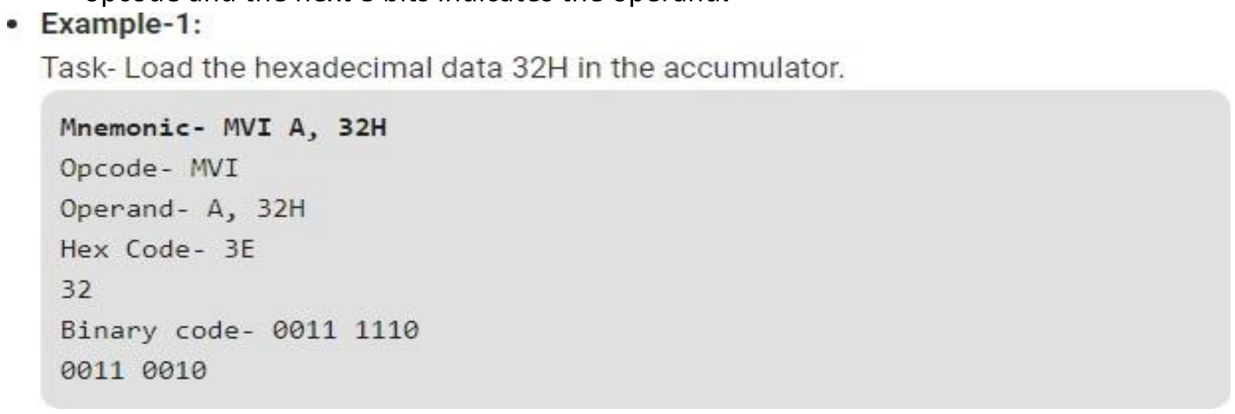
```
Mnemonic- MOV B, A
Opcode- MOV
Operand- B, A
Hex Code- 47H
Binary code- 0100 0111
```



- **Example-2:**
  Task- Add the contents of accumulator to the contents of register B.

```
 Mnemonic- ADD B
Opcode- ADD
Operand- B
Hex Code- 80H
Binary code- 1000 0000
```

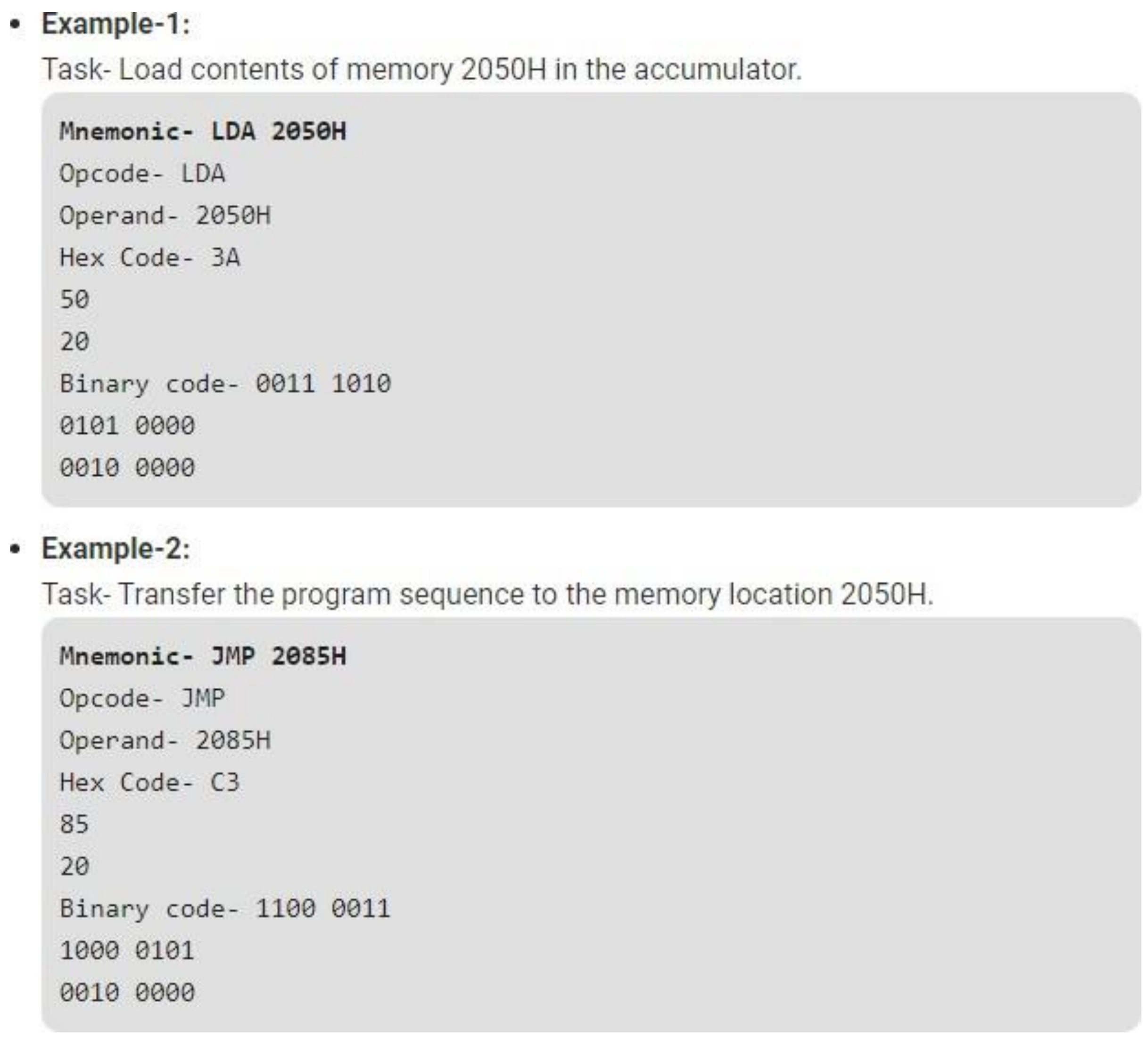
2. **Two-byte instructions**
   Two-byte instruction is the type of instruction in which the first 8 bits indicates the opcode and the next 8 bits indicates the operand.



- **Example-1:**
  Task- Load the hexadecimal data 32H in the accumulator.

```
Mnemonic- MVI A, 32H
Opcode- MVI
Operand- A, 32H
Hex Code- 3E
32
Binary code- 0011 1110
0011 0010
```

3. **Three-byte instructions**
   Three-byte instruction is the type of instruction in which the first 8 bits indicates the opcode and the next two bytes specify the 16-bit address. The low-order address is represented in second byte and the high-order address is represented in the third byte.



- **Example-1:**
  Task- Load contents of memory 2050H in the accumulator.

```
Mnemonic- LDA 2050H
Opcode- LDA
Operand- 2050H
Hex Code- 3A
50
20
Binary code- 0011 1010
0101 0000
0010 0000
```

- **Example-2:**
  Task- Transfer the program sequence to the memory location 2050H.

```
Mnemonic- JMP 2085H
Opcode- JMP
Operand- 2085H
Hex Code- C3
85
20
Binary code- 1100 0011
1000 0101
0010 0000
```

8085 Addressing Modes

The process of specifying the data to be operated on by the instruction is called addressing. The various formats for specifying operands are called addressing modes. The 8085 has the following five types of addressing:

I. Immediate addressing
II. Memory direct addressing
III. Register direct addressing
IV. Indirect addressing
V. Implicit addressing

Immediate Addressing:
In this mode, the operand given in the instruction - a byte or word – transfers to the destination register or memory location.
Ex: MVI A, 9AH
The operand is a part of the instruction.
The operand is stored in the register mentioned in the instruction.

Memory Direct Addressing:
Memory direct addressing moves a byte or word between a memory location and register. The memory location address is given in the instruction.
Ex: LDA 850FH
This instruction is used to load the content of memory address 850FH in the accumulator.

Register Direct Addressing:
Register direct addressing transfer a copy of a byte or word from source register to destination register.
Ex: MOV B, C
It copies the content of register C to register B.
Indirect Addressing:
Indirect addressing transfers a byte or word between a register and a memory location.
Ex: MOV A, M
Here the data is in the memory location pointed to by the contents of HL pair. The data is moved to the accumulator

Implicit Addressing
In this addressing mode the data itself specifies the data to be operated upon.
Ex: CMA
The instruction complements the content of the accumulator. No specific data or operand is mentioned in the instruction

8085 Instruction Set

DATA TRANSFER INSTRUCTIONS

Opcode	Operand	Description
MOV	Rd, Rs M, Rs Rd, M	This instruction copies the contents of the source register into the destination register; the contents of the source register are not altered. If one of the operands is a memory location, its location is specified by the contents of the HL registers Example: MOV B, C or MOV B,M
MVI	Rd, data M, data	The 8-bit data is stored in the destination register or memory. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: MVI B, 57H or MVI M, 57H
LDA	16-bit address	The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered. Example: LDA 2034H
LDAX	B/D Reg. pair	The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory

		location are not altered. Example: LDAX B
LXI	Reg. pair, 16-bit data	The instruction loads 16-bit data in the register pair designated in the operand. Example: LXI H, 2034H or LXI H, XYZ
LHLD	16-bit address	The instruction copies the contents of the memory location pointed out by the 16-bit address into register L and copies the contents of the next memory location into register H. The contents of source memory locations are not altered. Example: LHLD 2040H
STA	16-bit address	The contents of the accumulator are copied into the memory location specified by the operand. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. Example: STA 4350H
SHLD	16-bit address	The contents of register L are stored into the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand. The contents of registers HL are not altered. This is a 3-byte instruction, the second byte specifies the low-order address

		and the third byte specifies the high-order address. Example: SHLD 2470H
STAX	Reg. pair	The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered. Example: STAX B
XCHG		The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E. Example: XCHG
SPHL		The instruction loads the contents of the H and L registers into the stack pointer register, the contents of the H register provide the high-order address and the contents of the L register provide the low-order address. The contents of the H and L registers are not altered. Example: SPHL
XTHL		The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register. The contents of the H register are exchanged with the next stack location (SP+1); however, the contents of the stack pointer register are not altered. Example: XTHL
PUSH	Reg. pair	The contents of the register pair designated in the operand are

		copied onto the stack in the following sequence. The stack pointer register is decremented and the contents of the highorder register (B, D, H, A) are copied into that location. The stack pointer register is decremented again and the contents of the low-order register (C, E, L, flags) are copied to that location. Example: PUSH B or PUSH A
POP	POP Reg. pair	The contents of the memory location pointed out by the stack pointer register are copied to the low-order register (C, E, L, status flags) of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand. The stack pointer register is again incremented by 1. Example: POP H or POP A
OUT	8-bit port address	The contents of the accumulator are copied into the I/O port specified by the operand. Example: OUT F8H

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

ARITHMETIC INSTRUCTIONS

Opcode	Operand	Description
Add register or memory to accumulator		
ADD	R M	The contents of the operand (register or memory) are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. Example: ADD B or ADD M
Add register to accumulator with carry		
ADC	R M	The contents of the operand (register or memory) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. Example: ADC B or ADC M
Add immediate to accumulator		
ADI	8-bit data	The 8-bit data (operand) is added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ADI 45H
Add immediate to accumulator with carry		
ACI	8-bit data	The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ACI 45H

Add register pair to H and L registers

DAD	Reg. pair	The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register. The contents of the source register pair are not altered. If the result is larger than 16 bits, the CY flag is set. No other flags are affected. Example: DAD H

Subtract register or memory from accumulator

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

Subtract source and borrow from accumulator

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

Subtract immediate from accumulator

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

Subtract immediate from accumulator with borrow

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

Increment register or memory by 1

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

Increment register pair by 1

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

Decrement register or memory by 1

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

Decrement register pair by 1

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

Decimal adjust accumulator

DAA	none	The contents of the accumulator are changed from a binary value to two 4-bit binary coded decimal (BCD) digits. This is the only instruction that uses the auxiliary flag to perform the binary to BCD conversion, and the conversion procedure is described below. S, Z, AC, P, CY flags are altered to reflect the results of the operation. If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits. If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits. Example: DAA

BRANCHING INSTRUCTIONS

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

Jump conditionally

Operand: 16-bit address

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

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

Unconditional subroutine call

CALL 16-bit address

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

Call conditionally

Operand: 16-bit address

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

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

Return from subroutine unconditionally

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

Return from subroutine conditionally

Operand: none

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

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

Load program counter with HL contents		
PCHL	none	The contents of registers H and L are copied into the program counter. The contents of H are placed as the high-order byte and the contents of L as the low-order byte. Example: PCHL
Restart		
RST	0-7	The RST instruction is equivalent to a 1-byte call instruction to one of eight memory locations depending upon the number. The instructions are generally used in conjunction with interrupts and inserted using external hardware. However these can be used as software instructions in a program to transfer program execution to one of the eight locations. The addresses are:

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

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

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

LOGICAL INSTRUCTIONS

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

Exclusive OR immediate with accumulator

XRI	8-bit data	The contents of the accumulator are Exclusive ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: XRI 86H

Logical OR register or memory with accumulaotr

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

Logical OR immediate with accumulator

ORI	8-bit data	The contents of the accumulator are logically ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. Example: ORI 86H

Rotate accumulator left

RLC	none	Each binary bit of the accumulator is rotated left by one position. Bit D_7 is placed in the position of D_0 as well as in the Carry flag. CY is modified according to bit D_7. S, Z, P, AC are not affected. Example: RLC

Rotate accumulator right

RRC	none	Each binary bit of the accumulator is rotated right by one position. Bit D_0 is placed in the position of D_7 as well as in the Carry flag. CY is modified according to bit D_0. S, Z, P, AC are not affected. Example: RRC

Rotate accumulator left through carry

RAL	none	Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D_7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D_0. CY is modified according to bit D_7. S, Z, P, AC are not affected. Example: RAL

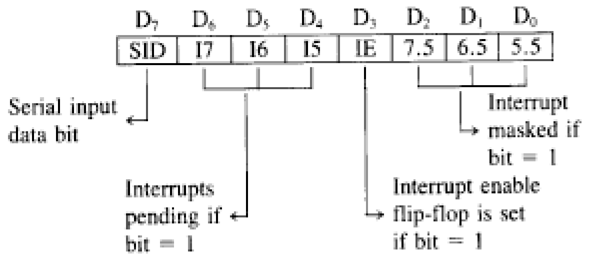
Rotate accumulator right through carry		
RAR	none	Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D_0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D_7. CY is modified according to bit D_0. S, Z, P, AC are not affected. Example: RAR
Complement accumulator		
CMA	none	The contents of the accumulator are complemented. No flags are affected. Example: CMA
Complement carry		
CMC	none	The Carry flag is complemented. No other flags are affected. Example: CMC
Set Carry		
STC	none	The Carry flag is set to 1. No other flags are affected. Example: STC

CONTROL INSTRUCTIONS

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

Read interrupt mask
RIM none

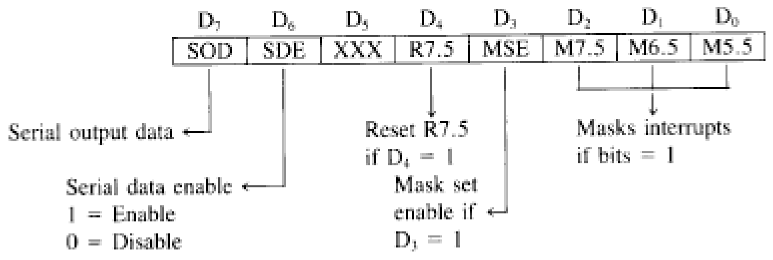
This is a multipurpose instruction used to read the status of interrupts 7.5, 6.5, 5.5 and read serial data input bit. The instruction loads eight bits in the accumulator with the following interpretations.
Example: RIM



D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
SID I7 I6 I5 IE 7.5 6.5 5.5
Serial input data bit
Interrupts pending if bit = 1
Interrupt enable flip-flop is set if bit = 1
Interrupt masked if bit = 1

Set interrupt mask
SIM none

This is a multipurpose instruction and used to implement the 8085 interrupts 7.5, 6.5, 5.5, and serial data output. The instruction interprets the accumulator contents as follows.
Example: SIM



D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
SOD SDE XXX R7.5 MSE M7.5 M6.5 M5.5
Serial output data
Serial data enable 1 = Enable 0 = Disable
Reset R7.5 if D_4 = 1
Mask set enable if D_3 = 1
Masks interrupts if bits = 1

- ☐ SOD — Serial Output Data: Bit D_7 of the accumulator is latched into the SOD output line and made available to a serial peripheral if bit $D_6 = 1$.
- ☐ SDE — Serial Data Enable: If this bit = 1, it enables the serial output. To implement serial output, this bit needs to be enabled.
- ☐ XXX — Don't Care
- ☐ R7.5 — Reset RST 7.5: If this bit = 1, RST 7.5 flip-flop is reset. This is an additional control to reset RST 7.5.
- ☐ MSE — Mask Set Enable: If this bit is high, it enables the functions of bits D_2, D_1, D_0. This is a master control over all the interrupt masking bits. If this bit is low, bits D_2, D_1, and D_0 do not have any effect on the masks.
- ☐ M7.5 — D_2 = 0, RST 7.5 is enabled.
  = 1, RST 7.5 is masked or disabled.
- ☐ M6.5 — D_1 = 0, RST 6.5 is enabled.
  = 1, RST 6.5 is masked or disabled.
- ☐ M5.5 — D_0 = 0, RST 5.5 is enabled.
  = 1, RST 5.5 is masked or disabled.

UNIT-3

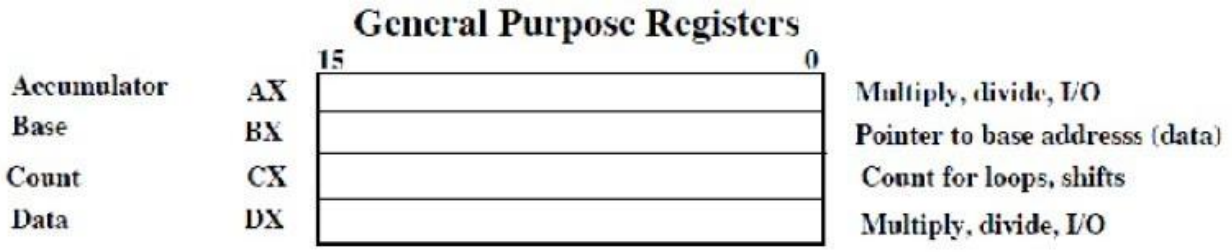
Unit-5

8086 microprocessor

Register Organization of 8086:

General purpose registers: The 8086 microprocessor has a total of fourteen registers that are accessible to the programmer. It is divided into four groups. They are:

- Four General purpose registers
- Four Index/Pointer registers
- Four Segment registers
- Two Other register



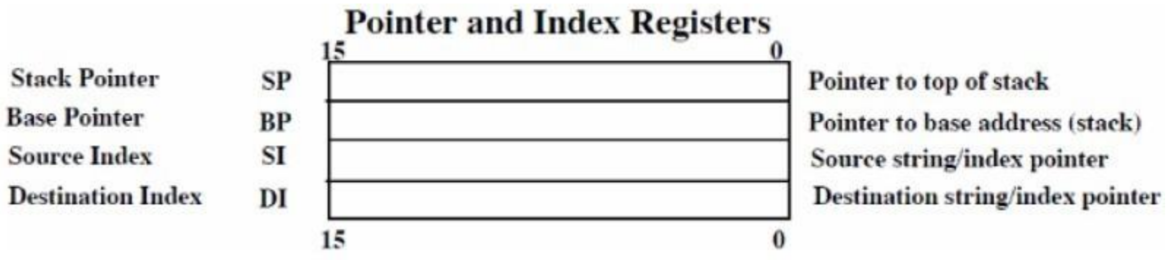
Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation.

Base register consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.

Count register consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low order byte of the word, and CH contains the high-order byte. Count register can be used in Loop, shift/rotate instructions and as a counter in string manipulation.

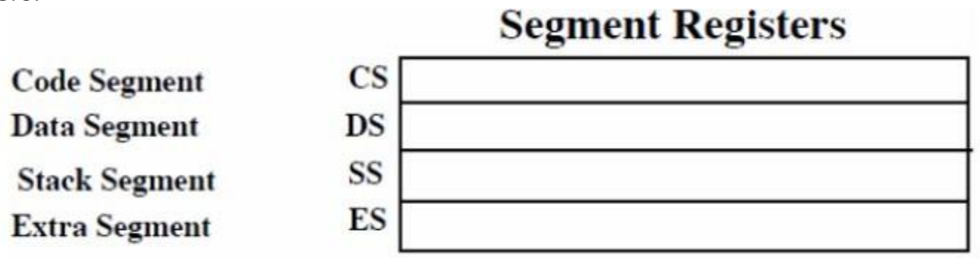
Data register consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX. When combined, DL register contains the low order byte of the word, and DH contains the high-order byte. Data register can be used as a port number in I/O operations. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

Index or Pointer Registers
These registers can also be called as Special Purpose registers.


Pointer and Index Registers
15
0
Stack Pointer
SP
Pointer to top of stack
Base Pointer
BP
Pointer to base address (stack)
Source Index
SI
Source string/index pointer
Destination Index
DI
Destination string/index pointer
15
0

Stack Pointer (SP) is a 16-bit register pointing to program stack, i.e. it is used to hold the address of the top of stack. The stack is maintained as a LIFO with its bottom at the start of the stack segment (specified by the SS segment register).Unlike the SP register, the BP can be used to specify the offset of other program segments. Base Pointer (BP) is a 16-bit register pointing to data in stack segment. It is usually used by subroutines to locate variables that were passed on the stack by a calling program. 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 address in string manipulation instructions. Used in conjunction with the DS register to point to data locations in the data segment. Destination Index (DI) is a 16-bit register. Used in conjunction with the ES register in string operations. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions. In short, Destination Index and SI Source Index registers are used to hold address.

Segment Registers
Most of the registers contain data/instruction offsets within 64 KB memory segment. There are four different 64 KB segments for instructions, stack, data and extra data. To specify where in 1 MB of processor memory these 4 segments are located the processor uses four segment registers.

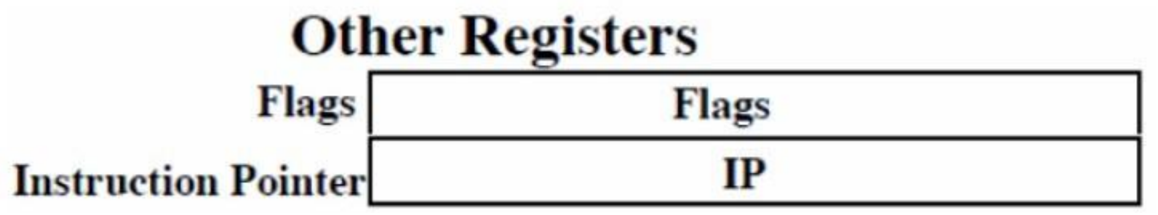


Code segment (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions.

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.

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.

Extra segment (ES) used to hold the starting address of Extra segment. Extra segment is provided for programs that need to access a second data segment. Segment registers cannot be used in arithmetic operations.



Instruction Pointer (IP) is a 16-bit register. This is a crucially important register which is used to control which instruction the CPU executes. The IP, or program counter, is used to store the memory location of the next instruction to be executed. The CPU checks the program counter to ascertain which instruction to carry out next. It then updates the program counter to point to the next instruction. Thus the program counter will always point to the next instruction to be executed.

Flag Register contains a group of status bits called flags that indicate the status of the CPU or the result of arithmetic operations. There are two types of flags:

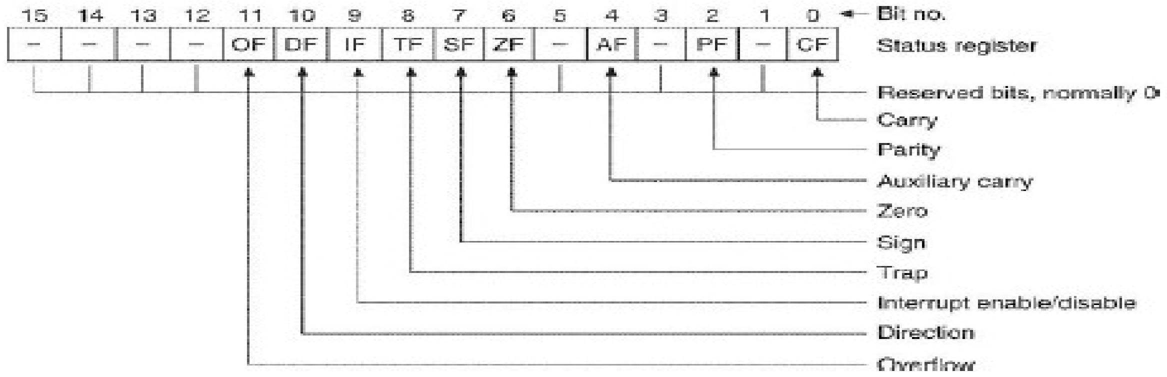
1. The status flags which reflect the result of executing an instruction. The programmer cannot set/reset these flags directly.
2. The control flags enable or disable certain CPU operations. The programmer can set/reset these bits to control the CPU's operation.

Nine individual bits of the status register are used as control flags (3 of them) and status flags (6 of them).The remaining 7 are not used.
A flag can only take on the values 0 and 1. We say a flag is set if it has the value 1.The status flags are used to record specific characteristics of arithmetic and of logical instructions.

Control Flags: There are three control flags

1. The Direction Flag (D): Affects the direction of moving data blocks by such instructions as MOVS, CMPS and SCAS. The flag values are 0 = up and 1 = down and can be set/reset by the STD (set D) and CLD (clear D) instructions.

2. The Interrupt Flag (I): Dictates whether or not system interrupts can occur. Interrupts are actions initiated by hardware block such as input devices that will interrupt the normal execution of programs. The flag values are 0 = disable interrupts or 1 = enable interrupts and can be manipulated by the CLI (clear I) and STI (set I) instructions.
3. The Trap Flag (T): Determines whether or not the CPU is halted after the execution of each instruction. When this flag is set (i.e. = 1), the programmer can single step through his program to debug any errors. When this flag = 0 this feature is off. This flag can be set by the INT 3 instruction.

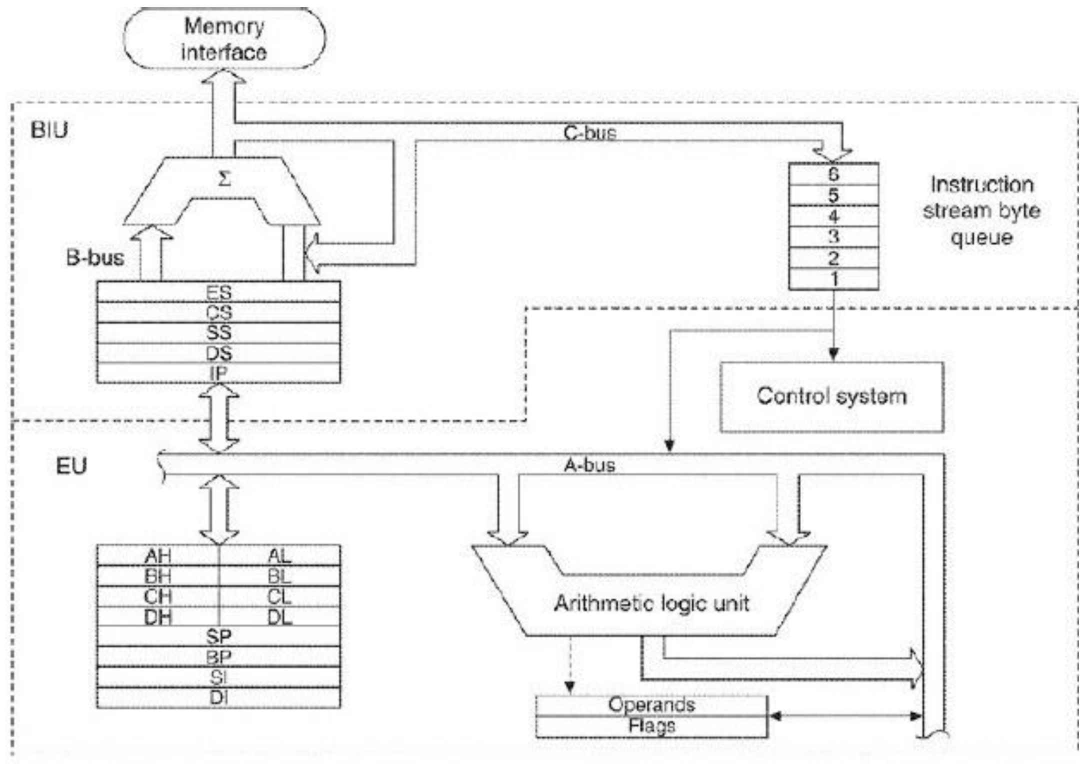


Status Flags: There are six status flags
1. The Carry Flag (C): This flag is set when the result of an unsigned arithmetic operation is too large to fit in the destination register. This happens when there is an end carry in an addition operation or there an end borrows in a subtraction operation. A value of 1 = carry and 0 = no carry.
2. The Overflow Flag (O): This flag is set when the result of a signed arithmetic operation is too large to fit in the destination register (i.e. when an overflow occurs). Overflow can occur when adding two numbers with the same sign (i.e. both positive or both negative). A value of 1 = overflow and 0 = no overflow.

3. The Sign Flag (S): This flag is set when the result of an arithmetic or logic operation is negative. This flag is a copy of the MSB of the result (i.e. the sign bit). A value of 1 means negative and 0 = positive.
4. The Zero Flag (Z): This flag is set when the result of an arithmetic or logic operation is equal to zero. A value of 1 means the result is zero and a value of 0 means the result is not zero.
5. The Auxiliary Carry Flag (A): This flag is set when an operation causes a carry from bit 3 to bit 4 (or a borrow from bit 4 to bit 3) of an operand. A value of 1 = carry and 0 = no carry.
6. The Parity Flag (P): This flags reflects the number of 1s in the result of an operation. If the number of 1s is even its value = 1 and if the number of 1s is odd then its value = 0.

Architecture of 8086:

8086 has two blocks Bus Interface Unit (BIU) and Execution Unit (EU).
The BIU performs all bus operations such as instruction fetching, reading and writing operands for memory and calculating the addresses of the memory operands. The instruction bytes are transferred to the instruction queue.
EU executes instructions from the instruction system byte queue.
Both units operate asynchronously to give the 8086 an overlapping instruction fetch and execution mechanism which is called as Pipelining. This results in efficient use of the system bus and system performance.
BIU contains Instruction queue, Segment registers, Instruction pointer, Address adder.
EU contains Control circuitry, Instruction decoder, ALU, Pointer and Index register, Flag register.



Intel 8086 internal architecture.

Bus Interface Unit:

1. It provides a full 16 bit bidirectional data bus and 20 bit address bus.
2. The bus interface unit is responsible for performing all external bus operations.

Specifically it has the following functions:

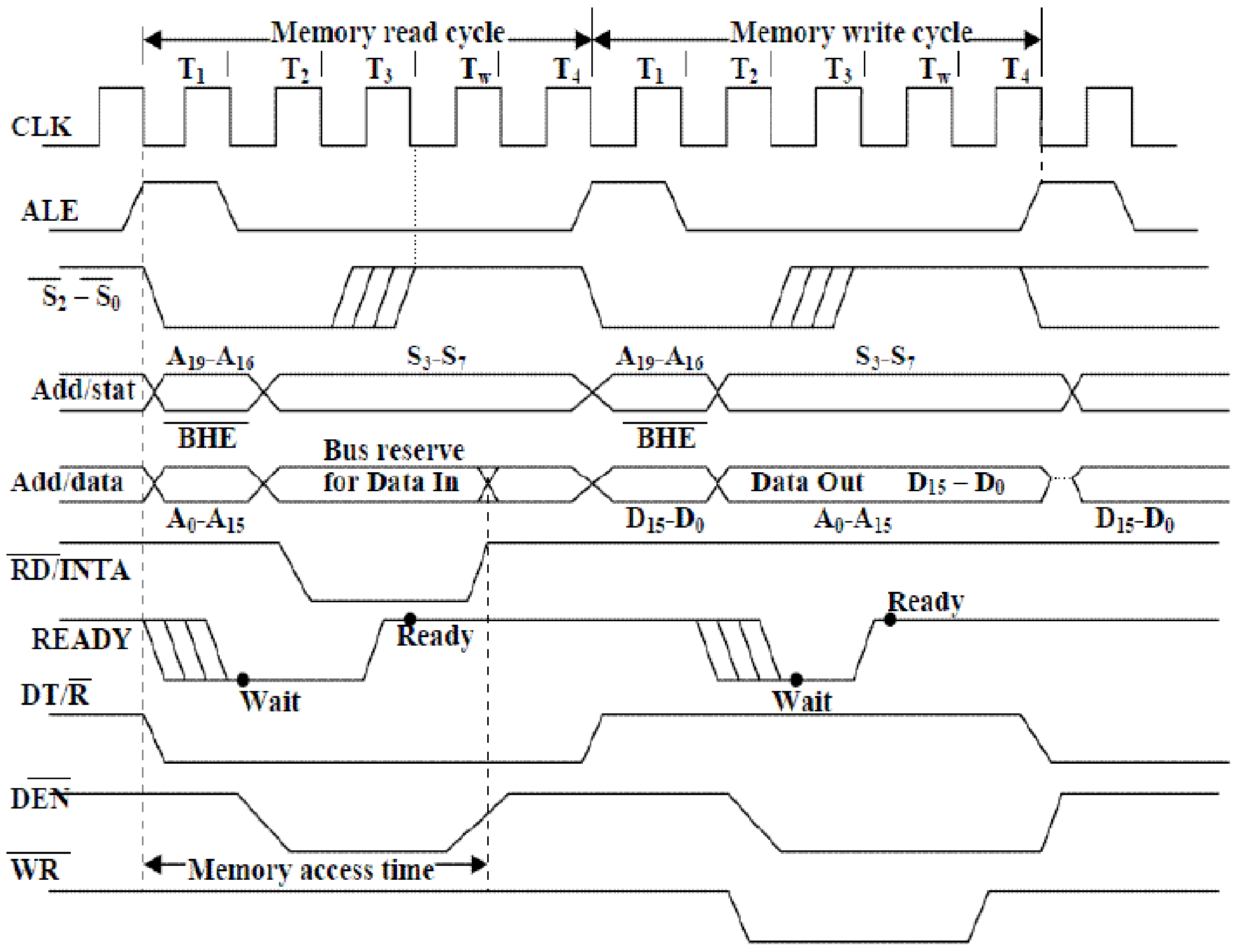
3. Instruction fetch Instruction queuing, Operand fetch and storage, Address relocation and Bus control.
4. The BIU uses a mechanism known as an instruction stream queue to implement pipeline architecture.
5. This queue permits prefetch of up to six bytes of instruction code. When ever the queue of the BIU is not full, it has room for at least two more bytes and at the same time the EU is not requesting it to read or write operands from memory, the BIU is free to look ahead in the program by prefetching the next sequential instruction.
6. These prefetching instructions are held in its FIFO queue. With its 16 bit data bus, the BIU fetches two instruction bytes in a single memory cycle.
7. After a byte is loaded at the input end of the queue, it automatically shifts up through the FIFO to the empty location nearest the output.
8. The EU accesses the queue from the output end. It reads one instruction byte after the other from the output of the queue. If the queue is full and the EU is not requesting access to operand in memory.
9. These intervals of no bus activity, which may occur between bus cycles are known as Idle state.
10. If the BIU is already in the process of fetching an instruction when the EU request it to read or write operands from memory or I/O, the BIU first completes the instruction fetch bus cycle before initiating the operand read / write cycle.
11. The BIU also contains a dedicated adder which is used to generate the 20bit physical address that is output on the address bus. This address is formed by adding an appended 16 bit segment address and a 16 bit offset address.
12. For example: The physical address of the next instruction to be fetched is formed by combining the current contents of the code segment CS register and the current contents of the instruction pointer IP register.
13. The BIU is also responsible for generating bus control signals such as those for memory read or write and I/O read or write.

Execution Unit:

1. The Execution unit is responsible for decoding and executing all instructions.
2. The EU extracts instructions from the top of the queue in the BIU, decodes them, generates operands if necessary, passes them to the BIU and requests it to perform the read or write bus cycles to memory or I/O and perform the operation specified by the instruction on the operands.
3. During the execution of the instruction, the EU tests the status and control flags and updates them based on the results of executing the instruction.
4. If the queue is empty, the EU waits for the next instruction byte to be fetched and shifted to top of the queue.
5. When the EU executes a branch or jump instruction, it transfers control to a location corresponding to another set of sequential instructions.
6. Whenever this happens, the BIU automatically resets the queue and then begins to fetch instructions from this new location to refill the queue.

General Bus Operation

1. The 8086 has a combined address and data bus commonly referred as a time multiplexed address and data bus.
2. The main reason behind multiplexing address and data over the same pins is the maximum utilization of processor pins and it facilitates the use of 40 pin standard DIP package.
3. The bus can be demultiplexed using a few latches and transceivers, when ever required.
4. Basically, all the processor bus cycles consist of at least four clock cycles. These are referred to as T1, T2, T3, and T4. The address is transmitted by the processor during T1. It is present on the bus only for one cycle.
5. The negative edge of this ALE pulse is used to separate the address and the data or status information. In maximum mode, the status lines S0, S1 and S2 are used to indicate the type of operation.
6. Status bits S3 to S7 are multiplexed with higher order address bits and the BHE signal. Address is valid during T1 while status bits S3 to S7 are valid during T2 through T4.



Maximum mode:

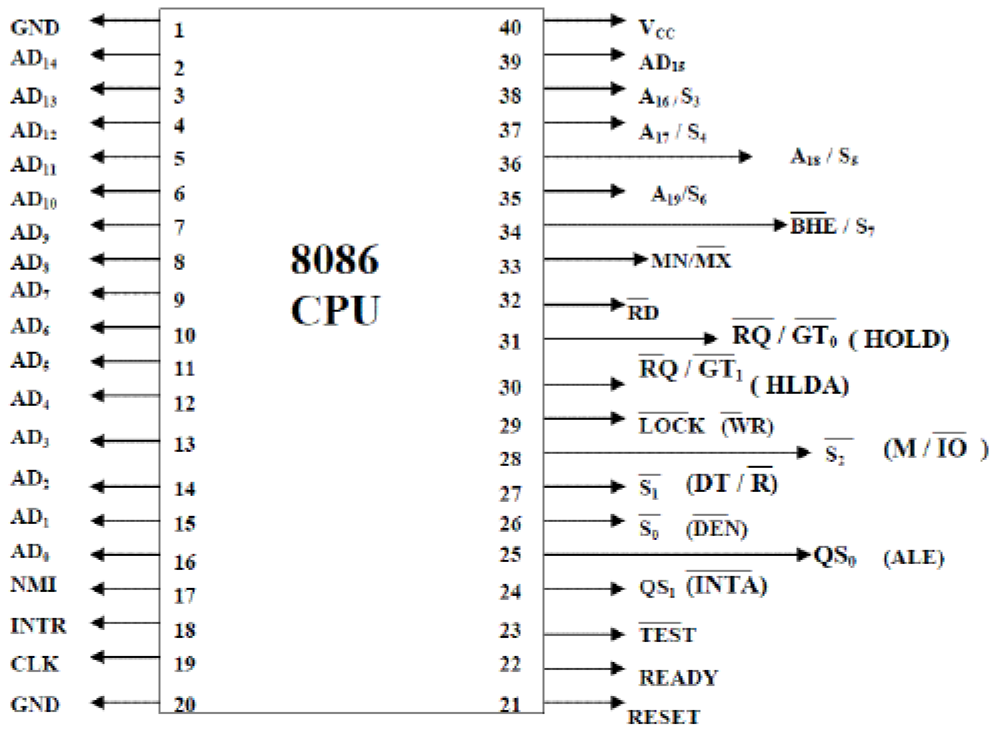
1. In the maximum mode, the 8086 is operated by strapping the MN/MX pin to ground.
2. In this mode, the processor derives the status signal S2, S1, S0. Another chip called bus controller derives the control signal using this status information.
3. In the maximum mode, there may be more than one microprocessor in the system configuration.

Minimum mode:

1. In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX pin to logic 1.
2. In this mode, all the control signals are given out by the microprocessor chip itself.
3. There is a single microprocessor in the minimum mode system.

Pin Diagram of 8086 and Pin description of 8086:

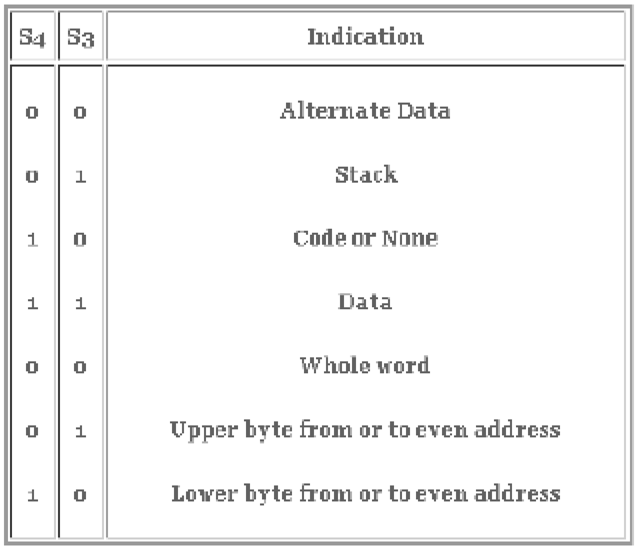
Pin Diagram of 8086



1. The Microprocessor 8086 is a 16-bit CPU available in different clock rates and packaged in a 40 pin CERDIP or plastic package.
2. The 8086 operates in single processor or multiprocessor configuration to achieve high performance. The pins serve a particular function in minimum mode (single processor mode) and other function in maximum mode configuration (multiprocessor mode).
3. The 8086 signals can be categorized in three groups.
   - The first are the signal having common functions in minimum as well as maximum mode.
   - The second are the signals which have special functions for minimum mode
   - The third are the signals having special functions for maximum mode.
4. The following signal descriptions are common for both 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. These lines are active high and float to a tristate during interrupt acknowledge and local bus hold acknowledge cycles.
5. **A19/S6, A18/S5, A17/S4, and A16/S3**: These are the time multiplexed address and status lines.
   - During T1 these are the most significant address lines for 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.

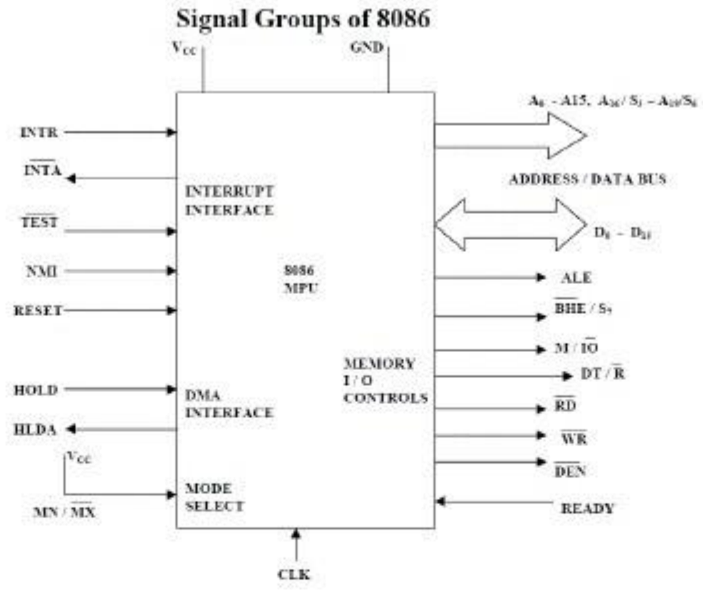
- The status of the interrupt enable flag bit is updated at the beginning of each clock cycle.
- The S4 and S3 combine indicate which segment registers is presently being used for memory accesses as in below figure
- These lines float to tri-state off during the local bus hold acknowledge.
- The status line S6 is always low. The address bit is separated from the status bit using latches controlled by the ALE signal.



S_4	S_3	Indication
0	0	Alternate Data
0	1	Stack
1	0	Code or None
1	1	Data
0	0	Whole word
0	1	Upper byte from or to even address
1	0	Lower byte from or to even address

6. BHE/S7: The bus high enable is used to indicate the transfer of data over the higher order (D15-D8) data bus as shown in table. It goes low for the data transfer over D15-D8 and is used to derive chip selects of odd address memory bank or peripherals. BHE is low during T1 for read, write and interrupt acknowledge cycles, whenever a byte is to be transferred on higher byte of data bus. The status information is available during T2, T3 and T4. The signal is active low and tristated during hold. It is low during T1 for the first pulse of the interrupt acknowledge cycle.
7. RD – Read: This signal on low indicates the peripheral that the processor is performing memory or I/O read operation. RD is active low and shows the state for T2, T3, Tw of any read cycle. The signal remains tristated during the hold acknowledge.
8. READY: This is the acknowledgement from the slow device 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.
9. INTR-Interrupt Request: This is a triggered input. This is sampled during the last clock cycles 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 resulting the interrupt enable flag. This signal is active high and internally synchronized.
10. TEST: This input is examined by a 'WAIT' instruction. If the TEST pin 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.

11. CLK- Clock Input: The clock input provides the basic timing for processor operation and bus control activity. It's an asymmetric square wave with 33% duty cycle.



The following pin functions are for the minimum mode operation of 8086.

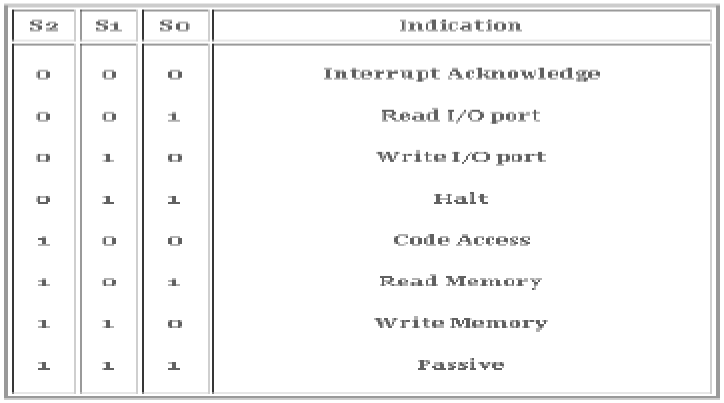
1. M/IO – Memory/IO: This is a status line logically equivalent to S2 in maximum mode. When it is low, it indicates the CPU is having an I/O operation, and when it is high, it indicates that the CPU is having a memory operation. This line becomes active high in the previous T4 and remains active till final T4 of the current cycle. It is tristated during local bus "hold acknowledge ".
2. INTA – Interrupt Acknowledge: This signal is used as a read strobe for interrupt acknowledge cycles. i.e. when it goes low, the processor has accepted the interrupt.
3. 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.
4. DT/R – Data Transmit/Receive: This output is used to decide the direction of data flow through the transceivers (bidirectional buffers). When the processor sends out data, this signal is high and when the processor is receiving data, this signal is low.
5.
6. DEN – Data Enable: This signal indicates the availability of valid data over the address/data lines. It is used to enable the transceivers (bidirectional buffers) to separate the data from the multiplexed address/data signal. It is active from the middle of T2 until the middle of T4. This is tristated during 'hold acknowledge' cycle.
7. HOLD, HLDA- 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 cycle.
8. 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 is should be externally synchronized. If the DMA request is made while the CPU is performing a memory or I/O cycle, it will release the local bus during T4 provided

- The request occurs on or before T2 state of the current cycle.
- The current cycle is not operating over the lower byte of a word.
- The current cycle is not the first acknowledge of an interrupt acknowledge sequence.
- A Lock instruction is not being executed.

The following pin functions are applicable for maximum mode operation of 8086:

1. S2, S1, and S0 – Status Lines: These are the status lines which reflect the type of operation, being carried out by the processor. These become activity during T4 of the previous cycle and active during T1 and T2 of the current bus cycles.
2. LOCK: This output pin indicates that other system bus master will be prevented from gaining the system bus, while the LOCK signal is low. The LOCK signal is activated by the 'LOCK' prefix instruction and remains active until the completion of the next instruction. 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.
3. The 8086, while executing the prefixed instruction, asserts the bus lock signal output, which may be connected to an external bus controller. By prefetching the instruction, there is a considerable speeding up in instruction execution in 8086. This is known as instruction pipelining.



S2	S1	S0	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

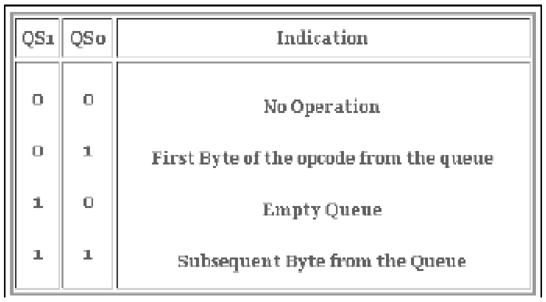
At the starting the CS: IP is loaded with the required address from which the execution is to be started. Initially, the queue will be empty and the microprocessor starts a fetch operation to bring one byte (the first byte) of instruction code, if the CS: IP address is odd or two bytes at a time, if the CS: IP address is even. The first byte is a complete opcode in case of some instruction (one byte opcode instruction.

The first byte is a complete opcode in case of some instruction (one byte opcode instruction) and is a part of opcode, in case of some instructions (two byte opcode instructions), the remaining part of code lie in second byte.

The second byte is then decoded in continuation with the first byte to decide the instruction length and the number of subsequent bytes to be treated as instruction data. The queue is updated after every byte is read from the queue but the fetch cycle is initiated by BIU only if at least two bytes of the queue are empty and the EU may be concurrently executing the fetched instructions.

The next byte after the instruction is completed is again the first opcode byte of the next instruction. A similar procedure is repeated till the complete execution of the program. The fetch operation of the next instruction is overlapped with the execution of the current instruction. As in the architecture, there are two separate units, namely Execution unit and Bus interface unit.

While the execution unit is busy in executing an instruction, after it is completely decoded, the bus interface unit may be fetching the bytes of the next instruction from memory, depending upon the queue status.



QS_1	QS_0	Indication
0	0	No Operation
0	1	First Byte of the opcode from the queue
1	0	Empty Queue
1	1	Subsequent Byte from the Queue

RQ/GT0, RQ/GT1 – Request/Grant: These pins are used by the other local bus master in maximum mode, to force the processor to release the local bus at the end of the processor current bus cycle.

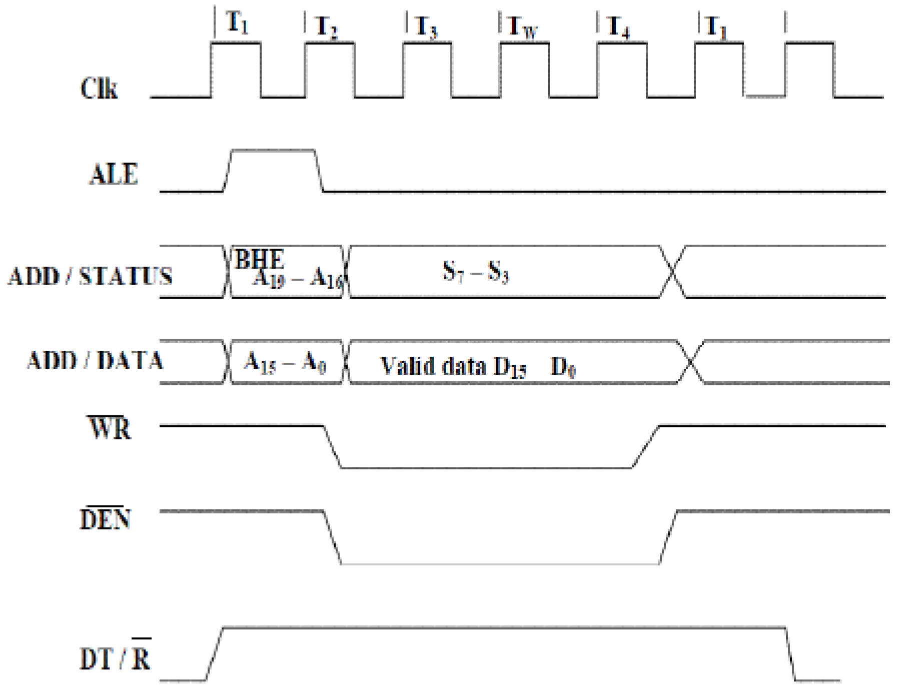
Each of the pin is bidirectional with RQ/GT0 having higher priority than RQ/GT1. RQ/GT pins have internal pull-up resistors and may be left unconnected. Request/Grant sequence is as follows.

1. A pulse of 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 cycle. The CPU bus interface unit is likely to be disconnected from the local bus of the system.
3. A one clock wide pulse from another master indicates to the 8086 that the hold request is about to end and the 8086 may regain control of the local bus at the next clock cycle. Thus each master to master exchange of the local bus is a sequence of 3 pulses. There must be at least one dead clock cycle after each bus exchange. The request and grant pulses are active low. For the bus request those are received while 8086 is performing memory or I/O cycle, the granting of the bus is governed by the rules as in case of HOLD and HLDA in minimum mode.

Write Cycle Timing Diagram for Minimum Mode:
The working of the minimum mode configuration system can be better described in terms of the timing diagrams rather than qualitatively describing the operations.

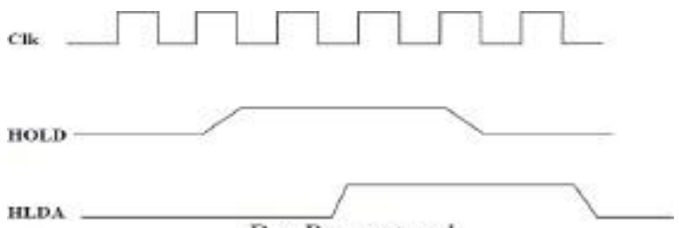
- The opcode fetch and read cycles are similar. Hence the timing diagram can be categorized in two parts, the first is the timing diagram for read cycle and the second is the timing diagram for write cycle.
- The read cycle begins in T1 with the assertion of address latch enable (ALE) signal and also M / IO signal. During the negative going edge of this signal, the valid address is

latched on the local bus.



Write Cycle Timing Diagram for Minimum Mode

- The BHE and A0 signals address low, high or both bytes. From T1 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 address device to enable its data bus drivers. After RD goes low, the valid data is available on the data bus. 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.
- 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 T1, 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 (unlike RD is somewhat delayed in T2 to provide time for floating).
- The BHE and A0 signals are used to select the proper byte or bytes of memory or I/O word to be read or write.



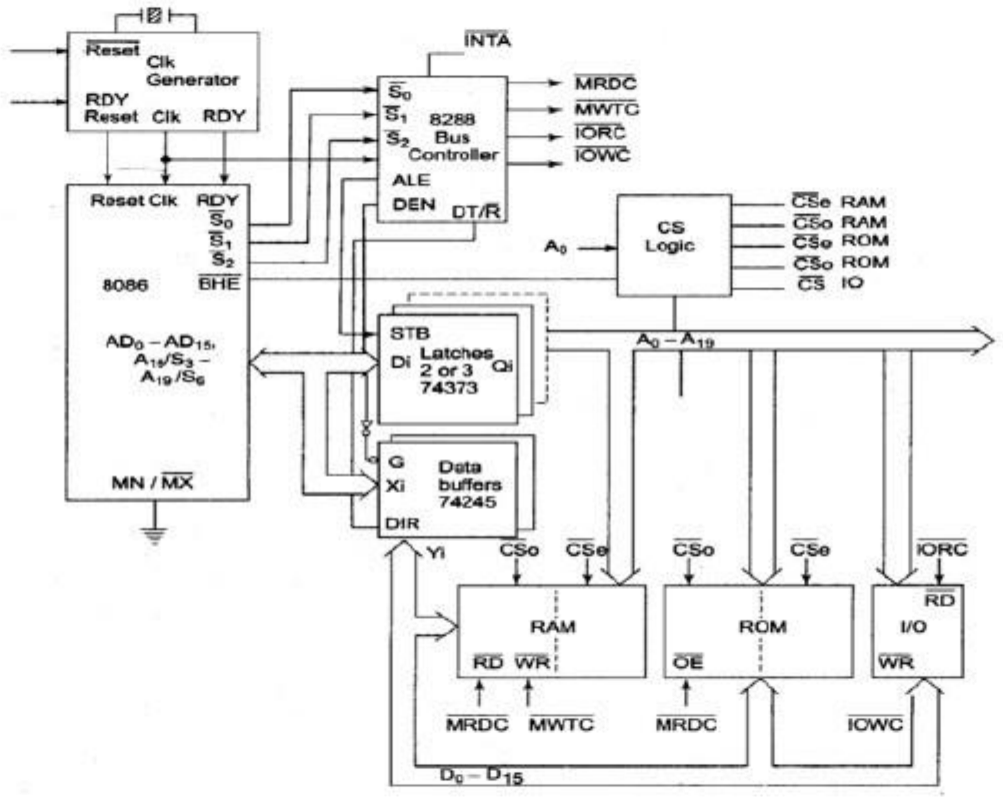
Bus Request and
Bus Grant Timings in Minimum Mode System

- Hold Response sequence: The HOLD pin is checked at leading edge of each clock pulse. If it is received active by the processor before T4 of the previous cycle or during T1 state of the current cycle, the CPU activates HLDA in the next clock cycle and for succeeding bus cycles, the bus will be given to another requesting master.
- The 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.

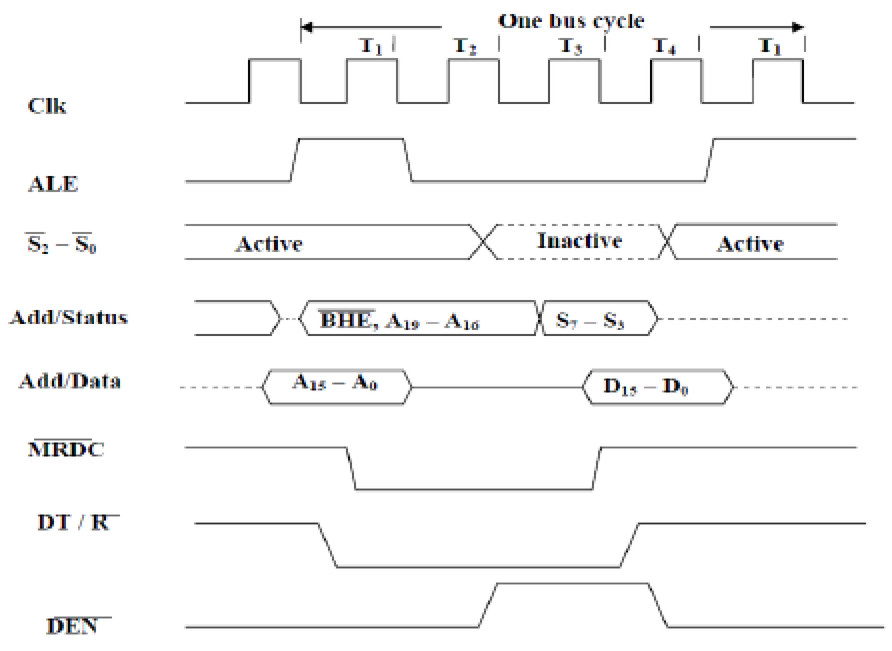
Maximum Mode 8086 System

1. In the maximum mode, the 8086 is operated by strapping the MN/MX pin to ground.
2. In this mode, the processor derives the status signal S2, S1, S0. Another chip called bus controller derives the control signal using this status information.
3. In the maximum mode, there may be more than one microprocessor in the system configuration.
4. The components in the system are same as in the minimum mode system.
5. The basic function 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 by the processor on the status lines.
6. The bus controller chip has input lines S2, S1, S0 and CLK. These inputs to 8288 are driven by CPU.
7. It derives the outputs ALE, DEN, DT/R, MRDC, MWTC, AMWC, IORC, IOWC and AIOWC. The AEN, IOB and CEN pins are especially useful for multiprocessor systems.
8. AEN and IOB are generally grounded. CEN pin is usually tied to +5V. The significance of the MCE/PDEN output depends upon the status of the IOB pin.
9. If IOB is grounded, it acts as master cascade enable to control cascade 8259A, else it acts as peripheral data enable used in the multiple bus configurations.
10. INTA pin used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device.
11. IORC, IOWC are I/O read command and I/O write command signals respectively.
12. These signals enable an IO interface to read or write the data from or to the address port.
13. The MRDC, MWTC are memory read command and memory write command signals respectively and may be used as memory read or write signals.
14. All these command signals instructs the memory to accept or send data from or to the bus.

15. For both of these write command signals, the advanced signals namely AIOWC and AMWTC are available.
16. Here the only difference between in timing diagram between minimum mode and maximum mode is the status signals used and the available control and advanced command signals.
17. R0, S1, S2 are set at the beginning of bus cycle.8288 bus controller will output a pulse as on the ALE and apply a required signal to its DT / R pin during T1.
18. In T2, 8288 will set DEN=1 thus enabling transceivers, and for an input it will activate MRDC or IORC. These signals are activated until T4. For an output, the AMWC or AIOWC is activated from T2 to T4 and MWTC or IOWC is activated from T3 to T4.
19. The status bit S0 to S2 remains active until T3 and become passive during T3 and T4.
20. If reader input is not activated before T3, wait state will be inserted between T3 and T4.

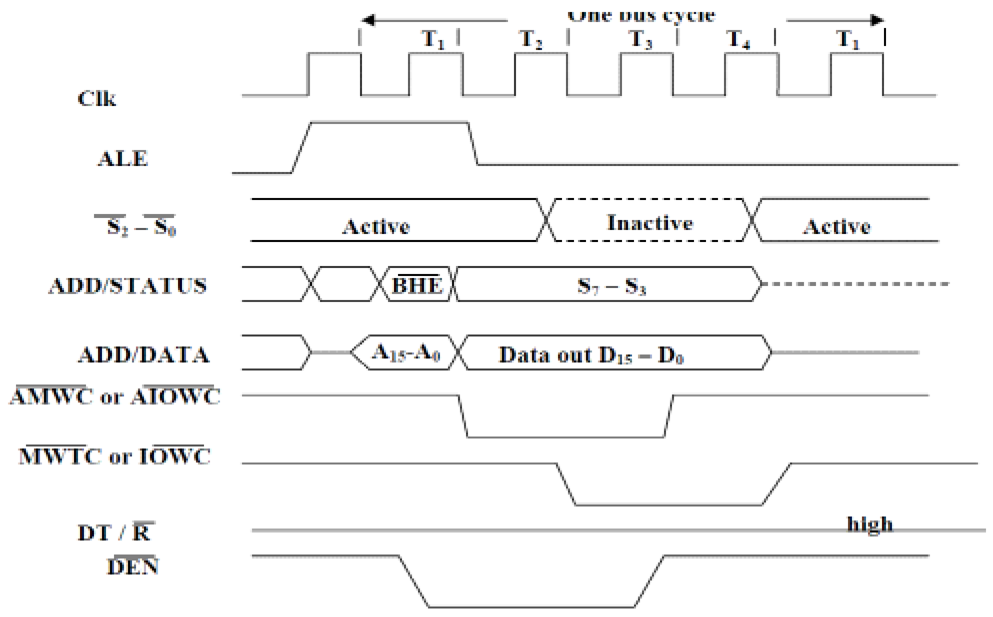


Memory Read Timing Diagram in Maximum Mode of 8086



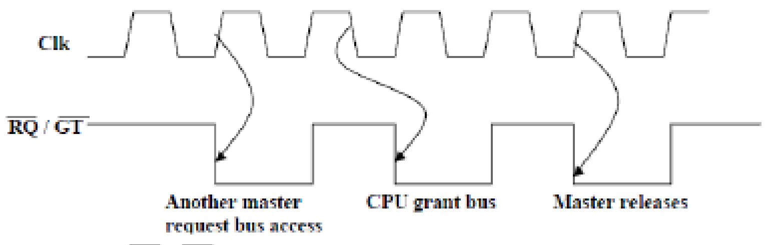
Memory Read Timing in Maximum Mode

Memory Write Timing in Maximum mode of 8086



Memory Write Timing in Maximum mode.

RQ/GT Timings in Maximum Mode:


Clk
$\overline{RQ}$ / $\overline{GT}$
Another master request bus access
CPU grant bus
Master releases

$\overline{RQ}/\overline{GT}$ **Timings in Maximum Mode**

- The request/grant response sequence contains a series of three pulses. The request/grant pins are checked at each rising pulse of clock input.
- When a request is detected and if the condition for HOLD request is satisfied, the processor issues a grant pulse over the RQ/GT pin immediately during T4 (current) or T1 (next) state.
- When the requesting master receives this pulse, it accepts the control of the bus, it sends a release pulse to the processor using RQ/GT pin.

Interrupts:

The meaning of 'interrupts' is to break the sequence of operation. While the CPU is executing a program, on 'interrupt' breaks the normal sequence of execution of instructions, diverts its execution to some other program called Interrupt Service Routine (ISR).After executing ISR , the control is transferred back again to the main program. Interrupt processing is an alternative to polling.

Need for Interrupt: Interrupts are particularly useful when interfacing I/O devices that provide or require data at relatively low data transfer rate.

Types of Interrupts: There are two types of Interrupts in 8086. They are:

(i)Hardware Interrupts

(ii) Software Interrupts

- Hardware Interrupts (External Interrupts). The Intel microprocessors support hardware interrupts through two pins that allow interrupt requests, INTR and NMI.

- One pin that acknowledges, INTA, the interrupt requested on INTR. INTR and NMI

- INTR is a maskable hardware interrupt. The interrupt can be enabled/disabled using STI/CLI instructions or using more complicated method of updating the FLAGS register with the help of the POP instruction.

- When an interrupt occurs, the processor stores FLAGS register into stack, disables further interrupts, fetches from the bus one byte representing interrupt type, and jumps to interrupt processing routine address of which is stored in location 4 * <interrupt type>. Interrupt processing routine should return with the IRET instruction.

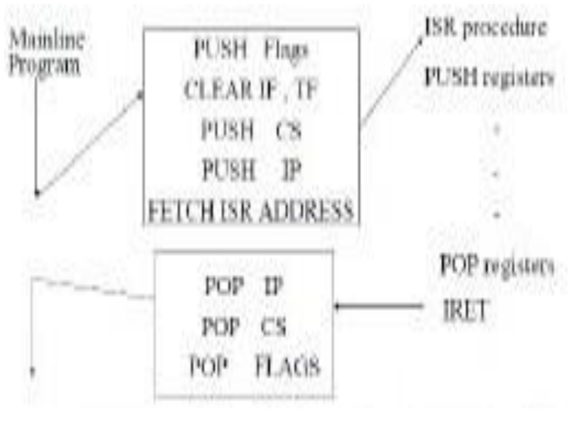
- NMI is a non-maskable interrupt. Interrupt is processed in the same way as the INTR interrupt. Interrupt type of the NMI is 2, i.e. the address of the NMI processing routine is stored in location 0008h. This interrupt has higher priority than the maskable interrupt.

Software Interrupts (Internal Interrupts and Instructions) :
Software interrupts can be caused by:

- INT instruction - breakpoint interrupt.

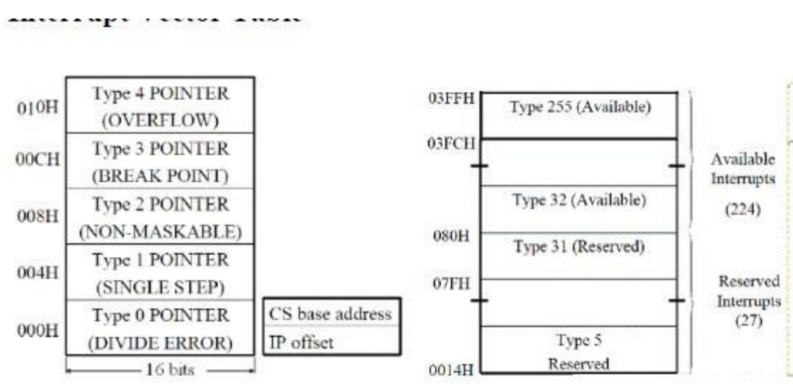
This is a type 3 interrupt.

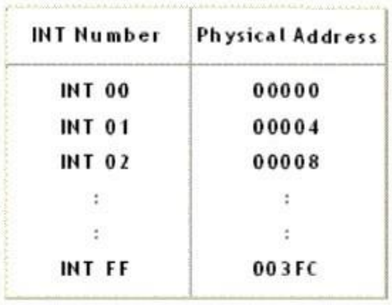
- INT <interrupt number> instruction - any one interrupt from available 256 interrupts.
- INTO instruction - interrupt on overflow
- Single-step interrupt - generated if the TF flag is set. This is a type 1 interrupt. When the CPU processes this interrupt it clears TF flag before calling the interrupt processing routine.
- Processor exceptions: Divide Error (Type 0), Unused Opcode (type 6) and Escape opcode (type 7).
- Software interrupt processing is the same as for the hardware interrupts.
- Ex: INT n (Software Instructions)
- Control is provided through:
    1. IF and TF flag bits
    2. IRET and IRETD



1. It decrements SP by 2 and pushes the flag register on the stack.
2. Disables INTR by clearing the IF.
3. It resets the TF in the flag Register.
5. It decrements SP by 2 and pushes CS on the stack.
6. It decrements SP by 2 and pushes IP on the stack.
7. Fetch the ISR address from the interrupt vector table

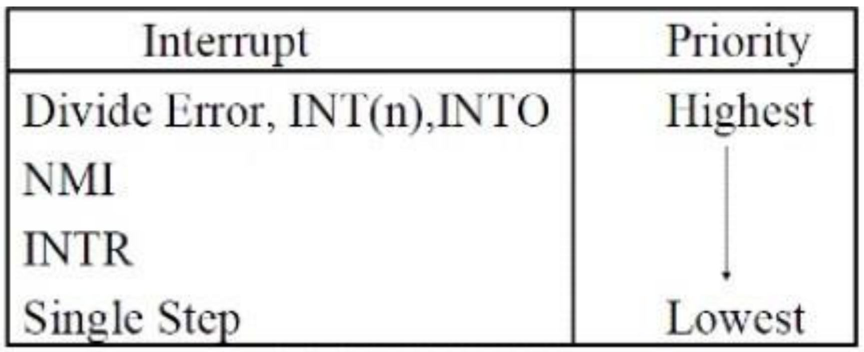
Interrupt Vector Table





INT Number	Physical Address
INT 00	00000
INT 01	00004
INT 02	00008
:	:
:	:
INT FF	003FC

Interrupt Priority Structure



Interrupt	Priority
Divide Error, INT(n),INTO	Highest
NMI	
INTR	
Single Step	Lowest

Addressing Modes of 8086:

Addressing mode indicates a way of locating data or operands. Depending up on the data type used in the instruction and the memory addressing modes, any instruction may belong to one or more addressing modes or same instruction may not belong to any of the addressing modes. The addressing mode describes the types of operands and the way they are accessed for executing an instruction. According to the flow of instruction execution, the instructions may be categorized as

1. Sequential control flow instructions and
2. Control transfer instructions.

Sequential control flow instructions are the instructions which after execution, transfer control to the next instruction appearing immediately after it (in the sequence) in the program. For example the arithmetic, logic, data transfer and processor control instructions are Sequential control flow instructions.

The control transfer instructions on the other hand transfer control to some predefined address or the address somehow specified in the instruction, after their execution. For example INT, CALL, RET & JUMP instructions fall under this category

The addressing modes for Sequential and control flow instructions are explained as follows.

1. Immediate addressing mode:

In this type of addressing, immediate data is a part of instruction, and appears in the form of successive byte or bytes.

Example: MOV AX, 0005H.

In the above example, 0005H is the immediate data. The immediate data may be 8- bit or 16-bit in size.

2. Direct addressing mode:

In the direct addressing mode, a 16-bit memory address (offset) directly specified in the instruction as a part of it

Example: MOV AX, [5000H].

3. Register addressing mode:

In the register addressing mode, the data is stored in a register and it is referred using the particular register. All the registers, except IP, may be used in this mode.

Example: MOV BX, AX

4. Register indirect addressing mode:

Sometimes, the address of the memory location which contains data or operands is determined in an indirect way, using the offset registers. The mode of addressing is known as register indirect mode.

In this addressing mode, the offset address of data is in either BX or SI or DI Register. The default segment is either DS or ES.

Example: MOV AX, [BX].

5. Indexed addressing mode:

In this addressing mode, offset of the operand is stored one of the index registers. DS & ES are the default segments for index registers SI & DI respectively.

Example: MOV AX, [SI]

Here, data is available at an offset address stored in SI in DS.

6. Register relative addressing mode:
In this addressing mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the register BX, BP, SI & DI in the default (either in DS & ES) segment.
Example: MOV AX, 50H [BX]
7. Based indexed addressing mode:
The effective address of data is formed in this addressing mode, by adding content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI). The default segment register may be ES or DS.
Example: MOV AX, [BX][SI]
8. Relative based indexed:
The effective address is formed by adding an 8 or 16-bit displacement with the sum of contents of any of the base registers (BX or BP) and any one of the index registers, in a default segment.
Example: MOV AX, 50H [BX] [SI]
For the control transfer instructions, the addressing modes depend upon whether the destination location is within the same segment or in a different one. It also depends upon the method of passing the destination address to the processor. Basically, there are two addressing modes for the control transfer instructions, viz. intersegment and intrasegment addressing modes.
If the location to which the control is to be transferred lies in a different segment other than the current one, the mode is called intersegment mode. If the destination location lies in the same segment, the mode is called intrasegment mode.

Addressing Modes for control transfer instructions:

1. Intersegment
   - Intersegment direct
   - Intersegment indirect.

2. Intrasegment
   - Intrasegment direct
   - Intrasegment indirect

1. Intersegment direct:

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 5000H, 2000H;
Jump to effective address 2000H in segment 5000H.

2. Intersegment indirect:

In this mode, the address to which the control is to be transferred lies in a different segment and it is passed to the instruction indirectly, i.e. contents of a memory block containing four bytes, i.e. 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 modes, except immediate mode.
Example: JMP [2000H].
Jump to an address in the other segment specified at effective address 2000H in DS.

3. Intrasegment direct mode:
In this mode, the address to which the control is to be transferred lies in the same segment in which the control transfers instruction lies and appears directly in the instruction as an immediate displacement value. In this addressing mode, the displacement is computed relative to the content of the instruction pointer.
The effective address to which the control will be transferred is given by the sum of 8 or 16 bit displacement and current content of IP. In case of jump instruction, if the signed displacement (d) is of 8-bits (i.e. -128<d<+127), it as short jump and if it is of 16 bits (i.e. -32768<d<+32767), it is termed as long jump.
Example: JMP SHORT LABEL.
4. 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 directly. Here, the branch address is found as the content of a register or a memory location.
This addressing mode may be used in unconditional branch instructions.
Example: JMP [BX]; Jump to effective address stored in BX.

<u>INSTRUCTION SET OF 8086</u>

The Instruction set of 8086 microprocessor is classified into 7, they are:-

- Data transfer instructions
- Arithmetic& logical instructions
- Program control transfer instructions
- Machine Control Instructions
- Shift / rotate instructions
- Flag manipulation instructions
- String instructions

Data Transfer instructions
Data transfer instruction, as the name suggests is for the transfer of data from memory to internal register, from internal register to memory, from one register to another register, from input port to internal register, from internal register to output port etc

1. MOV instruction
It is a general purpose instruction to transfer byte or word from register to register, memory to register, register to memory or with immediate addressing.
General Form:
MOV destination, source
Here the source and destination needs to be of the same size, that is both 8 bit or both 16 bit.
MOV instruction does not affect any flags.
Example:-
MOV BX, 00F2H ; load the immediate number 00F2H in BX register

MOV CL, [2000H] ; Copy the 8 bit content of the memory location, at a displacement of 2000H from data segment base to the CL register

MOV [589H], BX ; Copy the 16 bit content of BX register on to the memory location, which at a displacement of 589H from the data segment base.

MOV DS, CX ; Move the content of CX to DS.

2. PUSH instruction
The PUSH instruction decrements the stack pointer by two and copies the word from source to the location where stack pointer now points. Here the source must of word size data. Source can be a general purpose register, segment register or a memory location.
The PUSH instruction first pushes the most significant byte to sp-1, then the least significant to the sp-2.
Push instruction does not affect any flags

Example:-
PUSH CX ; Decrements SP by 2, copy content of CX to the stack (figure shows execution of this instruction)
PUSH DS ; Decrement SP by 2 and copy DS to stack

3. POP instruction
The POP instruction copies a word from the stack location pointed by the stack pointer to the destination. The destination can be a General purpose register, a segment register or a memory location. Here after the content is copied the stack pointer is automatically incremented by two.
The execution pattern is similar to that of the PUSH instruction.
Example:
POP CX ; Copy a word from the top of the stack to CX and increment SP by 2.

4. IN & OUT instructions
The IN instruction will copy data from a port to the accumulator. If 8 bit is read the data will go to AL and if 16 bit then to AX. Similarly OUT instruction is used to copy data from accumulator to an output port.
Both IN and OUT instructions can be done using direct and indirect addressing modes.

Example:
IN AL, 0F8H;   Copy a byte from the port 0F8H to AL
MOV DX, 30F8H; Copy port address in DX
IN AL, DX;   Move 8 bit data from 30F8H port
IN AX, DX; Move 16 bit data from 30F8H port
OUT 047H, AL;   Copy contents of AL to 8 bit port 047H
MOV DX, 30F8H; Copy port address in DX
OUT DX, AL; Move 8 bit data to the 30F8H port
OUT DX, AX; Move 16 bit data to the 30F8H port

5. XCHG instruction
The XCHG instruction exchanges contents of the destination and source. Here destination and source can be register and register or register and memory location, but XCHG cannot interchange the value of 2 memory locations.

General Format

XCHG Destination, Source

Example:
XCHG BX, CX; exchange word in CX with the word in BX
XCHG AL, CL; exchange byte in CL with the byte in AL
XCHG AX, SUM[BX];here physical address, which is DS+SUM+[BX]. The content at physical address and the content of AX are interchanged.

Arithmetic and Logic instructions:

The arithmetic and logic logical group of instruction include,

1. ADD instruction
Add instruction is used to add the current contents of destination with that of source and store the result in destination. Here we can use register and/or memory locations. AF, CF, OF, PF, SF, and ZF flags are affected

General Format:

ADD Destination, Source
Example:
ADD AL, 0FH ; Add the immediate content, 0FH to the content of AL and store the result in AL
ADD AX, BX ; AX <= AX+BX
ADD AX,0100H – IMMEDIATE
ADD AX,BX – REGISTER
ADD AX,[SI] – REGISTER INDIRECT OR INDEXED
ADD AX, [5000H] – DIRECT
ADD [5000H], 0100H – IMMEDIATE
ADD 0100H – DESTINATION AX (IMPLICT)

2. ADC: ADD WITH CARRY
This instruction performs the same operation as ADD instruction, but adds the carry flag bit (which may be set as a result of the previous calculation) to the result. All the condition code flags are affected by this instruction. The examples of this instruction along with the modes are as follows:
Example:
ADC AX,BX – REGISTER
ADC AX,[SI] – REGISTER INDIRECT OR INDEXED
ADC AX, [5000H] – DIRECT
ADC [5000H], 0100H – IMMEDIATE
ADC 0100H – IMMEDIATE (AX IMPLICT)

3. SUB instruction
SUB instruction is used to subtract the current contents of destination with that of source and store the result in destination. Here we can use register and/or memory locations. AF, CF, OF, PF, SF, and ZF flags are affected

General Format:
SUB Destination, Source
Example:
SUB AL, 0FH ; subtract the immediate content, 0FH from the content of AL and store the result in AL
SUB AX, BX ; AX <= AX-BX
SUB AX,0100H – IMMEDIATE (DESTINATION AX)
SUB AX,BX – REGISTER
SUB AX,[5000H] – DIRECT
SUB [5000H], 0100H – IMMEDIATE

4. SBB: SUBTRACT WITH BORROW

The subtract with borrow instruction subtracts the source operand and the borrow flag (CF) which may reflect the result of the previous calculations, from the destination operand. Subtraction with borrow, here means subtracting 1 from the subtraction obtained by SUB, if carry (borrow) flag is set.
The result is stored in the destination operand. All the flags are affected (condition code) by this instruction. The examples of this instruction are as follows:

Example:
SBB AX,0100H – IMMEDIATE (DESTINATION AX)
SBB AX,BX – REGISTER
SBB AX,[5000H] – DIRECT
SBB [5000H], 0100H – IMMEDIATE

5. CMP: COMPARE
The instruction compares the source operand, which may be a register or an immediate data or a memory location, with a destination operand that may be a register or a memory location. For comparison, it subtracts the source operand from the destination operand but does not store the result anywhere. The flags are affected depending upon the result of the subtraction. If both of the operands are equal, zero flag is set. If the source operand is greater than the destination operand,carry flag is set or else, carry flag is reset. The examples of this instruction are as follows:
Example:
CMP BX,0100H – IMMEDIATE
CMP AX,0100H – IMMEDIATE

6. INC & DEC instructions
INC and DEC instructions are used to increment and decrement the content of the specified destination by one. AF, CF, OF, PF, SF, and ZF flags are affected.
Example:
INC AL;AL<= AL + 1
INC AX;AX<=AX + 1
DEC AL;AL<= AL – 1
DEC AX;AX<=AX – 1

7. AND instruction
This instruction logically ANDs each bit of the source byte/word with the corresponding bit in the destination and stores the result in destination. The source can be an immediate number, register or memory location, register can be a register or memory location.
The CF and OF flags are both made zero, PF, ZF, SF are affected by the operation and AF is undefined.

General Format:

AND Destination, Source
Example:
AND BL, AL ; suppose BL=1000 0110 and AL = 1100 1010 then after the operation BL would be BL= 1000 0010.

AND CX, AX ; CX <= CX AND AX AND CL, 08 ; CL<= CL AND (0000 1000)

8. OR instruction
This instruction logically ORs each bit of the source byte/word with the corresponding bit in the destination and stores the result in destination. The source can be an immediate number, register or memory location, register can be a register or memory location.
The CF and OF flags are both made zero, PF, ZF, SF are affected by the operation and AF is undefined.

General Format:
OR Destination, Source

Example:
OR BL, AL ; suppose BL=1000 0110 and AL = 1100 1010 then after the operation BL would be BL= 1100 1110.

OR CX, AX; CX <= CX AND AX

OR CL, 08; CL<= CL AND (0000 1000)

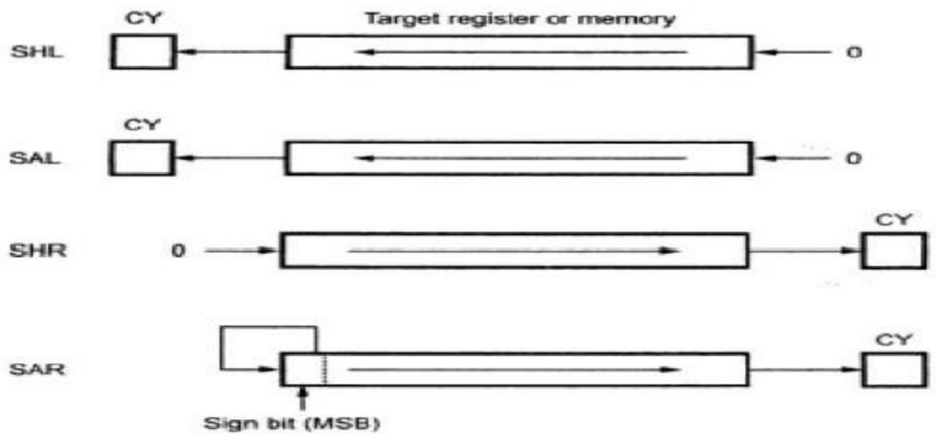
9. NOT instruction
The NOT instruction complements (inverts) the contents of an operand register or a memory location, bit by bit. The examples are as follows:
Example:
NOT AX (BEFORE AX= $(1011)_2$= $(B)_{16}$; AFTER EXECUTION AX= $(0100)_2$= $(4)_{16}$)

10. XOR instruction
The XOR operation is again carried out in a similar way to the AND and OR operation. The constraints on the operands are also similar. The XOR operation gives a high output, when the 2 input bits are dissimilar. Otherwise, the output is zero. The example instructions are as follows:
Example:
XOR AX,0098H

XOR AX,BX

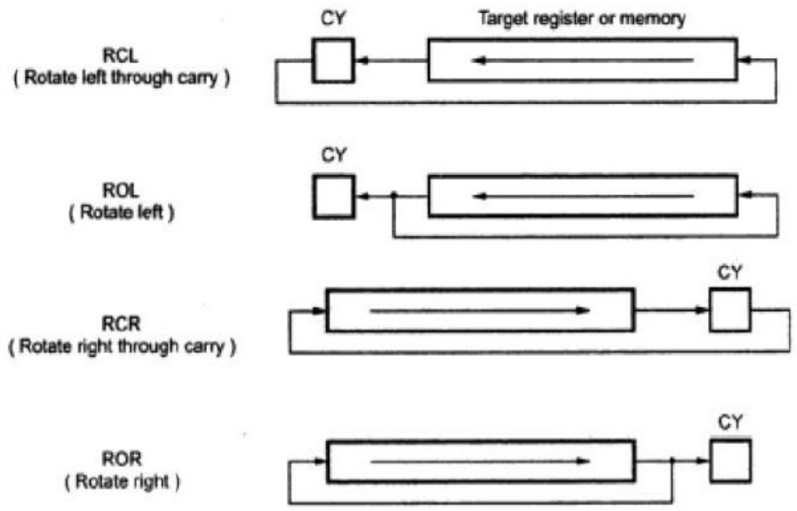
Shift / Rotate Instructions:

Shift instructions move the binary data to the left or right by shifting them within the register or memory location. They also can perform multiplication of powers of 2^{+n} and division of powers of 2^{-n}.
There are two type of shifts logical shifting and arithmetic shifting, later is used with signed numbers while former with unsigned.



. **Shift operations**

Rotate on the other hand rotates the information in a register or memory either from one end to another or through the carry flag.


CY
Target register or memory
RCL
(Rotate left through carry)
CY
ROL
(Rotate left)
CY
RCR
(Rotate right through carry)
CY
ROR
(Rotate right)
ROTATE OPERATIONS

SHL/SAL instruction

Both the instruction shifts each bit to left, and places the MSB in CF and LSB is made 0. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.
All flags are affected.
General Format:
SAL/SHL destination, count

Example:
MOV BL, B7H;BL is made B7H
SAL BL, 1; shift the content of BL register one place to left.

Before execution

cy	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
0	1	0	1	1	0	1	1	1

After execution

cy	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
1	0	1	1	0	1	1	1	0

1. SHR instruction
This instruction shifts each bit in the specified destination to the right and 0 is stored in the MSB position. The LSB is shifted into the carry flag. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

All flags are affected
General Format: SHR destination, count
Example:
MOV BL, B7H ; BL is made B7H
SHR BL, 1 ; shift the content of BL register one place to the right.

BEFORE:

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CY
1	0	1	1	0	1	1	1	0

AFTER:

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	CY
0	1	0	1	1	0	1	1	1

2. ROL instruction

This instruction rotates all the bits in a specified byte or word to the left some number of bit positions. MSB is placed as a new LSB and a new CF. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.
All flags are affected

General Format: ROL destination, count Example:
MOV BL, B7H ; BL is made B7H
ROL BL, 1 ; rotates the content of BL register one place to the left.

3. ROR instruction
This instruction rotates all the bits in a specified byte or word to the right some number of bit positions. LSB is placed as a new MSB and a new CF. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.
All flags are affected
General Format: ROR destination, count
Example:
MOV BL, B7H ; BL is made B7H
ROR BL, 1 ; shift the content of BL register one place to the right.

4. RCR instruction
This instruction rotates all the bits in a specified byte or word to the right some number of bit positions along with the carry flag. LSB is placed in a new CF and previous carry is placed in the new MSB. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

All flags are affected

General Format: RCR destination, count Example:
MOV BL, B7H ; BL is made B7H
RCR BL, 1 ; shift the content of BL register one place to the right.

Program control transfer instructions
There are 2 types of such instructions. They are:
1. Unconditional transfer instructions – CALL, RET, JMP
2. Conditional transfer instructions – J condition

1. CALL instruction

The CALL instruction is used to transfer execution to a subprogram or procedure. There are two types of CALL instructions, near and far.
A near CALL is a call to a procedure which is in the same code segment as the CALL instruction. 8086 when encountered a near call, it decrements the SP by 2 and copies the offset of the next instruction after the CALL on the stack. It loads the IP with the offset of the procedure then to start the execution of the procedure. A far CALL is the call to a procedure residing in a different segment. Here value of CS and offset of the next instruction both are backed up in the stack. And then branches to the procedure by changing the content of CS with the segment base containing procedure and IP with the offset of the first instruction of the procedure

Example:
Near call
CALL PRO ; PRO is the name of the procedure
CALL CX ; Here CX contains the offset of the first instruction of
the procedure, that is replaces the content of IP with the content of CX

Far call
CALL DWORD PTR[8X] ; New values for CS and IP are fetched from four
memory locations in the DS. The new value for CS is fetched from [8X] and [8X+1], the new IP is fetched from [8X+2] and [8X+3].

2. RET instruction
RET instruction will return execution from a procedure to the next instruction after the CALL instruction in the calling program. If it was a near call, then IP is replaced with the value at the top of the stack, if it had been a far call, then another POP of the stack is required. This second popped data from the stack is put in the CS, thus resuming the execution of the calling program. RET instruction can be followed by a number, to specify the parameters passed.

3. JMP instruction
This is also called as unconditional jump instruction, because the processor jumps to the specified location rather than the instruction after the JMP instruction. Jumps can be short jumps when the target address is in the same segment as the JMP instruction or far jumps when it is in a different segment.

General Format: JMP<targetaddress>

4. Conditional Jump :
Conditional jumps are always short jumps in 8086. Here jump is done only if the condition specified is true/false. If the condition is not satisfied, then the execution proceeds in the normal way.

JC : Jump on carry (CF=set)
JNC : Jump on non carry (CF=reset)
JZ : Jump on zero (ZF=set)

JNO: Jump on overflow (OF=set)

5. Iteration control instructions
These instructions are used to execute a series of instructions some number of times. The number is specified in the CX register, which will be automatically decremented in course of iteration. But here the destination address for the jump must be in the range of -128 to 127 bytes.
LOOP : loop through the set of instructions until CX is 0
LOOPE/LOOPZ : here the set of instructions are repeated until CX=0 or ZF=0
LOOPNE/LOOPNZ: here repeated until CX=0 or ZF=1

1. HLT instruction
The HLT instruction will cause the 8086 microprocessor to fetching and executing instructions. The 8086 will enter a halt state. The processor gets out of this Halt signal upon an interrupt signal in INTR pin/NMI pin or a reset signal on RESET input.
2. WAIT instruction
When this instruction is executed, the 8086 enters into an idle state. This idle state is continued till a high is received on the TEST input pin or a valid interrupt signal is received. Wait affects no flags. It generally is used to synchronize the 8086 with a peripheral device(s).
3. ESC instruction
This instruction is used to pass instruction to a coprocessor like 8087. There is a 6 bit instruction for the coprocessor embedded in the ESC instruction. In most cases the 8086 treats ESC and a NOP, but in some cases the 8086 will access data items in memory for the coprocessor
4. LOCK instruction
In multiprocessor environments, the different microprocessors share a system bus, which is needed to access external devices like disks. LOCK Instruction is given as prefix in the case when a processor needs exclusive access of the system bus for a particular instruction. It affects no flags

Example:
LOCK XCHG SEMAPHORE, AL :The XCHG instruction requires two
bus accesses. The lock prefix prevents another processor from taking control of thesystem bus between the 2 accesses
5. NOP instruction
At the end of NOP instruction, no operation is done other than the fetching and decoding of the instruction. It takes 3 clock cycles. NOP is used to fill in time delays or to provide space for instructions while trouble shooting. NOP affects no flags

Flag manipulation instructions:

1. STC instruction
This instruction sets the carry flag. It does not affect any other flag.
2. CLC instruction

This instruction resets the carry flag to zero. CLC does not affect any other flag.
3. CMC instruction
This instruction complements the carry flag. CMC does not affect any other flag.
4. STD instruction
This instruction is used to set the direction flag to one so that SI and/or DI can be decremented automatically after execution of string instruction. STD does not affect any other flag.
5. CLD instruction
This instruction is used to reset the direction flag to zero so that SI and/or DI can be incremented automatically after execution of string instruction. CLD does not affect any other flag.
6. STI instruction
This instruction sets the interrupt flag to 1. This enables INTR interrupt of the 8086. STI does not affect any other flag.
7. CLI instruction
This instruction resets the interrupt flag to 0. Due to this the 8086 will not respond to an interrupt signal on its INTR input. CLI does not affect any other flag.

String Instructions:
1. MOVS/MOVSB/MOVSW
These instructions copy a word or byte from a location in the data segment to a location in the extra segment. The offset of the source is in SI and that of destination is in DI. For multiple word/byte transfers the count is stored in the CX register.
When direction flag is 0, SI and DI are incremented and when it is 1, SI and DI are decremented. MOVS affect no flags. MOVSB is used for byte sized movements while MOVSW is for word sized.

Example:
CLD ; clear the direction flag to auto increment SI and DI MOV AX, 0000H ;
MOV DS, AX ; initialize data segment register to 0 MOV ES, AX ; initialize extra segment register to 0 MOV SI, 2000H ; Load the offset of the string1 in SI MOV DI, 2400H ; Load the offset of the string2 in DI MOV CX, 04H ; load length of the string in CX
REP MOVSB ; decrement CX and MOVSB until CX will be 0

2. REP/REPE/REP2/REPNE/REPNZ
REP is used with string instruction; it repeats an instruction until the specified condition becomes false.
3. LODS/LODSB/LODSW
This instruction copies a byte from a string location pointed to by SI to AL or a word from a string location pointed to by SI to AX.LODS does not affect any flags. LODSB copies byte and LODSW copies word.
STOS/STOSB/STOSW
The STOS instruction is used to store a byte/word contained in AL/AX to
the offset contained in the DI register. STOS does not affect any flags. After copying the content DI is automatically incremented or decremented, based on the value of direction flag.
5. CMPS/CMPSB/CMPSW

CMPS is used to compare the strings, byte wise or word wise. The comparison is affected by subtraction of content pointed by DI from that pointed by SI. The AF, CF, OF, PF, SF and ZF flags are affected by this instruction, but neither operand is affected.

ASSEMBLER DIRECTIVES:
There are some instructions in the assembly language program which are not a part of processor instruction set. These instructions are instructions to the assembler, linker and loader. These are referred to as pseudo-operations or as assembler directives. The assembler directives enable us to control the way in which a program assembles and lists. They act during the assembly of a program and do not generate any executable machine code.

There are many specialized assembler directives. Let us see the commonly used assembler directive in 8086 assembly language programming.
1. ASSUME:
It is used to tell the name of the logical segment the assembler to use for a specified segment. E.g.: ASSUME CS: CODE tells that the instructions for a program are in a logical segment named CODE.
2. DB -Define Byte:
The DB directive is used to reserve byte or bytes of memory locations in the available memory. While preparing the EXE file, this directive directs the assembler to allocate the specified number of memory bytes to the said data type that may be a constant, variable, string, etc. Another option of this directive also initializes the reserved memory bytes with the ASCII codes of the characters specified as a string. The following examples show how the DB directive is used for different purposes.
3. DD -Define Double word - used to declare a double word type variable or to reserve memory locations that can be accessed as double word.
4. DQ -Define Quad word
This directive is used to direct the assembler to reserve 4 words (8 bytes) of memory for the specified variable and may initialize it with the specified values.
5. DT -Define Ten Bytes:
The DT directive directs the assembler to define the specified variable requiring 10-bytes for its storage and initialize the 10-bytes with the specified values. The directive may be used in case of variables facing heavy numerical calculations, generally processed by numerical processors.
6. DW -Define Word:
The DW directives serves the same purposes as the DB directive, but it now makes the assembler reserve the number of memory words (16-bit) instead of bytes. Some examples are given to explain this directive
. 7. END-End of Program:
The END directive marks the end of an assembly language program. When the assembler comes across this END directive, it ignores the source lines available later on. Hence, it should be ensured that the END statement should be the last statement in the file and should not appear in between. Also, no useful program statement should lie in the file, after the END statement
8. ENDP-End Procedure - Used along with the name of the procedure to indicate the end of a procedure.

E.g.: SQUARE_ROOT PROC: start of procedure SQUARE_ROOT ENDP: End of procedure
9. EQU-Equate - Used to give a name to some value or symbol. Each time the assembler finds the given name in the program, it will replace the name with the vale.
E.g.: CORRECTION_FACTOR EQU 03H MOV AL, CORRECTION_FACTOR
10. LABEL- Used to give a name to the current value in the location counter.
This directive is followed by a term that specifies the type you want associated with that name.
E.g: ENTRY_POINT LABEL FAR
NEXT: MOV AL, BL

11. OFFSET- Used to determine the offset or displacement of a named data item or procedure from the start of the segment which contains it.
E.g.: MOV BX, OFFSET PRICES
12. ORG- The location counter is set to 0000 when the assembler starts reading a segment. The ORG directive allows setting a desired value at any point in the program.
E.g.: ORG 2000H

CODE FOR 8 BIT ADDER

```
DATA SEGMENT
A1 DB 50H
A2 DB 51H
```

```
RES DB ?
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS:DATA
START: MOV AX,DATA
MOV DS,AX
MOV AL,A1
MOV BL,A2
ADD AL,BL
MOV RES,AL
MOV AX,4C00H
INT 21H
CODE ENDS
END START
```

CODE FOR 16 BIT ADDER

```
DATA SEGMENT
A1 DW 0036H
A2 DW 0004H SUM DW ?
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
MOV DS,AX
MOV AX,A1
MOV BX,A2
DIV BX
MOV SUM,AX
MOV AX,0008H
INT 21H
CODE ENDS
END START
```

ARRAY SUM

```
.MODEL SMALL
.DATA
ARRAY DB 12H, 24H, 26H, 63H, 25H, 86H, 2FH, 33H, 10H, 35H
```

```
SUM DW 0
.CODE
START:MOV AX, @DATA
MOV DS, AX
MOV CL, 10
XOR DI, DI
LEA BX, ARRAY
BACK: MOV AL, [BX+DI]
MOV AH, 00H
ADD SUM, AX
INC DI
DEC CL
JNZ BACK
END START
```

ASCIITOHEX

```
DATA SEGMENT
A DB 41H
R DB ?
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS:DATA
START:  MOV AX,DATA
MOV DS,AX
MOV AL,A
SUB AL,30H
CMP AL,39H
JBE L1
SUB AL,7H
L1: MOV R,AL
INT 3H
CODE ENDS
END START
```

8BMUL

```
DATA SEGMENT
A1 DB 25H
A2 DB 25H
A3 DB ?
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS:DATA START:MOV AX,DATA
MOV DS,AX
MOV AL,A1
MOV BL,A2
MUL BL
MOV A3,AL
MOV AX,4C00H
INT 21H
CODE ENDS
END START
```

16BIT MUL

```
DATA SEGMENT
A1 DW 1000H
A2 DW 1000H
A3 DW ?
A4 DW ?
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS:DATA
START:MOV AX,DATA
MOV DS,AX
MOV AX,A1
MOV BX,A2
MUL BX
MOV A3,DX
MOV A4,AX
MOV AX,4C00H
INT 21H
CODE ENDS
END START
```

HEX TO ASCII

```
DATA SEGMENT
A DB 08H
C DB ?
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS: DATA
START: MOV AX,DATA
MOV DS,AX
MOV AL,A
ADD AL,30H
CMP AL,39H
JBE L1
ADD AL,7H
L1: MOV C,AL
```

```
INT 3H
CODE ENDS
END START
```

GREATEST NUMBER IN ARRAY

```
.MODEL SMALL
.STACK 100
.DATA
ARRAY DB 63H,32H,45H,75,12H,42H,09H,14H,56H,38H
MAX DB 0
.CODE
START:MOV AX,@DATA
MOV DS,AX
XOR DI,DI
MOV CL,10
LEA BX,ARRAY
MOV AL,MAX
BACK: CMP AL,[BX+DI]
JNC SKIP
MOV DL,[BX+DI]
MOV AL,DL
SKIP: INC DI
DEC CL
JNZ BACK
MOV MAX,AL
MOV AX,4C00H
INT 21H
END START
```

UNIT-6

<u>Difference between Microprocessor & Microcontroller</u>

MICROPROCESSOR	MICROCONTROLLER
Microprocessor widely used in the computer system.. If the microprocessor is the heart of computer system.	And microcontroller is used in embedded system. microcontroller is the heart of the embedded system.
The microprocessor uses Von Neumann architecture where data and program present in the same memory module	The microcontroller uses Harvard architecture. In this module, data and program get stored in separate memory. The microcontroller can access data and program at the same time as it is in a separate memory. This is one of the reasons microcontrollers is faster than the microprocessor.
The microprocessor cannot operate without peripheral components. It has only processing unit and we have to attach all the required components externally to operate	Whereas the microcontroller has small processing unit along with internal memory to store and I/O components to give input. So it can work independently.
As we have to connect components externally, microprocessor circuit becomes large and complex	In a microcontroller, all the components are internally connected, so the circuit becomes too small. The microcontroller can be used in the compact system as it has a small size. It provides better and efficient technique in the compact system than the microprocessor.
icroprocessor requires external components and its circuits also complex. It requires more power consumption. So it is difficult to operate microprocessor using battery power.	The microcontroller has very low external components. it manages all its operation inside the single chip. So it consumes very low power supply as compared to the microprocessor. We can operate microcontroller on the externally connected stored power such as a battery.
Most of the Microprocessor does not have this power saving feature.	The microcontroller can have multiple modes of operation such as higher performance, balance, idle or power saving mode. So if we operate microcontroller in power saving mode, the power consumption reduce even more.
The microprocessor has very less internal registers. It has to rely on external storage. So all the memory operations are carried out using memory-based external commands. Results in high processing time.	The microcontroller has many registers for instruction execution. Fetching data and storing data require internal commands. So its execution and processing time are lower than the microprocessor.

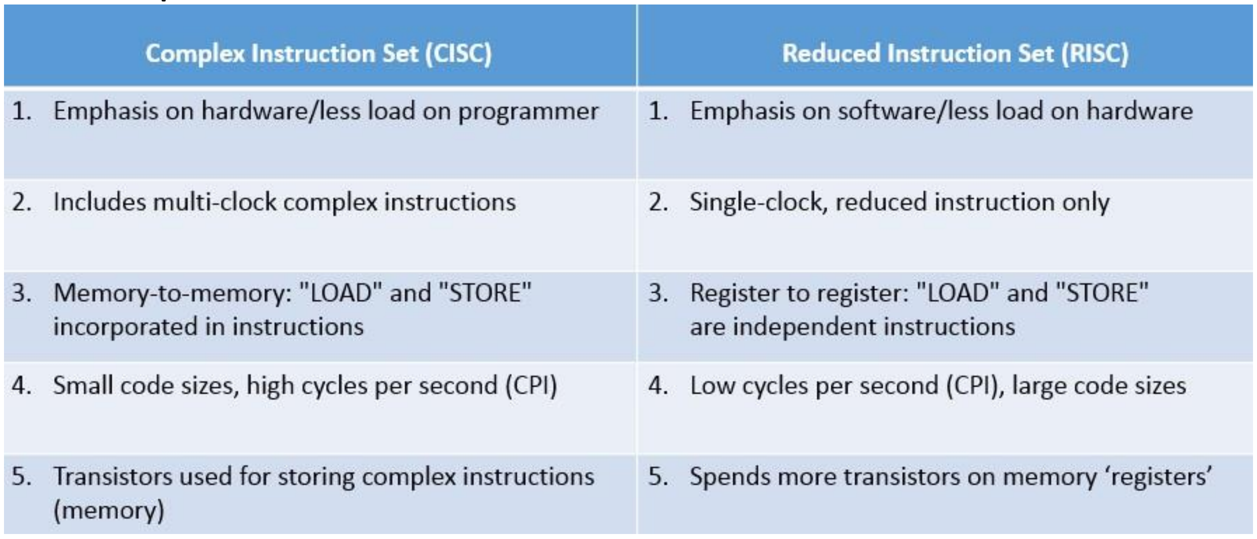
Microprocessor can perform variety of operations	Microcontroller are designed for single purpose

8 bit & 16 bit microcontroller

In 8-bit microcontroller, the point when the internal bus is 8-bit then the ALU is performs the arithmetic and logic operations. The examples of 8-bit microcontrollers are Intel 8031/8051, PIC1x and Motorola MC68HC11 families

The 16-bit microcontroller performs greater precision and performance as compared to 8-bit. For example 8 bit microcontrollers can only use 8 bits, resulting in a final range of 0×00 – 0xFF (0-255) for every cycle. In contrast, 16 bit microcontrollers with its 16 bit data width has a range of 0×0000 – 0xFFFF (0-65535) for every cycle. A longer timer most extreme worth can likely prove to be useful in certain applications and circuits. It can automatically operate on two 16 bit numbers. Some examples of 16-bit microcontroller are 16-bit MCUs are extended 8051XA, PIC2x, Intel 8096 and Motorola MC68HC12 families.

CISC & RISC processor



Complex Instruction Set (CISC)	Reduced Instruction Set (RISC)
1. Emphasis on hardware/less load on programmer	1. Emphasis on software/less load on hardware
2. Includes multi-clock complex instructions	2. Single-clock, reduced instruction only
3. Memory-to-memory: "LOAD" and "STORE" incorporated in instructions	3. Register to register: "LOAD" and "STORE" are independent instructions
4. Small code sizes, high cycles per second (CPI)	4. Low cycles per second (CPI), large code sizes
5. Transistors used for storing complex instructions (memory)	5. Spends more transistors on memory 'registers'

CISC advantages vs disadvantages

ADVANTAGES:

- Source code programs are short; fewer instructions; this saves memory space.
- Fewer instructions in a code means fewer instruction bytes to be fetched. (reducing fetch operations).
- A complex High-Level Language (HLL) operation will execute more quickly in a single machine instruction (cycle).

DIADVANTAGES

- Memory is now much inexpensive
- The number of bits of memory occupied by a CISC program (in machine language) may not be noticeably small (heavy programs.

- Using complex and fewer instructions, code size cannot be reduced/optimized by the programmer (inflexible).
- The individual instructions are complex, and they take a long time to decode and execute.
- CISC uses memory references for operands, which is much slower.
- There are more instructions on a CISC, longer opcodes are required, producing longer instructions. (instructions lengths vary)

- Each instruction takes different clock cycles to execute, so in CISC, the pipeline is not possible. (to execute multiple instructions in parallel).

RISC Advantages

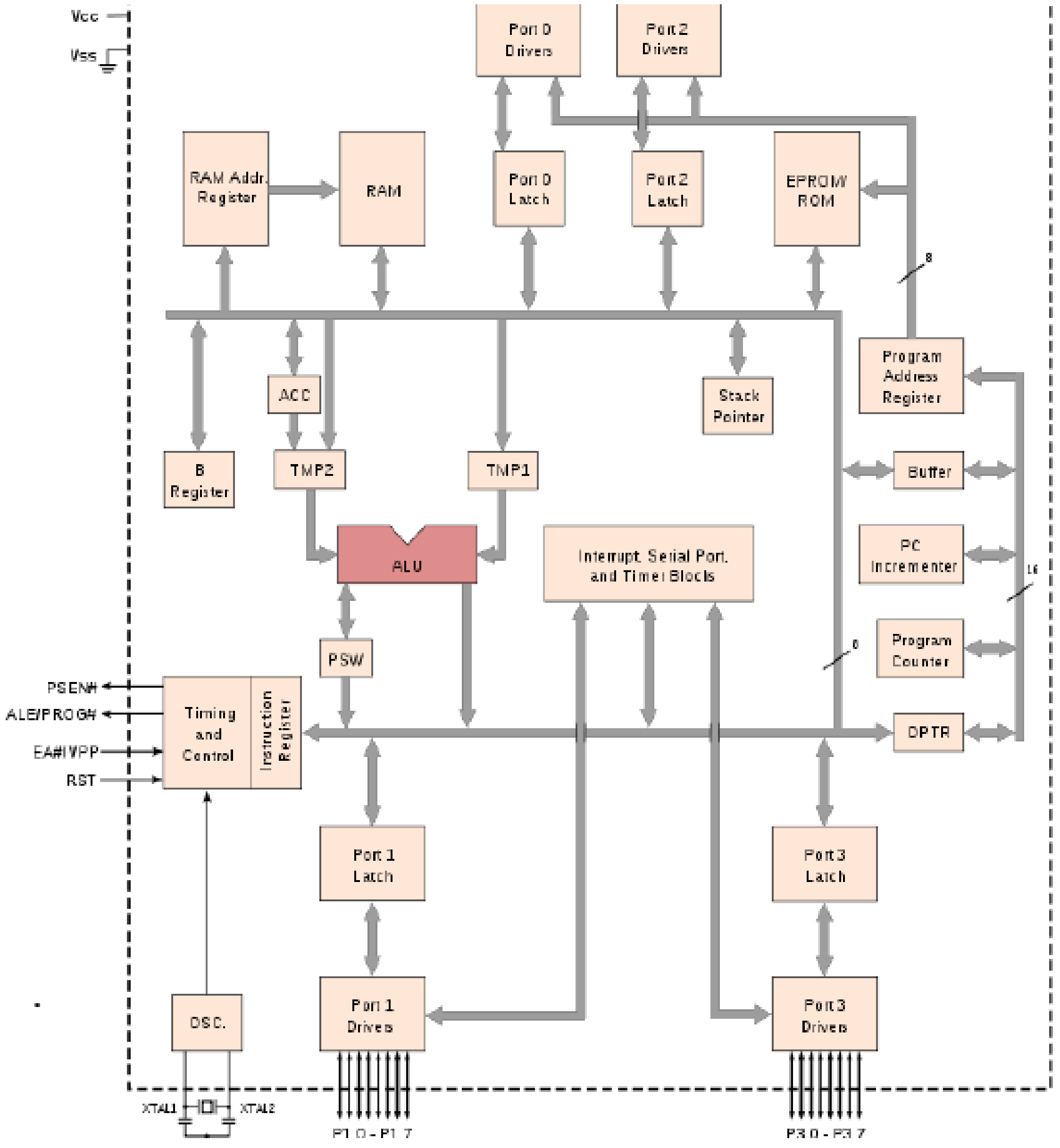
- The instructions are simple/primitive, and they execute fast.
- RISC uses registers to access operands, which are much faster to read.
- Instruction length is fixed, especially the opcode so opcode decoding and register operand accessing can occur simultaneously.
- The source code is long, but it is flexible enough to be optimized.
- Because each instruction takes one clock cycle, instruction pipelining is possible. This improves processor operating speed.
- RISC processors are more responsive to interrupts because interrupts are checked between each small instruction/every machine cycle.

Architecture of 8051Microcontroller

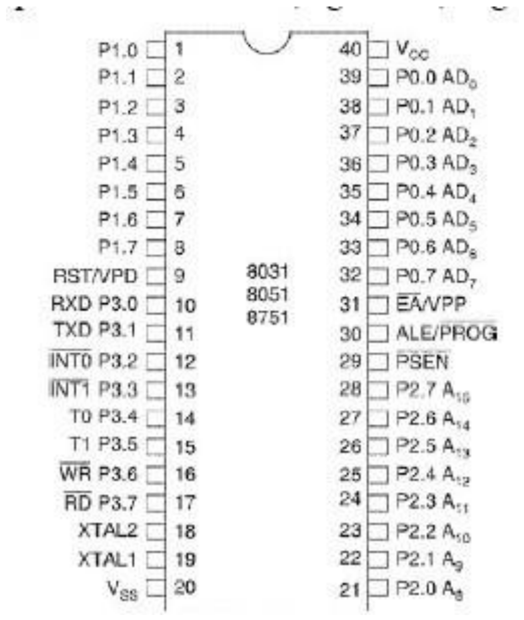
Some of the features that have made the 8051 popular are:

🎬 64 KB on chip program memory.

- 128 bytes on chip data memory (RAM).
- 4 register banks.
- 128 user defined software flags.
- 8-bit data bus
- 16-bit address bus
- 32 general purpose registers each of 8 bits
- 16 bit timers (usually 2, but may have more, or less).
- 3 internal and 2 external interrupts.
- Bit as well as byte addressable RAM area of 16 bytes.
- Four 8-bit ports, (short models have two 8-bit ports).
- 16-bit program counter and data pointer.
- 1 Microsecond instruction cycle with 12 MHz Crystal.


Vcc
Vss
Port 0 Drivers
Port 2 Drivers
RAM Addr. Register
RAM
Port 0 Latch
Port 2 Latch
EPROM/ROM
8
Program Address Register
ACC
Stack Pointer
B Register
TMP2
TMP1
Buffer
ALU
Interrupt, Serial Port, and Timer Blocks
PC Incrementer
16
PSW
Program Counter
8
PSEN#
ALE/PROG#
EA#/VPP
RST
Timing and Control
Instruction Register
DPTR
Port 1 Latch
Port 3 Latch
OSC.
Port 1 Drivers
Port 3 Drivers
XTAL1
XTAL2
P1.0 - P1.7
P3.0 - P3.7

8051 PIN DIAGRAM:



Basic Pins:
Pin 9: PIN 9 is the reset pin which is used reset the microcontroller‘s internal registers and ports upon starting up. (Pin should be held high for 2 machine cycles.)

Pins 18 & 19: The 8051 has a built-in oscillator amplifier hence we need to only connect a crystal at these pins to provide clock pulses to the circuit.

Pin 40 and 20: Pins 40 and 20 are VCC and ground respectively. The 8051 chip needs +5V 500mA to function properly, although there are lower powered versions like the Atmel 2051 which is a scaled down version of the 8051 which runs on +3V.

Pins 29, 30 & 31: As described in the features of the 8051, this chip contains a built-in flash memory. In order to program this we need to supply a voltage of +12V at pin 31. If external memory is connected then PIN 31, also called EA/VPP, should be connected to ground to indicate the presence of external memory. PIN 30 is called ALE (address latch enable), which is used when multiple memory chips are connected to the controller and only one of them needs to be selected. We will deal with this in depth in the later chapters. PIN 29 is called PSEN. This is "program store enable". In order to use the external memory it is required to provide the low voltage (0) on both PSEN and EA pins.

Ports:
There are 4 8-bit ports: P0, P1, P2 and P3.

PORT P1 (Pins 1 to 8): The port P1 is a general purpose input/output port which can be used for a variety of interfacing tasks. The other ports P0, P2 and P3 have dual roles or additional functions associated with them based upon the context of their usage.

PORT P3 (Pins 10 to 17): PORT P3 acts as a normal IO port, but Port P3 has additional functions such as, serial transmit and receive pins, 2 external interrupt pins, 2 external counter inputs, read and write pins for memory access.

P3.0	–	RXD (Serial input)
P3.1	–	TXD (Serial output)
P3.2	–	$\overline{INT0}$ (External interrupt)
P3.3	–	$\overline{INT1}$ (External interrupt)
P3.4	–	T0 (Timer 0 external input)
P3.5	–	T1 (Timer 1 external input)
P3.6	–	$\overline{WR}$ (External data memory write strobe)
P3.7	–	$\overline{RD}$ (External data memory read strobe)

PORT P2 (pins 21 to 28): PORT P2 can also be used as a general purpose 8 bit port when no external memory is present, but if external memory access is required then PORT P2 will act as an address bus in conjunction with PORT P0 to access external memory. PORT P2 acts as A8-A15, as can be seen from ABOVE fig

PORT P0 (pins 32 to 39) PORT P0 can be used as a general purpose 8 bit port when no external memory is present, but if external memory access is required then PORT P0 acts as a multiplexed address and data bus that can be used to access external memory in conjunction with PORT P2. P0 acts as AD0-AD7, as can be seen from above fig

Oscillator Circuits

The 8051 requires the existence of an external oscillator circuit. The oscillator circuit usually runs around 12MHz, although the 8051 (depending on which specific model) is capable of running at a maximum of 40MHz. Each machine cycle in the 8051 is 12 clock cycles, giving an effective cycle rate at 1MHz (for a 12MHz clock) to 3.33MHz (for the maximum 40MHz clock).

The 8051 has an internal clock circuit, hence acrystal of proper frequency can be connected directly to these two pins. The frequency range is 3.5–12 MHz. An external oscillator can also be connected instead of a crystal. In this case, the XTAL1 pin is grounded and the oscillator signal is given to the XTAL2 pin

$PS'EN$: This pin gives out active low output pulses. This signal is used for fetching the data from the external program memory. A pulse is generated after every six clock cycles. This is used as a read signal for reading from the external program memory. If the data to be fetched I inside the chip itself, then is $PS'EN$not generated. This signal is also not generated during the external data memory operation.

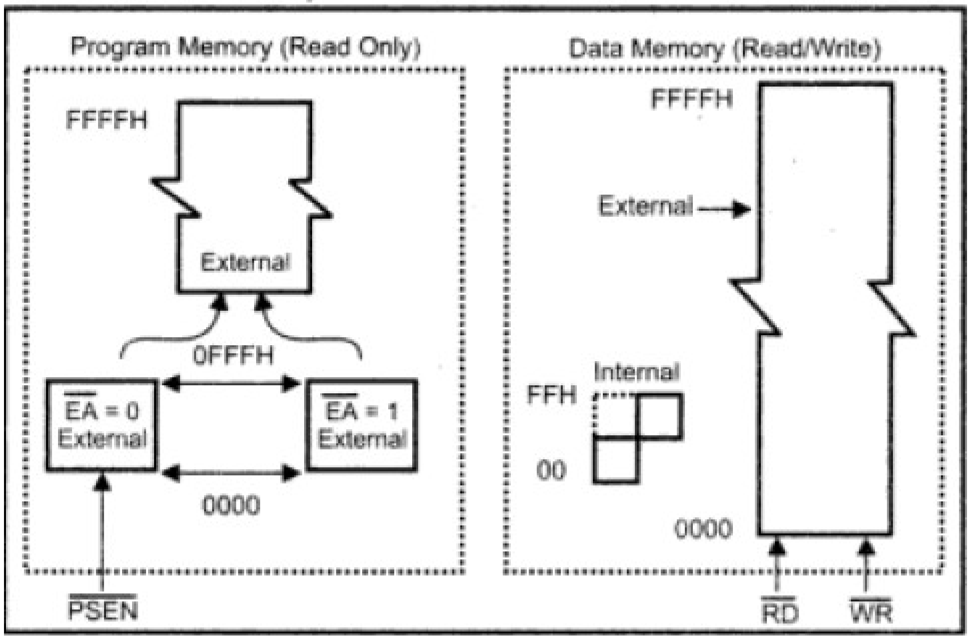
$E'A/VPP$: This pin, if connected to 5 V, will indicate to the processor that the 4 KB of the program memory within the chip should be used. Hence addresses 0000H to 0FFFH will be within the chip. Once the address exceeds this, i.e. 1000H to FFFFH, the external program memory is accessed. This pin, if pulled low, will inform the processor that the on chip
ROM is not being used. Hence the entire 64 KB will be external to the chip. So, depending on whether a designer is using the internal program memory or not, he/she will have to connect

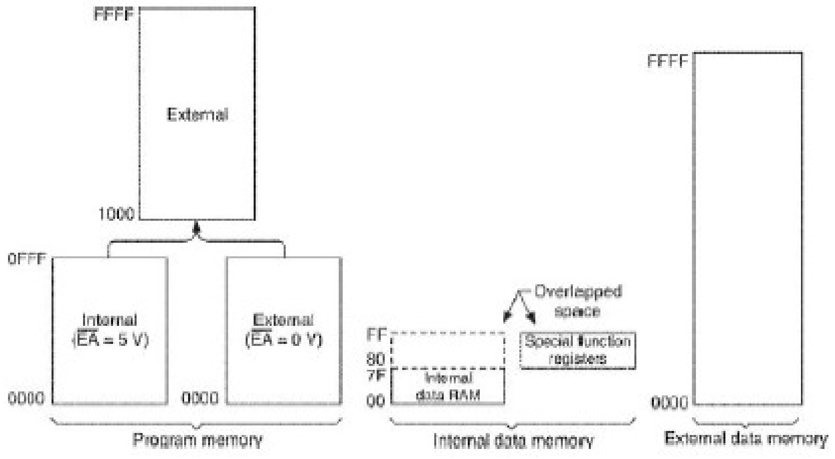
the pin to either 5 V or ground. This pin is also used while programming the 8751 as programming supply (VPP) voltage (21 V).

RST/VPD: A high on this pin, for more than 24 oscillator cycles, will reset the chip. VPD may be used to supply power to the internal RAM during power failure or power down modes.

Memory Organisation

The 8051 can access up to 64 KB of program memory and 64 KB of external data memory and also the internal data RAM locations. Figure below shows the memory mapping in this chip



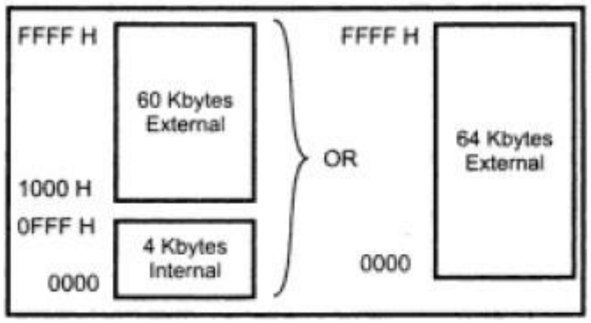


External Data Memory and Program Memory

We have seen that 8051 has internal data and code memory with limited memory capacity. This memory capacity may not be sufficient for some applications. In such situations, we have to connect external ROM/EPROM and RAM to 8051 microcontroller to increase the memory capacity. We also know that ROM is used as a program memory and RAM is used as a data memory. Let us see how 8051 accesses these memories.

External Program Memory

a map of the 8051 program memory.



The 8051 program memory

In 8051, when the $\overline{EA}$ pin is connected to V_{CC}, program fetches to addresses 0000H through 0FFFH are directed to the internal ROM and program fetches to addresses 1000H through FFFFH are directed to external ROM/EPROM. On the other hand when $\overline{EA}$ pin is grounded, all addresses (0000H to FFFFH) fetched by program are directed to the external ROM/EPROM. The $\overline{PSEN}$ signal is used to activate output enable signal of the external ROM/EPROM, as shown in the Fig.

SPECIAL FUNCTION REGISTERS:

All the resources in the 8051 can be accessed through special function registers maintained in the internal data memory. The resources like timers would require setting the modes and controlling them. Similarly, serial data input would require a serial data buffer as well as the control. The 8051 offers 4 ports which can be used for various purposes like input/output, memory interface, and so on. The functions of these resources can also be programmed through special function registersWe shall now describe all the special function registers available in the 8051. As the name itself suggests, these registers have some special functions like controlling the timer/counter, enabling interrupts, controlling the serial port operations, etc. There are 21 special function registers in the 8051. Some special function registers are bit addressable. The various special function registers are shown below:

ACC	-	Accumulator*
B	-	B Register*
PSW	-	Program Status Word*
SP	-	Stack Pointer
DPTR (Low)	-	Data Pointer Low
DPTR (High)	-	Data Pointer High
P0	-	Port 0*
P1	-	Port 1*
P2	-	Port 2*
P3	-	Port 3*
IP	-	Interrupt Priority*
IE	-	Interrupt Enable*
TMOD	-	Timer/Counter Mode
TCON	-	Timer/Counter Control*
TH0	-	(Timer/Counter) 0 High
TL0	-	(Timer/Counter) 0 Low
TH1	-	(Timer/Counter) 1 High
TL1	-	(Timer/Counter) 1 Low
SCON	-	Serial Control*
SBUF	-	Serial Data Buffer
PCON	-	Power Control

Accumulator: ACC (also called A register) is the accumulator. Thereare many instructions that use the accumulator as the destination. In nearly all arithmetic operations, ACC is used as one of the operands and after the operation, the result is also stored in the ACC.

B Register: This register is mainly used for multiply and divide operations in which this register acts as one of the operands and after the operations a part of the result is stored in this register. This register otherwise is just like a scratch pad, i.e. a temporary register.

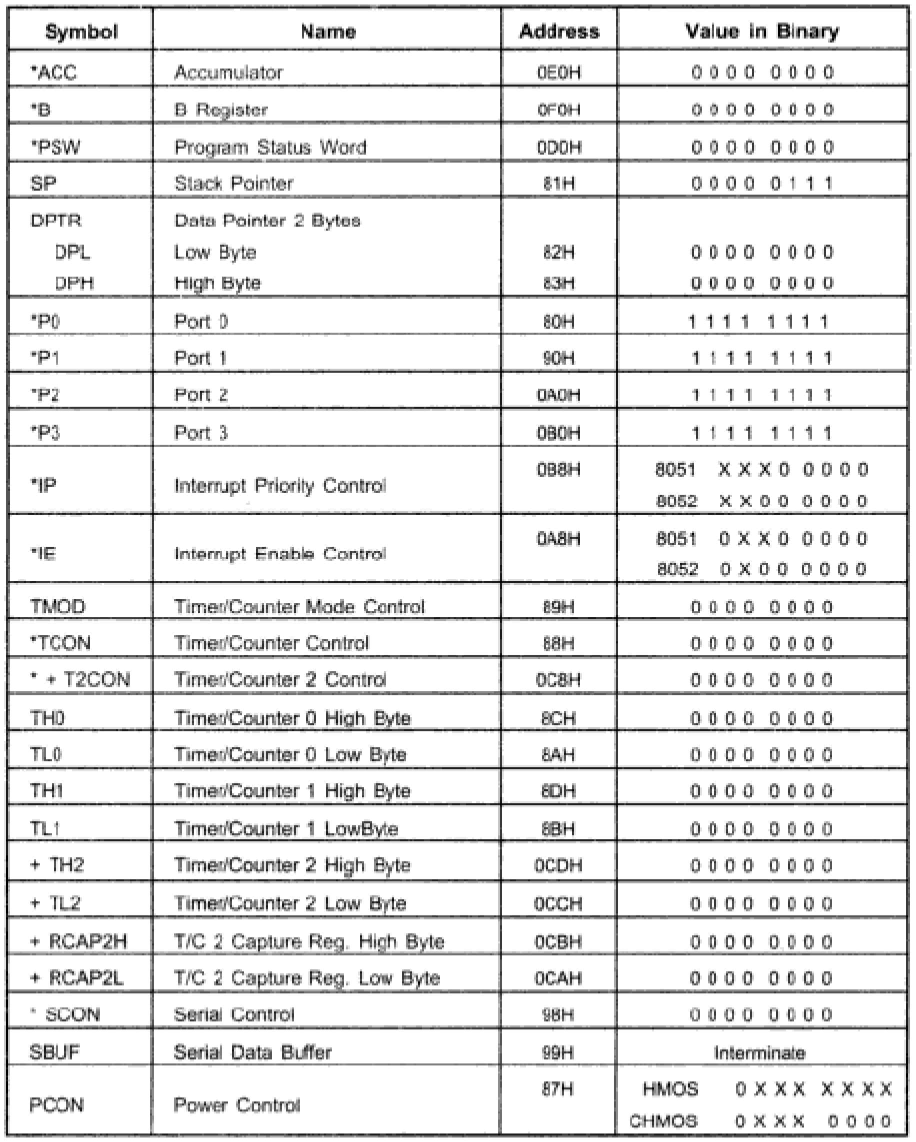
Stack Pointer: The stack pointer in the 8051 is an 8-bit wide register. This pointer can point to any location in the internal data RAM, i.e. locations from 0–127. When the chip is reset, this register is initialized to 07H. During PUSH and CALL instructions the stack pointer is first incremented and then the data is stored in the stack. That is, if initially (SP) = 20H, then the first bytes of the data will be stored at the 21H address.

Data Pointer: The data pointer (DPTR) is a 16-bit register, consisting of a high byte (DPH) and a low byte (DPL). This register normally contains a 16-bit address

Serial Data Buffer (SBUF): This register holds the data that has to be transmitted through the serial port and also holds the data that is received. This register is interconnected to two 8-bit shift registers. When the data is written into the SBUF, it is loaded in the transmit shift register and hence the process of moving a data byte into the SBUF starts the transmission process. During reception, the data coming to the 8051 is clocked into the receive shift register and once all the 8 bits of data or a frame is received, it is transferred to the SBUF.

Control and status registers: All the special function registers (like IP, IE, TMOD, TCON, PCON, SCON, etc.) that are used for controlling the internal resources or to see the status of these resources come under this category. These registers contain the control and status bits of the interrupt systems, timers, and serial port operations.

THE LIST OF ALL SFRS



Symbol	Name	Address	Value in Binary
*ACC	Accumulator	0E0H	0000 0000
*B	B Register	0F0H	0000 0000
*PSW	Program Status Word	0D0H	0000 0000
SP	Stack Pointer	81H	0000 0111
DPTR	Data Pointer 2 Bytes		
DPL	Low Byte	82H	0000 0000
DPH	High Byte	83H	0000 0000
*P0	Port 0	80H	1111 1111
*P1	Port 1	90H	1111 1111
*P2	Port 2	0A0H	1111 1111
*P3	Port 3	0B0H	1111 1111
*IP	Interrupt Priority Control	0B8H	8051 XXX0 0000 8052 XX00 0000
*IE	Interrupt Enable Control	0A8H	8051 0XX0 0000 8052 0X00 0000
TMOD	Timer/Counter Mode Control	89H	0000 0000
*TCON	Timer/Counter Control	88H	0000 0000
* + T2CON	Timer/Counter 2 Control	0C8H	0000 0000
TH0	Timer/Counter 0 High Byte	8CH	0000 0000
TL0	Timer/Counter 0 Low Byte	8AH	0000 0000
TH1	Timer/Counter 1 High Byte	8DH	0000 0000
TL1	Timer/Counter 1 LowByte	8BH	0000 0000
+ TH2	Timer/Counter 2 High Byte	0CDH	0000 0000
+ TL2	Timer/Counter 2 Low Byte	0CCH	0000 0000
+ RCAP2H	T/C 2 Capture Reg. High Byte	0CBH	0000 0000
+ RCAP2L	T/C 2 Capture Reg. Low Byte	0CAH	0000 0000
* SCON	Serial Control	98H	0000 0000
SBUF	Serial Data Buffer	99H	Interminate
PCON	Power Control	87H	HMOS 0XXX XXXX CHMOS 0XXX 0000

8051 TIMER AND COUNTER

TCON and TMOD Function Registers

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

THE TIMER CONTROL (TCON) SPECIAL FUNCTION REGISTER

Bit	Symbol	Function
7	TF1	Timer 1 Overflow flag. Set when timer rolls from all ones to zero. Cleared when processor vectors to execute interrupt service routine located at program address 001Bh.
6	TR1	Timer 1 run control bit. Set to 1 by program to enable timer to count; cleared to 0 by program to halt timer. Does not reset timer.
5	TF0	Timer 0 Overflow flag. Set when timer rolls from all ones to zero. Cleared when processor vectors to execute interrupt service routine located at program address 000Bh.
4	TR0	Timer 0 run control bit. Set to 1 by program to enable timer to count; cleared to 0 by program to halt timer. Does not reset timer.
3	IE1	External interrupt 1 edge flag. Set to 1 when a high to low edge signal is received on port 3 pin 3.3 ($\overline{\text{INT1}}$). Cleared when processor vectors to interrupt service routine located at program address 0013h. Not related to timer operations.
2	IT1	External interrupt 1 signal type control bit. Set to 1 by program to enable external interrupt 1 to be triggered by a falling edge signal. Set to 0 by program to enable a low level signal on external interrupt 1 to generate an interrupt.
1	IE0	External interrupt 0 edge flag. Set to 1 when a high to low edge signal is received on port 3 pin 3.2 ($\overline{\text{INT0}}$). Cleared when processor vectors to interrupt service routine located at program address 0003h. Not related to timer operations.
0	IT0	External interrupt 0 signal type control bit. Set to 1 by program to enable external interrupt 0 to be triggered by a falling edge signal. Set to 0 by program to enable a low level signal on external interrupt 0 to generate an interrupt.

7	6	5	4	3	2	1	0
Gate	C/$\overline{T}$	M1	M0	Gate	C/$\overline{T}$	M1	M0
[Timer 1			]	[Timer 0			]

THE TIMER MODE CONTROL (TMOD) SPECIAL FUNCTION REGISTER

Bit	Symbol	Function
7/3	Gate	OR gate enable bit which controls RUN/STOP of timer 1/0. Set to 1 by program to enable timer to run if bit TR1/0 in TCON is set and signal on external interrupt $\overline{\text{INT1}}$/0 pin is high. Cleared to 0 by program to enable timer to run if bit TR1/0 in TCON is set.
6/2	C/$\overline{T}$	Set to 1 by program to make timer 1/0 act as a counter by counting pulses from external input pins 3.5 (T1) or 3.4 (T0). Cleared to 0 by program to make timer act as a timer by counting internal frequency.
5/1	M1	Timer/counter operating mode select bit 1. Set/cleared by program to select mode.
4/0	M0	Timer/counter operating mode select bit 0. Set/cleared by program to select mode.

M1	M0	Mode
0	0	0
0	1	1
1	0	2
1	1	3

Many microcontroller applications require the counting of external events, such as the frequency of a pulse train, or the generation of precise internal time delays between computer actions. Both of these tasks can be accomplished using software techniques, but software loops for counting or timing keep the processor occupied so that other, perhaps more important, functions are not done. To relieve the processor of this burden, two 16-bit *up* counters, named T0 and T1, are provided for the general use of the programmer. Each counter may be programmed to count internal clock pulses, acting as a timer, or programmed to count external pulses as a counter.

The counters are divided into two 8-bit registers called the timer low (TL0, TL1) and high (TH0, TH1) bytes. All counter action is controlled by bit states in the timer mode control register (TMOD), the timer/counter control register (TCON), and certain program instructions.

TMOD is dedicated solely to the two timers and can be considered to be two duplicate 4-bit registers, each of which controls the action of one of the timers. TCON has control bits and flags for the timers in the upper nibble, and control bits and flags for the external interrupts in the lower nibble. Figure 2.10 shows the bit assignments for TMOD and TCON.

Timer Counter Interrupts

The counters have been included on the chip to relieve the processor of timing and counting chores. When the program wishes to count a certain number of internal pulses or external events, a number is placed in one of the counters. The number represents the maximum count *less* the desired count, plus one. The counter increments from the initial number to the maximum and then rolls over to zero on the final pulse and also sets a timer flag. The flag condition may be tested by an instruction to tell the program that the count has been accomplished, or the flag may be used to interrupt the program.

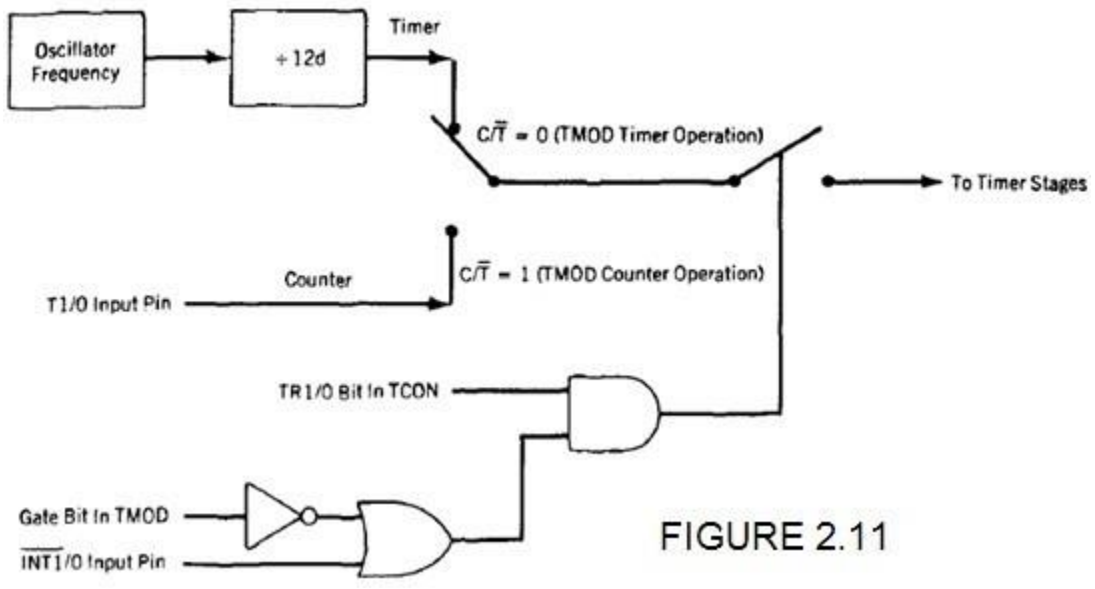

Oscillator Frequency
÷12d
Timer
C/$\overline{T}$ = 0 (TMOD Timer Operation)
To Timer Stages
Counter
C/$\overline{T}$ = 1 (TMOD Counter Operation)
T1/0 Input Pin
TR1/0 Bit In TCON
Gate Bit In TMOD
$\overline{INT1}$/0 Input Pin

FIGURE 2.11

Timing

If a counter is programmed to be a timer, it will count the internal clock frequency of the 8051 oscillator divided by 12d. As an example, if the crystal frequency is 6.0 megahertz, then the timer clock will have a frequency of 500 kilohertz.

The resultant timer clock is gated to the timer by means of the circuit shown in Figure 2.11. In order for oscillator clock pulses to reach the timer, the C/$\overline{T}$ bit in the TMOD register must be set to 0 (timer operation). Bit TRX in the TCON register must be set to 1 (timer run), and the *gate* bit in the TMOD register must be 0, or external pin $\overline{INTX}$ must be a 1. In other words, the counter is configured as a timer, then the timer pulses are gated to the counter by the run bit *and* the gate bit *or* the external input bits $\overline{INTX}$.

Timer Modes of Operation

The timers may operate in any one of four modes that are determined by the mode bits, M1 and M0, in the TMOD register. Figure 2.12 shows the four timer modes.

Timer Mode 0

Setting timer X mode bits to 00b in the TMOD register results in using the THX register as an 8-bit counter and TLX as a 5-bit counter; the pulse input is divided by 32d in TL so that TH counts the original oscillator frequency reduced by a total 384d. As an example, the 6 megahertz oscillator frequency would result in a final frequency to TH of 15625 hertz. The timer flag is set whenever THX goes from FFh to 00h, or in .0164 seconds for a 6 megahertz crystal if THX starts at 00h.

Timer Mode 1

Mode 1 is similar to mode 0 except TLX is configured as a full 8-bit counter when the mode bits are set to 01b in TMOD. The timer flag would be set in .1311 seconds using a 6 megahertz crystal.

Timer Mode 2

Setting the mode bits to 10b in TMOD configures the timer to use only the TLX counter as an 8-bit counter. THX is used to hold a value that is loaded into TLX every time TLX overflows from FFh to 00h. The timer flag is also set when TLX overflows.

This mode exhibits an auto-reload feature: TLX will count up from the number in THX, overflow, and be initialized again with the contents of THX. For example, placing 9Ch in THX will result in a delay of exactly .0002 seconds before the overflow flag is set if a 6 megahertz crystal is used.

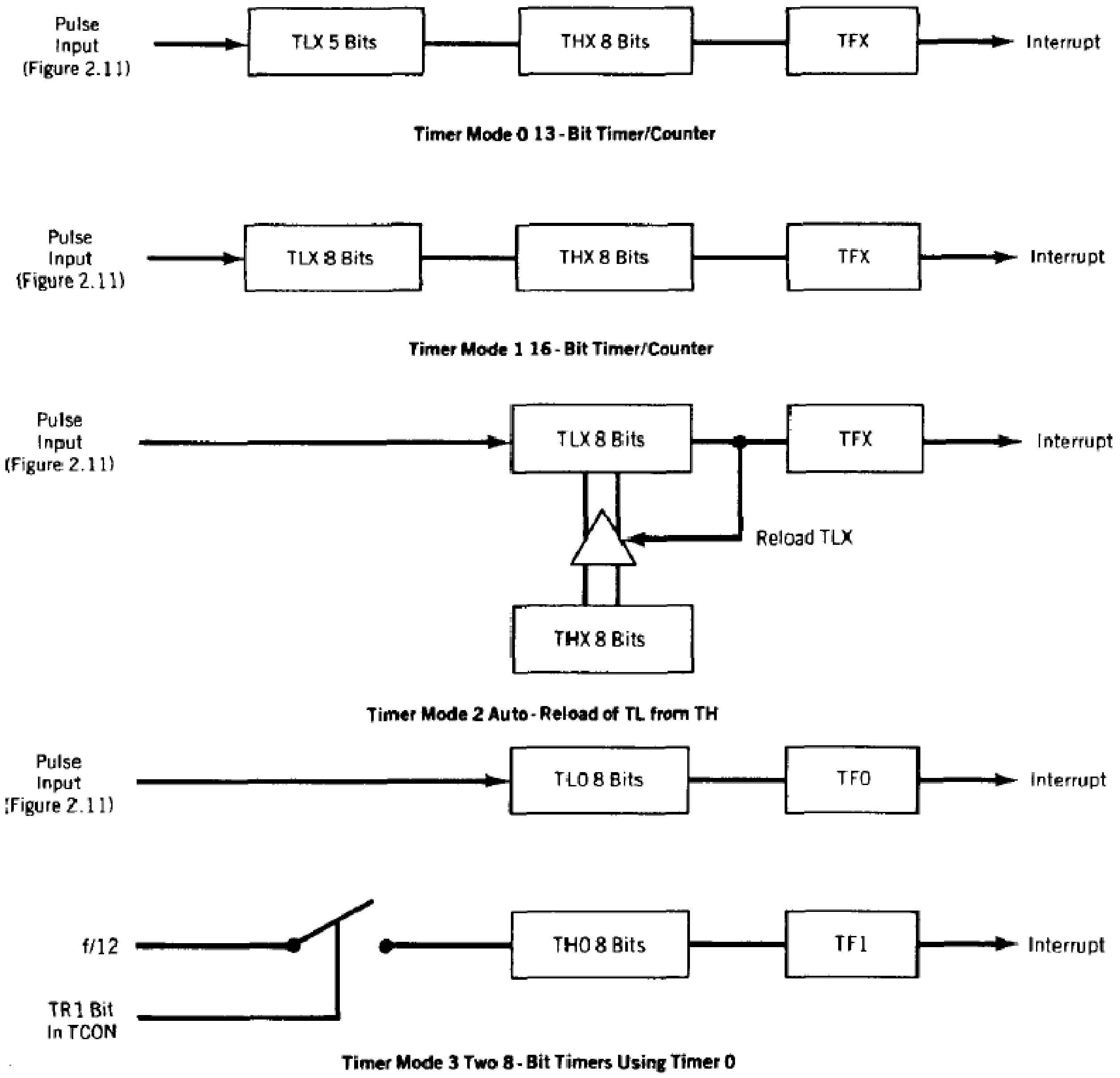
Timer Mode 3

Timers 0 and 1 may be programmed to be in mode 0, 1, or 2 independently of a similar mode for the other timer. This is not true for mode 3; the timers do not operate independently if mode 3 is chosen for timer 0. Placing timer 1 in mode 3 causes it to stop counting; the control bit TR1 and the timer 1 flag TF1 are then used by timer 0.

Timer 0 in mode 3 becomes two completely separate 8-bit counters. TL0 is controlled by the gate arrangement of Figure 2.11 and sets timer flag TF0 whenever it overflows from FFh to 00h. TH0 receives the timer clock (the oscillator divided by 12) under the control of TR1 only and sets the TF1 flag when it overflows.

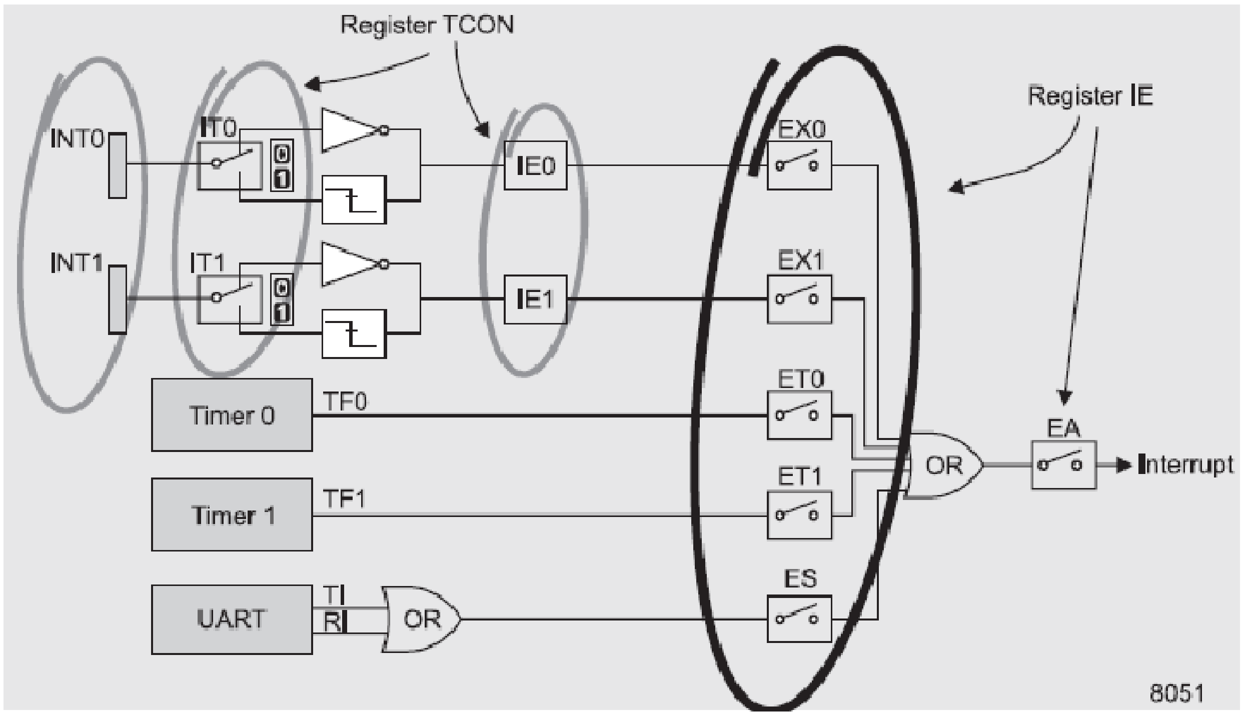
Timer 1 may still be used in modes 0, 1, and 2, while timer 0 is in mode 3 with one important exception: *No interrupts* will be generated by timer 1 while timer 0 is using the TF1 overflow flag. Switching timer 1 to mode 3 will stop it (and hold whatever count is in timer 1). Timer 1 can be used for baud rate generation for the serial port, or any other mode 0, 1, or 2 function that does not depend upon an interrupt (or any other use of the TF1 flag) for proper operation.

FIGURE 2.12 Timer 1 and Timer 0 Operation Modes

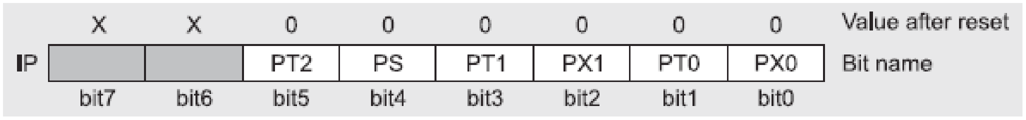

Pulse Input (Figure 2.11)
TLX 5 Bits
THX 8 Bits
TFX
Interrupt
Timer Mode 0 13 - Bit Timer/Counter
Pulse Input (Figure 2.11)
TLX 8 Bits
THX 8 Bits
TFX
Interrupt
Timer Mode 1 16 - Bit Timer/Counter
Pulse Input (Figure 2.11)
TLX 8 Bits
TFX
Interrupt
Reload TLX
THX 8 Bits
Timer Mode 2 Auto - Reload of TL from TH
Pulse Input (Figure 2.11)
TL0 8 Bits
TF0
Interrupt
f/12
TH0 8 Bits
TF1
Interrupt
TR1 Bit In TCON
Timer Mode 3 Two 8 - Bit Timers Using Timer 0

8051 Microcontroller Interrupts

There are five interrupt sources for the 8051, which means that they can recognize 5 different events that can interrupt regular program execution. Each interrupt can be enabled or disabled by setting bits in the IE register. Also, as seen from the picture below the whole interrupt system can be disabled by clearing bit EA from the same register. Namely, if the bits IT0 and IT1 stored in the TCON register areset, program interrupt will occur on changing logic state from 1 to 0, (only at the moment). If these bits are cleared, the same signal will generate interrupt request and it will be continuously executed as far as the pins are held low.



IE Register (Interrupt Enable)



	X	X	0	0	0	0	0	0	Value after reset
IP			PT2	PS	PT1	PX1	PT0	PX0	Bit name
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	

Interrupt Enable

- **EA** - bit enables or disables all other interrupt sources (globally)
  - 0 - (when cleared) any interrupt request is ignored (even if it is enabled)
  - 1 - (when set to 1) enables all interrupts requests which are individually enabled
- **ES** - bit enables or disables serial communication interrupt (UART)
  - 0 - UART System cannot generate interrupt
  - 1 - UART System enables interrupt
- **ET1** - bit enables or disables Timer 1 interrupt
  - 0 - Timer 1 cannot generate interrupt
  - 1 - Timer 1 enables interrupt
- **EX1** - bit enables or disables INT 0 pin external interrupt
  - 0 - change of the pin INT0 logic state cannot generate interrupt
  - 1 - enables external interrupt at the moment of changing the pin INT0 state
- **ET0** - bit enables or disables timer 0 interrupt

    - 0 - Timer 0 cannot generate interrupt
    - 1 - enables timer 0 interrupt
- **EX0** - bit enables or disables INT1 pin external interrupt
    - 0 - change of the INT1 pin logic state cannot cause interrupt
    - 1 - enables external interrupt at the moment of changing the pin INT1 state

Interrupt Priorities

It is not possible to predict when an interrupt will be required. For that reason, if several interrupts
are enabled. It can easily occur that while one of them is in progress, another one is requested. In such situation, there is a priority list making the microcontroller know whether to continue operating or meet a new interrupt request.
The priority list consists of 3 levels:

1. Reset! The absolute master of the situation. If an request for Reset omits, everything is stopped and the microcontroller starts operating from the beginning.

2. Interrupt priority 1 can be stopped by Reset only.

3. Interrupt priority 0 can be stopped by both Reset and interrupt priority 1.

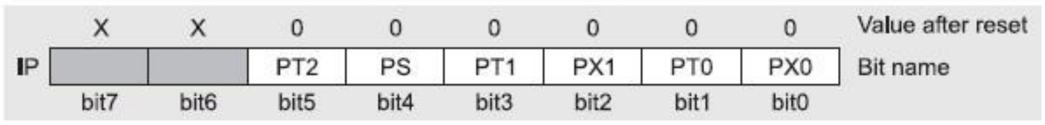
Which one of these existing interrupt sources has higher and which one has lower priority is defined in the IP Register (Interrupt Priority Register). It is usually done at the beginning of the program. According to that, there are several possibilities:

- Once an interrupt service begins. It cannot be interrupted by another interrupt at the same or lower priority level, but only by a higher priority interrupt.
- If two interrupt requests, at different priority levels, arrive at the same time then the higher priority interrupt is serviced first.
- lf the both interrupt requests, at the same priority level, occur one after another, the onewho came later has to wait until routine being in progress ends.
- If two interrupts of equal priority requests arrive at the same time then the interrupt to be serviced is selected according to the following priority list :

1. External interrupt INT0
2. Timer 0 interrupt
3. External Interrupt INT1
4. Timer 1 interrupt
5. Serial Communication Interrupt

IP Register (Interrupt Priority)

The IP register bits specify the priority level of each interrupt (high or low priority).



	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	
	X	X	0	0	0	0	0	0	Value after reset
IP			PT2	PS	PT1	PX1	PT0	PX0	Bit name

- PS - Serial Port Interrupt priority bit
    1. Priority 0
    2. Priority 1

- PT1 - Timer 1 interrupt priority
1. Priority 0
2. Priority 1

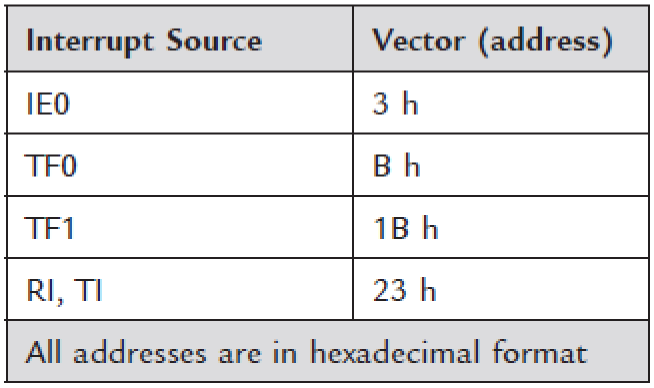
- PX1 - External Interrupt INT1 priority
1. Priority 0
2. Priority 1

- PT0 - Timer 0 Interrupt Priority
1. Priority 0
2. Priority 1

- PX0 - External Interrupt INT0 Priority
1. Priority 0
2. Priority 1

Handling Interrupt

Once some of interrupt requests arrives, everything occurs according to the following order:
1. Instruction in progress is ended
2. The address of the next instruction to execute is pushed on the stack
3. Depending on which interrupt is requested, one of 5 vectors (addresses) is written to the program counter in accordance to the following table:



Interrupt Source	Vector (address)
IE0	3 h
TF0	B h
TF1	1B h
RI, TI	23 h
All addresses are in hexadecimal format	

4. The appropriate subroutines processing interrupts should be located at these addresses. Instead of them, there are usually jump instructions indicating the location where the subroutines reside.
5. When interrupt routine is executed, the address of the next instruction to execute is poped from the stack to the program counter and interrupted program continues operating from where it left off.

ADDRESSING MODES

The instructions of 8051 may be classified based on the source or destination type

1. Register addressing
2. Direct addressing
3. Register-indirect addressing
4. Immediate addressing
5. Base register plus index register indirect addressing

Register Addressing
Register addressing permits access to eight registers (R0–R7) of the register bank. There are four banks of eight registers. One of the four banks is selected by a 2-bit field in PSW (Program Status Word). Other registers used are A, B, AB and DPTR.
For example,

```
MOV A, Rn       ;Move contents of register Rn to accumulator.
```

Hence, after the execution of this instruction, the registers Rn and A have the original contents of register Rn.

Direct Addressing

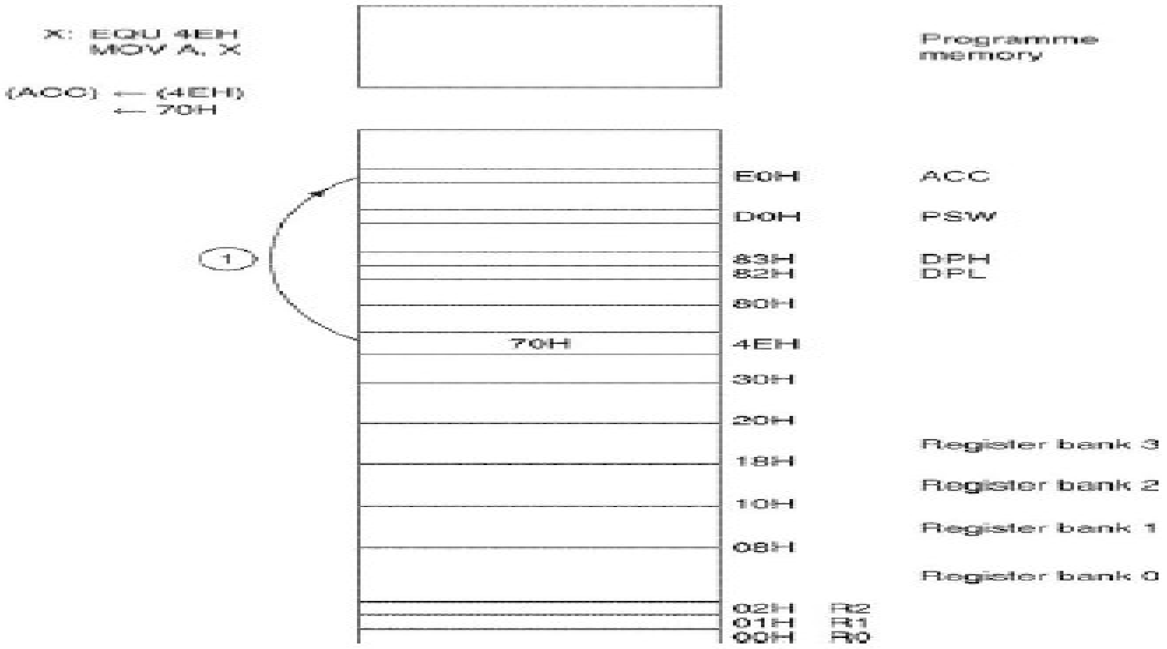
In direct addressing, the address of the operand is specified in the instruction. Direct addressing has operands as byte or bit. Direct addressing of byte provides operation on one of the following.

1. Lower 128 bytes of internal data RAM
2. Special function registers

Direct bit addressing provides operation on the following.

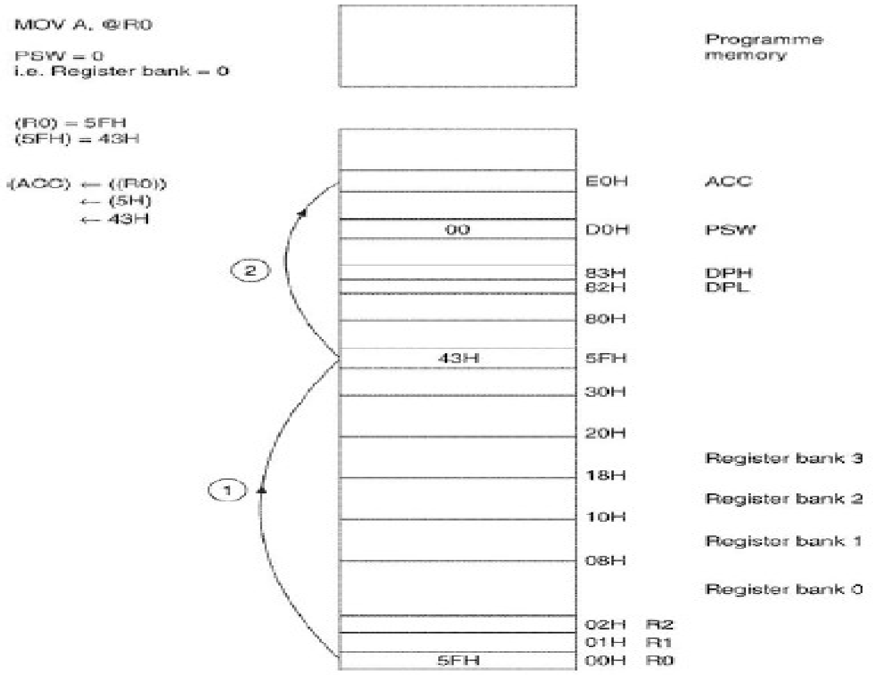
1. 128 bits subset of internal data RAM (20H to 2FH)

2. 128 bits subset of special function register address space (80H to FFH)



Indirect Addressing

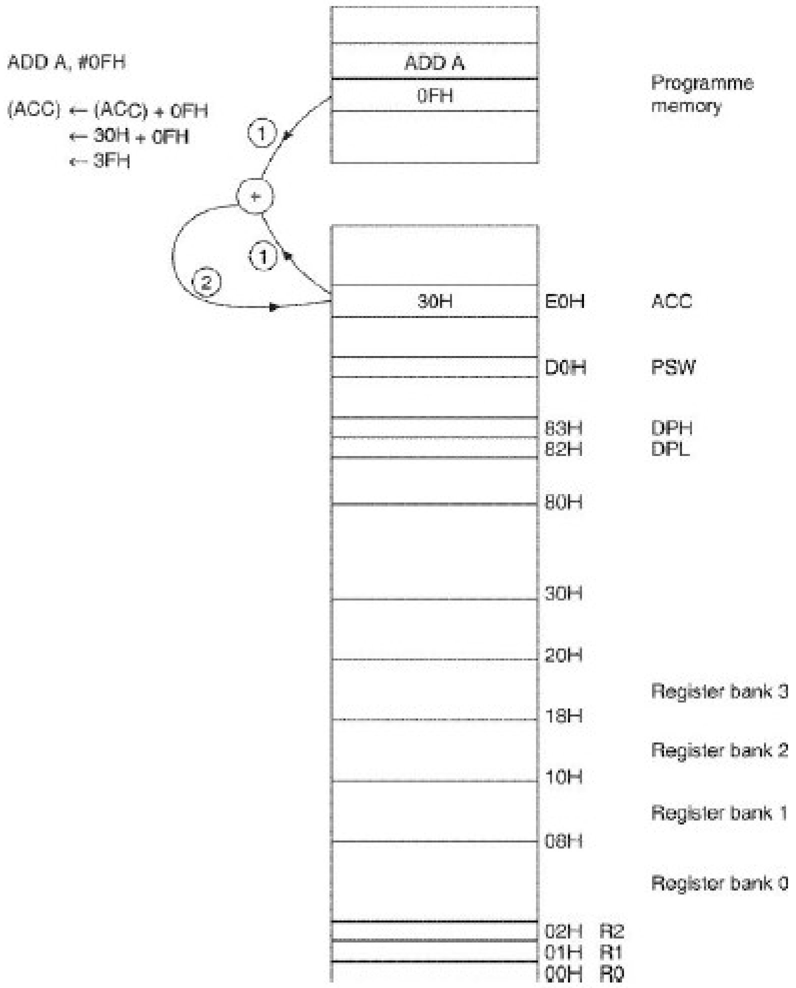
In this addressing, the address of the operand is not specified directly. Instead, the address of the operand is specified as the contents of the register mentioned in the instruction. For example, in the instruction MOV A, @Ri, the contents of register Ri give the address of the location from where the operand is picked up. The above instruction causes the operand to be picked up from the mentioned location for transfer to register A. The operation sequence has been explained in Figure for the instruction MOV A, @R0.



This type of instruction is useful when the same set of operations is performed on different data sets stored in memory. In the present case either R0 or R1 may be used for accessing the locations within the 256 byte locations. However, register indirect addressing is also used for accessing external data memory. The 16-bit pointer (DTPR) can be used for accessing any location within the full 64 KB memory space.

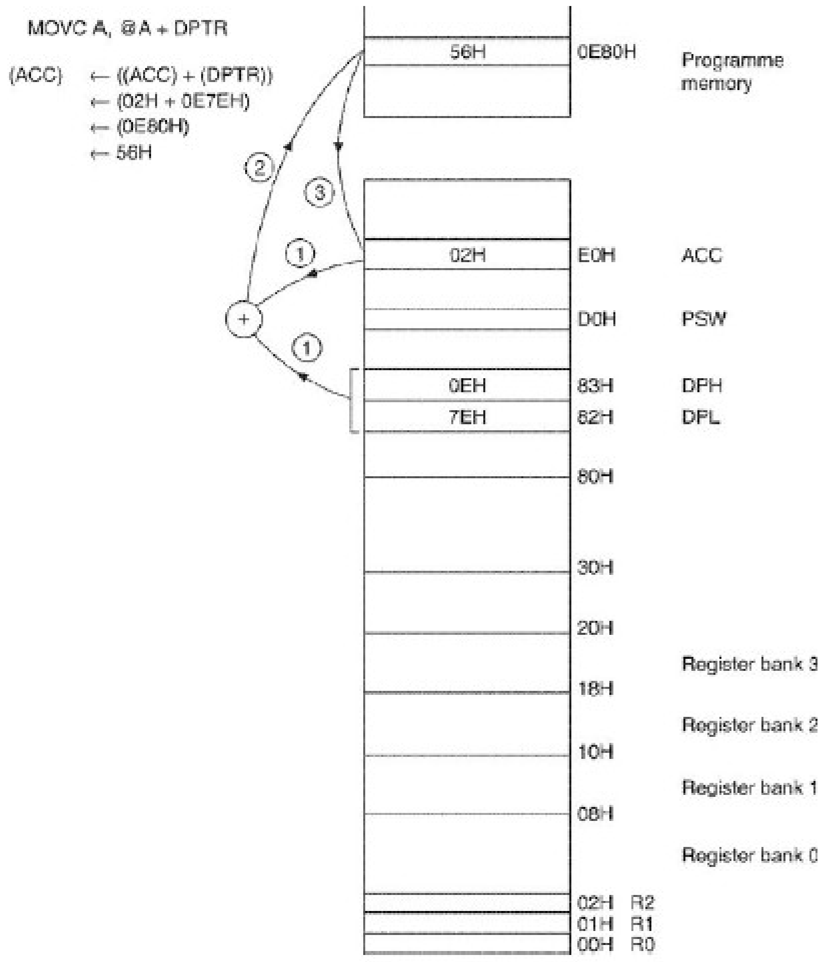
Immediate Addressing

In this case the operand on which the operation has to be performed according to the instruction, is specified in the instruction itself. For example, the instruction ADD A, #data, adds the 8-bit data specified in the instruction to the contents of register A and the result is placed in register A itself .



Base Register Plus Index Register Indirect Addressing

This is an indirect instruction used to access the program memory. In the instruction the operand on which the operation is to be performed, is not specified directly. The summation of contents of base register and index register determines the operand address. The DPTR or PC register may act as the base register and register A acts as the index register. Thus the base register (DPTR or PC) and the index register A are added to get the memory location. The operand is the contents of the memory location calculated above. For example, the instruction MOVC A, @ A+DPTR. moves a byte from a specified address to register A. The address of the byte to be moved in register A is the sum of the original 8 bits of accumulator contents and 16 bits of base register (which is the DPTR). The operation sequence for instruction MOVC A, @A+DPTR is explained in Figure BELOW.

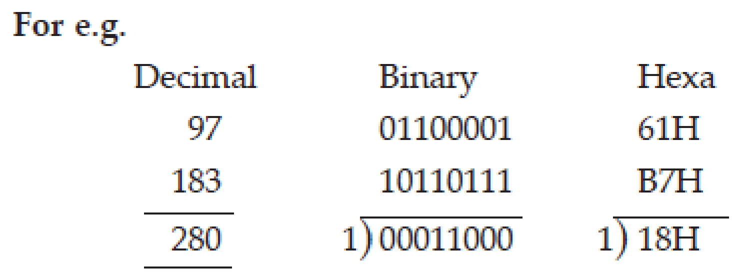


8051 Assembly Language Programming

ARITHMETIC INSTRUCTION

Unsinged and signed Addition:
Number may be unsigned or signed number for addition signed numbers use 7th bit as a sign it(MSB) remaining 0 to 6th bit, expresses the magnitude of the number for signed number 7th bit shows one for negative sign and shows zero for positive sign.

Unsigned Addition
Unsigned numbers make use of the carry flag to detect when the reset an ADD operation is a number larger than FFh. If the carry is set to 1 after an ADD, then the carry can be added to a higher order byte. So that the sum is not lost,


For e.g.

Decimal	Binary	Hexa
97	01100001	61H
183	10110111	B7H
280	1) 00011000	1) 18H

Signed Addition

Signed numbers may be added two ways additions of like signed number and addition of unlike signed numbers. If unlike signed numbers are added then it is not possible for the result to be larger than – 128d or +127d, and the sign of the result will always be corrected.

ADD A, <src-bytes>

Function: ADD

Description: ADD adds the byte variables indicated to the accumulator, leaving the result in the accumulator. The carry and auxiliary carry flags are set, respectively. If there is carry out from bit-7 or bit-3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands.

Four source operand addressing modes are allowed:

Register, direct, register-direct, immediate.

byte: 1 OR 2

cycle: 1

Example: The accumulator holds 89H (10001001B) and register 0 holds 95H (10010101B) .The instruction,

ADD A, R0

Will leave 1E (00011110B) in the accumulator with the AC flag cleared and both the carry flag and OV set to 1.

ADD A, direct

Operation: (A) ← (A) + (direct)

Example: ADD A, 20H: Adds contents of A and memory whose address is 20H and store result in A.

2 bytes instruction

1 machine cycle

ADD A, @Ri
Operation: (A) ← (A) + ((Ri))
Example: ADD A, @R2; Adds contents of A and memory whose address is given by register R2, and stores result in A.
1 byte instruction
1 machine cycle

ADD A, #data
Operation: (A) ← (A) + #data
Example: ADD A, # 20H; Adds the contents of A and 20H.
2 bytes instruction
1 machine cycle

ADDC A, <src – byte>
Function: Add with carry
Description: ADDC simultaneously adds the byte variable indicated, the carry flag and the accumulator contents, leaving the result in the accumulator. The carry and auxiliary – carry flags
are set respectively, if there is a carry-out from bit-7 or bit-3 and cleared otherwise. When adding
unsigned integers, the carry flag indicates an overflow occurred.
OV is set if there is a carry-out of bit-6 but now of bit-7, or a carry-out of bit-7 but not out of bit-6;
Otherwise, OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands.
Four source operand addressing modes are allowed: register, direct, register – indirect, or immediate.

BYTE: 1 OR 2
1 machine cycle

ADDC A, R0
Will leave 6EH (01101110B) in the accumulator with AC cleared and both the carry flag and or set to 1.
ADDC A, Rn
Operation: (A) ← (A) + (C) + (Rn)
1 byte instruction
1 machine cycle
ADDC A, direct
Operation: (A) ← (A) + (C) + (direct)
Example: ADDC A, 20H; adds the contents of A, memory location whose address is 20H and the carry flag and stores result in A.
2 bytes instruction
1 machine cycle

ADDC A, @Ri
Operation: (A) ← (A) + (C) + ((Ri))
Example: ADDC A, @R2; Adds the contents of A, memory location whose address is given by register R2 and the carry flag and stores result in the A.

1 byte instruction
1 machine cycle

ADDC A, #data
Operation: (A) ← (A) + (C) + #data
Example: ADDC A, #20H; Adds the contents of A and carry flag and 20H and stores result in A.

2 bytes instruction
1 machine cycle

SUBB A, <src-byte>
Function: Subtract with borrow.
Description: SUBB subtracts the indicated variable and the carry flag together from the accumulator, leaving the result in the accumulator, SUBB sets the carry (borrow) flag if a borrow is needed for bit-7 and clears otherwise. If C was set before executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction. So, the carry is subtracted from the accumulator along with the source operand1. AC is set if a borrow is needed for bit 3 and cleared otherwise. OV is set if a borrow is needed into bit-6, but Not into bit-7 or into bit but not bit-6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value or a positive result when a positive number is subtracted from a negative number. The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

SUBB A, Rn
Operation: (A) ← (A) – (C) – (Rn)
Example: SUBB A, R3; Subtracts contents of R3 and carry together from A and stores results in A.
1 byte instruction
1 machine cycle

SUBB A, direct
Operation: (A) ← (A) – (C) – (direct)
Example: SUBB A, 20H; Subtracts the contents of memory location 20H and carry together from A and stores result in A.
2 bytes instruction
1 machine cycle
SUBB A, @Ri
Operation: (A) ← (A) – (C) – ((Ri))
Example: SUBB A, @R2; subtracts the contents of memory location whose address is given by R2 and carry together from A and stores result in A.
1 byte instruction
1 machine cycle

SUBB A, #data
Operation: (A) ← (A) – (C) - #data
Example: SUBB A, #20H; Subtracts 20H from A and stores result in A.

2 bytes instruction
1 machine cycle

Increment And Decrement InstructionS
INC<byte>
Function: Increment
Description: INC increment the indicated variable by 1. An original value of 0FFH will overflow to 00H. No flags are affected. Three addressing modes are allowed. Register, direct or register-indirect.
Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

INC Rn
Operation: (A) ← (A) + 1
Rn → 5F

Before Execution	After Execution
Rn	Rn
5F	60

INC direct
Operation: INC
(direct) ← (direct) + 1
Add a 1 to content of the direct memory address. Assume initially the internal memory

location 25H contains data 60H. After executing INC 25H, the location will have the updated value 26H.
2 bytes instruction.
1 machine cycle.

INC @Ri
Operation: ((Ri)) ← ((Ri)) +1
Example: INC @ R1; increment contents of memory location whose address is given by register R1 by 1.

Note :Ri may be R0 or R1
Assume the pointer R0 contains 42H and the internal memory location 42 contains 2AH. After executing INC @ R0, the internal memory location 42H will have updated value 2BH.
1 byte instruction.
1 machine cycle.
INC DPTR
Function: Increment Data pointer
Description: Increment the 16-bit data pointers by 1. A 16-bit increment (modulo 216) is performed; an overflow of the low order byte of the data pointer (DPL) from FFH to 00H will increment the high-order byte DPH. No flags are affected.

Example: Registers DPH and DPL contain 12H and FEH respectively. The instruction sequence,
INC DPTR
INC DPTR
INC DPTR
Will change DPH and DPL to 13H and 01H
Operation: (DPTR) ← (DPTR) + 1
1 byte instruction.
2 machine cycles.

DEC byte
Function: Decrement
Description: The variable indicate is decremented by 1. An original value of 00H will underflow to 0FFH. No flags are affected. Four operand addressing modes are allowed: accumulator, register-direct, register-indirect.
Note: Register0 contains 7FH (01111111B). Internal RAM locations 7EH and 7FH contain 00H and 40H respectively. The instruction sequence,
DEC @R0
DEC R0
DEC @R0
Will leave register 0 set to 7EH and internal RAM locations 7EH & 7FH and 7FH set to 0FFH and 3FH.

DEC A
Operation: (A) ← (A)–1
Subtract a 1from accumulator
Let us assume A = 23H, after executing the instruction DEC A, there will contain 22H.
1 byte instruction.
1 machine cycle.
DEC Rn
Operation: (Rn) ← (Rn)–1
Example: DEC R3; decrements the contents of R3 by 1

Before Execution	After Execution
R3	R3
3E	3D

1 byte instruction.
1 machine cycle

DEC direct
Operation: (direct) ← (direct) –1
Example: DEC 20H; decrements the contents of memory location whose address is 20H
by 1.
2 bytes instruction.
1 machine cycle
DEC @Rn
Operation: ((R1)) ← ((Ri)) – 1
Example: DEC @R2; decrements the contents of memory location whose address is given by register R2 by 1.
1 byte instruction.
1 machine cycle.
Note:
1. No math flags are affected.
2. All 8-bit address contents overflow FFh to 00h.
3. DPTR is 16-bit; DPTR overflows from FFFFh to 0000h.
4. The 8-bit address contents underflow from 00h to FFh.
5. There is no DEC DPTR to match the INC DPTR.

MULTIPLICATION AND DIVISION

MUL AB
Function: Multiply
Description: MUL AB the unsigned eight bit integers in the accumulators and register B. The low-order byte of the sixteen-bit product is left in the accumulator and the high-order byte in B. If the product is greater than 255(FFH) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.

Example: Originally, the accumulator holds the value 80(50H). Register B holds the value 160(0A0H).
The instruction,
MUL AB
Will give the product 12,800(3200H), 80B is changed to 32H (00110010B) and the accumulator is cleared. The overflow flag is set, carry is cleared.

Operation: $(A)_{7-0} \leftarrow (A) \times (B)$
$(B)_{15-8}$

$A = 58H$
$B = 11H$

Before Execution		After Execution	
A	B	A	B
58	11H	D8	05

MOV A, #12
MOV B, #05
MUL AB
;A = 5A; B = 00
1 byte instruction.
4 machine cycles.

DIV AB
Function: Divide
Description: DIV AB divides the unsigned eight bit integer in the accumulator by the unsigned eight-bit integer in register B. The accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and or flags will be cleared.

Exception if B had originally contained 00H, the values returned in the accumulator and B register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

Example: The accumulator contains 250(0FBH or 11111010B) and B contains 18(12H or 00010010B). The instruction,
DIV AB
Will leave B in the accumulator (0DH or 00001101B) and the value 16(10H or 00010000B) in B. Since, 250 = (13 × 18) + 16 carry and OV will both be cleared.

<u>Logical Instructions</u>

Logical AND

ANL destination, source: ANL does a bitwise "AND" operation between *source* and *destination*, leaving the resulting value in *destination*. The value in source is not affected. "AND" instruction logically AND the bits of source and destination.

ANL A,#DATA ANL A, Rn
ANL A,DIRECT ANL A,@Ri
ANL DIRECT,A ANL DIRECT, #DATA

Logical OR

ORL destination, source; ORL does a bitwise "OR" operation between source and destination,

leaving the resulting value in destination. The value in source is not affected. " OR " instruction logically OR the bits of source and destination.
ORL A,#DATA ORL A, Rn
ORL A,DIRECT ORL A,@Ri
ORL DIRECT,A ORL DIRECT, #DATA

Logical Ex-OR

XRL destination, source: XRL does a bitwise "EX-OR" operation between source and destination, leaving the resulting value in destination. The value in source is not affected.

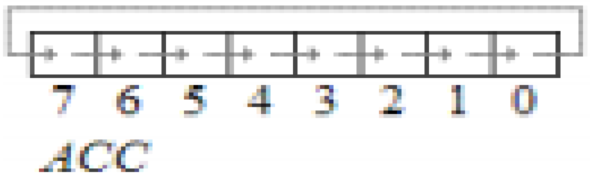
" XRL" instruction logically EX-OR the bits of source and destination.
XRL A,#DATA XRL A,Rn
XRL A,DIRECT XRL A,@Ri
XRL DIRECT,A XRL DIRECT, #DATA

Logical NOT

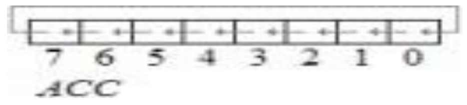
CPL complements operand, leaving the result in operand. If operand is a single bit then the state of the bit will be reversed. If operand is the Accumulator then all the bits in the Accumulator will be reversed.

CPL A, CPL C, CPL bit address

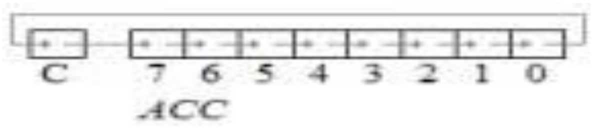
SWAP A – Swap the upper nibble and lower nibble of A
Rotate Instructions
RR A
This instruction is rotate right the accumulator. Its operation is illustrated below. Each bit is shifted one location to the right, with bit 0 going to bit 7.



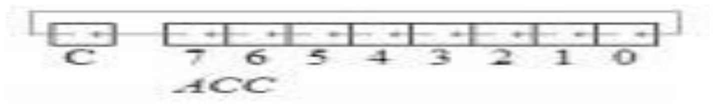
RL A
Rotate left the accumulator. Each bit is shifted one location to the left, with bit 7 going to bit 0



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

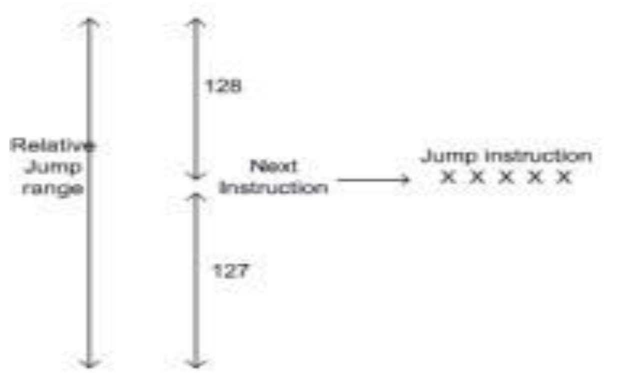


RLC A
Rotate left through the carry. Each bit is shifted one location to the left, with bit 7 going into the carry bit in the PSW, while the carry goes into bit 0.



Branch (JUMP) Instructions
Jump and Call Program Range
There are 3 types of jump instructions. They are:-
1. Relative Jump
2. Short Absolute Jump
3. Long Absolute Jump

Relative Jump
Jump that replaces the PC (program counter) content with a new address that is greater than (the address following the jump instruction by 127 or less) or less than (the address following the jump by 128 or less) is called a relative jump. Schematically, the relative jump can be shown as follows: -



The advantages of the relative jump are as follows:-
1. Only 1 byte of jump address needs to be specified in the 2's complement form, ie. For jumping ahead, the range is 0 to 127 and for jumping back, the range is -1 to -128.
2. Specifying only one byte reduces the size of the instruction and speeds up program execution.
3. The program with relative jumps can be relocated without reassembling to generate absolute jump addresses.

Disadvantages of the absolute jump: -
1. Short jump range (-128 to 127 from the instruction following the jump instruction)

Instructions that use Relative Jump
SJMP <relative address>; this is unconditional jump
The remaining relative jumps are conditional jumps
JC <relative address>
JNC <relative address>
JB bit, <relative address>
JNB bit, <relative address>
JBC bit, <relative address>
CJNE <destination byte>, <source byte>, <relative address>
DJNZ <byte>, <relative address>
JZ <relative address>
JNZ <relative address>

Short Absolute Jump
In this case only 11bits of the absolute jump address are needed. The absolute jump address is calculated in the following manner.

In 8051, 64 kbyte of program memory space is divided into 32 pages of 2 kbyte each. The hexadecimal addresses of the pages are given as follows:-

Page (Hex)	Address (Hex)
00	0000 - 07FF
01	0800 - 0FFF
02	1000 - 17FF
03	1800 - 1FFF
.	
.	
1E	F000 - F7FF
1F	F800 – FFFF

It can be seen that the upper 5bits of the program counter (PC) hold the page number and the lower

11bits of the PC hold the address within that page. Thus, an absolute address is formed by takingpage numbers of the instruction (from the program counter) following the jump and attaching the specified 11bits to it to form the 16-bit address.

Advantage: The instruction length becomes 2 bytes.
Example of short absolute jump: -
ACALL <address 11>
AJMP <address 11>

Long Absolute Jump/Call
Applications that need to access the entire program memory from 0000H to FFFFH use long absolute jump. Since the absolute address has to be specified in the op-code, the instruction length is 3 bytes (except for JMP @ A+DPTR). This jump is not re-locatable.

Example: -
LCALL <address 16>
LJMP <address 16>
JMP @A+DPTR
Another classification of jump instructions is
1. Unconditional Jump
2. Conditional Jump

1. The unconditional jump is a jump in which control is transferred unconditionally to the target location.
a. LJMP (long jump). This is a 3-byte instruction. First byte is the op-code and second and third bytes represent the 16-bit target address which is any memory location from 0000 to FFFFH
eg: LJMP 3000H
b. AJMP: this causes unconditional branch to the indicated address, by loading the 11 bit address to
0 -10 bits of the program counter. The destination must be therefore within the same 2K blocks.
c. SJMP (short jump). This is a 2-byte instruction. First byte is the op-code and second byte is the relative target address, 00 to FFH (forward +127 and backward -128 bytes from the current PC value). To calculate the target address of a short jump, the second byte is added to the PC value which is address of the instruction immediately below the jump

2. Conditional Jump instructions.

JBC	Jump if bit = 1 and clear bit
JNB	Jump if bit = 0
JB	Jump if bit = 1
JNC	Jump if CY = 0
JC	Jump if CY = 1
CJNE reg,#data	Jump if byte ≠ #data

CJNE A,byte Jump	if A ≠ byte
DJNZ	Decrement and Jump if A ≠ 0
JNZ	Jump if A ≠ 0
JZ	Jump if A ＝ 0

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

JB bit, rel ; jump if the direct bit is set to the relative address specified.
JNB bit, rel ; jump if the direct bit is clear to the relative address specified.
JBC bit, rel ; jump if the direct bit is set to the relative address specified and then clear the bit.

Subroutine CALL And RETURN Instructions
Subroutines are handled by CALL and RET instructions
There are two types of CALL instructions
1. LCALL address(16 bit)
This is long call instruction which unconditionally calls the subroutine located at the indicated 16 bit address. This is a 3 byte instruction. The LCALL instruction works as follows.
a. During execution of LCALL, [PC] = [PC]+3; (if address where LCALL resides is say, 0x3254; during execution of this instruction [PC] = 3254h + 3h = 3257h
b. [SP]=[SP]+1; (if SP contains default value 07, then SP increments and [SP]=08
c. [[SP]] = [PC7-0]; (lower byte of PC content ie., 57 will be stored in memory location 08.
d. [SP]=[SP]+1; (SP increments again and [SP]=09)
e. [[SP]] = [PC15-8]; (higher byte of PC content ie., 32 will be stored in memory location 09.
With these the address (0x3254) which was in PC is stored in stack.
f. [PC]= address (16 bit);the new address of subroutine is loaded to PC. No flags are affected.
2. ACALL address(11 bit)
This is absolute call instruction which unconditionally calls the subroutine located at the indicated 11 bit address. This is a 2 byte instruction. The SCALL instruction works as follows.

a. During execution of SCALL, [PC] = [PC]+2; (if address where LCALL resides is say, 0x8549; during execution of this instruction [PC] = 8549h + 2h = 854Bh

b. [SP]=[SP]+1; (if SP contains default value 07, then SP increments and [SP]=08

c. [[SP]] = [PC7-0]; (lower byte of PC content ie., 4B will be stored in memory location 08.
d. [SP]=[SP]+1; (SP increments again and [SP]=09)
e. [[SP]] = [PC15-8]; (higher byte of PC content ie., 85 will be stored in memory location 09.
With these the address (0x854B) which was in PC is stored in stack
f. [PC10-0]= address (11 bit); the new address of subroutine is loaded to PC. No flags are affected.

RET instruction

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

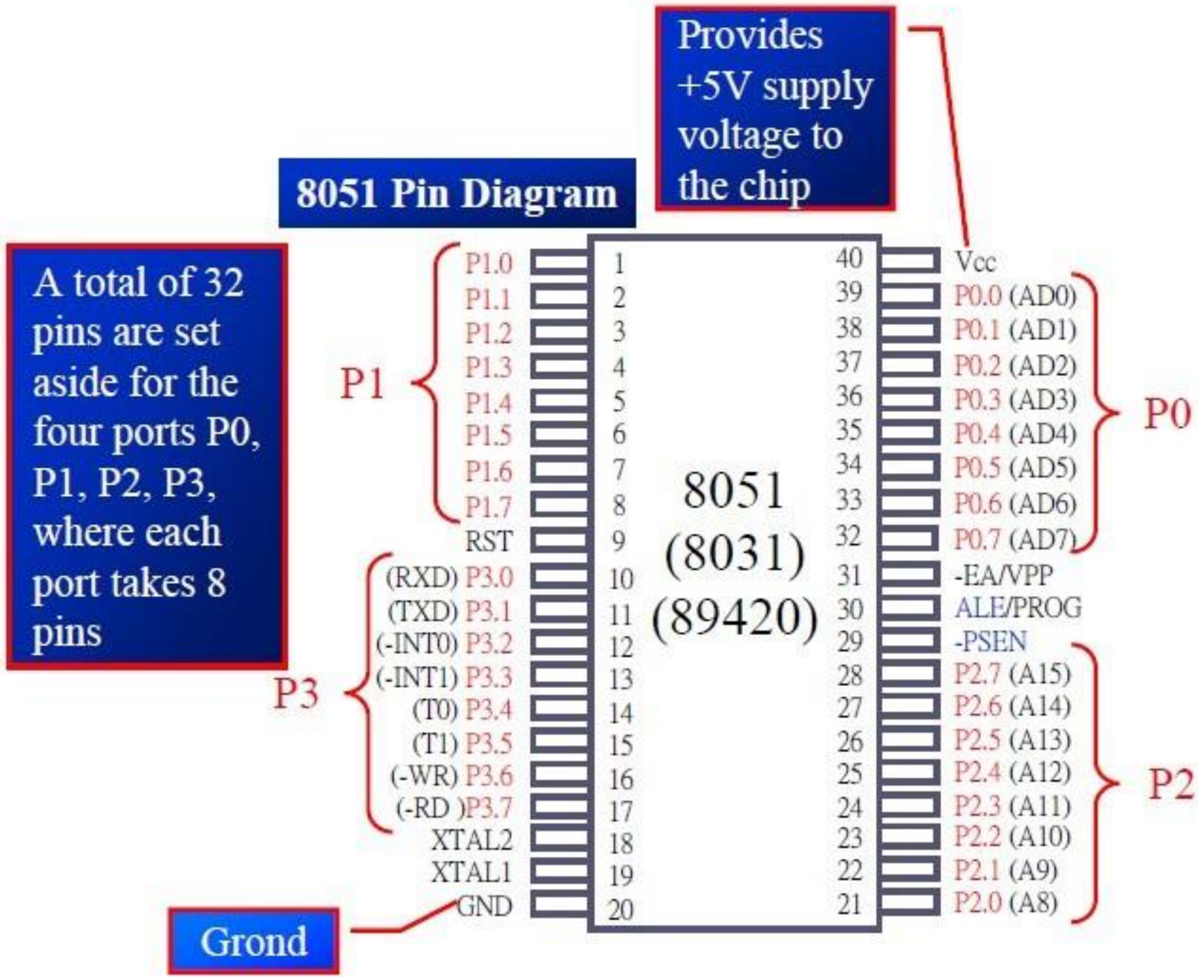
g. $[PC_{15-8}]$ = [[SP]] ;content of current top of the stack will be moved to higher byte of PC.

h. [SP]=[SP]-1; (SP decrements)

i. $[PC_{7-0}]$ = [[SP]] ;content of bottom of the stack will be moved to lower byte of PC.

j. [SP]=[SP]-1; (SP decrements again)

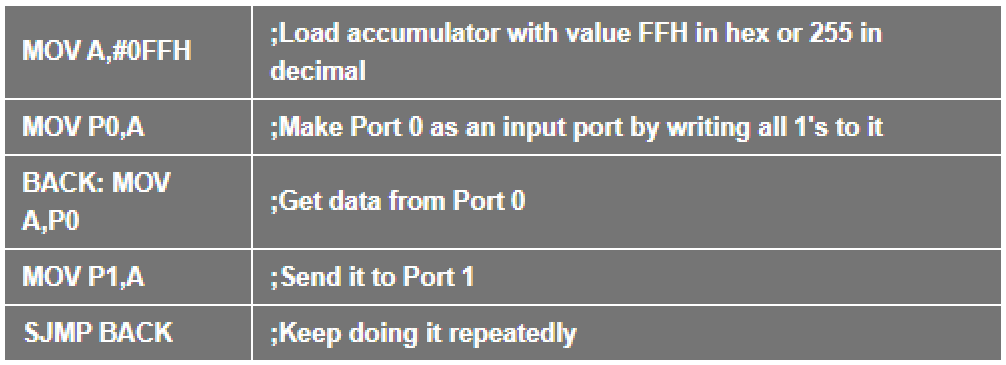
I/O Port Programming



- The four 8-bit I/O ports P0, P1, P2 and P3 each uses 8 pins.
- All the ports upon RESET are configured as output , ready to be used as output ports.
- To use all these ports as an input port, it must be programmed. Means all the bits of the port are needed to be written 1 on them.

PORT 0:
Port 0 has 8 pins P0.0, P0.1, P0.2, P0.3, P0.4, P0.5, P0.6, and P0.7, pins 39, 38, 37, 36, 35, 34, 33, 32. This port can be used for input or output. Each pin of the port 0 must be connected externally to a 10K Ohm pull-up resistor, to use Port 0 as both input or output ports. This is because the Port 0 [P0] is an open drain. This is not the case in other ports P1, P2 and P3. 'Open drain' is a term used for MOS chips, as 'open collector' is used for TTL chips. To use Port 0 for both input and output, have to connect Port 0 to pull-up resistors. When external pull-up resistors connected upon reset, Port 0 is configures as output port**.**

Role of Port 0 as Input Port:
To make Port 0 as input port, the pull up resistors are connected to port 0. The port must be programmed by writing 1 to all the bits. Lets examine the code example below in which the Port 0 is configured first as an input port by writing 1's to it and then data is received from that port and send to Port 1 [P1].



MOV A,#0FFH	;Load accumulator with value FFH in hex or 255 in decimal
MOV P0,A	;Make Port 0 as an input port by writing all 1's to it
BACK: MOV A,P0	;Get data from Port 0
MOV P1,A	;Send it to Port 1
SJMP BACK	;Keep doing it repeatedly

The Dual Role of Port 0:

Port 0 can be used to configures for both data and address. The Port 0 is also designated as AD0 - AD7. When connecting an 8051 to an external memory, port 0 provides both address and data. The 8051 multiplexes address and data through Port 0 to save pins. Address latch enable [ALE] indicates if Port 0 has address or data. When ALE=0, it provides data D0 - D7, but when ALE = 1, it has address A0 - A7. With the help of 74LS373 latch, ALE is used for demultiplexing address and data.

PORT 1:

Port 1 has a total of 8 pins. P1.0, P1.1, P1.2, P.13, P1.4, P1.5, P1.6, and P1.7. Port 1 can be use as an input or output. As compares to Port 0, this port does not need any pull-up resistors. Port 1 already has internally connected pull-up resistors. Port 1 is configured as a output port when reset.

Example # 1:

Lets examine the code, which will continuously send Output to Port 1 the alternating values 55H and AAH

MOV A,#55H
BACK MOV P1,A
ACALL DELAY
CPL A
SJMP BACK

The Role of Port 1 as Input Port:
The Port 1 must be programmed by writing 1's to all thr bits in order to make Port 1 as an input port.

EXAMPLE-2

MOV A,#0FFH	; Load Accumulator with FFH in hex
MOV P1,A	;Make P1 an input Port, by writing all 1's to Port 1
MOV A,P1	;Get data from Port 1
MOV R3,A	;Save data in register R3
ACALL DELAY	;Wait
MOV A,P1	;Get another data from Port 1
MOV R4,A	;Save data in register R4
ACALL DELAY	;Wait
MOV A,P1	;Get another data from Port 1
MOV R5,A	;Save data in register R5

In Example # 2 above, the Port 1 is configures as input port by writing 1's to it, then data is received from that port and saved in R3, R4 and R5

PORT 2:
Port 2 also has total of 8 pins. P2.0, P2.1, P2.2, P2.3, P2.4, P2.5, P2.6, and P2.7. Port 2 can be used for input or output port. To make Port 2 as an input, the Port 2 must be programmed by writing 1's to all bits.

Example # 3:

MOV A,#55H
BACK: MOV P2,A
ACALL DELAY
CPL A
SJMP BACK

In Example # 3 above, the code will send out continuously to Port 2 an alternating values 55H and AAH to toggle the bits or Port 2 continuously.
The Role of Port 2 as an Input Port:

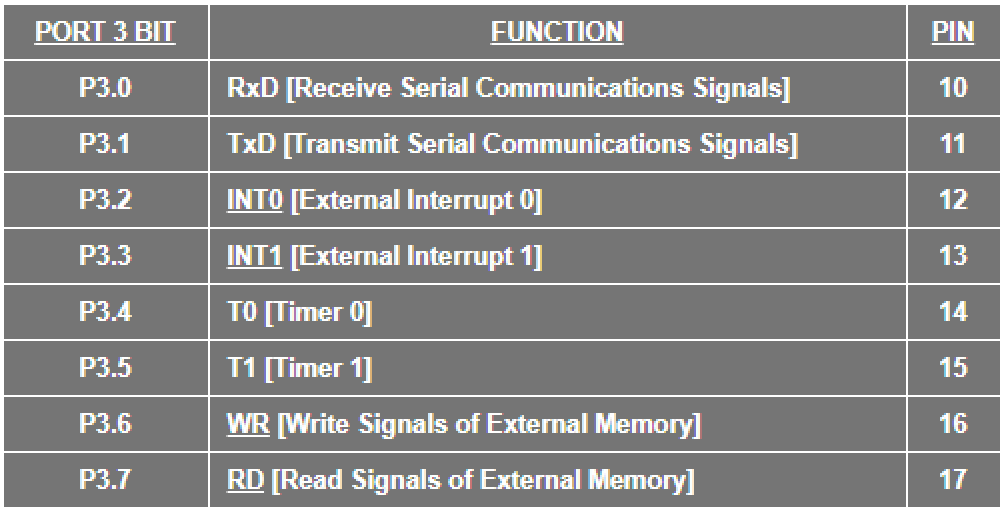
The Port 2 must be programmed by writing 1's to all the port 2 bits to make Port 2 as an input port.
Example #4:

MOV A,#0FFH	;Load accumulator with value 0FFH in hex
MOV P2,A	;Make Port 2 an input port by writing 1's to all bits of port 2
BACK: MOV A,P2	;Get data from Port 2
MOV P1,A	;Send data to Port 1
SJMP BACK	;Keep doing it repeatedly

In Example # 4 above, the Port 2 is configures as input port by writing all 1's, then data is received from that port and is sent to Port 1 continuously**.**
The Dual Role or Port 2:
Port 2 has dual role. Port 2 is also designated as A8 - A15. This indicates that Port 2 has a dual function. An 8031 microcontroller is capable of accessing 64K bytes of external memory, it needs a path for the 16 bits of the address. P0 provides the lower 8 bits via A0 - A7 while Port 2 provides bits A8 - A15 of the address. When the 8031 is connected to external memory, Port 2 is used for the upper 8 bits of the 16-bit address, and it cannot be used for input / output operations.

Port 3:
Port 3 has total 8 Pins. P3.0, P3.1, P3.2, P3.3, P3.4, P3.5, P3.6, and P3.7. Pins 10, 11, 12, 13, 14, 15, 16, and 17. Port 3 can be sued as an input port or output port. Port 3 does need any pull-up resisters, just like the Port 1 and Port 2 does not require pull-up resistors. Only Port 0 require Pull-up resistors. Port 3 configured as an output port upon system reset. Port 3 has additional function of providing signals of interrupts.

Alternate Functions of Port 3:



PORT 3 BIT	FUNCTION	PIN
P3.0	RxD [Receive Serial Communications Signals]	10
P3.1	TxD [Transmit Serial Communications Signals]	11
P3.2	INT0 [External Interrupt 0]	12
P3.3	INT1 [External Interrupt 1]	13
P3.4	T0 [Timer 0]	14
P3.5	T1 [Timer 1]	15
P3.6	WR [Write Signals of External Memory]	16
P3.7	RD [Read Signals of External Memory]	17

Serial Communication

BASICS OF SERIAL COMMUNICATION:

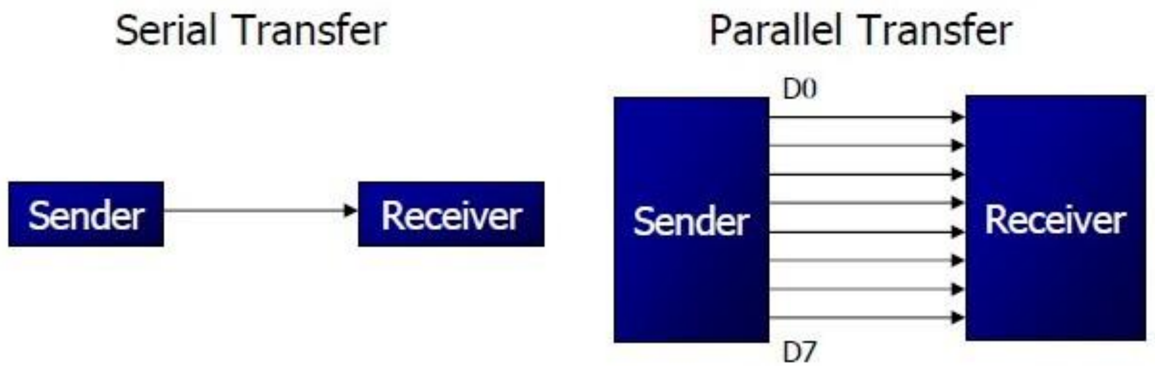
Computers transfer data in two ways:

Parallel:

- Often 8 or more lines (wire conductors) are used to transfer data to a device that is only a few feet away.

Serial:

- To transfer to a device located many meters away, the serial method is used
- The data is sent one bit at a time.



- ➢ At the transmitting end, the byte of data must be converted to serial bits using parallel-in-serial-out shift register.
- ➢ At the receiving end, there is a serialin- parallel-out shift register to receive the serial data and pack them into byte.
- ➢ When the distance is short, the digital signal can be transferred as it is on a simple wire and requires no modulation.

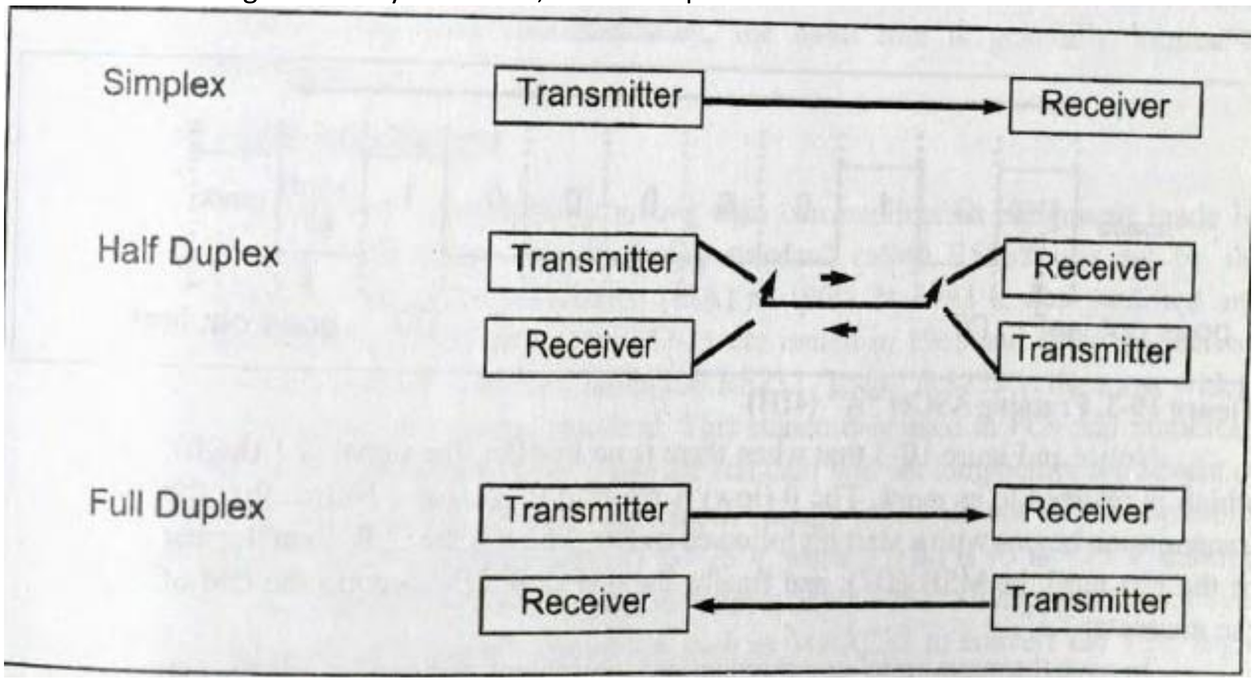
- ➢ If data is to be transferred on thetelephone line, it must be convertedfrom 0s and 1s to audio tones.
    1. This conversion is performed by a device called a modem, “Modulator/demodulator”.

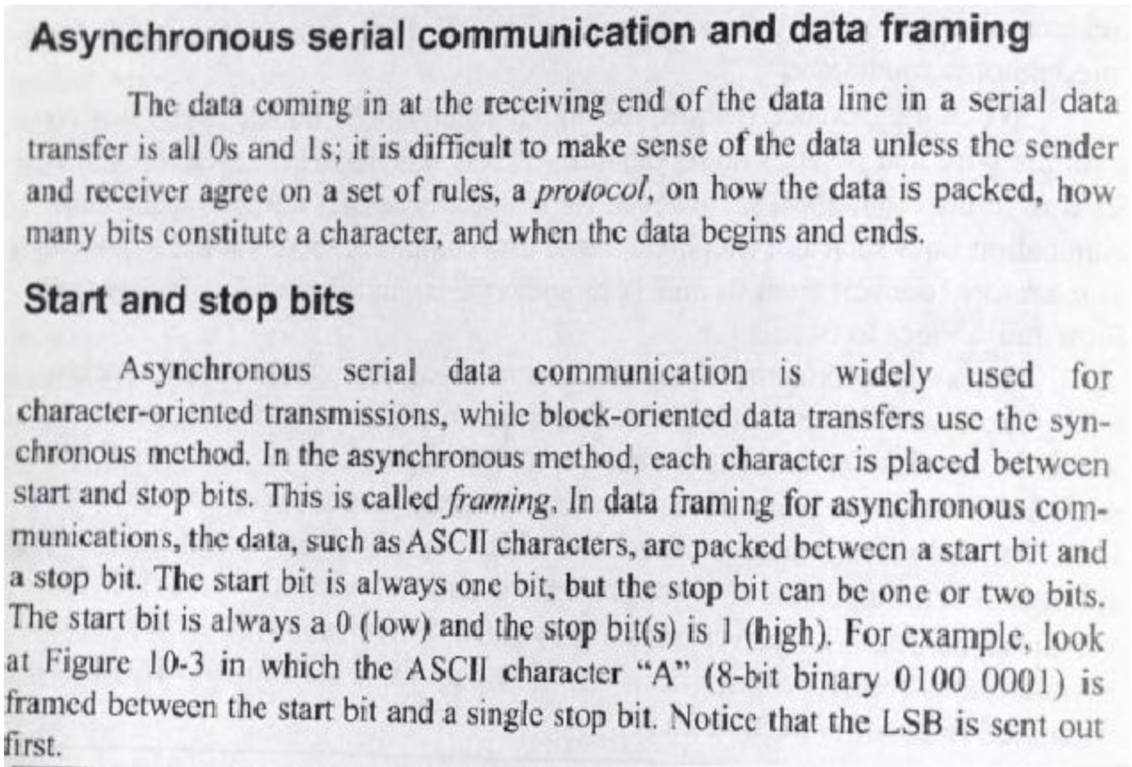
Serial data communication uses two methods

1. Synchronous method transfers a block of data at a time
2. Asynchronous method transfers a single byte at a time

It is possible to write software to use either of these methods, but the programs can be tedious and long There are special IC chips made by many manufacturers for serial communications. UART (universal asynchronous Receivertransmitter) USART (universal synchronous asynchronous Receiver-transmitter).

- ➢ If data can be transmitted and received, it is a duplex transmission
- ➢ If data transmitted one way a time, it is referred to as half duplex
- ➢ If data can go both ways at a time, it is full duplex.



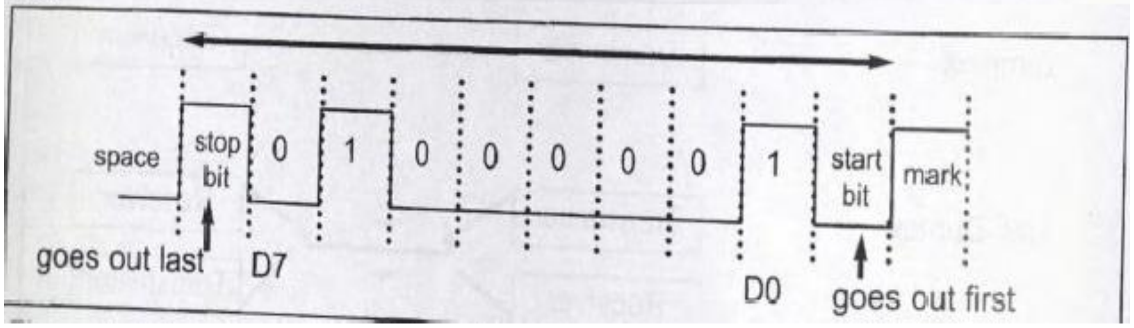


Asynchronous serial communication and data framing

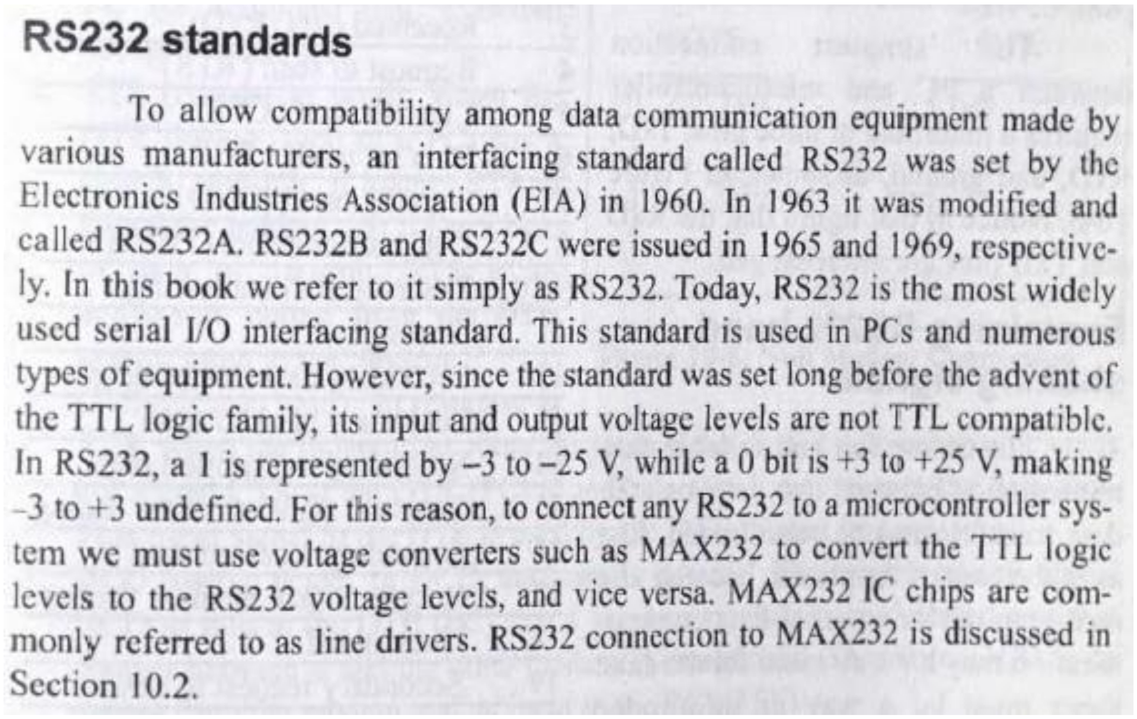
The data coming in at the receiving end of the data line in a serial data transfer is all 0s and 1s; it is difficult to make sense of the data unless the sender and receiver agree on a set of rules, a *protocol*, on how the data is packed, how many bits constitute a character, and when the data begins and ends.

Start and stop bits

Asynchronous serial data communication is widely used for character-oriented transmissions, while block-oriented data transfers use the synchronous method. In the asynchronous method, each character is placed between start and stop bits. This is called *framing*. In data framing for asynchronous communications, the data, such as ASCII characters, are packed between a start bit and a stop bit. The start bit is always one bit, but the stop bit can be one or two bits. The start bit is always a 0 (low) and the stop bit(s) is 1 (high). For example, look at Figure 10-3 in which the ASCII character "A" (8-bit binary 0100 0001) is framed between the start bit and a single stop bit. Notice that the LSB is sent out first.



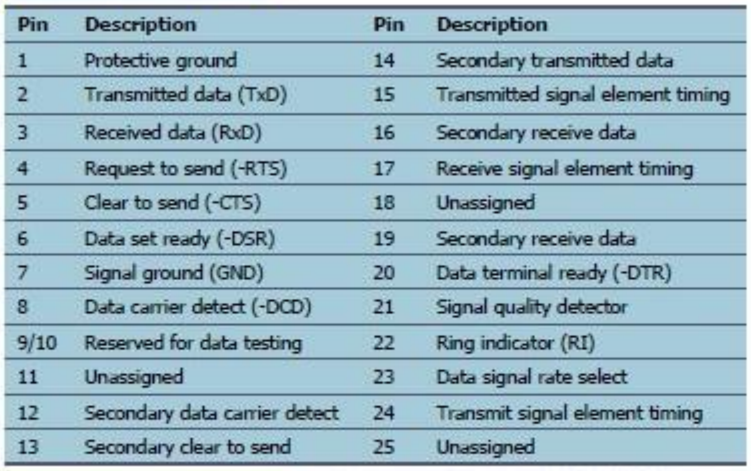
- ➢ The rate of data transfer in serial data communication is stated in bps (bits per second)
- ➢ Another widely used terminology for bps is baud rate
- ➢ It is modem terminology and is defined as the number of signal changes per second
- ➢ In modems, there are occasions when a single change of signal transfers several bits of data
- ➢ As far as the conductor wire is concerned, the baud rate and bps are the same, and we use the terms interchangeably.



RS232 standards

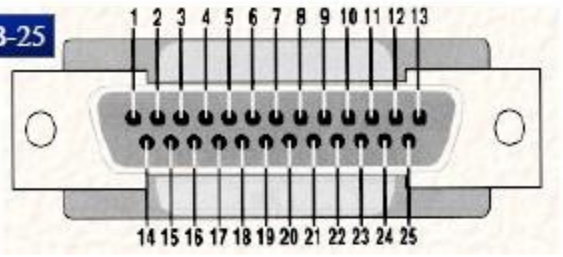
To allow compatibility among data communication equipment made by various manufacturers, an interfacing standard called RS232 was set by the Electronics Industries Association (EIA) in 1960. In 1963 it was modified and called RS232A. RS232B and RS232C were issued in 1965 and 1969, respectively. In this book we refer to it simply as RS232. Today, RS232 is the most widely used serial I/O interfacing standard. This standard is used in PCs and numerous types of equipment. However, since the standard was set long before the advent of the TTL logic family, its input and output voltage levels are not TTL compatible. In RS232, a 1 is represented by −3 to −25 V, while a 0 bit is +3 to +25 V, making −3 to +3 undefined. For this reason, to connect any RS232 to a microcontroller system we must use voltage converters such as MAX232 to convert the TTL logic levels to the RS232 voltage levels, and vice versa. MAX232 IC chips are commonly referred to as line drivers. RS232 connection to MAX232 is discussed in Section 10.2.

RS232 DB-25 Pins



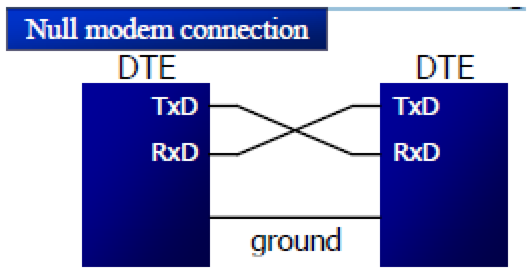
Pin	Description	Pin	Description
1	Protective ground	14	Secondary transmitted data
2	Transmitted data (TxD)	15	Transmitted signal element timing
3	Received data (RxD)	16	Secondary receive data
4	Request to send (-RTS)	17	Receive signal element timing
5	Clear to send (-CTS)	18	Unassigned
6	Data set ready (-DSR)	19	Secondary receive data
7	Signal ground (GND)	20	Data terminal ready (-DTR)
8	Data carrier detect (-DCD)	21	Signal quality detector
9/10	Reserved for data testing	22	Ring indicator (RI)
11	Unassigned	23	Data signal rate select
12	Secondary data carrier detect	24	Transmit signal element timing
13	Secondary clear to send	25	Unassigned

RS232 Connector DB-25



Current terminology classifies data communication equipment as

- DTE (data terminal equipment) refers to terminal and computers that send and receive data
- DCE (data communication equipment) refers to communication equipment, such as modems
- The simplest connection between a PC and microcontroller requires a minimum of three pins, TxD, RxD, and ground



DTR (data terminal ready)
When terminal is turned on, it sends out signal DTR to indicate that it is ready for communication

DSR (data set ready)
When DCE is turned on and has gone through the self-test, it assert DSR to indicate that it is ready to communicate

RTS (request to send)
When the DTE device has byte to transmit, it assert RTS to signal the modem that it has a byte of data to transmit.

CTS (clear to send)
When the modem has room for storing the data it is to receive, it sends out signal CTS to DTE to indicate that it can receive the data now

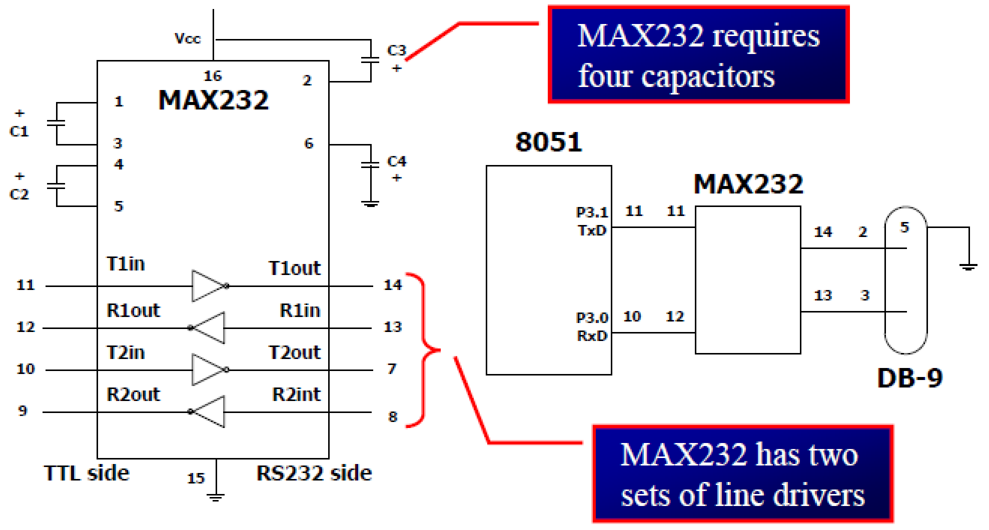
DCD (data carrier detect)
The modem asserts signal DCD to inform the DTE that a valid carrier has been detected and that contact between it and the other modem is established

RI (ring indicator)
An output from the modem and an input to a PC indicates that the telephone is ringing
It goes on and off in synchronous with the ringing sound.

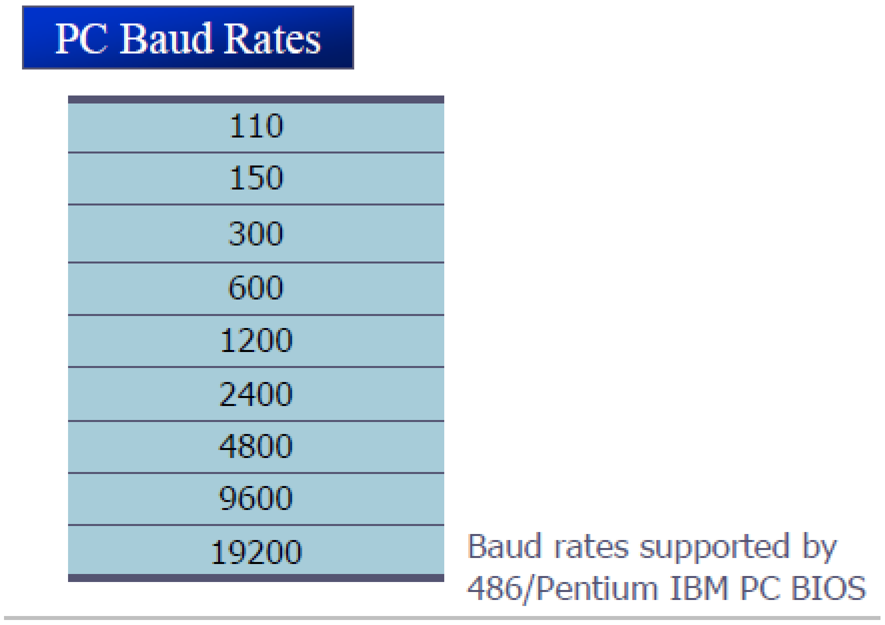
A line driver such as the MAX232 chip is required to convert RS232 voltage levels to TTL levels, and vice versa. 8051 has two pins that are used specifically for transferring and receiving data serially

- These two pins are called TxD and RxD andare part of the port 3 group (P3.0 and P3.1)
- These pins are TTL compatible; therefore, they require a line driver to make them RS232 compatible

We need a line driver (voltage converter) to convert the R232's signals to TTL voltage levels that will be acceptable to 8051's TxD and RxD pins



To allow data transfer between the PC and an 8051 system without any error, we must make sure that the baud rate of 8051 system matches the baud rate of the PC's COM port Hyperterminal function supports baud rates much higher than listed below



PC Baud Rates

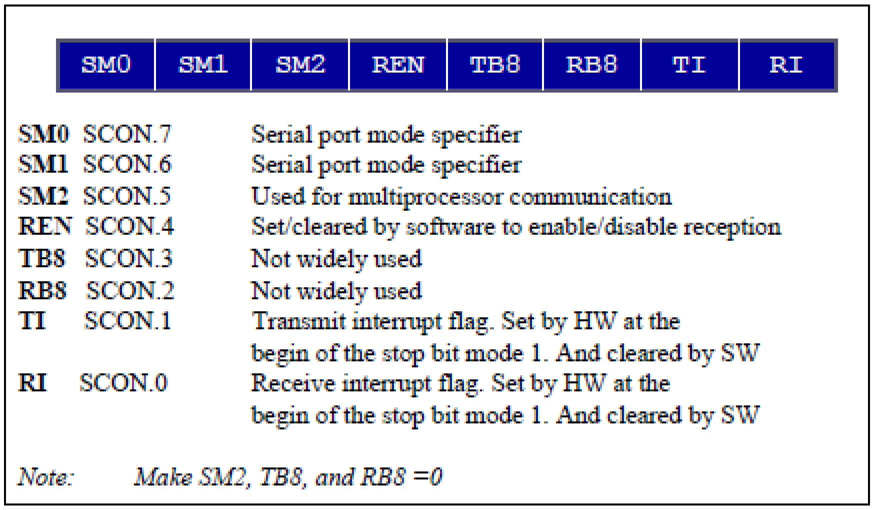
110
150
300
600
1200
2400
4800
9600
19200

Baud rates supported by 486/Pentium IBM PC BIOS

SBUF :
It is an 8-bit register used solely for serial communicationFor a byte data to be transferred via the TxD line, it must be placed in the SBUF Register The moment a byte is written into SBUF, it is framed with the start and stop bits and transferred serially via the TxD line SBUF holds the byte of data when it is received by 8051 RxD line .When the bits are received serially via RxD, the 8051 deframes it by eliminating the stop and start bits, making a byte out of the data received, and then placing it in SBUF

```
    MOV SBUF,#'D'    ;load SBUF=44h, ASCII for 'D'
    MOV SBUF,A       ;copy accumulator into SBUF
    MOV A,SBUF       ;copy SBUF into accumulator
```

SCON
It is an 8-bit register used to program the start bit, stop bit, and data bits of data framing, among other things.

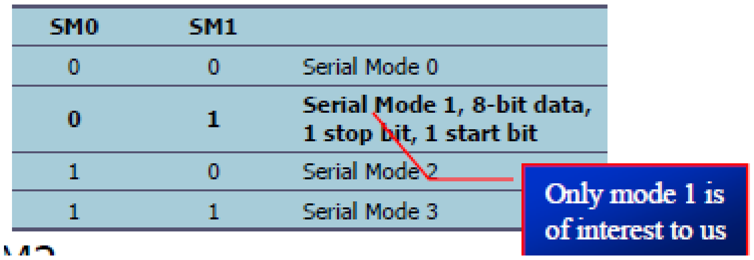


SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	SCON.7	Serial port mode specifier
SM1	SCON.6	Serial port mode specifier
SM2	SCON.5	Used for multiprocessor communication
REN	SCON.4	Set/cleared by software to enable/disable reception
TB8	SCON.3	Not widely used
RB8	SCON.2	Not widely used
TI	SCON.1	Transmit interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW
RI	SCON.0	Receive interrupt flag. Set by HW at the begin of the stop bit mode 1. And cleared by SW

Note: *Make SM2, TB8, and RB8 =0*

SM0, SM1
They determine the framing of data by specifying the number of bits per character, and the start and stop bits



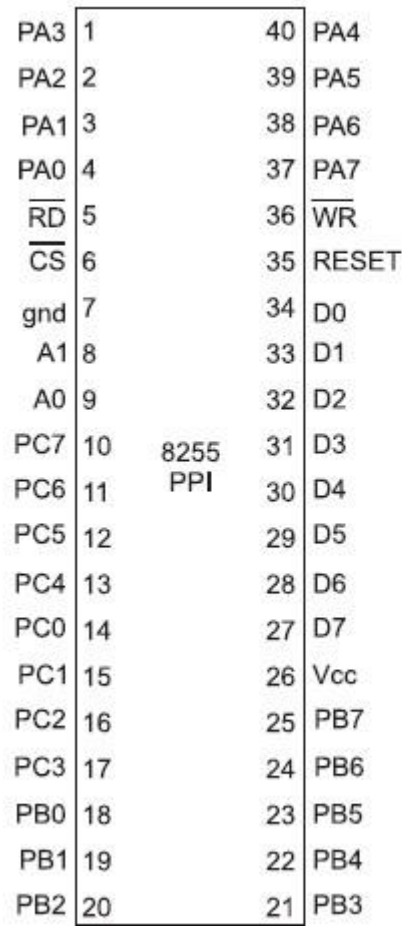
SM0	SM1	
0	0	Serial Mode 0
0	**1**	**Serial Mode 1, 8-bit data, 1 stop bit, 1 start bit**
1	0	Serial Mode 2
1	1	Serial Mode 3

Only mode 1 is of interest to us

SM2
This enables the multiprocessing capability of the 8051
REN (receive enable)
It is a bit-adressable register. When it is high, it allows 8051 to receive data on RxD pin. If low, the receiver is disable

TI (transmit interrupt)
When 8051 finishes the transfer of 8-bit character. It raises TI flag to indicate that it is ready to transfer another byte. TI bit is raised at the beginning of the stop bit
RI (receive interrupt)
When 8051 receives data serially via RxD, it gets rid of the start and stop bits and places the byte in SBUF register. It raises the RI flag bit to indicate that a byte has been received and should be picked up before it is lost. RI is raised halfway through the stop bit

PIN DIAGRAM OF 8255 A



DATA BUS (PIN27-PIN34):
There bi-directional, tri state data bus lines are connected to the system data bus. They are used to transfer data and control word from microprocessor (8085) or Microcontroller (8051) to
8255 or to receive data or status word from 8255 to the 8051 and 8085.
PORT A (PA0 – PA) [Pin 1-4 & Pin 37-40]:
There 8 bit bi-directional Input/Output pins are used to send data to output device and to receive data from input device. It functions as an 8bit data output latch / buffer, when used in output mode and an 8bit data input buffer, when used in input mode.
PORT B (PB0-PB7) (PIN18-PIN25):
There 8bit bi-directional input pins are used to send data to output device and to receive data from input device. It functions as 8bit data output latch/buffer when used in output mode and on 8bit data in input buffer – when used in input mode.
PORT C (PC0-PC7) (PIN10-PIN17):
There 8bit bi-directional input pins are divided into two groups PCL(PC3-PC0)and PC4(PC7-PC4) these groups individually can transfer data in or out when transformed for simple inputs and used as handshake signals when transformed for handshake for bi-directional modes.
RD (read) (pin5):

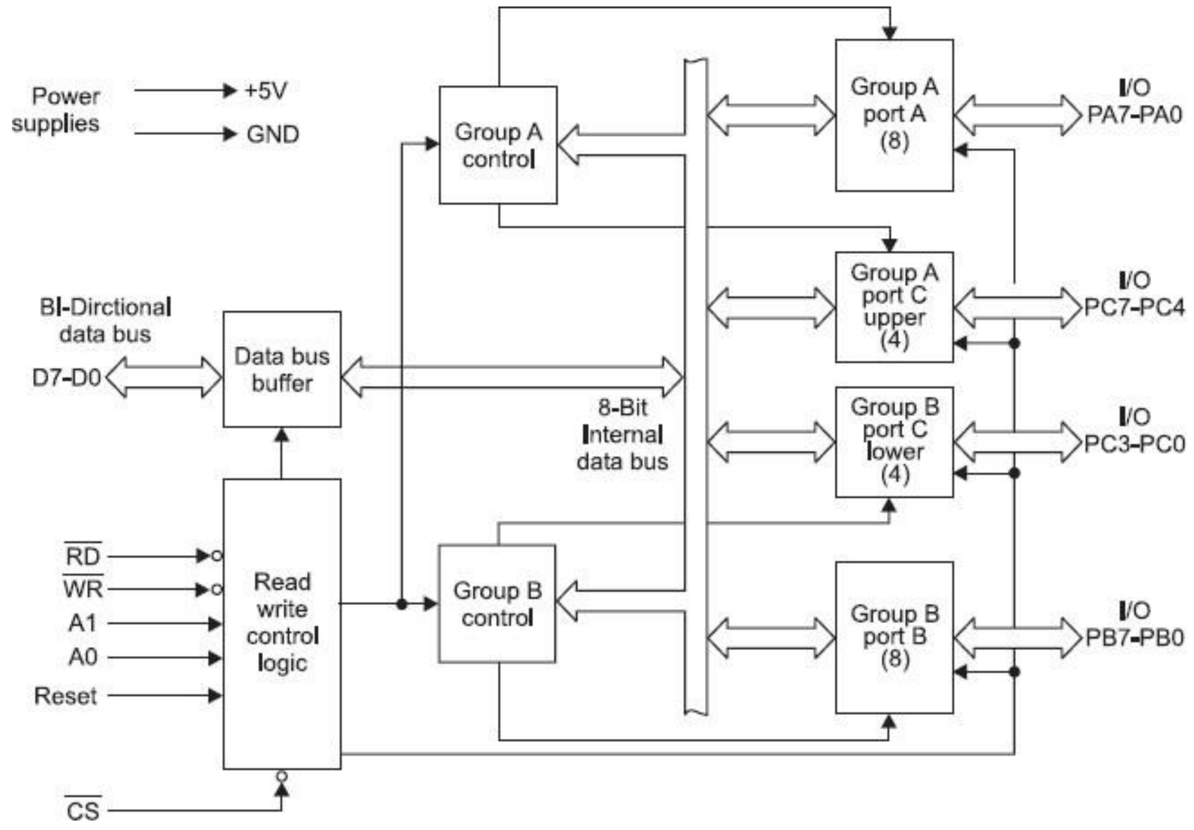
When this pin is low, the CPU can read the data in the ports or the states word through the data buffer.
WR (write) (pin 36):
When this pin is low the CPU can write the data on the ports or in the control register through the data bus buffer.
CS (chip select) (pin 6):
This is an active low input which can be enabled for data transfer operator between the CPU and the 8255.
RESET (pin 35):
This is an active high input used to reset 8255, when reset input is high the control register is cleaned and all the ports are set to the input mode. Usually reset output signal from 8085 or 8051 is used to reset 8255.
A0 and A1 (Pin8, 9):
These input signal and along with read and write inputs control the selection of the control/ states word register or one of the three parts A0 and A1 are generally connected to the A0 ,A1 pins
of the address bus; the 8255 therefore occupies from consecutive location in the input space

Ports and Registor Select Signals

A_1	A_0	RD	WR	CS	Operation
0	0	0	1	0	I/P (Read) Operation
					PORT A to data bus
0	1	0	1	0	I/P (Read) Operation
					PORT B to data bus
1	0	0	1	0	I/P (Read) Operation
					PORT C to data bus
0	0	1	0	0	O/P (Write) Operation
					Data bus to PORT A
0	1	1	0	0	O/P (Write) Operation
					Data bus to PORT B
1	0	1	0	0	O/P (Write) Operation
					Data bus to PORT C
1	1	1	0	0	O/P (Write) Operation
					Data bus to control register
X	x	x	x	1	Disable Function
					Data bus Tri-stated
1	1	0	1	0	Disable Function
					Illegal condition
X	x	1	1	0	Disable Function
					Data bus Tri-stated

Data Bus Buffer
This three state bi-directional 8bit buffer is used to interface the 8255 to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status informations are also transferred through the data bus buffer

READ/WRITE AND CONTROL LOGIC
The function of this block is to manage all of the internal and external transfers of both data and Control or status words. It accepts input from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.
(CS) Chip select: A "low" on this input pin enables the communication between the 8255 and the CPU.
(RD) Read: A "low" on this input pin enables 8255 to send the data or status information to the CPU on the data bus. In essence, it allows the CPU to "read from" the 8255.
(WR) Write: A "low" on this input pin enables the CPU to write data or control words into the 8255.



READ AND WRITE CONTROL LOGIC

(A0 and A1) Port Select 0 and Port Select 1. These input signals, in conjunction with the RD and WR inputs, control the selection of one of the three ports or the control word register. They
are normally connected to the least significant bits of the address bus. (A0 and A1).

(RESET) Reset: A “high” on this input initializes the control register to 9Bh and all ports (A, B, C) are set to the input mode.

A_1	A_0	Selection
0	0	Port a
0	1	Port b
1	0	Port c
1	1	control

Group A And Group B Controls

The functional configuration of each port is programmed by the system software. In essence, the CPU “outputs” a control word to the 8255. The control word contains information such as ”mode”, ”bit set”, etc. That initializes the functional configuration of the 8255.Each of the Control blocks(GROUP A and GROUP B)accepts “commands” from the Read/Write control logic, receives “control words” from the internal data bus and issues the proper commands to its associated ports.

PORTS A, B and C

The 8255 contains three 8-bit ports (A, B and C)all can be configured to a wide variety of functional characteristics by the system software but each has its own special features or “personality” to further enhance the power and flexibility of the 8255.

PORT A

One 8-bit data output latch/buffer and one 8-bit data input latch. Both “Pull-up” and “Pulldown” bus-hold devices are present on the Port A.

PORT B

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

PORT C

One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal output and status signal inputs in conjunction with ports A and B.

OPERATION MODES

Bit set-reset (BSR) Mode

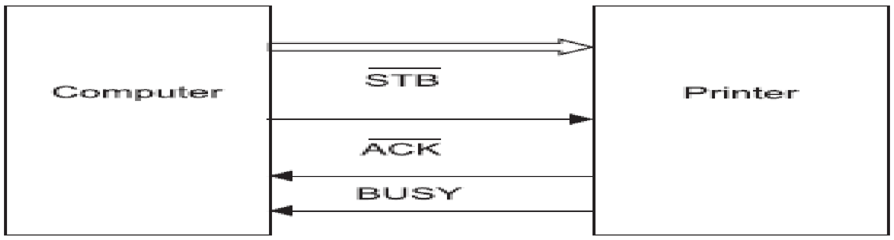
The individual bits of Port C can be set or reset by sending out a single OUT instruction to the control register. When Port C is used for control/status operations features can be used to set or reset individual bits.

I/O Modes

Mode 0 : Simple Input/Output.

In this mode, ports A and B are used as two simple 8-bit I/O ports and Port C as two 4-bit

ports. Each port (or half port, in case of C) can be programmed to function as simply an input port or an output port. The input/output features in Mode 0 are as follows
1. Outputs are latched.
2. Input are buffered, not latched.
3. Ports do not have handshake or interrupt capability
Mode 1: Input/Output with handshake
In this mode, input or output data transfer is controlled by hand shaking signals. Handshaking signals are used to transfer data between devices whose data transfer speeds are not same. For example, computer can send data to the printer with large speed but printer can’t accept data and print data with this rate.

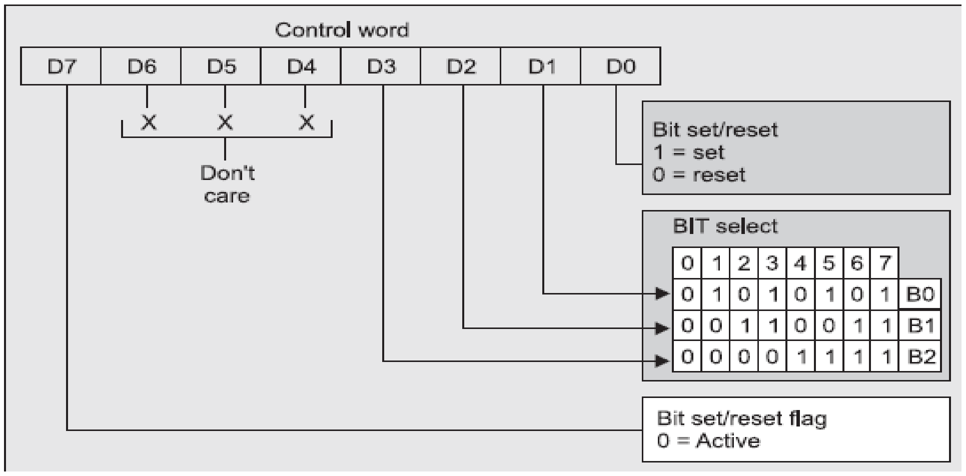


So, computer has to send data with the speed with which printer can accept. This type of data transfer is achieved by using hand shaking signals along with data signals. Fig 8.3 shows data transfer between computer and printer using handshaking signals.
These handshaking signals are used to tell computer whether printer is ready to accept the data or not. If printer is ready to accept the data, then after sending data on data bus, computer
uses another handshaking signal (STB) to tell printer that valid data is available on the data bus.
The 8255 mode 1 which supports handshaking has following features.

- Two ports (A and B) functions as 8-bit I/O ports. They can be configured either as input or as output ports.
- Each port uses three lines from port C as handshake signals. The remaining two lines of port C can be used for simple I/O function.
- Input and output data are latched.
- Interrupt logic is supported.

Mode 2: Bi-directional I/O data transfer
This mode allows bi-directional data transfer (transmission and reception) over a single 8-bit data bus using handshaking signals. This feature is available only in Group A with Port A as the 8-bit bidirectional data bus and PC3– PC7 are used for handshaking purpose. In this mode, both inputs and outputs are latched. Due to use of a simple 8-bit data bus for bidirectional data transfer, the data sent out by the CPU through Port A appears on the bus connecting it to the peripheral, only when the peripherals requests it. The remaining lines of Port C i.e., PC0 – PC2 can be used simple I/O function. This Port B can be programmed in mode 0 or in mode 1. When Port A is programmed in mode 1, PC0 - PC2 lines of Ports C are used as hand shaking signals.
CONTROL WORD FORMATS

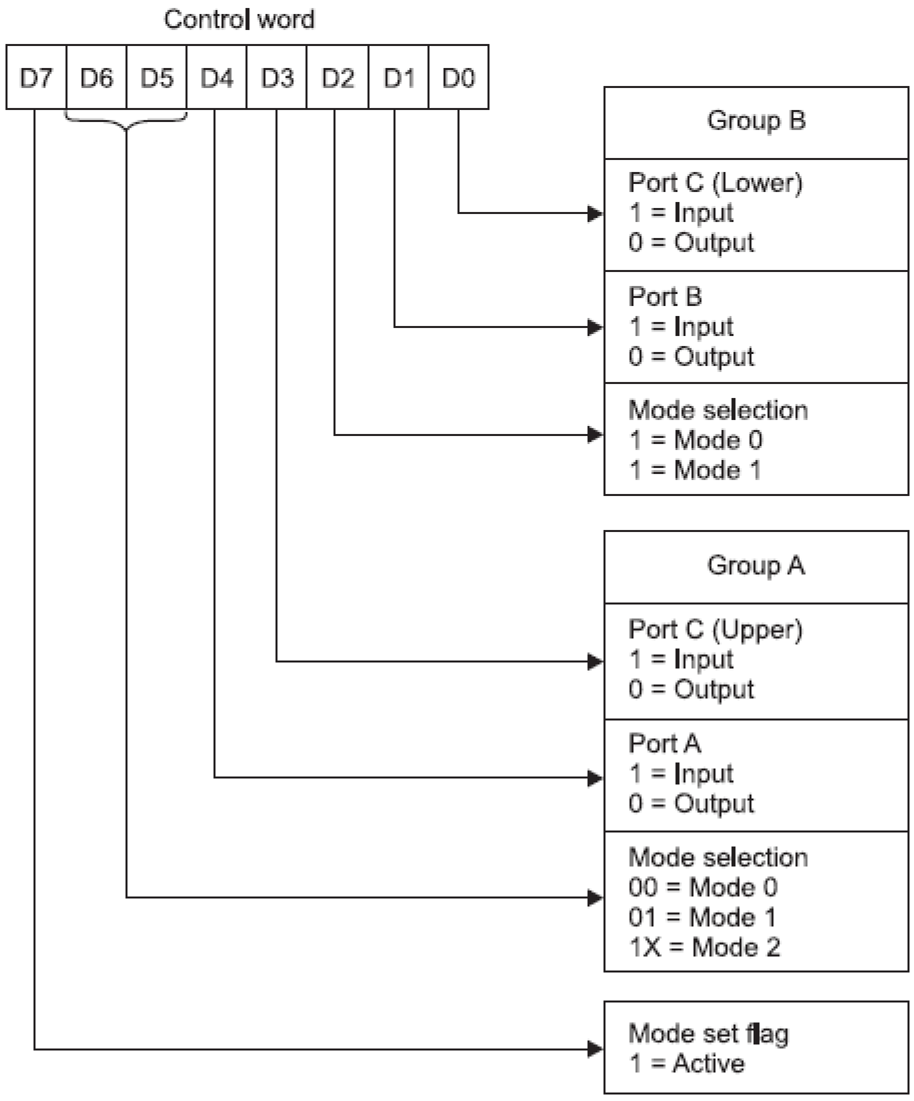
A high on the RESET pin causes all 24 lines of the three 8-bit ports to be in the input mode. All flip flops are cleared and the interrupts are reset. This condition is maintained even after the RESET goes low. The ports of the 8255 can then be programmed for any other mode by writing a single control word into the control register, when required.



The eight possible combinations of the status of bits D3-D1 (B2,B1,B0) in the Bit Set-Reset format (BSR) determine particular bit in PC0 – PC7 being set or reset as per the status of bit D0. A BSR word is to be written for each bit that is to be set or reset. For example if bit PC3 is to be set and bit PC4 is to be reset, the appropriate BSR words that will have to be loaded into the control register will be 0xxx0111 and 0xxx1000 respectively, where x is don't care.
The BSR word can also be used for enabling or disabling interrupt signal generated by Port C when the 8255 is programmed for Mode 1 or 2 operations. This is done by setting or resetting the associated bits of the interrupts.

For I/O Modes

The mode definition format for input mode is shown in figure 8.5. The control words for both, mode definition and bit set-reset are loaded into the same control register, with bit D7 used for specifying whether the word loaded into the control register is a mode definition word or Bit Set-Reset word.


Control word
D7
D6
D5
D4
D3
D2
D1
D0
Group B
Port C (Lower)
1 = Input
0 = Output
Port B
1 = Input
0 = Output
Mode selection
1 = Mode 0
1 = Mode 1
Group A
Port C (Upper)
1 = Input
0 = Output
Port A
1 = Input
0 = Output
Mode selection
00 = Mode 0
01 = Mode 1
1X = Mode 2
Mode set flag
1 = Active