UNIT I – 8085 MICROPROCESSOR

1.1 Introduction

Microcomputer: The term microcomputer is generally synonymous with personal computer, or a computer that depends on a microprocessor.

- Microcomputers are designed to be used by individuals, whether in the form of PCs, workstations or notebook computers.
- A microcomputer contains a CPU on a microchip (the microprocessor), a memory system (typically ROM and RAM), a bus system and I/O ports, typically housed in a motherboard.

Microprocessor: A silicon chip that contains a CPU. In the world of personal computers, the terms microprocessor and CPU are used interchangeably.

- A microprocessor (sometimes abbreviated μP) is a digital electronic component with miniaturized transistors on a single semiconductor integrated circuit (IC).
- One or more microprocessors typically serve as a central processing unit (CPU) in a computer system or handheld device.
- Microprocessors made possible the advent of the microcomputer.
- At the heart of all personal computers and most working stations sits a microprocessor.
- Microprocessors also control the logic of almost all digital devices, from clock radios to fuel-injection systems for automobiles.
- Three basic characteristics differentiate microprocessors:
- Instruction set: The set of instructions that the microprocessor can execute.
- Bandwidth: The number of bits processed in a single instruction.
- Clock speed: Given in megahertz (MHz), the clock speed determines how many instructions per second the processor can execute.
- In both cases, the higher the value, the more powerful the CPU. For example, a 32-bit microprocessor that runs at 50MHz is more powerful than a 16-bit microprocessor that runs at 25MHz.

- In addition to bandwidth and clock speed, microprocessors are classified as being either RISC (reduced instruction set computer) or CISC (complex instruction set computer)

1.2 8085 Microprocessor

The Intel 8085 is an 8-bit microprocessor introduced by Intel in 1977. It was binary compatible with the more-famous Intel 8080 but required less supporting hardware, thus allowing simpler and less expensive microcomputer systems to be built. The "5" in the model number came from the fact that the 8085 requires only a +5-Volt (V) power supply rather than the +5 V, −5 V and +12 V supplies the 8080 needed. The main features of 8085 µP are:

- It is an 8-bit microprocessor.
- It is manufactured with N-MOS technology.
- It has 16-bit address bus and hence can address up to 2^{16}= 65536 bytes (64KB) memory locations through A0–A15.
- The first 8 lines of address bus and 8 lines of data bus are multiplexed AD0–AD7
- Data bus is a group of 8 lines D0–D7.
- It supports external interrupt request.
- A 16-bit program counter (PC)
- A 16-bit stack pointer (SP)
- Six 8-bit general purpose register arranged in pairs: BC, DE, HL.
- It requires a signal +5V power supply and operates at 3.2 MHZ single phase clock.
- It is enclosed with 40 pins DIP (Dual in line package).

1.3 8085 Architecture

8085 consists of various units as shown in Fig. 1 and each unit performs its own functions. The various units of a microprocessor are listed below

- Accumulator
- Arithmetic and logic Unit
- General purpose register
- Program counter
- Stack pointer
- Temporary register
- Flags
- Instruction register and Decoder

- Timing and Control unit
- Interrupt control
- Address buffer and Address-Data buffer
- Address bus and Data bus

Accumulator

Accumulator is nothing but a register which can hold 8-bit data. Accumulator aids in storing two quantities. The data to be processed by arithmetic and logic unit is stored in accumulator. It also stores the result of the operation carried out by the Arithmetic and Logic unit. The accumulator is also called an 8-bit register. The accumulator is connected to Internal Data bus and ALU (arithmetic and logic unit). The accumulator can be used to send or receive data from the Internal Data bus.

Arithmetic and Logic Unit

There is always a need to perform arithmetic operations like +, -, *, / and to perform logical operations like AND, OR, NOT etc. So, there is a necessity for creating a separate unit which can perform such types of operations. These operations are performed by the Arithmetic and Logic Unit (ALU). ALU performs these operations on 8-bit data. But these operations cannot be performed unless we have an input (or) data on which the desired operation is to be performed. So, from where do these inputs reach the ALU? For this purpose, accumulator is used. ALU gets its Input from accumulator and temporary register. After processing the necessary operations, the result is stored back in accumulator.

General Purpose Registers

Apart from accumulator 8085 consists of six special types of registers called General Purpose Registers. These general-purpose registers are used to hold data like any other registers. The general-purpose registers in 8085 processors are B, C, D, E, H and L. Each register can hold 8-bit data. Apart from the above function these registers can also be used to work in pairs to hold 16-bit data. They can work in pairs such as B-C, D-E and H-L to store 16-bit data. The H-L pair works as a memory pointer. A memory pointer holds the address of a particular memory location. They can store 16-bit address as they work in pair.

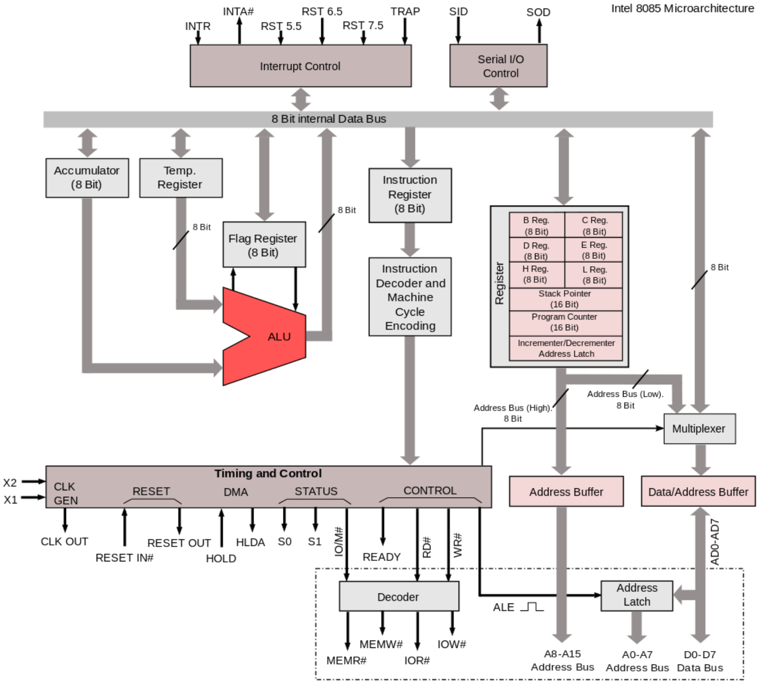


Fig. 1.1 8085 Architecture

Program Counter and Stack Pointer

Program counter is a special purpose register.

Consider that an instruction is being executed by processor. As soon as the ALU finished executing the instruction, the processor looks for the next instruction to be executed. So, there is a necessity for holding the address of the next instruction to be executed in order to save time. This is taken care by the program counter. A program counter stores the address of the next instruction to be executed. In other words, the program counter keeps track of the memory address of the instructions that are being executed by the microprocessor and the memory address of the next instruction that is going to be executed.

Microprocessor increments the program whenever an instruction is being executed, so that the program counter points to the memory address of the next instruction that is going to be executed. Program counter is a 16-bit register.

Stack pointer is also a 16-bit register which is used as a memory pointer. A stack is nothing but the portion of RAM (Random access memory).

So, does that mean the stack pointer points to portion of RAM?

Yes. Stack pointer maintains the address of the last byte that is entered into stack.

Each time when the data is loaded into stack, Stack pointer gets decremented. Conversely it is incremented when data is retrieved from stack.

Temporary Register

As the name suggests this register acts as a temporary memory during the arithmetic and logical operations. Unlike other registers, this temporary register can only be accessed by the microprocessor and it is completely inaccessible to programmers. Temporary register is an 8-bit register.

Flags

Flags are nothing but a group of individual Flip-flops. The flags are mainly associated with arithmetic and logic operations. The flags will show either a logical (0 or 1) (i.e.) a set or reset depending on the data conditions in accumulator or various other registers. A flag is actually a latch which can hold some bits of information. It alerts the processor that some event has taken place.

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

Fig. 1.2 Flag Register

Intel processors have a set of 5 flags.

1. Carry flag
2. Parity flag
3. Auxiliary carry flag
4. Zero flag
5. Sign flag

Consider two binary numbers.

For example

1100 0000

1000 0000

When we add the above two numbers, a carry is generated in the most significant bit. The number in the extreme right is least significant bit, while the number in extreme left is most significant bit. So, a ninth bit is generated due to the carry. So how to accommodate 9th bit in an 8-bit register?

For this purpose, the Carry flag is used. The carry flag is set whenever a carry is generated and reset whenever there is no carry. But there is an auxiliary carry flag? What is the difference between the carry flag and auxiliary carry flag?

Let's discuss with an example. Consider the two numbers given below

0000 1100

0000 1001

When we add both the numbers a carry is generated in the fourth bit from the least significant bit. This sets the auxiliary carry flag. When there is no carry, the auxiliary carry flag is reset. So, whenever there is a carry in the most significant bit Carry flag is set. While an auxiliary carry flag is set only when a carry is generated in bits other than the most significant bit.

Parity checks whether it's even or add parity. This flag returns a 0 if it is odd parity and returns a 1 if it is an even parity. Sometimes they are also called as parity bit which is used to check errors while data transmission is carried out.

Zero flag shows whether the output of the operation is 0 or not. If the value of Zero flag is 0 then the result of operation is not zero. If it is zero the flag returns value 1.

Sign flag shows whether the output of operation has positive sign or negative sign. A value 0 is returned for positive sign and 1 is returned for negative sign.

Instruction Register and Decoder

Instruction register is 8-bit register just like every other register of microprocessor. Consider an instruction. The instruction may be anything like adding two data's, moving a data, copying a data etc. When such an instruction is fetched from memory, it is directed to Instruction register. So, the instruction registers are specifically to store the instructions that are fetched from memory. There is an Instruction decoder which decodes the information present in the Instruction register for further processing.

Timing and Control Unit

Timing and control unit is a very important unit as it synchronizes the registers and flow of data through various registers and other units. This unit consists of an oscillator and controller sequencer which sends control signals needed for internal and external control of data and other units. The oscillator generates two-phase clock signals which aids in synchronizing all the registers of 8085 microprocessor.

Signals that are associated with Timing and control unit are:

Control Signals: RD’, WR’, ALE

- ALE is used for provide control signal to synchronize the components of microprocessor and timing for instruction to perform the operation.
- RD (Active low) and WR (Active low) are used to indicate whether the operation is reading the data from memory or writing the data into memory respectively.

Status Signals: S0, S1, IO/M’

- IO/M (Active low) is used to indicate whether the operation belongs to the memory or peripherals.

Table 1.1 Status signals and the status of data bus

IO/M’ (Active Low)	S1	S2	Data Bus Status (Output)
0	0	0	Halt
0	0	1	Memory WRITE
0	1	0	Memory READ
1	0	1	IO WRITE
1	1	0	IO READ
0	1	1	Op code fetch
1	1	1	Interrupt acknowledge

DMA Signals: HOLD, HLDA, READY

- HOLD: Indicates that another master is requesting the use of the address and data buses. The CPU, upon receiving the hold request, will relinquish the use of the bus as soon as the completion of the current bus transfer. Internal processing can continue. The processor can regain the bus only after the HOLD is removed. When the HOLD is acknowledged, the Address, Data RD, WR and IO/M’ lines are tri-stated.
- HLDA: Hold Acknowledge: Indicates that the CPU has received the HOLD request and that it will relinquish the bus in the next clock cycle HLDA goes low after the Hold request is removed. The CPU takes the bus one half-clock cycle after HLDA goes low.
- READY: This signal synchronizes the fast CPU and the slow memory, peripherals. If READY is high during a read or write cycle, it indicates that the memory or peripheral is ready to send or receive

data. If READY is low, the CPU will wait an integral number of clock cycle for READY to go high before completing the read or write cycle. READY must conform to specified setup and hold times.

Reset Signals: Reset in, Reset Out

RESET IN: A low on this pin;

- Sets the program counter to zero (0000H)
- Resets the interrupt enables and HLDA flip-flops.
- Tri-states the data bus, address bus and control bus.
- Affects the content of processors internal registers randomly.

On Reset, The Program counter sets to 0000h which causes the 8085 to execute; the first instruction from address 0000H.

- RESET OUT: This active high signal indicates that the processor; is being reset. This signal is synchronized to the processor clock and it can be used to reset other devices connected in the system.

Interrupt control

As the name suggests this control interrupts a process. Consider that a microprocessor is executing the main program. Now whenever the interrupt signal is enabled or requested the microprocessor shifts the control from main program to process the incoming request and after the completion of request, the control goes back to the main program. For example, an Input/output device may send an interrupt signal to notify that the data is ready for input. The microprocessor temporarily stops the execution of main program and transfers control to I/O device. After collecting the input data, the control is transferred back to main program. Interrupt signals present in 8085 are:

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

INTR is maskable 8080A compatible interrupt. When the interrupt occurs the processor fetches from the bus one instruction, usually one of these instructions: One of the 8 RST instructions (RST0 - RST7). The processor saves current program counter into stack and branches to memory location N * 8 (where N is a 3 - bit number from 0 to 7 supplied with the RST instruction).

CALL instruction (3-byte instruction). The processor calls the subroutine, address of which is specified in the second and third bytes of the instruction.

RST5.5 is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 2CH (hexadecimal) address.

RST6.5 is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 34H (hexadecimal) address.

RST7.5 is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 3CH (hexadecimal) address.

TRAP is a non-maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 24H (hexadecimal) address.

All maskable interrupts can be enabled or disabled using EI and DI instructions. RST5.5, RST6.5 and RST7.5 interrupts can be enabled or disabled individually using SIM instruction.

Serial Input/output control

The input and output of serial data can be carried out using 2 instructions in 8085.

- SID-Serial Input Data
- SOD-Serial Output Data

Two more instructions are used to perform serial-parallel conversion needed for serial I/O devices.

- SIM
- RIM

Address buffer and Address-Data buffer

The contents of the stack pointer and program counter are loaded into the address buffer and address-data buffer. These buffers are then used to drive the external address bus and address-data bus. As the memory and I/O chips are connected to these buses, the CPU can exchange desired data to the memory and I/O chips.

The address-data buffer is not only connected to the external data bus but also to the internal data bus which consists of 8-bits. The address data buffer can both send and receive data from internal data bus.

Address bus and Data bus

We know that 8085 is an 8-bit microprocessor. So, the data bus present in the microprocessor is also 8-bits wide. So, 8-bits of data can be transmitted from or to the microprocessor. But 8085 processor requires 16-bit address bus as the memory addresses are 16-bit wide. The 8 most significant bits of the address are transmitted with the help of address bus and the 8 least significant bits are transmitted with the help of multiplexed address/data bus. The eight-bit data bus is multiplexed with the eight least significant bits of address bus. The address/data bus is time multiplexed. This means for few microseconds, the 8 least significant bits of address are generated, while for next few seconds the same pin generates the data. This is called Time multiplexing. But there are situations where there is a need to transmit both data and address simultaneously. For this purpose, a signal called ALE (address latch enables) is used. ALE signal holds the obtained address in its latch for a long time until the data is obtained and so when the microprocessor sends the data next time the address is also available at the output latch. This technique is called Address/Data demultiplexing.

1.4 Pin Diagram of 8085

The signals can be grouped as follows

1. Power supply and clock signals
2. Address bus
3. Data bus
4. Control and status signals
5. Interrupts and externally initiated signals
6. Serial I/O ports

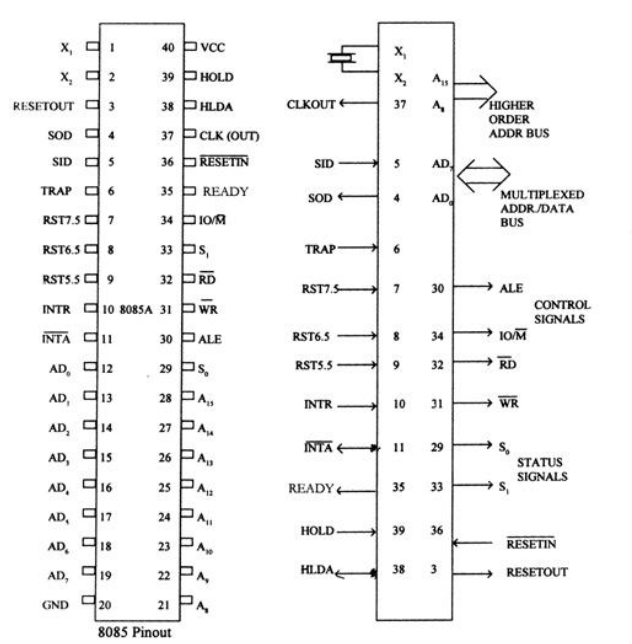


Fig. 1.3 Pin diagram of 8085

Power supply and Clock frequency signals

- Vcc + 5-volt power supply
- Vss Ground
- X1, X2: Crystal or R/C network or LC network connections to set the frequency of internal clock generator. The frequency is internally divided by two. Since the basic operating timing frequency is 3 MHz, a 6 MHz crystal is connected externally.
- CLK (output) – Clock Output is used as the system clock for peripheral and devices interfaced with the microprocessor.

Data Bus and Address Bus

AD0-AD7:-These are multiplexed address and data bus. So, it can be used to carry the lower order 8-bit address as well as the data. Generally, these lines are demultiplexed using the Latch. During the opcode fetch operation, in the first clock cycle the lines deliver the lower order address bus A0-A7. In the subsequent IO/M read or write it is used as data bus D0-D7. CPU can read or write data through these lines. A8-A15:- These are address bus used to address the memory location.

1.5 Instruction Set

The 8085 instruction set can be classified into the following five functional headings.

Data Transfer Instructions: Includes the instructions that moves (copies) data between registers or between memory locations and registers. In all data transfer operations, the content of source register is not altered. Hence the data transfer is copying operation.

Arithmetic Instructions: Includes the instructions, which performs the addition, subtraction, increment or decrement operations. The flag conditions are altered after execution of an instruction in this group.

Logical Instructions: The instructions which performs the logical operations like AND, OR, EXCLUSIVE-OR, complement, compare and rotate instructions are grouped under this heading. The flag conditions are altered after execution of an instruction in this group.

Branching Instructions: The instructions that are used to transfer the program control from one memory location to another memory location are grouped under this heading.

Machine Control Instructions: Includes the instructions related to interrupts and the instruction used to halt program execution.

1.6 Data Transfer Instructions

- These instructions move data between registers, or between memory and registers.
- These instructions copy data from source to destination.
- While copying, the contents of source are not modified.

Opcode	Operand	Description
MOV	Rd, Rs M, Rs Rd, M	Copy from source to destination.
MVI	Rd, Data M, Data	Move immediate 8-bit
LDA	16-bit address	Load Accumulator
LDAX	B/D Register Pair	Load accumulator indirect
LXI	Reg. pair, 16-bit data	Load register pair immediate
STA	16-bit address	Store accumulator direct
STAX	Reg. pair	Store accumulator indirect
XCHG	None	Exchange H-L with D-E

1.7 Arithmetic Instructions

These instructions perform arithmetic operations such as addition, subtraction, increment, and decrement.

Opcode	Operand	Description
ADD	R M	Add register or memory to accumulator
ADC	R M	Add register or memory to accumulator with carry
ADI	8-bit data	Add immediate to accumulator
ACI	8-bit data	Add immediate to accumulator with carry
SUB	R M	Subtract register or memory from accumulator
SUI	8-bit data	Subtract immediate from accumulator
INR	R M	Increment register or memory by 1
INX	R	Increment register pair by 1
DCR	R M	Decrement register or memory by 1
DCX	R	Decrement register pair by 1

1.8 Logical Instructions

These instructions perform various logical operations with the contents of the accumulator.

Opcode	Operand	Description
CMP	R M	Compare register or memory with accumulator
CMP	R M	Compare register or memory with accumulator
CPI	8-bit data	Compare immediate with accumulator
ANA	R M	Logical AND register or memory with accumulator
ANI	8-bit data	Logical AND immediate with accumulator
XRA	R M	Exclusive OR register or memory with accumulator
ORA	R M	Logical OR register or memory with accumulator
ORI	8-bit data	Logical OR immediate with accumulator
XRA	R M	Logical XOR register or memory with accumulator
XRI	8-bit data	XOR immediate with accumulator

1.9 Branching Instructions

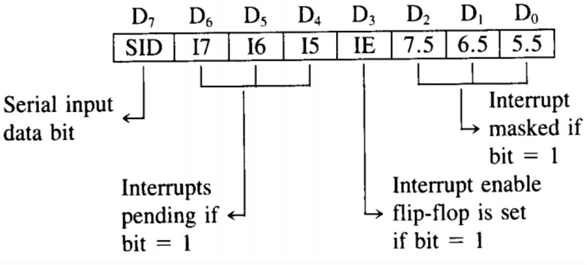
This group of instructions alters the sequence of program execution either conditionally or unconditionally.

Opcode	Operand	Description
JMP	16-bit address	Jump unconditionally
Jx	16-bit address	Jump conditionally

1.10 Machine Control Instructions

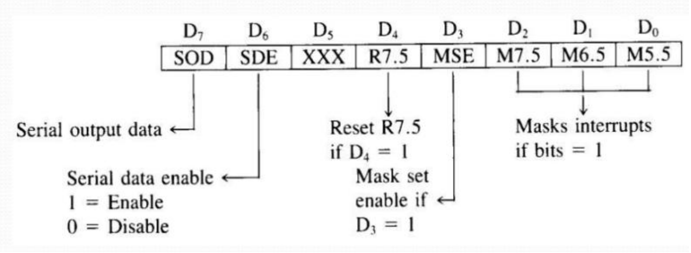
These instructions control machine functions such as Halt, Interrupt, or do nothing.

Opcode	Operand	Description
HLT	None	Halt
NOP	None	No operation
EI	None	The interrupt enable flip-flop is set and all interrupts are enabled. No flags are affected.
DI	None	The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected.
SIM	None	This is a multipurpose instruction and used to implement the 8085 interrupts 7.5, 6.5, 5.5, and serial data output.
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.



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

Fig. 1.4 SIM Instruction



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

Fig. 1. 5 RIM Instruction

1.11 Addressing Modes

Every instruction of a program has to operate on a data. The method of specifying the data to be operated by the instruction is called Addressing. The 8085 has the following 5 different types of addressing.

- Immediate Addressing
- Direct Addressing
- Register Addressing
- Register Indirect Addressing
- Implied Addressing

Immediate Addressing

In immediate addressing mode, the data is specified in the instruction itself. The data will be a part of the program instruction.

EX. MVI B, 3EH - Move the data 3EH given in the instruction to B register; LXI SP, 2700H.

Direct Addressing

In direct addressing mode, the address of the data is specified in the instruction. The data will be in memory. In this addressing mode, the program instructions and data can be stored in different memory.

EX. LDA 1050H - Load the data available in memory location 1050H in to accumulator; SHLD 3000H

Register Addressing

In register addressing mode, the instruction specifies the name of the register in which the data is available.

EX. MOV A, B - Move the content of B register to A register; SPHL; ADD C.

Register Indirect Addressing

In register indirect addressing mode, the instruction specifies the name of the register in which the address of the data is available. Here the data will be in memory and the address will be in the register pair.

EX. MOV A, M - The memory data addressed by H L pair is moved to A register. LDAX B.

Implied Addressing

In implied addressing mode, the instruction itself specifies the data to be operated. EX. CMA - Complement the content of accumulator; RAL

1.12 Timing Diagrams

Timing diagram is the display of initiation of read/write and transfer of data operations under the control of 3-status signals IO/M’, S1 and S0. Each machine cycle is composed of many clock cycles. Since, the data and instructions, both are stored in the memory, the µP performs fetch operation to read the instruction or data and then execute the instruction. The 3-status signals: IO / M’, S1 and S0 are generated at the beginning of each machine cycle. The unique combination of these 3-status signals identifies read or write operation and remain valid for the duration of the cycle. Thus, time taken by any µP to execute one instruction is calculated in terms of the clock period. The execution of instruction always requires read and writes operations to transfer data to or from the µP and memory or I/O devices. Each read/ write operation constitutes one machine cycle. Each machine cycle consists of many clock periods/ cycles, called T-states.

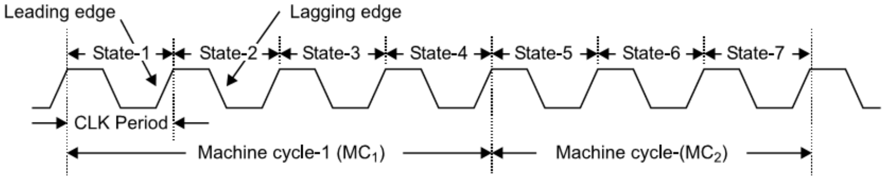


Fig. 1.6 Machine cycle showing clock periods

Each and every operation inside the microprocessor is under the control of the clock cycle. The clock signal determines the time taken by the microprocessor to execute any instruction. State is defined as the time interval between 2-trailing or leading edges of the clock. Machine cycle is the time required to transfer data to or from memory or I/O devices.

The 8085 microprocessor has 5 basic machine cycles. They are

- Opcode fetch cycle (4T)
- Memory read cycle (3 T)
- Memory write cycle (3 T)
- I/O read cycle (3 T)
- I/O write cycle (3 T)

Processor Cycle

The function of the microprocessor is divided into fetch and execute cycle of any instruction of a program. The program is nothing but number of instructions stored in the memory in sequence. In the normal process of operation, the microprocessor fetches (receives or reads) and executes one instruction at a time in the sequence until it executes the halt (HLT) instruction. Thus, an instruction cycle is defined as the time required to fetch and execute an instruction. For executing any program, basically 2-steps are followed sequentially with the help of clocks

- Fetch, and
- Execute.

The time taken by the µP in performing the fetch and execute operations are called fetch and execute cycle. Thus, sum of the fetch and execute cycle is called the instruction cycle as indicated in Fig.

Instruction Cycle (IC) = Fetch cycle (FC) + Execute Cycle (EC)

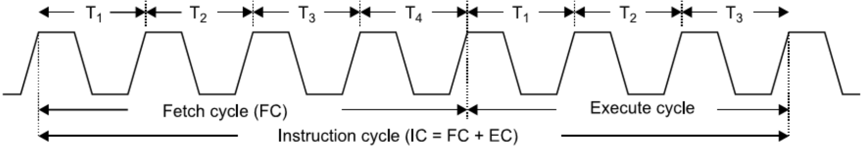


Fig. 1.7 Processor cycle

The 1st machine cycle of any instruction is always an Opcode fetch cycle in which the processor decides the nature of instruction. It is of at least 4-states. It may go up to 6-states.

In the opcode fetch cycle, the processor comes to know the nature of the instruction to be executed. The processor during (M1 cycle) puts the program counter contents on the address bus and reads the opcode of the instruction through read process. The T1, T2, and T3 clock cycles are used for the basic memory read operation and the T4 clock and beyond are used for its interpretation of the opcode. Based on these interpretations, the µP comes to know the type of additional information/data needed for the execution of the instruction and accordingly proceeds further for 1 or 2-machine cycle of memory read and writes.

Instruction Fetch (FC)⇒An instruction of 1 or 2 or 3-bytes is extracted from the memory locations during the fetch and stored in the µP's instruction register.

Instruction Execute (EC)⇒The instruction is decoded and translated into specific activities during the execution phase.

Opcode Fetch

The 1st step in communicating between the microprocessor and memory is reading from the memory. This reading process is called opcode fetch. The process of opcode fetch operation requires minimum 4-clock cycles T1, T2, T3, and T4and is the 1st machine cycle (M1) of every instruction. In order to differentiate between the data byte pertaining to an opcode or an address, the machine cycle takes help of the status signal IO/ M, S1, and S0. The IO/ M= 0 indicates memory operation and S1 = S0 = 1 indicates Opcode fetch operation. The opcode fetch machine cycle M1 consists of 4-states (T1, T2, T3, and T4). The 1st 3-states are used for fetching (transferring) the byte from the memory and the 4th-state is used to decode it.

Example

Fetch a byte 41H stored at memory location 2105H.

For fetching a byte, the microprocessor must find out the memory location where it is stored. Then provide condition (control) for data flow from memory to the microprocessor. The process of data flow and timing diagram of fetch operation are shown in Figs. 5.3 (a), (b), and (c). The µP fetches opcode of the instruction from the memory as per the sequence below

- A low IO/ M’ means microprocessor wants to communicate with memory.
- The µP sends a high on status signal S1 and S0 indicating fetch operation.
- The µP sends 16-bit address. AD bus has address in 1st clock of the 1st machine cycle, T1.
- AD7to AD0 address is latched in the external latch when ALE = 1.
- AD bus now can carry data.
- In T2, the RD control signal becomes low to enable the memory for read operation.
- The memory places opcode on the AD bus
- The data is placed in the data register (DR) and then it is transferred to IR.
- During T3the RDsignal becomes high and memory is disabled.

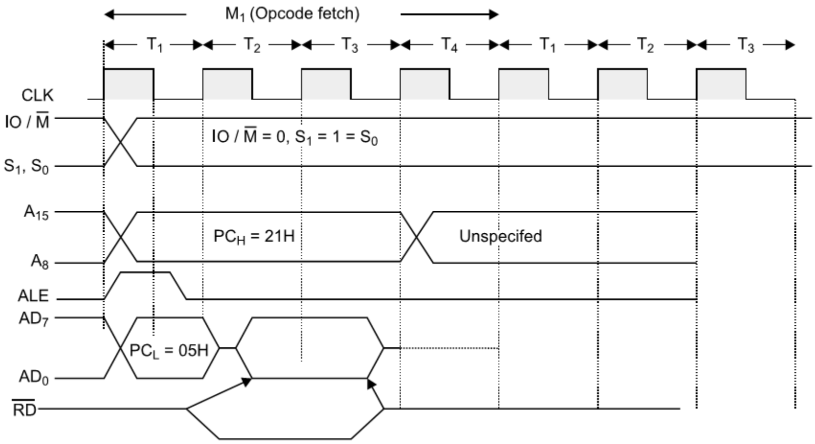


Fig. 1.8 Opcode Fetch

- During T4 the opcode is sent for decoding and decoded in T4.
- The execution is also completed in T4if the instruction is single byte.
- More machine cycles are essential for 2- or 3-byte instructions. The 1st machine cycleM1is meant for fetching the opcode. The machine cycles M2and M3are required either to read/ write data or address from the memory or I/O devices.

Memory and I/O Read Cycle

The memory read machine cycle is executed by the processor to read a data byte from memory. The processor takes 3T states to execute this cycle. The instructions which have more than one-byte word size will use the machine cycle after the opcode fetch machine cycle.

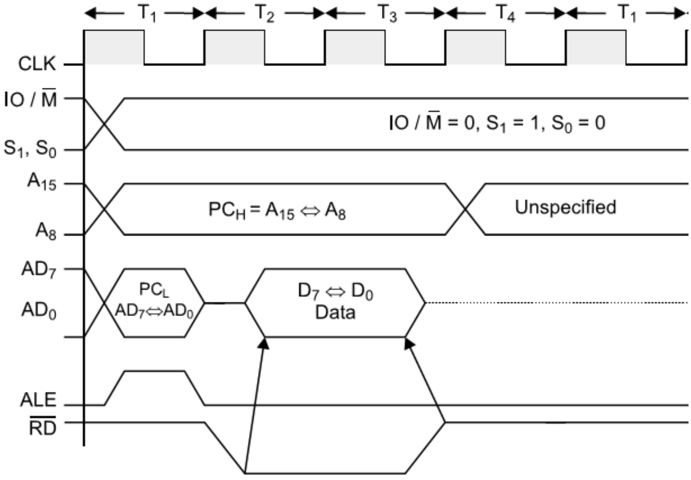


Fig. 1.9 Memory Read Cycle

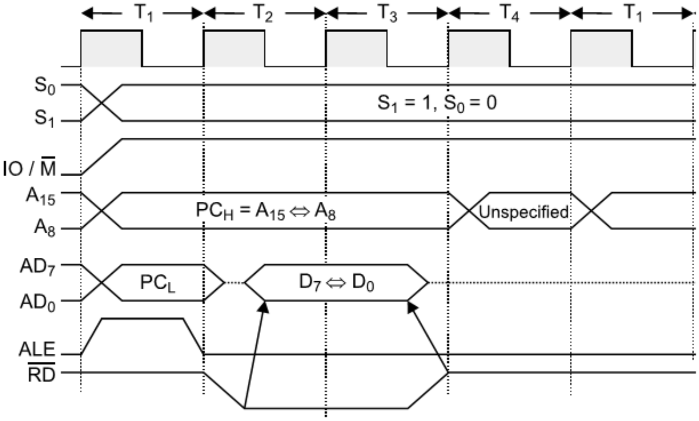


Fig. 1.10 I/O Read Cycle

The high order address (A15 ⇔A8) and low order address (AD7 ⇔AD0) are asserted on 1st low going transition of the clock pulse. The timing diagram for IO/M read are shown in Fig. The A15 ⇔A8 remains valid in T1, T2, and T3 i.e. duration of the bus cycle, but AD7⇔AD0 remains valid only in T1. Since it has to remain valid for the whole bus cycle, it must be saved for its use in the T2 and T3. ALE is asserted at the beginning of T1 of each bus cycle and is negated towards the end of T1. ALE is active during T1 only and is used as the clock pulse to latch the address (AD7⇔AD0) during T1. The RD' is asserted near the beginning of T2. It ends at the end of T3. As soon as the RD' becomes active, it forces the memory or I/O port to assert data. RD' becomes inactive towards the end of T3, causing the port or memory to terminate the data.

Memory and I/O Write Cycle

Immediately after the termination of the low order address, at the beginning of the T2, data is asserted on the address/data bus by the processor. WR' control is activated near the start of T2 and becomes inactive at the end of T3. The processor maintains valid data until after WR' is terminated. This ensures that the memory or port has valid data while WR' is active. It is clear from figures that for READ bus cycle, the data appears on the bus as a result of activating RD' and for the WR' bus cycle, the time the valid data is on the bus overlaps the time that the WR' is active.

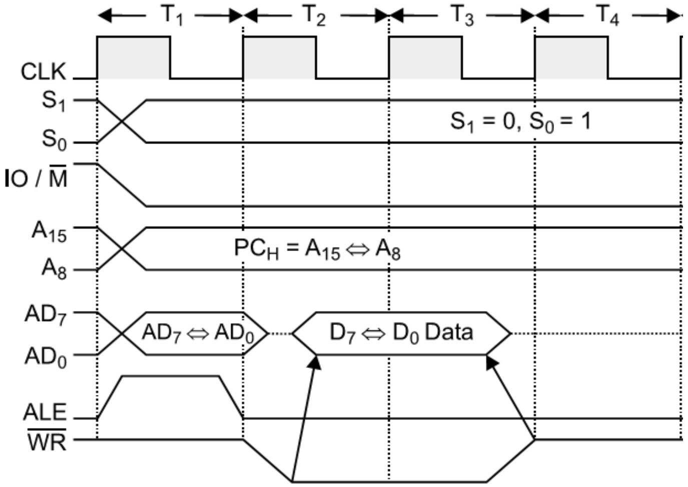


Fig. 1.11 Memory Write Cycle

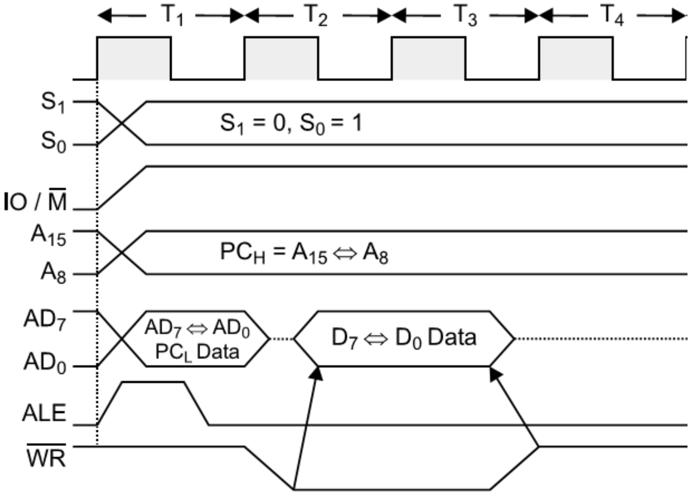


Fig. 1.12 I/O Write Cycle

Examples

Opcode fetch MOV B, C.

T1: The 1st clock of 1st machine cycle (M1) makes ALE high indicating address latch enabled which loads low-order address 00H on AD7⇔AD0 and high-order address 10H simultaneously on A15 ⇔A8. The address 00H is latched in T1.

T2: During T2 clock, the microprocessor issues RD control signal to enable the memory and memory places 41H from 1000H location on the data bus.

T3: During T3, the 41H is placed in the instruction register and RD= 1 (high) disables signal. It means the memory is disabled in T3 clock cycle. The opcode cycle is completed by end of T3 clock cycle.

T4: The opcode is decoded in T4 clock and the action as per 41H is taken accordingly. In other word, the content of C-register is copied in B-register.

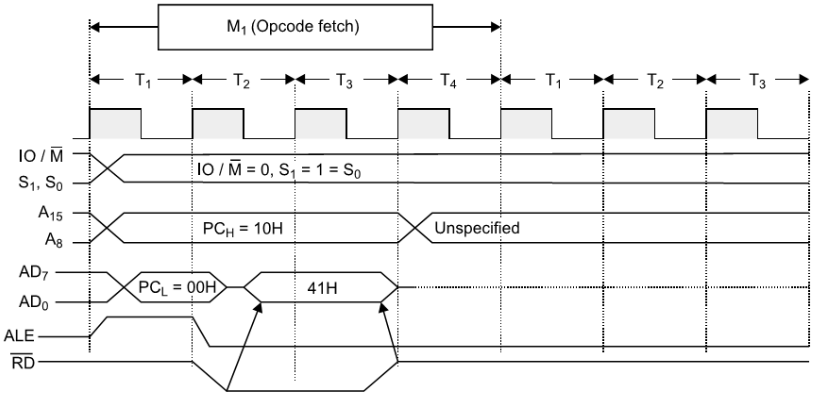
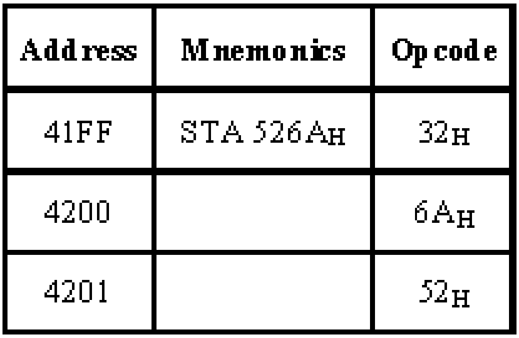


Fig. 1.13 Opcode Fetch (MOV B, C)

Timing diagram for STA $526A_H$

- STA means Store Accumulator -The contents of the accumulator is stored in the specified address (526A).
- The opcode of the STA instruction is said to be 32H. It is fetched from the memory 41FFH (see fig). - *OF machine cycle*
- Then the lower order memory address is read (6A). - *Memory Read Machine Cycle*
- Read the higher order memory address (52). -*Memory Read Machine Cycle*
- The combination of both the addresses are considered and the content from accumulator is written in 526A. - *Memory Write Machine Cycle*

- Assume the memory address for the instruction and let the content of accumulator is C7H. So, C7H from accumulator is now stored in 526A.



Address	Mnemonics	Opcode
41FF	STA 526A$_H$	32$_H$
4200		6A$_H$
4201		52$_H$

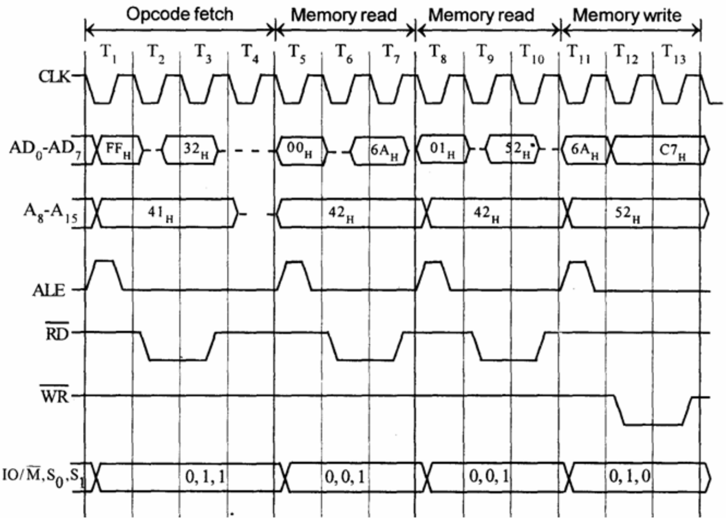
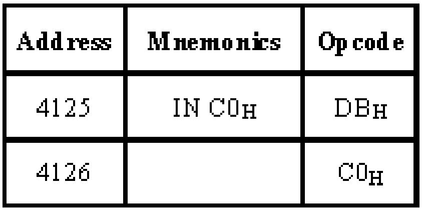


Fig. 1.14 Timing Diagram for STA 526A$_H$

Timing diagram for IN C0$_H$

- Fetching the Opcode DB$_H$ from the memory 4125$_H$.
- Read the port address C0$_H$ from 4126$_H$.
- Read the content of port C0$_H$ and send it to the accumulator.
- Let the content of port is 5E$_H$.



Address	Mnemonics	Opcode
4125	IN C0H	DBH
4126		C0H

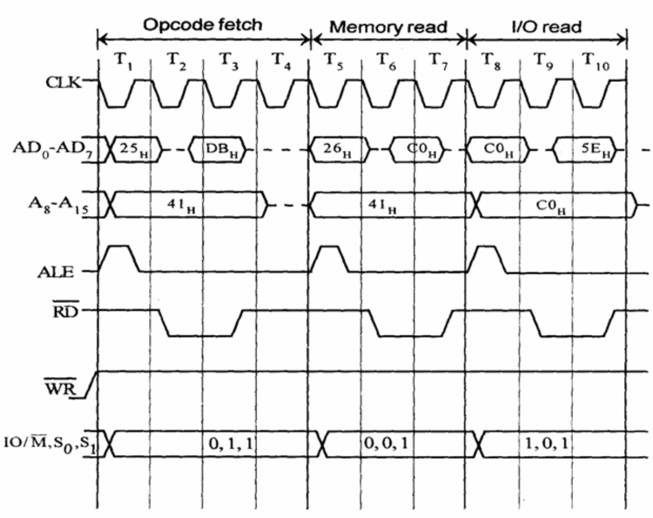


Fig. 1.15 Timing Diagram for IN C0$_H$

1.13 Assembly Language Programming

An assembly language is a low-level programming language for a computer, or other programmable device, in which there is a very strong (generally one-to-one) correspondence between the language and the architecture's machine code instructions. Each assembly language is specific to a particular computer architecture, in contrast to most high-level programming languages, which are generally portable across multiple architectures, but require interpreting or compiling.

Assembly language is converted into executable machine code by a utility program referred to as an assembler; the conversion process is referred to as assembly, or assembling the code.

Assembly language uses a mnemonic to represent each low-level machine operation or opcode. Some opcodes require one or more operands as part of the instruction, and most assemblers can take labels and symbols as operands to represent addresses and constants, instead of hard coding them into the program.

What is an Assembler?

An assembler is a software tool - a program -- designed to simplify the task of writing computer programs. If you have ever written a computer program directly in a machine-recognizable form such as binary or hexadecimal code, you will appreciate the advantages of programming in a symbolic assembly language.

Assembly language operation codes (opcodes) are easily remembered (MOV for move instructions, JMP for jump). You can also symbolically express addresses and values referenced in the operand field of instructions. Since you assign these names, you can make them as meaningful as the mnemonics for the instructions. For example, if your program manipulates a date as data, you can assign it the symbolic name DATE. If your program contains a set of instructions used as a timing loop (a set of instructions executed repeatedly until a specific amount of time has passed), you can name the instruction group TIMER.

What the Assembler Does

To use the assembler, you first need a source program. The source program consists of programmer written assembly language instructions. These instructions are written using mnemonic opcodes and labels. Assembly language source programs must be in a machine-readable form when passed to the assembler. TheIntellec development system includes a text editor that will help you maintain source programs as paper tape files or diskette files. You can then pass the resulting *source program file* to the assembler. The assembler program performs the clerical task of translating symbolic code into *object code* which can be executed by the 8080 and 8085 microprocessors. Assembler output consists of three possible files: the *object file*containing your program translated into object code; the *list file* printout of your source code, the assemble generated object code, and the symbol table; and the *symbol-crass-reference file,* a listing of the symbol-cross reference records.

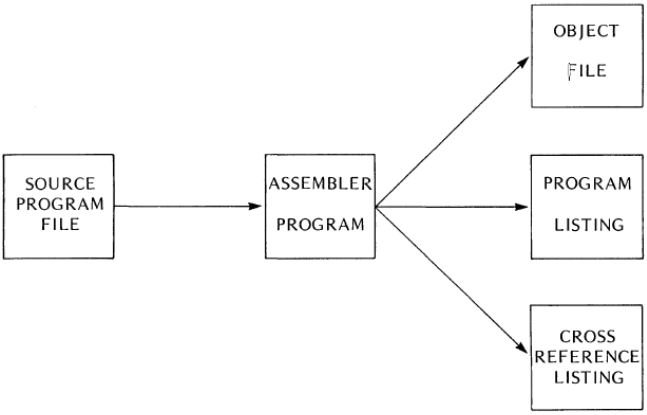


Fig. 1.16 Function of an Assembler

Example Programs

1. Statement: Store the data byte 32H into memory location 4000H.

Program 1

MVI A, 32H : Store 32H in the accumulator
STA 4000H : Copy accumulator contents at address 4000H
HLT : Terminate program execution

Program 2

LXI H : Load HL with 4000H
MVI M : Store 32H in memory location pointed by HL register
pair (4000H)
HLT : Terminate program execution

Statement: Exchange the contents of memory locations 2000H and 4000H

Program 1

LDA 2000H : Get the contents of memory location 2000H into
accumulator
MOV B, A : Save the contents into B register
LDA 4000H : Get the contents of memory location 4000H into
accumulator
STA 2000H : Store the contents of accumulator at address 2000H
MOV A, B : Get the saved contents back into A register
STA 4000H : Store the contents of accumulator at address 4000H

Program 2

LXI H 2000H : Initialize HL register pair as a pointer to memory
location
2000H.

```
        LXI D 4000H     : Initialize DE register pair as a pointer to memory
location                                                      4000H.
        MOV B, M        : Get the contents of memory location 2000H into B
register.
        LDAX D          : Get the contents of memory location 4000H into A
register.
        MOV M, A        : Store the contents of A register into memory location
2000H.
        MOV A, B        : Copy the contents of B register into accumulator.
        STAX D          : Store the contents of A register into memory location
4000H.
        HLT             : Terminate program execution.
```

Sample problem

(4000H) = 14H

(4001H) = 89H

Result = 14H + 89H = 9DH

Source program

```
LXI H 4000H     : HL points 4000H
MOV A, M        : Get first operand
INX H           : HL points 4001H
ADD M           : Add second operand
INX H           : HL points 4002H
MOV M, A        : Store result at 4002H
HLT             : Terminate program execution
```

Statement: Subtract the contents of memory location 4001H from the memorylocation 2000H and place the result in memory location 4002H.

Program - 4: Subtract two 8-bit numbers

Sample problem

(4000H) = 51H

(4001H) = 19H

Result = 51H - 19H = 38H

Source program

```
LXI H, 4000H    : HL points 4000H
```

```
MOV A, M      : Get first operand
INX H         : HL points 4001H
SUB M         : Subtract second operand
INX H         : HL points 4002H
MOV M, A      : Store result at 4002H.
HLT           : Terminate program execution
```

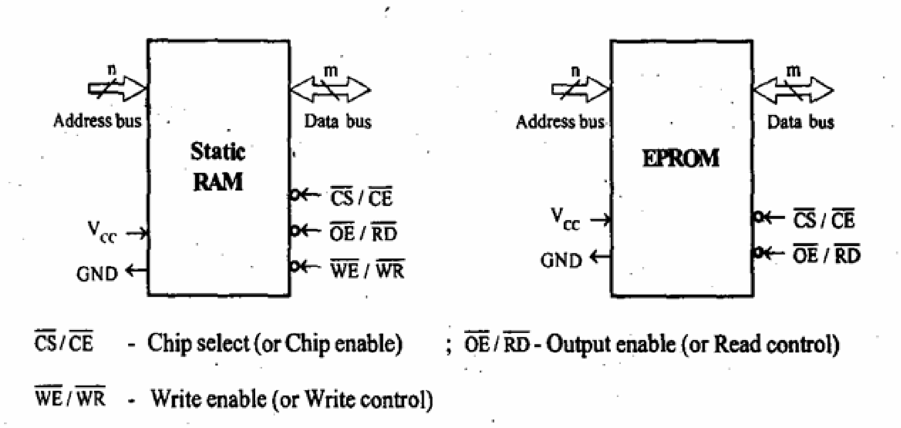
1.14 Memory Interfacing

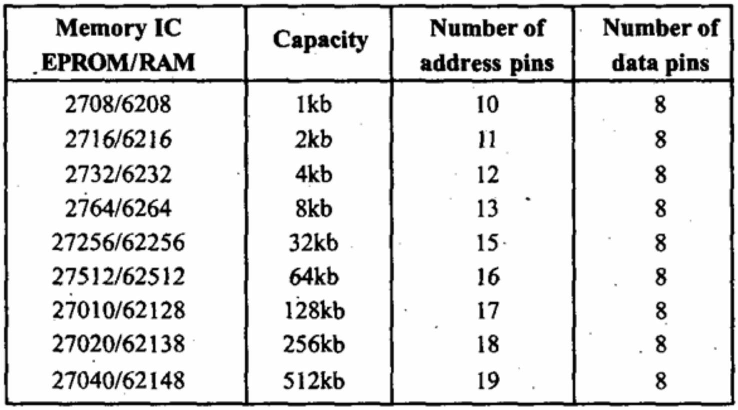
The memory is made up of semiconductor material used to store the programs and data. Three types of memory are,

- Process memory
- Primary or main memory
- Secondary memory

Typical EPROM and Static RAM

- A typical semiconductor memory IC will have ‘n’ address pins, ‘m’ data pins (or output pins).
- Having two power supply pins (one for connecting required supply voltage (V and the other for connecting ground).
- The control signals needed for static RAM are chip select (chip enable), read control (output enable) and write control (write enable).
- The control signals needed for read operation in EPROM are chip select (chip enable) and read control (output enable).


n
Address bus
Static RAM
m
Data bus
V_{cc}
GND
$\overline{CS}$ / $\overline{CE}$
$\overline{OE}$ / $\overline{RD}$
$\overline{WE}$ / $\overline{WR}$
n
Address bus
EPROM
m
Data bus
V_{cc}
GND
$\overline{CS}$ / $\overline{CE}$
$\overline{OE}$ / $\overline{RD}$
$\overline{CS}$/$\overline{CE}$ - Chip select (or Chip enable) ; $\overline{OE}$/$\overline{RD}$ - Output enable (or Read control)
$\overline{WE}$/$\overline{WR}$ - Write enable (or Write control)

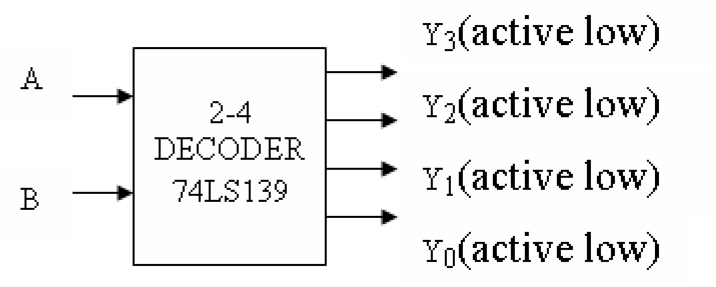


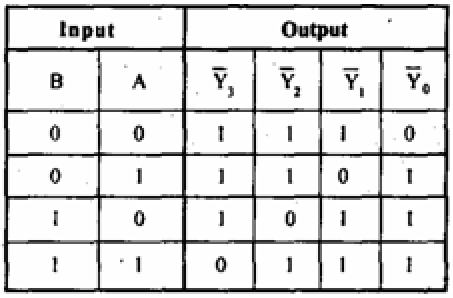
Memory IC EPROM/RAM	Capacity	Number of address pins	Number of data pins
2708/6208	1kb	10	8
2716/6216	2kb	11	8
2732/6232	4kb	12	8
2764/6264	8kb	13	8
27256/62256	32kb	15	8
27512/62512	64kb	16	8
27010/62128	128kb	17	8
27020/62138	256kb	18	8
27040/62148	512kb	19	8

Decoder

It is used to select the memory chip of processor during the execution of a program. No of IC's used for decoder is,

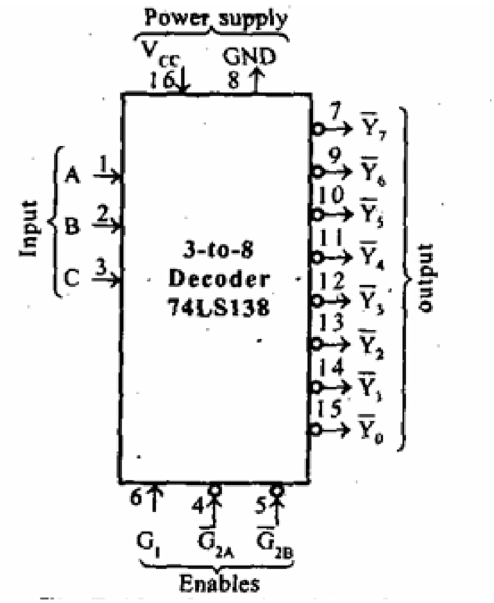
- 2-4 decoder (74LS139)
- 3-8 decoder (74LS138)

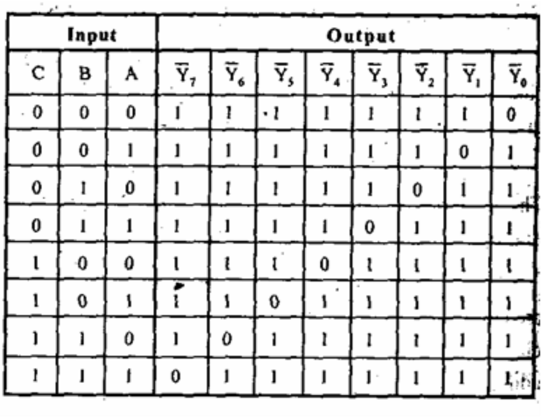




Input		Output			
B	A	$\overline{Y}_3$	$\overline{Y}_2$	$\overline{Y}_1$	$\overline{Y}_0$
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1

Fig 1.17 Block diagram and Truth table of 2-4 decoder





Input			Output							
C	B	A	$\overline{Y}_7$	$\overline{Y}_6$	$\overline{Y}_5$	$\overline{Y}_4$	$\overline{Y}_3$	$\overline{Y}_2$	$\overline{Y}_1$	$\overline{Y}_0$
0	0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0	1
0	1	0	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	1	1	0	1	1	1	1
1	0	1	1	1	0	1	1	1	1	1
1	1	0	1	0	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1

Fig 1.18 Block diagram and Truth table of 3-8 decoder

Memory Interfacing

The following are the steps involved in interfacing memory with 8085 processor.

1. First decide the size of memory requires to be interfaced. Depending on this we can say how many address lines are required for it. For example, if you want to interface 4KB (2^{12}) memory it requires 12 address lines. Remaining address lines can be used in address decoding.
2. Depending on the size of memory required and given address range, construct address decoding circuitry. This address decoding circuitry

can be implemented with NAND gates and/or decoders or using PAL (when board size is a constraint).

3. Connect data bus of memory to processor data bus.
4. Generate the control signals required for memory using IO/M’, WR’, RD’ signals of 8085 processor.

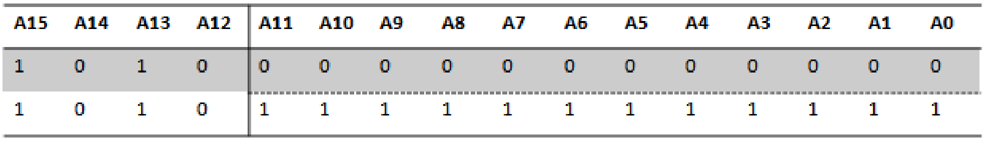
Address Decoding

- The result of ‘address decoding’ is the identification of a register for a given address.
- A large part of the address bus is usually connected directly to the address inputs of the memory chip.
- This portion is decoded internally within the chip.
- What concerns us is the other part that must be decoded externally to select the chip.
- This can be done either using logic gates or a decoder.

Example

Interface 4KB memory to 8085 with starting address A000H.

1. 4KB memory requires 12 address lines for addressing as already mentioned. But 8085 has 16 address lines. Hence four of address lines are used for address decoding
2. Given that starting address for memory is A000H. So, for 4KB memory ending address becomes A000H+0FFFH (4KB) = AFFFH.



A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1

A0-A11 address lines are directly connected to address bus of memory chip. A12-A15 are used for generating chip select signal for memory chip.

Address decoding circuit using 3X8 decoder

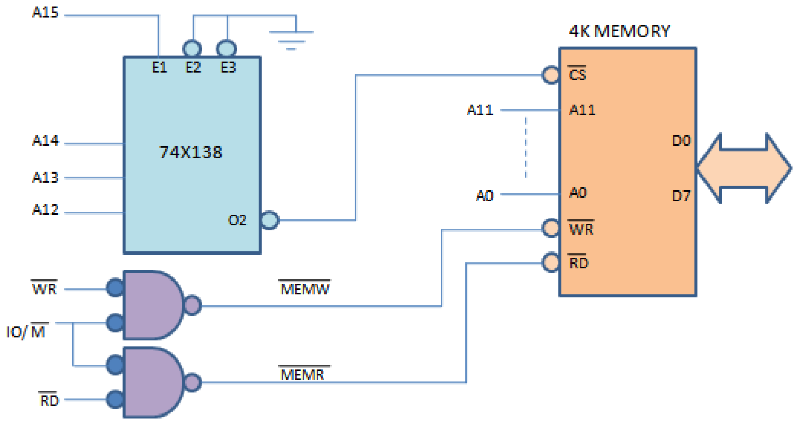


Fig. 1.19Address decoding circuit using 3X8 decoder

A15 line is use for enabling 74x138 decoder chip. A12, A13, A14 lines are connected to 74X138 chip as inputs. When these lines are 010 output should be '0'. This is provided at O2 pin of 74X138 chip.

Address decoding circuit using only NAND gates:

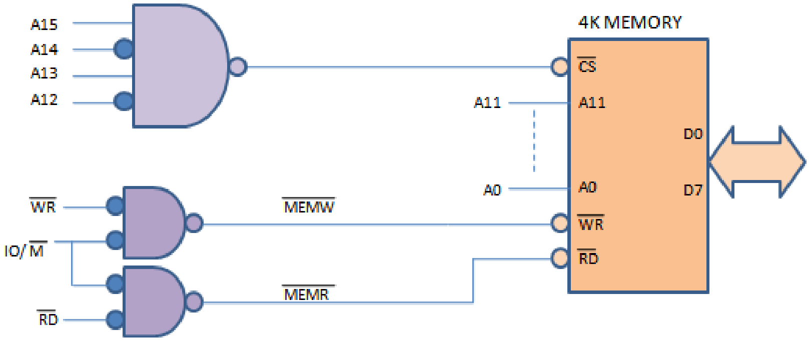


Fig. 1.20Address decoding circuit using NAND gates

A15, A14, A13, A12 inputs should be 1010, for enabling the chip. So, the circuit for this is as shown above.

Types of address decoding

There are two types of address decoding mechanism, based on address lines used for generating chip select signal.

1. Absolute decoding
2. Partial decoding

Absolute decoding

All the higher order lines of microprocessor, left after using the required signals for memory are completely used for generating chip select signal as shown in above example. This type of decoding is called absolute decoding.

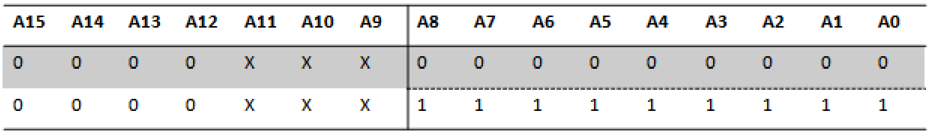
Partial decoding

Only some of the address lines of microprocessor left after using the required signals for memory are used for generating chip select signal. Because of this multiple address ranges will be formed. If total memory space is not required for the system then, this type of address decoding can be used. The advantage of this technique is fewer components are required for memory interfacing because of this board size reduces and in turn cost reduces.

Example

Connect 512 bytes of memory to 8085

1. For interfacing 512 bytes 9 address lines are required. So A0-A8 can be used to directly connect to address bus of memory.
2. In the remaining A9-A15 for example only A15-A12 are used for generating chip select signal. A11-A9 are don't care signals.



A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	0	0	0	X	X	X	0	0	0	0	0	0	0	0	0
0	0	0	0	X	X	X	1	1	1	1	1	1	1	1	1

Because of the don't care signals the address range can be

0000 to 01FF
0200 to 03FF
0400 to 05FF
0600 to 07FF
0800 to 09FF
0A00 to 0BFF
0C00 to 0DFF
0E00 to 0FFF

Address decoding circuit

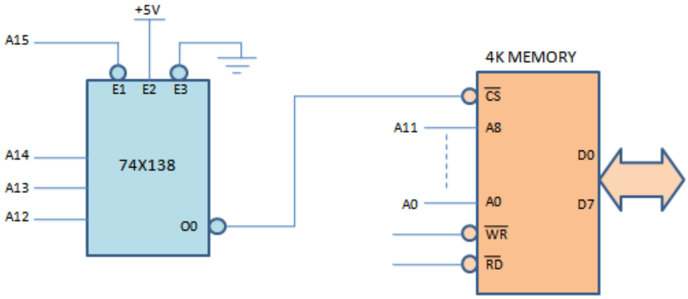


Fig. 1.21 Address decoding

Example

Consider a system in which 32kb memory space is implemented using four numbers of 8kb memory. Interface the EPROM and RAM with 8085 processor.

The total memory capacity is 32Kb. So, let two number of 8kb n memory be EPROM and the remaining two numbers be RAM. Each 8kb memory requires 13 address lines and so the address lines A0- A12 of the processor are connected to 13 address pins of all the memory. The address lines and A13 - A14 can be decoded using a 2-to-4 decoder to generate four chips select signals. These four chips select signals can be used to select one of the four memory IC at any one time. The address line A15 is used as enable for decoder. The simplified schematic memory organization is shown.

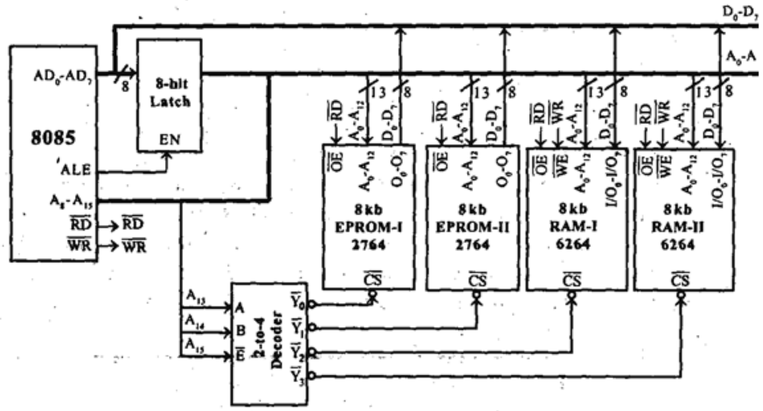
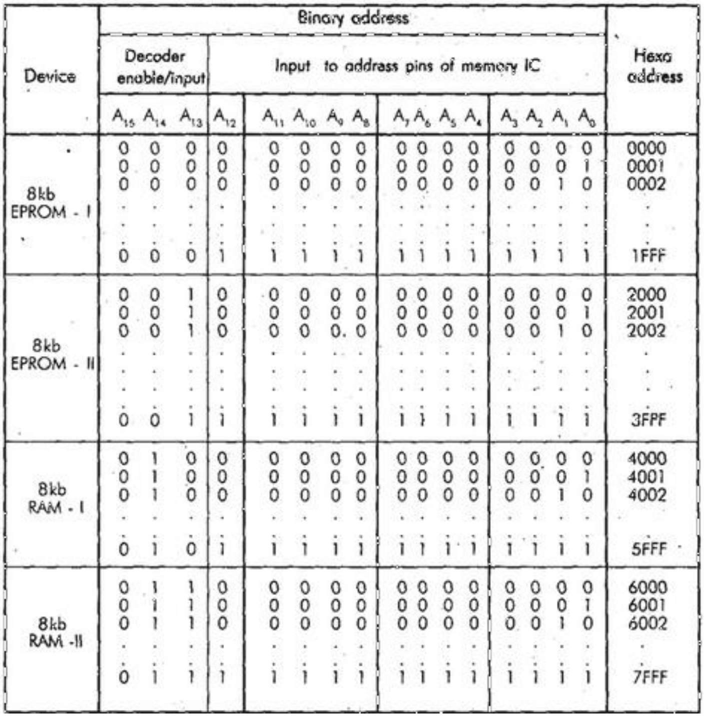


Fig.1.22 Interfacing 16KB EPROM and 16KB RAM with 8085

The address allotted to each memory IC is shown in following table.



Device	Binary address																Hexa address
	Decoder enable/input			Input to address pins of memory IC													
	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
8kb EPROM - I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF
8kb EPROM - II	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	2001
	0	0	1	0	0	0	0.	0	0	0	0	0	0	0	1	0	2002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFF
8kb RAM - I	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4001
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	4002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	5FFF
8kb RAM -II	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6000
	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	6001
	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	6002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7FFF

1.15 Interfacing I/O Devices

- Using I/O devices data can be transferred between the microprocessor and the outside world.
- This can be done in groups of 8 bits using the entire data bus. This is called parallel I/O.
- The other method is serial I/O where one bit is transferred at a time using the SID and SOD pins on the Microprocessor.
- There are two ways to interface 8085 with I/O devices in parallel data transfer mode:
- Memory Mapped IO
- IO Mapped IO

Memory mapped I/O

I/O devices are interfaced using address from memory space. That means IO device address are part of addresses given to memory locations.8085 uses 16-bit address to memory interfacing. So, any address between 0000H-FFFFH can be given to each peripheral. But the addresses given to peripheral can’t be used for memory. Memory control signals are used as read and write control signals for peripherals. And all the operations that can be performed on memory can also be performed on peripherals. No need of using IO instructions such as IN, OUT.

IO mapped I/O

In this method separate address space is given to IO devices. Each IO device is given an 8-bit address. Hence maximum 256 devices can be interfaced to the processor. The address range for the IO devices is 00H-FFH. IO control signals are used to perform read, write operations. For reading data from IO device or writing data to IO device IN, OUT instructions needs to be used. Arithmetic and logical operations can't be performed directly on IO devices as in memory mapped IO. IO devices can be interfaced, by using buffers for simple IO i.e. by using address decoding circuit to enable buffer. For handshake IO or to interface more peripherals ICs like 8255 peripheral programmable interface (PPI) can be used.

IO mapped IO vs. Memory Mapped IO

Memory Mapped IO	IO mapped IO
IO is treated as memory.	IO is treated IO.
16-bit addressing.	8- bit addressing.
More Decoder Hardware.	Less Decoder Hardware.
Can address 2^{16}=64k locations.	Can address 2^{8}=256 locations.
Less memory is available.	Whole memory address space is available.
Memory Instructions are used.	Special Instructions are used like IN, OUT.
Memory control signals are used.	Special control signals are used.
Arithmetic and logic operations can be performed on data.	Arithmetic and logic operations cannot be performed on data.
Data transfer b/w register and IO.	Data transfer b/w accumulator and IO.

The interfacing of output devices

Output devices are usually slow.

- Also, the output is usually expected to continue appearing on the output device for a long period of time.
- Given that the data will only be present on the data lines for a very short period (microseconds), it has to be latched externally.

- To do this the external latch should be enabled when the port's address is present on the address bus, the IO/M signal is set high and WR is set low.
- The resulting signal would be active when the output device is being accessed by the microprocessor.
- Decoding the address bus (for memory-mapped devices) follows the same techniques discussed in interfacing memory.

Interfacing of Input Devices

- The basic concepts are similar to interfacing of output devices.
- The address lines are decoded to generate a signal that is active when the particular port is being accessed.
- An IORD signal is generated by combining the IO/M and the RD signals from the microprocessor.
- A tri-state buffer is used to connect the input device to the data bus.
- The control (Enable) for these buffers is connected to the result of combining the address signal and the signal IORD.

1.16 Applications of Microprocessor in General Life

There are a lot of applications of Microprocessor in general life. Some of the applications are given below

- Mobile Phones
- Digital Watches
- Washing Machine
- Computer
- Lighting Control
- Traffic Control
- LAPTOP
- Modems
- Power Stations
- Television
- CD Player
- Multimeter
- CRO
- Wave generator
- More applications in medical

UNIT II – PERIPHERALS INTERFACING

2.1 Introduction

Peripheral Interfacing is considered to be a main part of Microprocessor, as it is the only way to interact with the external world. The interfacing happens with the ports of the Microprocessor.

The main IC's which are to be interfaced with 8085 are:

- 8255 PPI
- 8259 PIC
- 8251 USART
- 8279 Key board display controller
- 8253 Timer/ Counter
- A/D and D/A converter interfacing.

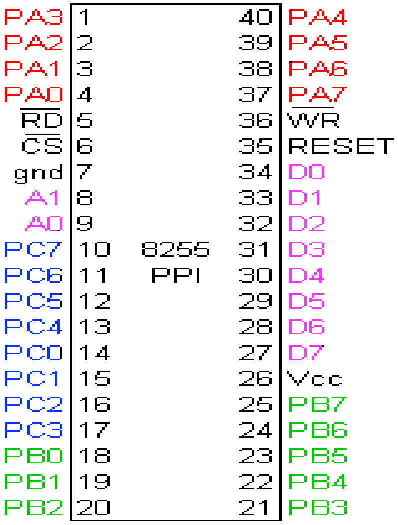
2.2 Programmable Peripheral Interface 8255

The 8255 is a widely used, programmable, parallel I/O device. It can be programmed to transfer data under various conditions, from simple I/O to interrupt I/O. It is flexible, versatile and economical and complex.

Features

- Three 8-bit IO ports PA, PB, PC
- PA can be set for Modes 0, 1, 2. PB for 0,1 and PC for mode 0 and for BSR. Modes 1 and 2 are interrupt driven.
- PC has 2 4-bit parts: PC upper (PCU) and PC lower (PCL), each can be set independently for I or O. Each PC bit can be set/reset individually in BSR mode.
- PA and PCU are Group A (GA) and PB and PCL are Group B (GB)
- Address/data bus must be externally demultiplexed.
- TTL compatible.
- Improved dc driving capability.

Pin diagram



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

Fig. 2.1 8255 Pin Diagram

Block Diagram

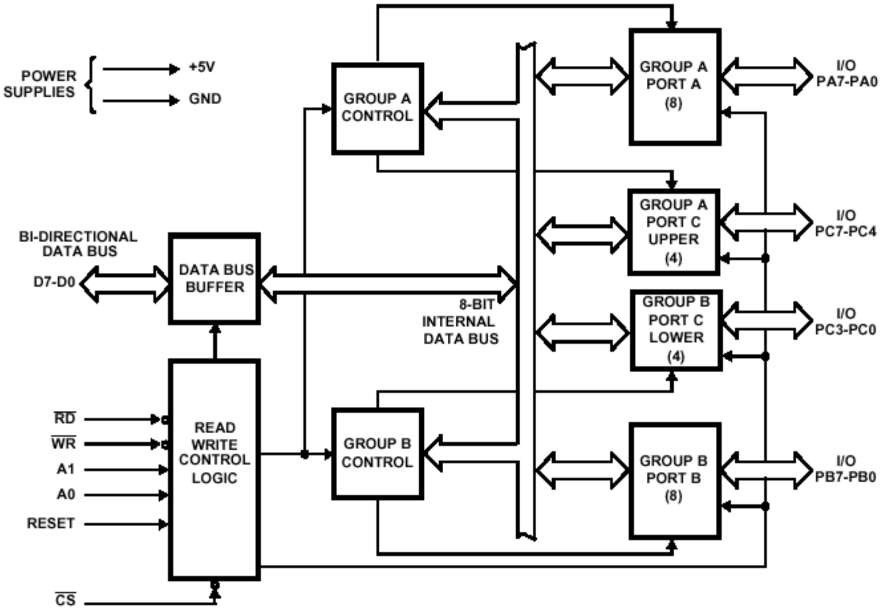


Fig. 2.2 8255 Block Diagram

Data Bus Buffer: This three-state bi-directional 8-bit 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 information 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 inputs 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.

(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.

A1	A0	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 systems software. In essence, the CPU "outputs" a control word to the 8255. The control word contains information such as "mode", "bit set", "bit reset", 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 "pull-down" bus-hold devices are present on 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.

Operational modes of 8255

There are two basic operational modes of 8255:

- Bit set/reset Mode (BSR Mode).
- Input/output Mode (I/O Mode).

The two modes are selected on the basis of the value present at the D7 bit of the Control Word Register. When D7 = 1, 8255 operates in I/O mode and when D7 = 0, it operates in the BSR mode.

Bit set/reset (BSR) mode

The Bit Set/Reset (BSR) mode is applicable to port C only. Each line of port C (PC0 - PC7) can be set/reset by suitably loading the control word register. BSR mode and I/O mode are independent and selection of BSR mode does not affect the operation of other ports in I/O mode.

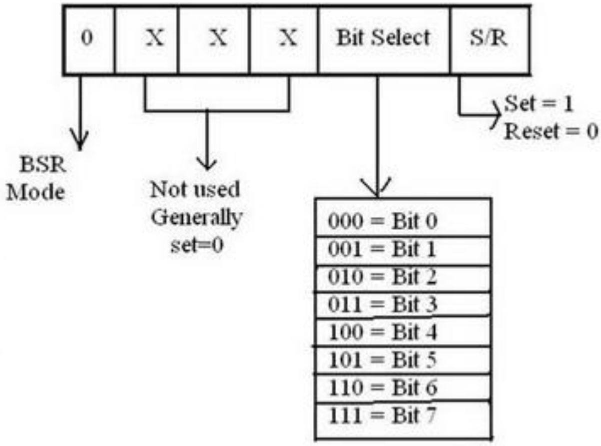


Fig. 2.3 control word BSR mode

- D7 bit is always 0 for BSR mode.
- Bits D6, D5 and D4 are don't care bits.
- Bits D3, D2 and D1 are used to select the pin of Port C.
- Bit D0 is used to set/reset the selected pin of Port C.

Selection of port C pin is determined as follows

B3	B2	B1	Bit/pin of port C selected
0	0	0	PC0
0	0	1	PC1
0	1	0	PC2
0	1	1	PC3
1	0	0	PC4
1	0	1	PC5
1	1	0	PC6
1	1	1	PC7

Input/Output mode

This mode is selected when D7 bit of the Control Word Register is 1.

There are three I/O modes:

- Mode 0 - Simple I/O
- Mode 1 - Strobed I/O
- Mode 2 - Strobed Bi-directional I/O

D0, D1, D3, D4 are assigned for lower port C, port B, upper port C and port A respectively. When these bits are 1, the corresponding port acts as an input port. For e.g., if D0 = D4 = 1, then lower port C and port A act as input ports. If these bits are 0, then the corresponding port acts as an output port. For e.g., if D1 = D3 = 0, then port B and upper port C act as output ports. D2 is used for mode selection of Group B (port B and lower port C). When D2 = 0, mode 0 is selected and when D2 = 1, mode 1 is selected.

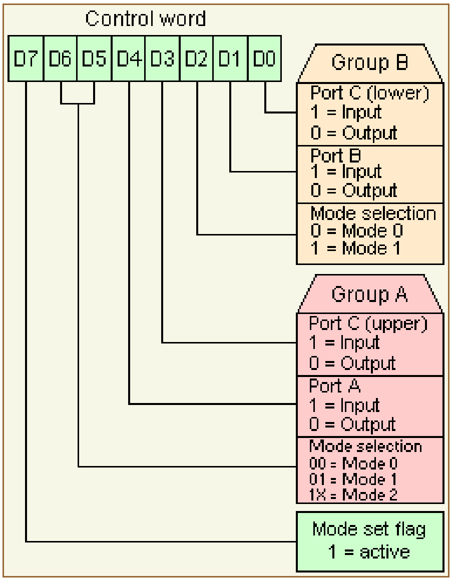


Fig. 2.4 Control word I/O mode

D5 & D6 are used for mode selection of Group A (port A and upper port C). The selection is done as follows:

D6	D5	Mode
0	0	0
0	1	1
1	X	2

As it is I/O mode, D7 = 1.

Mode 0 - Simple I/O

In this mode, the ports can be used for simple I/O operations without handshaking signals. Port A, port B provide simple I/O operation. The two halves of port C can be either used together as an additional 8-bit port, or they can be used as individual 4-bit ports. Since the two halves of port C are independent, they may be used such that one-half is initialized as an input port while the other half is initialized as an output port.

- The input/output features in mode 0 are as follows:
- Output ports are latched.
- Input ports are buffered, not latched.
- Ports do not have handshake or interrupt capability.
- With 4 ports, 16 different combinations of I/O are possible.

Mode 0 – input mode

- In the input mode, the 8255 gets data from the external peripheral ports and the CPU reads the received data via its data bus.
- The CPU first selects the 8255 chip by making CS' low. It then selects the desired port using A0 and A1 lines.
- The CPU then issues an RD' signal to read the data from the external peripheral device via the system data bus.

Mode 0 - Output mode

- In the output mode, the CPU sends data to 8255 via system data bus and then the external peripheral ports receive this data via 8255 port.
- CPU first selects the 8255 chip by making CS' low. It then selects the desired port using A0 and A1 lines.
- CPU then issues a WR' signal to write data to the selected port via the system data bus. This data is then received by the external peripheral device connected to the selected port.

Mode 1

When we wish to use port A or port B for handshake (strobed) input or output operation, we initialise that port in mode 1 (port A and port B can be initialised to operate in different modes, i.e., for e.g., port A can operate in mode 0 and port B in mode 1). Some of the pins of port C function as handshake lines. For port B in this mode (irrespective of whether is acting as an input port or output port), PC0, PC1 and PC2 pins function as handshake lines. If port A is initialised as mode 1 input port, then, PC3, PC4 and PC5 function as handshake signals. Pins PC6 and PC7 are available for use as input/output lines.

The mode 1 which supports handshaking has following features:

- Two ports i.e. port A and B can be used as 8-bit i/o ports.
- Each port uses three lines of port c as handshake signal and remaining two signals can be used as i/o ports.
- Interrupt logic is supported.
- Input and Output data are latched.

Input Handshaking signals

1. IBF(Input Buffer Full)-It is an output indicating that the input latch contains information.

2. STB(Strobed Input)-The strobe input loads data into the port latch, which holds the information until it is input to the microprocessor via the IN instruction.
3. INTR(Interrupt request)-It is an output that requests an interrupt. The INTR pin becomes a logic 1 when the STB input returns to a logic 1, and is cleared when the data are input from the port by the microprocessor.
4. INTE(Interrupt enable)-It is neither an input nor an output; it is an internal bit programmed via the port PC4(port A) or PC2(port B) bit position.

Output Handshaking signals

1. OBF(Output Buffer Full)-It is an output that goes low whenever data are output(OUT) to the port A or port B latch. This signal is set to a logic 1 whenever the ACK pulse returns from the external device.
2. ACK(Acknowledge)-It causes the OBF pin to return to a logic 1 level. The ACK signal is a response from an external device, indicating that it has received the data from the 82C55 port.
3. INTR(Interrupt request)-It is a signal that often interrupts the microprocessor when the external device receives the data via the signal. this pin is qualified by the internal INTE(interrupt enable) bit.
4. INTE(Interrupt enable)-It is neither an input nor an output; it is an internal bit programmed to enable or disable the INTR pin. The INTE A bit is programmed using the PC6 bit and INTE B is programmed using the PC2 bit.

<table>
<tr><th colspan="8">PC bits in input mode:</th></tr>
<tr><td>D7</td><td>D6</td><td>D5</td><td>D4</td><td>D3</td><td>D2</td><td>D1</td><td>D0</td></tr>
<tr><td>PC7</td><td>PC6</td><td>IBF-A</td><td>INTE-A / STB-A-bar</td><td>INTR-A</td><td>INTE-B / STB-B-bar</td><td>IBF-B</td><td>INTR-B</td></tr>
<tr><th colspan="8">PC bits in output mode:</th></tr>
<tr><td>D7</td><td>D6</td><td>D5</td><td>D4</td><td>D3</td><td>D2</td><td>D1</td><td>D0</td></tr>
<tr><td>OBF-A-bar</td><td>INTE-A / ACK-A-bar</td><td>PC5</td><td>PC4</td><td>INTR-A</td><td>INTE-B / ACK-B-bar</td><td>OBF-B-bar</td><td>INTR-B</td></tr>
</table>

Mode 2

Only group A can be initialised in this mode. Port A can be used for bidirectional handshake data transfer. This means that data can be input or output on the same eight lines (PA0 - PA7). Pins PC4 - PC7 are used as handshake lines for port A. The remaining pins of port C (PC0 - PC3) can be used as input/output lines if group B is initialised in mode 0 or as handshaking for port b if group B is initialised in mode 1. In this mode, the 8255 may be used to extend the system bus to a slave microprocessor or to transfer data bytes to and from a floppy disk controller.

D7	D6	D5	D4	D3	D2	D1	D0
OBF-A-bar	INTE1(O/P) / ACK-A-BAR	IBF-A	INTE2(I/P) / STB-A-bAR	INTR-A	X	X	X

2.3 Programmable Interval Timer - 8253

The 8253 solves one of most common problem in any microcomputer system, the generation of accurate time delays under software control. Instead of setting up timing loops in system software, the programmer configures the 8253 to match his requirements, initializes one of the counters of the 8253 with the desired quantity, then upon command the 8253 will count out the delay and interrupt the CPU when it has completed its tasks.

It is easy to see that the software overhead is minimum and that multiple delays can be easily be maintained by assignment of priority levels. The 8253 includes three identical 16-bit counters that can operate independently. To operate a counter, a 16-bit count is loaded in its register and, on command, it begins to decrement the count until it reaches 0. At the end of the count, it generates a pulse that can be used to interrupt the CPU. The counter can count either in binary or BCD. In addition, a count can be read by the CPU while the counter is decrementing.

Features

1. Three independent 16-bit down counters.
2. 8253 can operate from DC up to 2.6 MHz
3. Three counters are identical presettable, and can be programmed for either binary or BCD count.
4. Counter can be programmed in six different modes.
5. Compatible with all Intel and most other microprocessors.

Pin Diagram

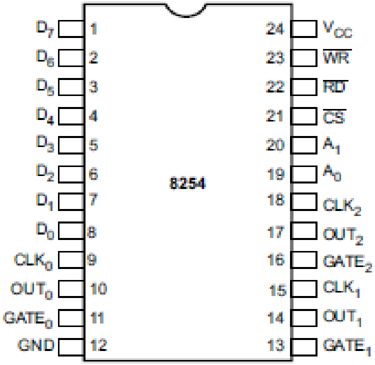


Fig. 2.5 8254 Pin Diagram

Block Diagram

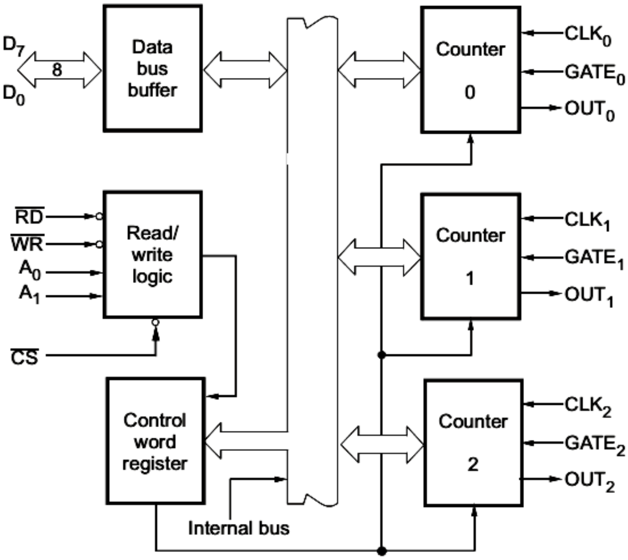


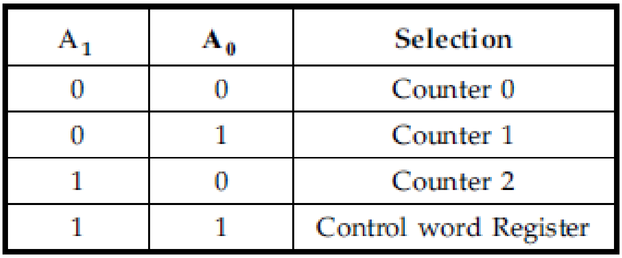
Fig. 2.6 8254 Block Diagram

It includes three counters, a data bus buffer, Read/Write control logic, and a control register. Each counter has two input signals CLOCK and GATE and one output signal OUT.

Data Bus Buffer: This tri-state, bi-directional, 8-bit buffer is used to interface the 8253 to the system data bus. The Data bus buffer has three basic functions.

1. Programming the modes of 8253.
2. Loading the count registers.
3. Reading the count values.

Read/Write Logic: The Read/Write logic has five signals: RD, WR, CS and the address lines A0 and A1. In the peripheral I/O mode, the RD, and WR signals are connected to IOR and IOW, respectively. In memory-mapped I/O, these are connected to MEMR and MEMW. Address lines A0 and A1 of the CPU are usually connected to lines A0 and A1 of the 8253, and CS is tied to a decoded address. The control word register and counters are selected according to the signals on lines A0 and A1. It includes three counters, a data bus buffer, Read/Write control logic, and a control register. Each counter has two input signals CLOCK and GATE and one output signal OUT.



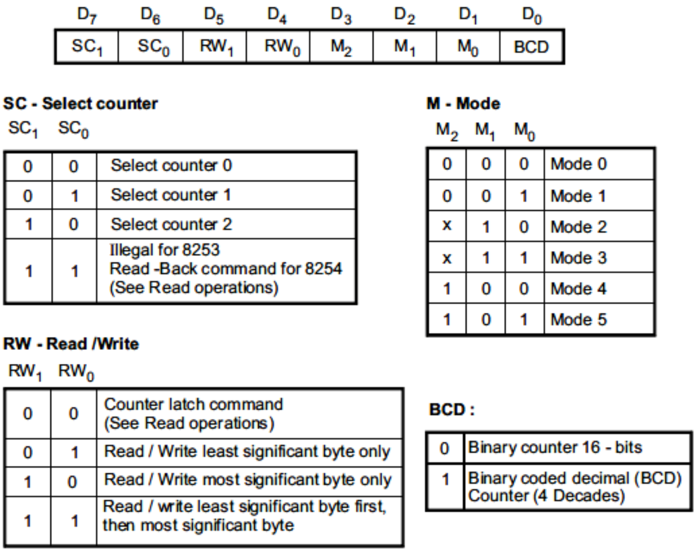
A_1	A_0	Selection
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Control word Register

Control Word Register: This register is accessed when lines A0 and A1 are at logic 1. It is used to write a command word which specifies the counter to be used (binary or BCD), its mode, and either a read or write operation.

Counters: These three functional blocks are identical in operation. Each counter consists of a single, 16-bit, pre-settable, down counter. The counter can operate in either binary or BCD and its input, gate and output are configured by the selection of modes stored in the control word register. The counters are fully independent. The programmer can read the contents of any of the three counters without disturbing the actual count in process.

Operation Description

The complete functional definition of the 8253 is programmed by the system software. Once programmed, the 8253 is ready to perform whatever timing tasks it is assigned to accomplish.



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC_1	SC_0	RW_1	RW_0	M_2	M_1	M_0	BCD

SC - Select counter

SC_1	SC_0	
0	0	Select counter 0
0	1	Select counter 1
1	0	Select counter 2
1	1	Illegal for 8253 Read -Back command for 8254 (See Read operations)

M - Mode

M_2	M_1	M_0	
0	0	0	Mode 0
0	0	1	Mode 1
x	1	0	Mode 2
x	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

RW - Read /Write

RW_1	RW_0	
0	0	Counter latch command (See Read operations)
0	1	Read / Write least significant byte only
1	0	Read / Write most significant byte only
1	1	Read / write least significant byte first, then most significant byte

BCD :

0	Binary counter 16 - bits
1	Binary coded decimal (BCD) Counter (4 Decades)

Mode 0: Interrupt on terminal count

1. The output will be initially low after the mode set operation.
2. After the count is loaded into the selected count Register the output will remain low and the counter will count.
3. When the terminal count is reached the output will go high and remain high until the selected count is reloaded.

Mode 1: Hardware Retriggerable One-shot

1. The output will be initially high
2. The output will go low on the CLK pulse following the rising edge at the gate input.
3. The output will go high on the terminal count and remain high until the next rising edge at the gate input.

Mode 2: Rate generator

This mode functions like a divide by-N counter.

1. The output will be initially high.
2. The output will go low for one clock pulse before the terminal count.
3. The output then goes high, the counter reloads the initial count and the process is repeated.
4. The period from one output pulse to the next equals the number of input counts in the count register.

Mode 3: Square wave mode

1. Initially output is high.
2. For even count, counter is decremented by 2 on the falling edge of each clock pulse. When the counter reaches terminal count, the state of the output is changed and the counter is reloaded with the full count and the whole process is repeated.
3. If the count is odd and the output is high the first clock pulse (after the count is loaded) decrements the count by 1. Subsequent clock pulses decrement the clock by 2. After timeout, the output goes low and the full count is reloaded. The first clock pulse decrements the count by 3 and subsequent clock pulse decrement the count by two. Then the whole process is repeated. In this way, if the count is odd, the output will be high for (n+1)/2 counts and low for (n-1)/2 counts.

Mode 4: Software Triggered Strobe

1. The output will be initially high
2. The output will go low for one CLK pulse after the terminal count (TC).

Mode 5: Hardware triggered strobe (Retriggerable)

1. The output will be initially high.
2. The counting is triggered by the rising edge of the Gate.
3. The output will go low for one CLK pulse after the terminal count (TC).

Programming the 8253

Each counter of the 8253 is individually programmed by writing a control word into the control word register (A0 - A1 = 11). The above figure shows the control word format. Bits SC1 and SC0 select the counter, bits RW1 and RW0 select the read, write or latch command, bits M2, M1 and M0 select the mode of operation and bit BCD decides whether it is a BCD counter or binary counter.

WRITE Operation

1. Write a control word into control register.
2. Load the low-order byte of a count in the counter register.
3. Load the high-order byte of count in the counter register.

READ Operation

In some applications, especially in event counters, it is necessary to read the value of the count in process. This can be done by following possible methods:

Simple Read: It involves reading a count after inhibiting the counter by controlling the gate input or the clock input of the selected counter, and two I/O read operations are performed by the CPU. The first I/O operation reads the low-order byte, and the second I/O operation reads the high order byte.

Counter Latch Command: In the second method, an appropriate control word is written into the control register to latch a count in the output latch, and two I/O read operations are performed by the CPU. The first I/O operation reads the low-order byte, and the second I/O operation reads the high order byte.

2.4 Programmable Interrupt Controller-8259

The Intel 8259A Programmable Interrupt Controller handles up to eight vectored priority interrupts for the CPU. It is cascadable for up to 64 vectored priority interrupts without additional circuitry. It is packaged in a 28-pin DIP, uses NMOS technology and requires a single a 5V supply. Circuitry is static, requiring no clock input. The 8259A is designed to minimize the software and real time overhead in handling multi-level priority interrupts. It has several modes, permitting optimization for a variety of system requirements. The PIC receives an interrupt request from an I/O device and tells the microprocessor. The CPU completes whatever instruction it is currently executing and then fetches a new routine that will service the requesting device. Once this peripheral service is completed, the CPU resumes doing exactly what it was doing when the interrupt request occurred. The PIC functions as an overall manager of hardware interrupt requests in an interrupt driven system environment.

Features

- 8 levels of interrupts.
- Can be cascaded in master-slave configuration to handle 64 levels of interrupts.
- Internal priority resolver, Fixed priority mode and rotating priority mode.
- Individually maskable interrupts.
- Modes and masks can be changed dynamically.

- Accepts IRQ, determines priority, checks whether incoming priority > current level being serviced, issues interrupt signal.
- In 8085 mode, provides 3-byte CALL instruction. In 8086 mode, provides 8-bit vector number.
- Polled and vectored mode.
- Starting address of ISR or vector number is programmable.
- No clock required.

Pin Diagram

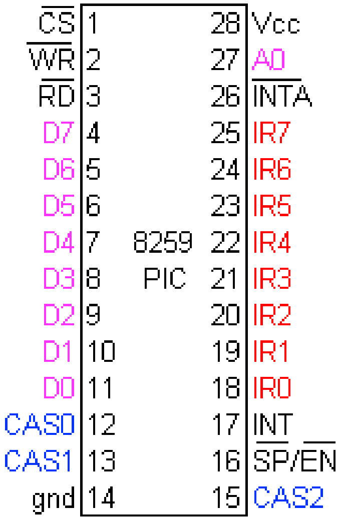


Fig. 2.7 8259 Pin Diagram

D0-D7	Bi-directional, tristate, buffered data lines. Connected to data bus directly or through buffers
RD-bar	Active low read control
WR-bar	Active low write control
A0	Address input line, used to select control register
CS-bar	Active low chip select
CAS0-2	Bi-directional, 3-bit cascade lines. In master mode, PIC places slave ID no. on these lines. In slave mode, the PIC reads slave ID no. from master on these lines. It may be regarded as slave-select.
SP-bar / EN-bar	Slave program / enable. In non-buffered mode, it is SP-bar input, used to distinguish master/slave PIC. In buffered mode, it is output line used to enable buffers

INT	Interrupt line, connected to INTR of microprocessor
INTA-bar	Interrupt ack, received active low from microprocessor
IR0-7	Asynchronous IRQ input lines, generated by peripherals.

Block Diagram

Interrupt Request Register (IRR): The interrupts at IRQ input lines are handled by Interrupt Request Register internally. IRR stores all the interrupt requests in it in order to serve them one by one on the priority basis.

In-Service Register (ISR): This register stores all the interrupt requests those are being served, i.e. ISR keeps a track of the requests being served.

Priority Resolver: This unit determines the priorities of the interrupt requests appearing simultaneously. The highest priority is selected and stored into the corresponding bit of ISR during INTA pulse. The IR0 has the highest priority while the IR7 has the lowest one, normally in fixed priority mode.The priorities however may be altered by programming the 8259A in rotating priority mode.

Interrupt Mask Register (IMR): This register stores the bits required to mask the interrupt puts. IMR operates on IRR at the direction of the Priority Resolver.

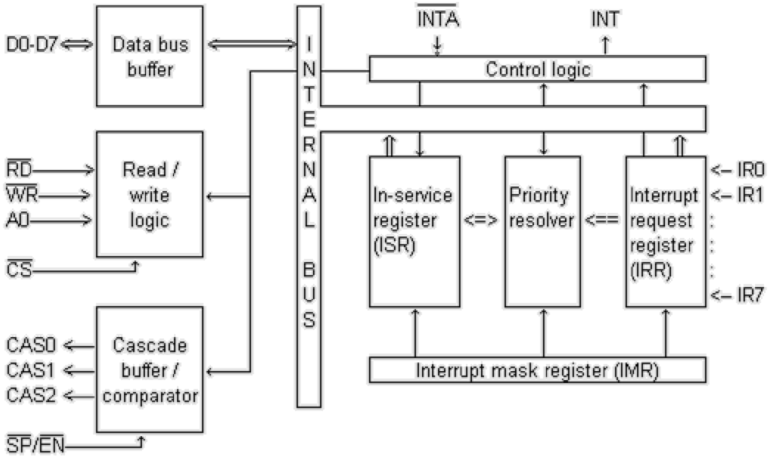


Fig. 2.8 8259 Block Diagram

Interrupt Control Logic: This block manages the interrupt and interrupt acknowledge signals to be sent to the CPU for serving one of the eight interrupt requests. This also accepts interrupt acknowledge (INTA) signal from CPU that causes the 8259A to release vector address on to the data bus.

Data Bus Buffer: This tristate bidirectional buffer interfaces internal 8259A bus to the microprocessor system data bus. Control words, status and vector information pass through buffer during read or write operations.

Read write Control Logic: This circuit accepts and decodes commands from the CPU. This also allows the status of the 8259A to be transferred on to the data bus.

Cascade Buffer/Comparator: This block stores and compares the ID's of all the 8259As used in the system. The three I/O pins CAS0-2 are outputs, when the 8259A is used as a master. The same pins act as inputs when the 8259A is in slave mode. The 8259A in master mode sends the ID of the interrupting slave device on these lines. The slave thus selected, will send its pre-programmed vector address on the data bus during the next INTA pulse.

Interrupt Sequence

The powerful features of the 8259A in a microcomputer system are its programmability and the interrupt routine addressing capability. The latter allows direct or indirect jumping to the specific interrupt routine requested without any polling of the interrupting devices. The normal sequence of events during an interrupt depends on the type of CPU being used. The events occur as follows in an 8085 system:

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

7. This completes the 3-byte CALL instruction released by the 8259A. In the AEOI mode the ISR bit is reset at the end of the third INTA pulse.
8. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt sequence.

When the 8259A PIC receives an interrupt, INT becomes active and an interrupt acknowledge cycle is started. If a higher priority interrupt occurs between the two INTA pulses, the INT line goes inactive immediately after the second INTA pulse. After an unspecified amount of time the INT line is activated again to signify the higher priority interrupt waiting for service. This inactive time is not specified and can vary between parts.

Programming the 8259A

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

Initialization Command Words (ICWs)

Before normal operation can begin, each 8259A in the system must be brought to a starting pointed by a sequence of 2 to 4 bytes timed by WR pulses.

Operation Command Words (OCWs)

These are the command words which command the 8259A to operate in various interrupt modes. These modes are:

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

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

Initialization Command Words (ICWS)

Whenever a command is issued with A0 = 0 and D4 = 1, this is interpreted as Initialization Command Word 1 (ICW1). ICW1 starts the initialization sequence during which the following automatically occur.

a. The edge sense circuit is reset, which means that following initialization, an interrupt request (IR) input must make a low-to-high transition to generate an interrupt.
b. The Interrupt Mask Register is cleared.
c. IR7 input is assigned priority 7.
d. The slave mode address is set to 7.

e. Special Mask Mode is cleared and Status Read is set to IRR.

f. If IC4 e 0, then all functions selected in ICW4 are set to zero.

ICW 1

This is the primary control word used to initialize the PIC. this is a 7-bit value that must be put in the primary PIC command register. This is the format:

Initialization Control Word (ICW) 1		
Bit	Value	**Description**
0	IC4	If set(1), the PIC expects to receive IC4 during initialization.
1	SNGL	If set(1), only one PIC in system. If cleared, PIC is cascaded with slave PICs, and ICW3 must be sent to controller.
2	ADI	If set (1), CALL address interval is 4, else 8. This is usually ignored by x86, and is default to 0
3	LTIM	If set (1), Operate in Level Triggered Mode. If Not set (0), Operate in Edge Triggered Mode
4	1	Initialization bit. Set 1 if PIC is to be initialized
5	0	MCS-80/85: Interrupt Vector Address. x86 Architecture: Must be 0
6	0	MCS-80/85: Interrupt Vector Address. x86 Architecture: Must be 0
7	0	MCS-80/85: Interrupt Vector Address. x86 Architecture: Must be 0

As you can see, there is a lot going on here. We have seen some of these before. This is not as hard as it seems, as most of these bits are not used on the x86 platform. To initialize the primary PIC, all we need to do is create the initial ICW and set the appropriate bits.

ICW 2

This control word is used to map the base address of the IVT of which the PIC is to use.

Initialization Control Word (ICW) 2		
Bit	**Value**	**Description**
0-2	A8/A9/A10	Address bits A8-A10 for IVT when in MCS-80/85 mode.
3-7	A11(T3)/A12(T4)/ A13(T5)/A14(T6)/ A15(T7)	Address bits A11-A15 for IVT when in MCS-80/85 mode. **In 80x86 mode, spe the interrupt vector address.** May be set to 0 in x86 mode.

During initialization, we need to send ICW 2 to the PICs to tell them where the base address of the IRQ's to use. If an ICW1 was sent to the PICs (With the initialization bit set), you must send ICW2 next. **Not doing so can result in undefined results.** Most likely the incorrect interrupt handler will be executed. Unlike ICW 1, which is placed into the PIC's data registers, ICW 2 is sent to the data Registers, as software ports 0x21 for the primary PIC, and port 0xA1 for the secondary PIC.

ICW 3

This is an important command word. It is used to let the PICs know what IRQ lines to use when communicating with each other.

ICW 3 Command Word for Primary PIC

Initialization Control Word (ICW) 3 - Primary PIC		
Bit	Value	Description
0-7	S0-S7	Specifies what Interrupt Request (IRQ) is connected to slave PIC

ICW 3 Command Word for Secondary PIC

Initialization Control Word (ICW) 3 - Secondary PIC		
Bit	Value	Description
0-2	ID0	IRQ number the master PIC uses to connect to (**In binary notation**)
3-7	0	Reserved, must be 0

We must send an ICW 3 whenever we enable cascading within ICW 1. This allows us to set which IRQ to use to communicate with each other. Remember that the 8259A Microcontroller relies on the IR0-IR7 pins to connect to other PIC devices. With this, it uses the CAS0-CAS2 pins to communicate with each other. We need to let each PIC know about each other and how they are connected. We do this by sending the ICW 3 to both PICs containing which IRQ line to use for both the master and associated PICs.

IRQ Lines for ICW 2 (Primary PIC)	
Binary	IRQ Line
000	IR0
001	IR1
010	IR2
011	IR3
100	IR4
101	IR5
110	IR6
111	IR7

IRQ Lines for ICW 2 (Secondary PIC)	
Binary	**IRQ Line**
000	IR0
001	IR1
010	IR2
011	IR3
100	IR4
101	IR5
110	IR6
111	IR7

ICW 4

This is the final initialization control word. This controls how everything is to operate.

Initialization Control Word (ICW) 4		
Bit	Value	Description
0	uPM	If set (1), it is in 80x86 mode. Cleared if MCS-80/86 mode
1	AEOI	If set, on the last interrupt acknowledge pulse, controller automatically performs End of Interrupt (EOI) operation
2	M/S	Only use if BUF is set. If set (1), selects buffer master. Cleared if buffer slave.
3	BUF	If set, controller operates in buffered mode
4	SFNM	Special Fully Nested Mode. Used in systems with cascaded controllers.
5-7	0	Reserved, must be 0

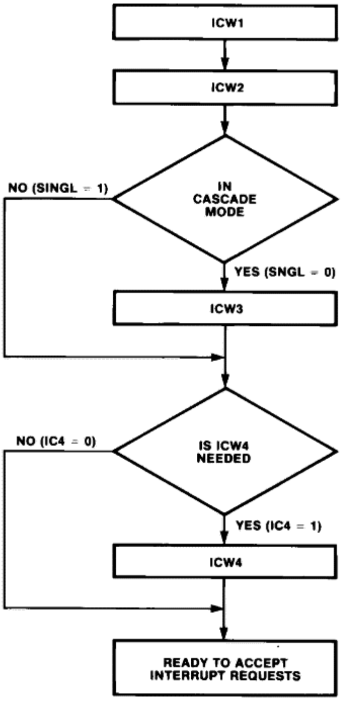


Fig. 2.9 8259 Flow chart of command Words

Operation Command Words (OCWs)

After the Initialization Command Words (ICWs) are programmed into the 8259A, the chip is ready to accept interrupt requests at its input lines. However, during the 8259A operation, a selection of algorithms can command the 8259A to operate in various modes through the Operation Command Words (OCWs).

OCW 1

OCW1 sets and clears the mask bits in the interrupt Mask Register (IMR). M7–M0 represent the eight mask bits. M = 1 indicates the channel is masked (inhibited), M = 0 indicates the channel is enabled.

OCW 2

R, SL, EOI – These three bits control the Rotate and End of Interrupt modes and combinations of the two. A chart of these combinations can be found on the Operation Command Word Format.

L2, L1, L0 – These bits determine the interrupt level acted upon when the SL bit is active.

Operation Command Word (OCW) 2		
Bit	Value	Description
0-2	L0/L1/L2	Interrupt level upon which the controller must react
3-4	0	Reserved, must be 0
5	EOI	End of Interrupt (EOI) request
6	SL	Selection
7	R	Rotation option

Bits 0-2 represents the interrupt level for the current interrupt. Bits 3-4 are reserved. Bits 5-7 are the interesting bits. Let's take a look at each combination for these bits.

OCW2 Commands			
R Bit	SL Bit	EOI Bit	Description
0	0	0	Rotate in Automatic EOI mode (CLEAR)
0	0	1	Non-specific EOI command
0	1	0	No operation
0	1	1	Specific EOI command
1	0	0	Rotate in Automatic EOI mode (SET)
1	0	1	Rotate on non-specific EOI
1	1	0	Set priority command
1	1	1	Rotate on specific EOI

OCW3

ESMM – Enable Special Mask Mode. When this bit is set to 1 it enables the SMM bit to set or reset the Special Mask Mode. When ESMM = 0 the SMM bit becomes a "don't care".

SMM – Special Mask Mode. If ESMM = 1 and SMM = 1 the 8259A will enter Special Mask Mode. If

ESMM = 1 and SMM = 0 the 8259A will revert to normal mask mode. When ESMM = 0, SMM has no effect.

D7	D6	D5	D4	D3	D2	D1	D0
D7	ESMM	SMM	0	1	MODE	RIR	RIS

ESMM	**SMM**	**Effect**
0	X	No effect
1	0	Reset special mask
1	1	Set special mask

2.5 Keyboard and Display Controller (8279)

8279 is a general-purpose Keyboard Display controller that simultaneously drives the display of a system and interfaces a Keyboard with the CPU. The Keyboard Display interface scans the Keyboard to identify if any key has been pressed and sends the code of the pressed key to the CPU. It also transmits the data received from the CPU, to the display device. Both of these functions are performed by the controller in repetitive fashion without involving the CPU. The Keyboard is interfaced either in the interrupt or the polled mode. In the interrupt mode, the processor is requested service only if any key is pressed, otherwise the CPU can proceed with its main task. In the polled mode, the CPU periodically reads an internal flag of 8279 to check for a key pressure.

Pin Diagram

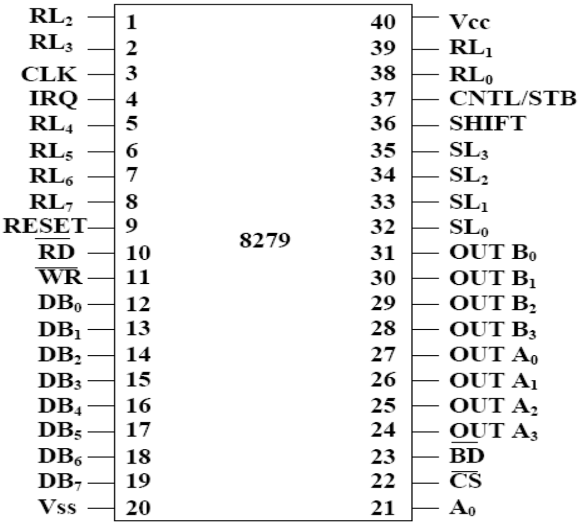


Fig. 2.10 8279 Pin Diagram

DB0 - DB7: These are bidirectional data bus lines. The data and command words to and from the CPU are transferred on these lines.

CLK: This is a clock input used to generate internal timings required by 8279.

RESET: This pin is used to reset 8279. A high on this line resets 8279. After resetting 8279, it's in sixteen 8-bit display, left entry encoded scan, 2-key lock out mode. The clock prescaler is set to 31.

CS chip select:A low on this line enables 8279 for normal read or write operations. Otherwise this pin should be high.

Ao:A high on the Ao line indicates the transfer of a command or status information. A low on this line indicates the transfer of data. This is used to select one of the internal registers of 8279.

RD, WR:(Input/Output) READ/WRITE input pins enable the data buffer to receive or send data over the data bus.

IRQ: This interrupt output line goes high when there is data in the FIFO sensor RAM. The interrupt line goes low with each FIFO RAM read-operation. However, if the FIFO RAM further contains any Key-code entry to be read by the CPU, this pin again goes high to generate an interrupt to the CPU.

Vss, Vcc: These are the ground and power supply lines for the circuit.

SL0-SL3 – Scan Lines: These lines are used to scan the keyboard matrix and display digits. These lines can be programmed as encoded or decoded, using the mode control register.

RL0-RL7 – Return Lines: These are the input lines which are connected to one terminal of keys, while the other terminal of the keys is connected to the decoded scan lines. These are normally high,but pulled low when a key is pressed.

SHIFT: The status of the Shift input line is stored along with each key code in FIFO in the scanned keyboard mode. Till it is pulled low with a key closure it is pulled up internally to keep it high.

CNTL/STB-CONTROL/STROBED I/P Mode: In the Keyboard mode, this line is used as a control input and stored in FIFO on a key closure. The line is a strobe line that enters the data into FIFO RAM, in the strobed input mode. It has an internal pull up. The line is pulled down with a Key closure.

BD – Blank Display: This output pin is used to blank the display during digit switching or by a blanking command.

OUTA0 – OUTA3 and OUTB0 – OUTB3: These are the output ports for two 16x4 (or one 16 x 8) internal display refresh registers. The data from these lines is synchronized with the scan lines to scan the display and keyboard. The two 4-bit ports may also be used as one 8-bit port.

Block Diagram

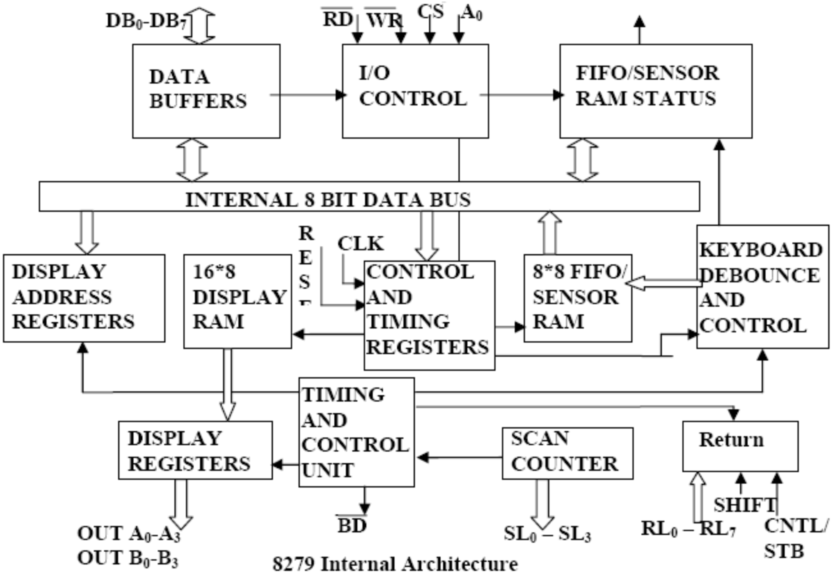


Fig. 2.11 8279 Block Diagram

Control and Timing Register and Timing Control

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

Scan Counter

The Scan Counter has two modes to scan the key matrix and refresh the display. In the Encoded mode, the counter provides a binary count that is to be externally decoded to provide the scan lines for keyboard and display (four externally decoded scan lines may drive up to 16 displays).In the decoded scan mode, the counter internally decodes the least significant 2 bits and provides a decoded 1 out of 4 scan on SL0-SL3 (four internally decoded scan lines may drive up to 4 Displays). The Keyboard and Display both are in the same mode at a time.

Return Buffers and Keyboard Debounce and Control

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

FIFO/Sensor RAM and Status Logic

In Keyboard or strobed input mode, this block acts as 8-byte first-in-first-out (FIFO) RAM. Each key code of the pressed key is entered in the order of the entry, and in the meantime, read by the CPU, till the RAM becomes empty. The status logic generates an interrupt request after each FIFO read operation till the FIFO is empty. In scanned sensor matrix mode, this unit acts as sensor RAM. Each row of the sensor RAM is loaded with the status of the corresponding row of sensors in the matrix. If a sensor changes its state, the IRQ line goes high to interrupt the CPU.

Display Address Registers and Display RAM

The Display address registers hold the addresses of the word currently being written or read by the CPU to or from the display RAM. The contents of the registers are automatically updated by 8279 to accept the next data entry by CPU. The 16-byte display RAM contains the 16-byte of data to be displayed on the sixteen 7-seg displays in the encoded scan mode to provide the scan lines for keyboard and display (four externally decoded scan lines may drive up to 16 displays). In the decoded scan mode, the counter internally decodes the least significant 2 bits and provides a decoded 1 out of 4 scans on SL0-SL3 (four internally decoded scan lines may drive up to 4 Displays). The Keyboard and Display both are in the same mode at a time.

Return Buffers and Keyboard Debounce and Control

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

FIFO/Sensor RAM and Status Logic

In Keyboard or strobed input mode, this block acts as 8-byte first-in-first-out (FIFO) RAM. Each key code of the pressed key is entered in the order of the entry, and in the meantime, read by the CPU, till the RAM becomes empty. The status logic generates an interrupt request after each FIFO read operation till the FIFO is empty. In scanned sensor matrix mode, this unit acts as sensor RAM. Each row of the sensor RAM is loaded with the status of the corresponding row

of sensors in the matrix. If a sensor changes its state, the IRQ line goes high to interrupt the CPU.

Display Address Registers and Display RAM

The Display address registers hold the addresses of the word currently being written or read by the CPU to or from the display RAM. The contents of the registers are automatically updated by 8279 to accept the next data entry by CPU. The 16-byte display RAM contains the 16-byte of data to be displayed on the sixteen 7-seg displays in the encoded scan mode.

Modes of Operation of 8279

The Modes of operation of 8279 are

- Input (Keyboard) modes
- Output (Display) modes

Input (Keyboard) modes

8279 provides three input modes, they are:

Scanned Keyboard Mode

This mode allows a key matrix to be interfaced using either encoded or decoded scans. In the encoded scan, an 8 x 8 keyboard or in decoded scan, a 4 x 8 Keyboard can be interfaced. The code of key pressed with SHIFT and CONTROL status is stored into the FIFO RAM.

Scanned Sensor Matrix

In this mode, a sensor array can be interfaced with 8279 using either encoder or decoder scans. With encoder scan 8 x 8 sensor matrix or with decoder scan 4 x 8 sensor matrix can be interfaced. The sensor codes are stored in the CPU addressable sensor RAM.

Strobed Input

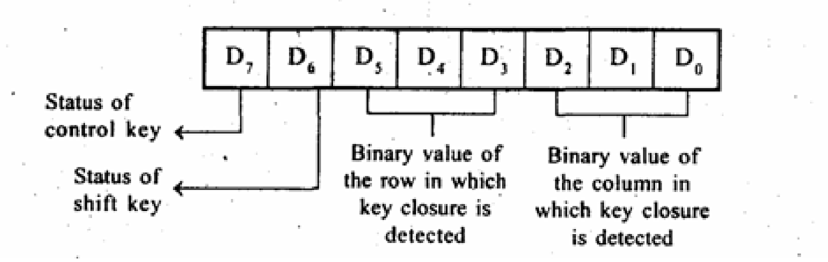
In this mode, if the control line goes low, the data on return lines, is stored in the FIFO byte by byte.

Major Section

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

Keyboard section

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


D_7
D_6
D_5
D_4
D_3
D_2
D_1
D_0
Status of control key
Status of shift key
Binary value of the row in which key closure is detected
Binary value of the column in which key closure is detected

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

Display section

- The display section has eight output lines divided into two groups A0-A3 and B0-B2.3.
- The output lines can be used either as a single group of eight lines or as two groups of four lines, in conjunction with the scan lines for a multiplexed display.

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

Scan section

- The scan section has a scan counter and four scan lines, SL0 to SL2.3.
- In decoded scan mode, the output of scan lines will be similar to a 2-to-4 decoder.
- In encoded scan mode, the output of scan lines will be binary count, and so an external decoder should be used to convert the binary count to decoded output.
- The scan lines are common for keyboard and display.
- The scan lines are used to form the rows of a matrix keyboard and also connected to digit drivers of a multiplexed display, to turn ON/OFF.

CPU interface section

- The CPU interface section takes care of data transfer between 8279 and the processor.
- This section has eight bidirectional data lines DB0 to DB7 for data transfer between 8279 and CPU.
- It requires two internal address A =0 for selecting data buffer and A = 1 for selecting control register of8279.
- The control signals WR (low), RD (low), CS (low) and A0 are used for read/write to 8279.
- It has an interrupt request line IRQ, for interrupt driven data transfer with processor.
- The 8279 require an internal clock frequency of 100 kHz. This can be obtained by dividing the input clock by an internal prescaler.
- The RESET signal sets the 8279 in 16-character display with two -key lockout keyboard modes.

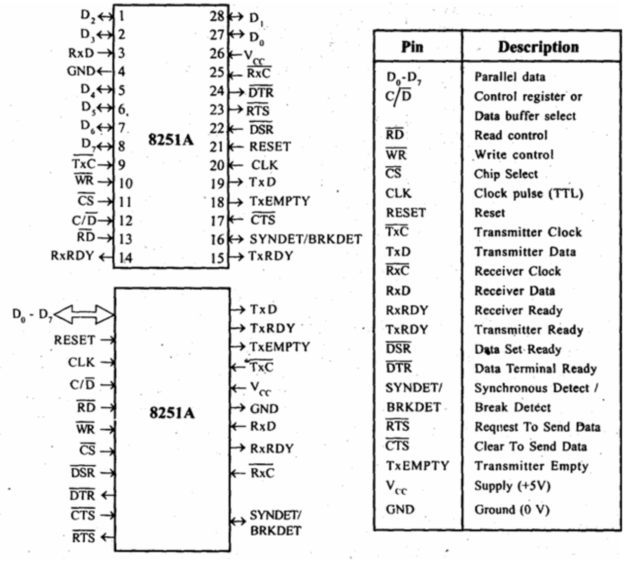
In a microprocessor system, when keyboard and 7-segment LED display is interfaced using ports or latches then the processor has to carry the following task.

- Keyboard scanning
- Key debouncing
- Key code generation
- Sending display code to LED
- Display refreshing

2.6 Interfacing Serial I/O (8251)

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

Pin Description



Pin	Description
D_0-D_7	Parallel data
$C/\overline{D}$	Control register or Data buffer select
$\overline{RD}$	Read control
$\overline{WR}$	Write control
$\overline{CS}$	Chip Select
CLK	Clock pulse (TTL)
RESET	Reset
$\overline{TxC}$	Transmitter Clock
TxD	Transmitter Data
$\overline{RxC}$	Receiver Clock
RxD	Receiver Data
RxRDY	Receiver Ready
TxRDY	Transmitter Ready
$\overline{DSR}$	Data Set Ready
$\overline{DTR}$	Data Terminal Ready
SYNDET/ BRKDET	Synchronous Detect / Break Detect
$\overline{RTS}$	Request To Send Data
$\overline{CTS}$	Clear To Send Data
TxEMPTY	Transmitter Empty
V_{CC}	Supply (+5V)
GND	Ground (0 V)

Fig. 2.12 8251 Pin Diagram

D0 to D7 (l/O terminal): This is bidirectional data bus which receive control words and transmits data from the CPU and sends status words and received data to CPU.

RESET (Input terminal): A "High" on this input forces the 8251 into "reset status." The device waits for the writing of "mode instruction." The min. reset width is six clock inputs during the operating status of CLK.

CLK (Input terminal): CLK signal is used to generate internal device timing. CLK signal is independent of RXC or TXC. However, the frequency of CLK must be greater than 30 times the RXC and TXC at Synchronous mode and

Asynchronous "x1" mode, and must be greater than 5 times at Asynchronous "x16" and "x64" mode.

WR (Input terminal): This is the "active low" input terminal which receives a signal for writing transmit data and control words from the CPU into the 8251.

RD (Input terminal): This is the "active low" input terminal which receives a signal for reading receive data and status words from the 8251.

C/D (Input terminal): This is an input terminal which receives a signal for selecting data or command words and status words when the 8251 is accessed by the CPU. If C/D = low, data will be accessed. If C/D = high, command word or status word will be accessed.

CS (Input terminal): This is the "active low" input terminal which selects the 8251 at low level when the CPU accesses. Note: The device won't be in "standby status"; only setting CS = High.

TXD (output terminal): This is an output terminal for transmitting data from which serial-converted data is sent out. The device is in "mark status" (high level) after resetting or during a status when transmit is disabled. It is also possible to set the device in "break status" (low level) by a command.

TXRDY (output terminal): This is an output terminal which indicates that the 8251is ready to accept a transmitted data character. But the terminal is always at low level if CTS = high or the device was set in "TX disable status" by a command. Note: TXRDY status word indicates that transmit data character is receivable, regardless of CTS or command. If the CPU writes a data character, TXRDY will be reset by the leading edge or WR signal.

TXEMPTY (Output terminal): This is an output terminal which indicates that the 8251 has transmitted all the characters and had no data character. In "synchronous mode," the terminal is at high level, if transmit data characters are no longer remaining and sync characters are automatically transmitted. If the CPU writes a data character, TXEMPTY will be reset by the leading edge of WR signal. Note: As the transmitter is disabled by setting CTS "High" or command, data written before disable will be sent out. Then TXD and TXEMPTY will be "High". Even if a data is written after disable, that data is not sent out and TXE will be "High".After the transmitter is enabled, it sent out. (Refer to Timing Chart of Transmitter Control and Flag Timing)

TXC (Input terminal): This is a clock input signal which determines the transfer speed of transmitted data. In "synchronous mode," the baud rate will be the same as the frequency of TXC. In "asynchronous mode", it is possible to select the baud rate factor by mode instruction. It can be 1, 1/16 or 1/64 the TXC. The falling edge of TXC sifts the serial data out of the 8251.

RXD (input terminal): This is a terminal which receives serial data.

RXRDY (Output terminal): This is a terminal which indicates that the 8251 contains a character that is ready to READ. If the CPU reads a data character, RXRDY will be reset by the leading edge of RD signal. Unless the CPU reads a data character before the next one is received completely, the preceding data will be lost. In such a case, an overrun error flag status word will be set.

RXC (Input terminal): This is a clock input signal which determines the transfer speed of received data. In "synchronous mode," the baud rate is the same as the frequency of RXC. In "asynchronous mode," it is possible to select the baud rate factor by mode instruction. It can be 1, 1/16, 1/64 the RXC.

SYNDET/BD (Input or output terminal): This is a terminal whose function changes according to mode. In "internal synchronous mode." this terminal is at high level, if sync characters are received and synchronized. If a status word is read, the terminal will be reset. In "external synchronous mode, "this is an input terminal. A "High" on this input forces the 8251 to start receiving data characters. In "asynchronous mode," this is an output terminal which generates "high level" output upon the detection of a "break" character if receiver data contains a "low-level" space between the stop bits of two continuous characters. The terminal will be reset, if RXD is at high level. After Reset is active, the terminal will be output at low level.

DSR (Input terminal): This is an input port for MODEM interface. The input status of the terminal can be recognized by the CPU reading status words.

DTR (Output terminal): This is an output port for MODEM interface. It is possible to set the status of DTR by a command.

CTS (Input terminal): This is an input terminal for MODEM interface which is used for controlling a transmit circuit. The terminal controls data transmission if the device is set in "TX Enable" status by a command. Data is transmittable if the terminal is at low level.

RTS (Output terminal): This is an output port for MODEM interface. It is possible to set the status RTS by a command.

Block Diagram

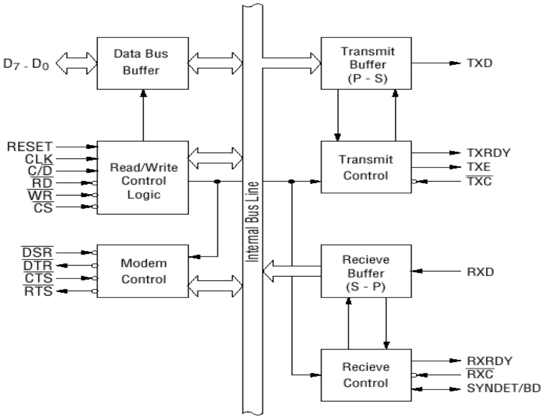


Fig. 2.13 8251 Block Diagram

The functional block diagram of 825 1A consists five sections. They are:

- Read/Write control logic
- Transmitter
- Receiver
- Data bus buffer
- Modem control.

Read/Write control logic

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

Transmitter section

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

Receiver Section

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

MODEM Control

- The MODEM control unit allows to interface a MODEM to 8251A and to establish data communication through MODEM over telephone lines.
- This unit takes care of handshake signals for MODEM interface.
- The 825 1A can be either memory mapped or I/O mapped in the system.
- 8251A in I/O mapped in the system is shown in the figure.
- Using a 3-to-8 decoder generates the chip select signals for I/O mapped devices.
- The address lines A4, A5 and A6 are decoded to generate eight chip-select signals (IOCS-0 to IOCS-7) and in this, the chip select signal IOCS-2 is used to select 8251A.
- The address line A7 and the control signal IO / M(low) are used as enable for decoder.
- The address line A0 of 8085 is connected to C/D(low) of 8251A to provide the internal addresses.
- The data lines D0 - D7 are connected to D0 - D7 of the processor to achieve parallel data transfer.
- The RESET and clock signals are supplied by the processor. Here the processor clock is directly connected to 8251A. This clock controls the parallel data transfer between the processor and 8251A.
- The output clock signal of 8085 is divided by suitable clock dividers like programmable timer 8254 and then used as clock for serial transmission and reception.
- The TTL logic levels of the serial data lines and the control signals necessary for serial transmission and reception are converted to RS232 logic levels using MAX232 and then terminated on a standard 9-pin D-type connector.
- In 8251A the transmission and reception baud rates can be different or same.

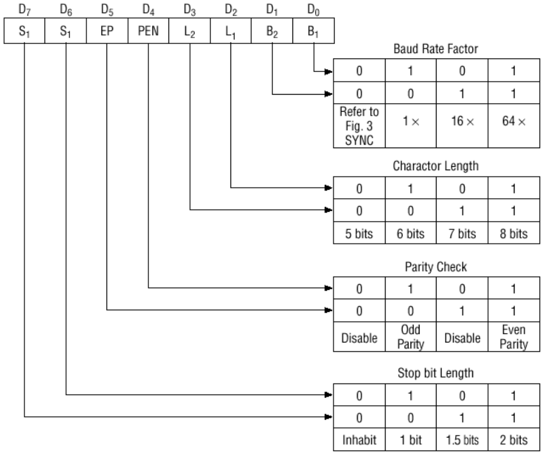
Control Words

There are two types of control word.

1. Mode instruction (setting of function)
2. Command (setting of operation)

Mode Instruction

Mode instruction is used for setting the function of the 8251. Mode instruction will be in "wait for write" at either internal reset or external reset. That is, the writing of a control word after resetting will be recognized as a "mode instruction."



Baud Rate Factor

0	1	0	1
0	0	1	1
Refer to Fig. 3 SYNC	1 ×	16 ×	64 ×

Charactor Length

0	1	0	1
0	0	1	1
5 bits	6 bits	7 bits	8 bits

Parity Check

0	1	0	1
0	0	1	1
Disable	Odd Parity	Disable	Even Parity

Stop bit Length

0	1	0	1
0	0	1	1
Inhabit	1 bit	1.5 bits	2 bits

Fig. 2.14 Bit Configuration of Mode Instruction(Asynchronous)

Items set by mode instruction are as follows

- Synchronous/asynchronous mode
- Stop bit length (asynchronous mode)
- Character length
- Parity bit
- Baud rate factor (asynchronous mode)
- Internal/external synchronization (synchronous mode)
- Number of synchronous characters (Synchronous mode)

The bit configuration of mode instruction is shown in Figures.

In the case of synchronous mode, it is necessary to write one-or two-byte sync characters. If sync characters were written, a function will be set because the writing of sync characters constitutes part of mode instruction.

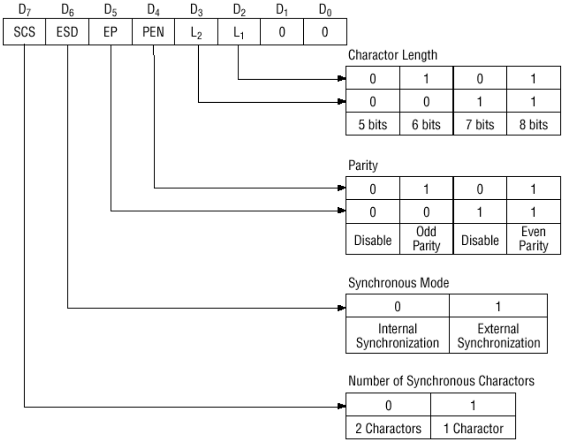

D7 D6 D5 D4 D3 D2 D1 D0
SCS ESD EP PEN L2 L1 0 0
Charactor Length
0 1 0 1
0 0 1 1
5 bits 6 bits 7 bits 8 bits
Parity
0 1 0 1
0 0 1 1
Disable Odd Parity Disable Even Parity
Synchronous Mode
0 1
Internal Synchronization External Synchronization
Number of Synchronous Charactors
0 1
2 Charactors 1 Charactor

Fig. 2.15 Bit Configuration of Mode Instruction(synchronous)

Command

Command is used for setting the operation of the 8251. It is possible to write a command whenever necessary after writing a mode instruction and sync characters.

Items to be set by command are as follows

- Transmit Enable/Disable
- Receive Enable/Disable
- DTR, RTS Output of data.
- Resetting of error flag.
- Sending to break characters
- Internal resetting
- Hunt mode (synchronous mode)

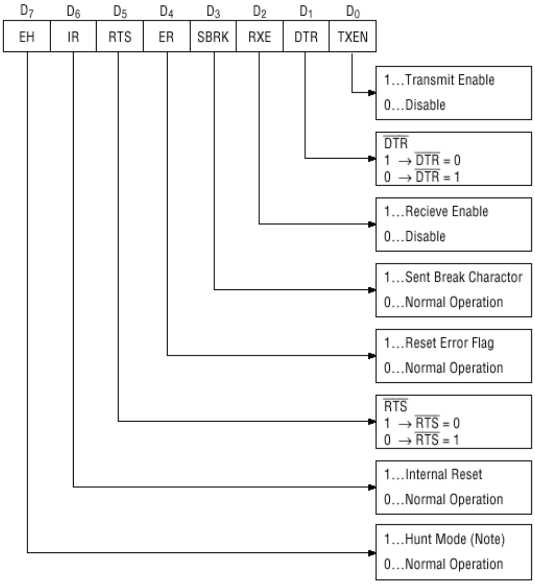


Fig. 2.16 Bit Configuration of command

Status Word

It is possible to see the internal status of the 8251 by reading a status word. The bit configuration of status word is shown in Fig. 2.5.

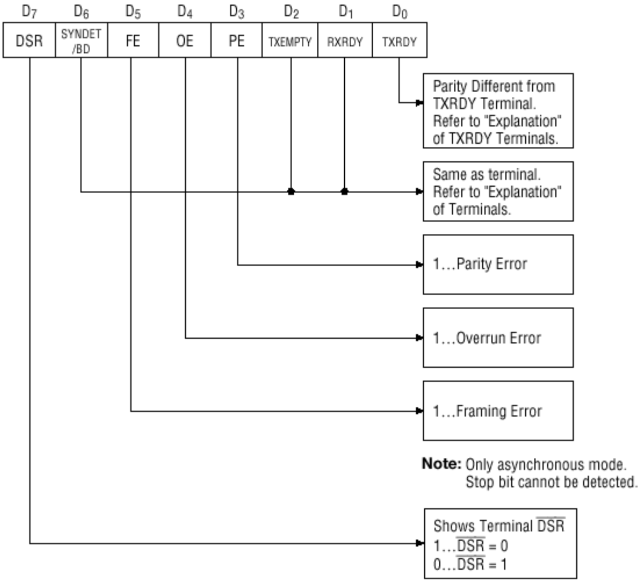

D7 D6 D5 D4 D3 D2 D1 D0
DSR SYNDET /BD FE OE PE TXEMPTY RXRDY TXRDY
Parity Different from TXRDY Terminal. Refer to "Explanation" of TXRDY Terminals.
Same as terminal. Refer to "Explanation" of Terminals.
1…Parity Error
1…Overrun Error
1…Framing Error
Note: Only asynchronous mode. Stop bit cannot be detected.
Shows Terminal DSR
1…DSR = 0
0…DSR = 1

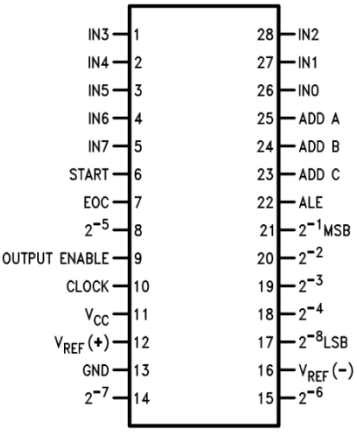
Fig. 2.17 Bit Configuration of status word

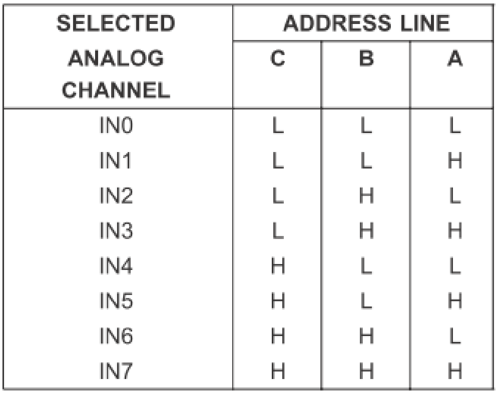
2.7 ADC/DAC Interfacing

ADC0808/ADC0809

The ADC0808, ADC0809 data acquisition component is a monolithic CMOS device with an 8-bit analog-to-digital converter, 8-channel multiplexer and microprocessor compatible control logic. The 8-bit A/D converter uses successive approximation as the conversion technique. The converter features a high impedance chopper stabilized comparator, a 256R voltage divider with analog switch tree and a successive approximation register. The 8-channel multiplexer can directly access any of 8-single-ended analog signals. The device eliminates the need for external zero and full-scale adjustments. Easy interfacing to microprocessors is provided by the latched and decoded multiplexer address inputs and latched TTL tri-state outputs. The design of the ADC0808, ADC0809 has been optimized by incorporating the most desirable aspects of several A/D conversion techniques. The ADC0808, ADC0809 offers high speed, high accuracy, minimal temperature dependence, excellent long-term accuracy and repeatability, and consumes minimal power. These features make this device ideally suited to applications from process and machine control to consumer and automotive applications.

Pin Diagram





SELECTED ANALOG CHANNEL	ADDRESS LINE		
	C	B	A
IN0	L	L	L
IN1	L	L	H
IN2	L	H	L
IN3	L	H	H
IN4	H	L	L
IN5	H	L	H
IN6	H	H	L
IN7	H	H	H

Functional Description

Multiplexer: The device contains an 8-channel single-ended analog signal multiplexer. A particular input channel is selected by using the address decoder. Table 1 shows the input states for the address lines to select any channel. The address is latched into the decoder on the low-to-high transition of the address latch enable signal.

The Converter: The heart of this single chip data acquisition system is its 8-bit analog-to-digital converter. The converter is designed to give fast, accurate, and repeatable conversions over a wide range of temperatures. The converter is partitioned into 3 major sections: the 256R ladder network, the successive approximation register, and the comparator. The converter's digital outputs are positive true.

The 256R ladder network approach was chosen over the conventional R/2R ladder because of its inherent monotonicity, which guarantees no missing digital codes. Additionally, the 256R network does not cause load variations on the reference voltage. The bottom resistor and the top resistor of the ladder network are not the same value as the remainder of the network. The difference in these resistors causes the output characteristic to be symmetrical with the zero and full-scale points of the transfer curve. The first output transition occurs when the analog signal has reached + 1 / 2 LSB and succeeding output transitions occur every 1 LSB later up to full-scale.

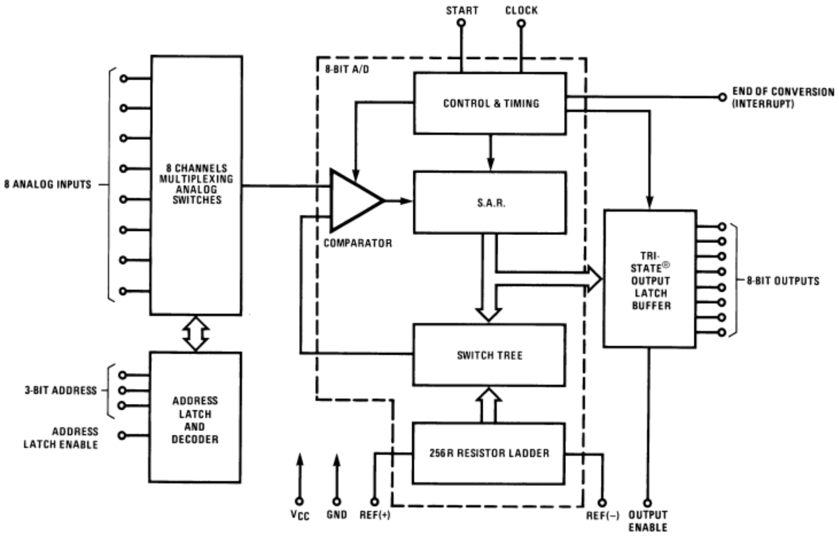


Fig. 2.18 Functional Block Diagram

The successive approximation register (SAR) performs 8 iterations to approximate the input voltage. For any SAR type converter, n-iterations are required for an n-bit converter. The A/D converter's successive approximation register (SAR) is reset on the positive edge of the start conversion (SC) pulse. The conversion is begun on the falling edge of the start conversion pulse. A conversion in process will be interrupted by receipt of a new start conversion pulse. Continuous conversion may be accomplished by tying the end-of-conversion (EOC) output to the SC input. If used in this mode, an external start conversion pulse should be applied after power up. End-of-conversion will go low between 0 and 8 clock pulses after the rising edge of start conversion. The most important section of the A/D converter is the comparator. It is this section which is responsible for the ultimate accuracy of the entire converter. It is also the comparator drift which has the greatest influence on the repeatability of the device. A chopper-stabilized comparator provides the most effective method of satisfying all the converter requirements.

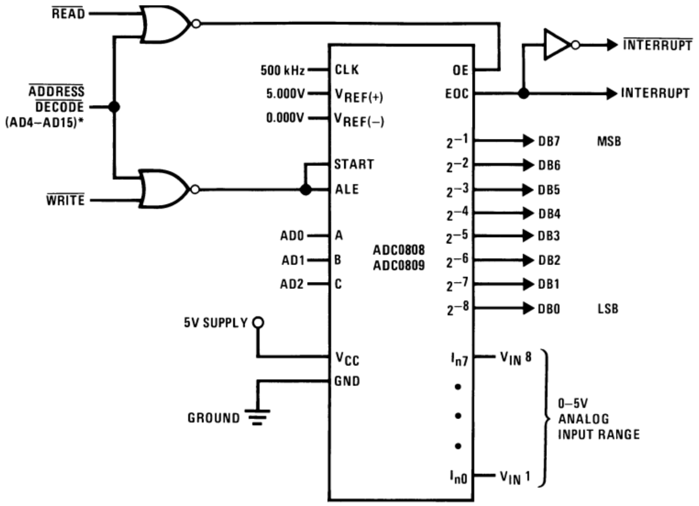
ADC Interfacing with 8085

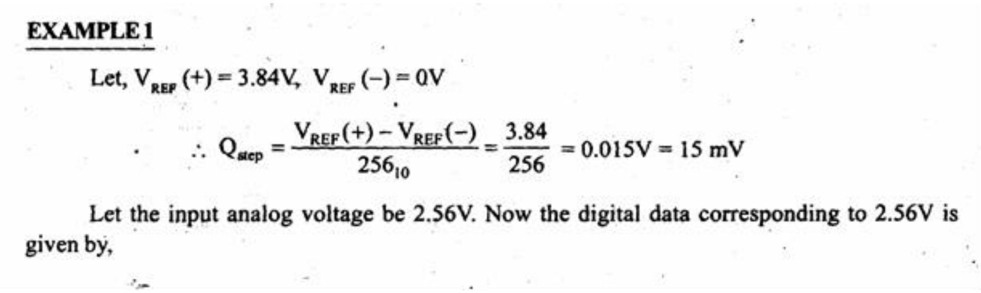
In ADC conversion process the input analog value is quantized and each quantized analog value will have a unique binary equivalent. The quantization step in ADC0809/ADC0808 is given by,

$$Q_{step} = \frac{V_{REF}}{2^8} = \frac{V_{REF}(+) - V_{REF}(-)}{256_{10}}$$

The digital data corresponding to an analog input (V_{in}) is given by,

$$\text{Digital data} = \left(\frac{V_{in}}{Q_{step}} - 1\right)_{10}$$



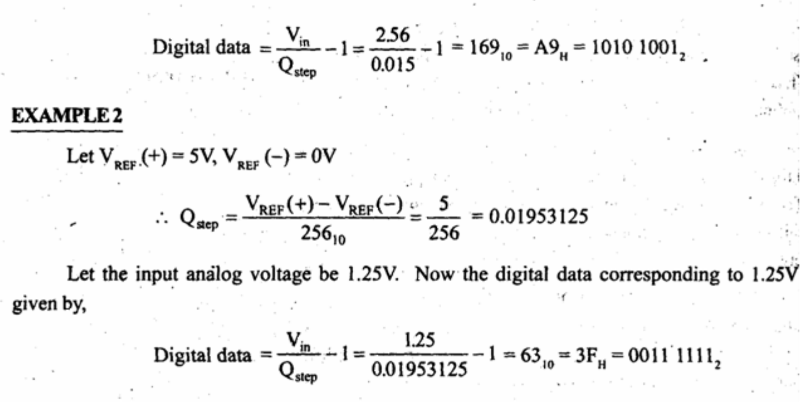


EXAMPLE 1

Let, $V_{REF}(+) = 3.84V$, $V_{REF}(-) = 0V$

$$\therefore\ Q_{step} = \frac{V_{REF}(+) - V_{REF}(-)}{256_{10}} = \frac{3.84}{256} = 0.015V = 15\ mV$$

Let the input analog voltage be 2.56V. Now the digital data corresponding to 2.56V is given by,



$$\text{Digital data} = \frac{V_{in}}{Q_{step}} - 1 = \frac{2.56}{0.015} - 1 = 169_{10} = A9_{H} = 1010\ 1001_{2}$$

EXAMPLE 2

Let $V_{REF}(+) = 5V$, $V_{REF}(-) = 0V$

$$\therefore\ Q_{step} = \frac{V_{REF}(+) - V_{REF}(-)}{256_{10}} = \frac{5}{256} = 0.01953125$$

Let the input analog voltage be 1.25V. Now the digital data corresponding to 1.25V given by,

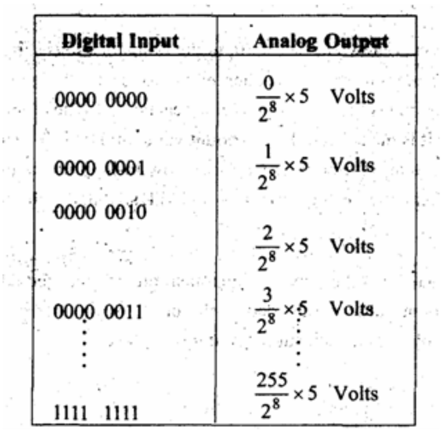
$$\text{Digital data} = \frac{V_{in}}{Q_{step}} - 1 = \frac{1.25}{0.01953125} - 1 = 63_{10} = 3F_{H} = 0011\ 1111_{2}$$

Example Program

```
MVI A, 10H; CHANNEL 0
OUT E0
MVI A, 18H; CONTROL WORD
OUT E0
MVI A, 01    ; START PULSE
OUT D0
XRA A
XRA A
XRA A
XRA A
MVI A, 00    ;STOP PULSE
OUT D0
LOOP:        IN D8             ;EOC
ANI 01
CPI 01
JNZ LOOP
INC0
STA 4200
HLT
```

DAC0800/DAC0802

The DAC will accept a digital (binary) input and convert to analog voltage or current. Every DAC will have "n" input lines and an analog output. The DAC require a reference analog voltage (Vref) or current (Iref) source. The smallest possible analog value that can be represented by the n-bit binary code is called resolution. The resolution of DAC with n-bit binary input is 1/2nof reference analog value. Every analog output will be a multiple of the resolution. For example, consider an 8-bit DAC with reference analog voltage of 5 volts. The analog values for all possible digital input are as shown.

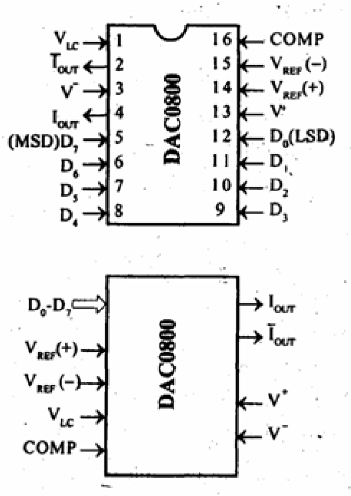


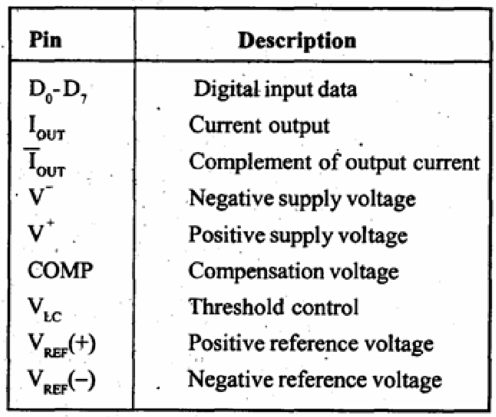
Digital Input	Analog Output
0000 0000	$\frac{0}{2^8} \times 5$ Volts
0000 0001	$\frac{1}{2^8} \times 5$ Volts
0000 0010	$\frac{2}{2^8} \times 5$ Volts
0000 0011	$\frac{3}{2^8} \times 5$ Volts
⋮	⋮
1111 1111	$\frac{255}{2^8} \times 5$ Volts

Pin diagram & block diagram of DAC0800

The DAC0800series are monolithic 8-bithigh-speed current-outputdigital-to-analog converters (DAC) featuring typical settling times of 100ns. When used As a multiplying DAC, monotonic performance over a 40 to 1 reference current range is possible. The DAC0800 series also features high compliance complementary current outputs to allow differential outputvoltagesof20Vp-pwithsimpleresistorloads.

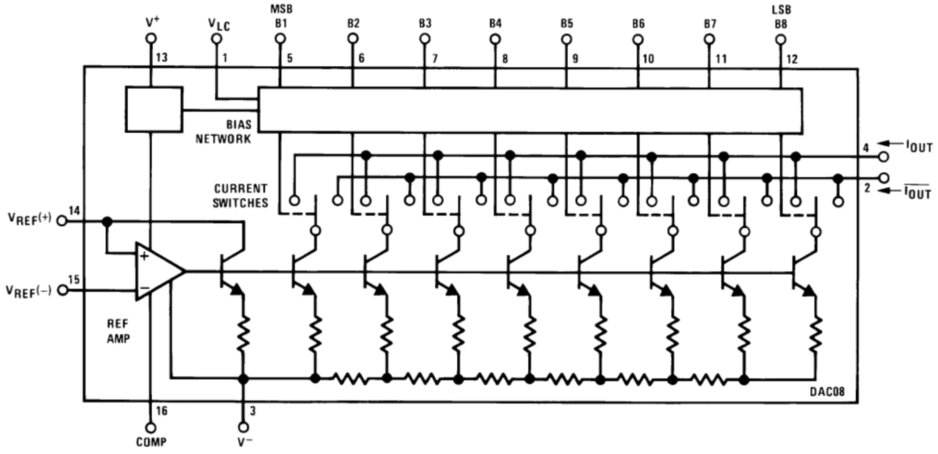
The reference-to-full-scale current matching of better than±1LS Be liminates the need for full-scale trim sin most applications, while the non-linearities of better than ±0.1% over temperature minimizes system error accumulations. The noise immune inputs will accept a variety of logic levels. The performance and characteristics of the device are essentially unchanged over the ±4.5V to ±18V power supply range and power consumption at only 33m W with ±5V supplies is independent of logic input levels.





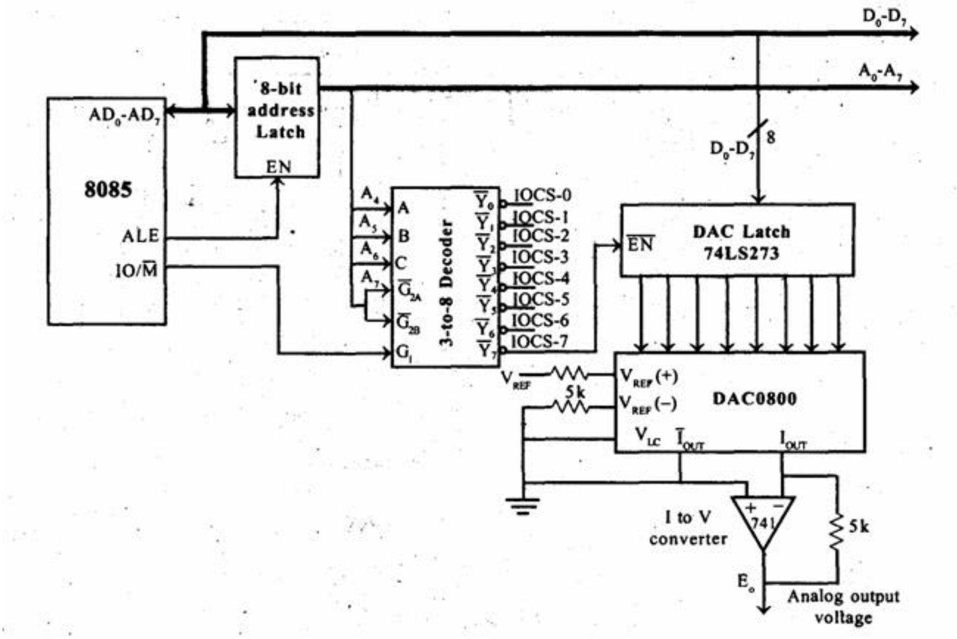
Pin	Description
D_0-D_7	Digital input data
I_{OUT}	Current output
$\overline{I}_{OUT}$	Complement of output current
V^-	Negative supply voltage
V^+	Positive supply voltage
COMP	Compensation voltage
V_{LC}	Threshold control
$V_{REF}(+)$	Positive reference voltage
$V_{REF}(-)$	Negative reference voltage

MSD - Most Significant Digit
LSD - Least Significant Digit

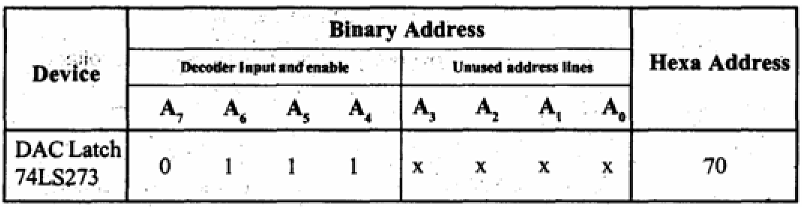


Interfacing with 8085

The DAC0800 can be interfaced to 8085 system bus by using an 8-bit latch and the latch can be enabled by using one of the chip-select signals generated for I/O devices. A simple schematic for interfacing DAC0800 with 8085 is shown in figure.



In this schematic the DAC0800 is interfaced using an 8-bit latch 74LS273 to the system bus. The 3-to-8 decoder 74LS 138 is used to generate chip select signals for I/O devices. The address lines A4, A5 and A6 are used as input to decoder. The address line A7 and the control signal IO/M (low) are used as enable for decoder. The decoder will generate eight chip-select signals and in this the signal IOCS-7 is used as enable for latch of DAC. The I/O address of the DAC is shown in table.



Device	Binary Address								Hexa Address
	Decoder Input and enable				Unused address lines				
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
DAC Latch 74LS273	0	1	1	1	x	x	x	x	70

In order to convert a digital data to analog value, the processor has to load the data to latch. The latch will hold the previous data until next data is loaded. The DAC will take definite time to convert the data. The software should take care of loading successive data only after the conversion time. The DAC 0800 produces a current output, which is converted to voltage output using I to V converter.

Example Programs

ALP to generate Square Wave

```
START   :   MVI A, 00
            OUT C0H
            CALL DELAY
            MVI A, FF
            OUT COH
            CALL DELAY
            JMP START
DELAY   :   MVI B, FF
L1      :   DCR B
            JNZ L1
            RET
```

ALP to generate Saw tooth Wave

```
START   :   MVI A, 00
L1      :   OUT C0
INR A       JNZ L1
            JMP START
```

ALP to generate triangular wave

```
START   :   MVI A, 00H
L1      :   OUT C0
            INR A
            JNZ L1
            MVI A, 0FFH
L2      :   OUT C0
            DCR A
            JNZ L2
            JMP START
```

2.8 Stepper Motor Interfacing

A stepper motor is a brushless, synchronous electric motor that converts digital pulses into mechanical shaft rotation. Every revolution of the stepper motor is divided into a discrete number of steps, and the motor must be sent a separate pulse for each step.

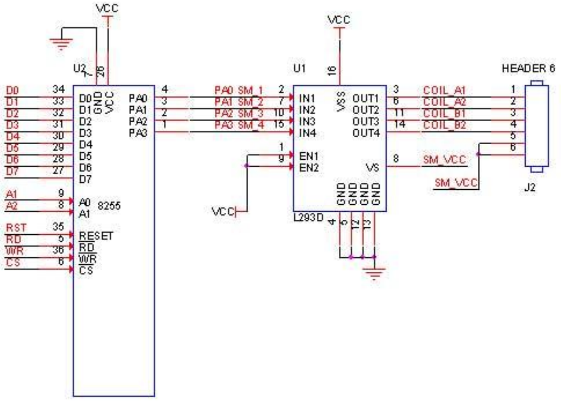


Fig. 2.19 Stepper motor control circuit

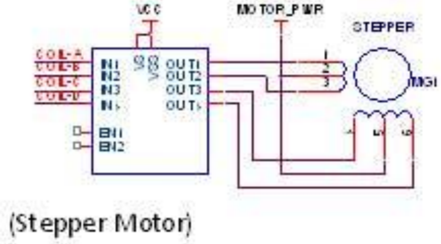
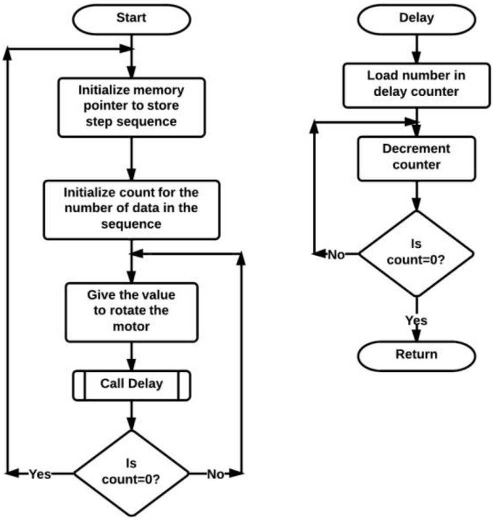
	Stepper Motor (5V)	8255Lines	Stepper Motor PWR Select
STEPPER MOTOR	COIL-A	PA.0	 VCC MOTOR_PWR STEPPER COIL-A COIL-B COIL-C COIL-D IN1 IN2 IN3 IN4 VSS VS OUT1 OUT2 OUT3 OUT4 EN1 EN2 1 2 3 MG1 (Stepper Motor)
	COIL-B	PA.1	
	COIL-C	PA.2	
	COIL-D	PA.3	

Fig. shows the typical 2 phase motor rated 12V/0.67 A/ph interfaced with the 8085-microprocessor system using 8255. Motor shown in the circuit has two phases, with center-tap winding. The center taps of these windings are connected to the 12V supply. Due to this, motor can be excited by grounding four terminals of the two windings. Motor can be rotated in steps by giving proper excitation sequence to these windings. The lower nibble of port A of the 8255 is used to generate excitation signals in the proper sequence. These excitation signals are buffered using driver transistors. The transistors are selected such that they can source rated current for the windings. Motor is rotated by 1.80 per excitation.


Start
Initialize memory pointer to store step sequence
Initialize count for the number of data in the sequence
Give the value to rotate the motor
Call Delay
Is count=0?
Yes
No
Delay
Load number in delay counter
Decrement counter
Is count=0?
No
Yes
Return

Software for Stepper Motor Control

As port A is used as an output port, control word for 8255 is 80H.

6000H Excite code DB 03_H, 06_H, 09_H, OC_H: This is the code sequence for clockwise rotation

Subroutine to rotate a stepper motor clockwise by 360° - Set the counts:

```
MVI C, 32H      : Set repetition count to 50₁₀
START           : MVI B, 04H
                : Counts excitation sequence
LXI H, 6000H    : Initialize pointer
BACK1           : MOV A, M
                : Get the Excite code
OUT PORTA       : Send Excite code
CALL DELAY      : Wait
INX H           : Increment pointer
DCR B           : Repeat 4 times
                JNZ BACK1
DELAY           : LXI D, Count
BACK            : DCX D
MOV A, D
ORA E
JNZ BACK
RET
```

2.9 Traffic Light Controller

The traffic light arrangement is as shown in Fig. The traffic should be controlled in the following manner. 1) Allow traffic from W to E and E to W transition for 20 seconds. 2) Give transition period of 5 seconds (Yellow bulbs ON) 3) Allow traffic from N to 5 and 5 to N for 20 seconds 4) Give transition period of 5 seconds (Yellow bulbs ON) 5) Repeat the process.

Hardware for Traffic Light Control

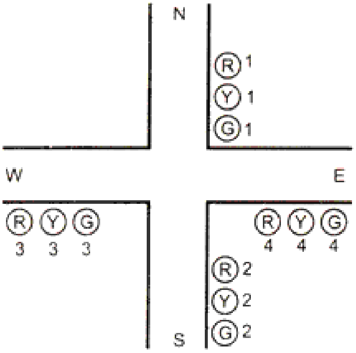
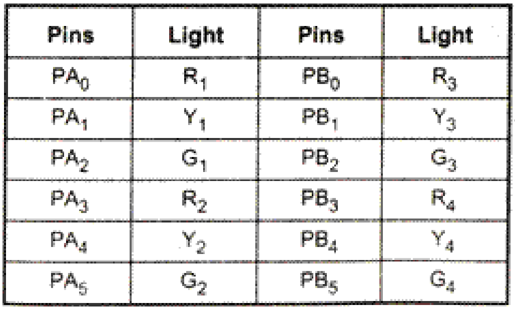


Fig. shows the interfacing diagram to control 12 electric bulbs. Port A is used to control lights on N-S road and Port B is used to control lights on W-E road. Actual pin connections are listed in Table 1 below.

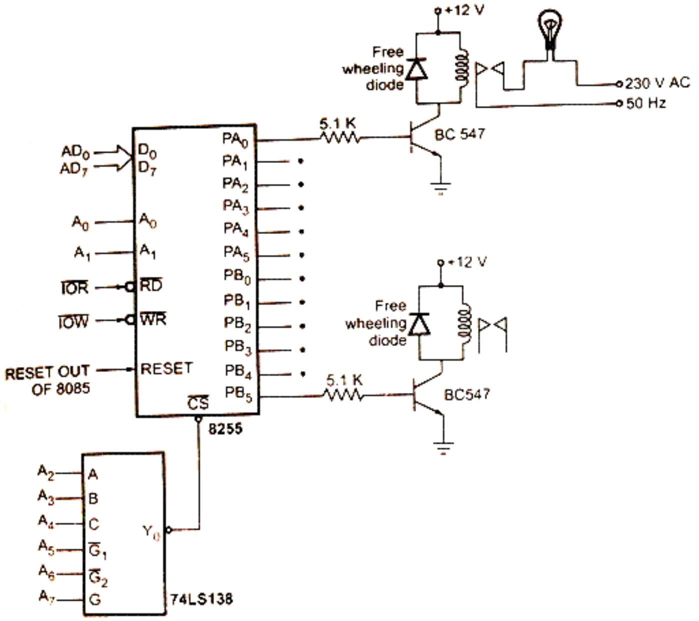


Pins	Light	Pins	Light
PA_0	R_1	PB_0	R_3
PA_1	Y_1	PB_1	Y_3
PA_2	G_1	PB_2	G_3
PA_3	R_2	PB_3	R_4
PA_4	Y_2	PB_4	Y_4
PA_5	G_2	PB_5	G_4

Table 1

The electric bulbs are controlled by relays. The 8255 pins are used to control relay on-off action with the help of relay driver circuits. The driver circuit includes 12 transistors to drive 12 relays. Fig. also shows the interfacing of 8255 to the system.

Interfacing Diagram



I/O MAP:

Ports / Control Register	Address lines A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Address
Port A	1	0	0	0	0	0	0	0	80H
Port B	1	0	0	0	0	0	0	1	81H
Port C	1	0	0	0	0	0	1	0	82H
Control Register	1	0	0	0	0	0	1	1	83H

Table 2

Software for Traffic Light Control

Control word : For initialization of 8255.

BSR/IO	$MODE_A$		P_A	PC_H	MODE B	P_B	PC_L
1	0	0	0	X	0	0	X

= 80H

Fig. Control word

Table shows the data bytes to be sent for specific combinations.

To glow	PB_7	PB_6	PB_5	PB_4	PB_3	PB_2	PB_1	PB_0	PA_7	PA_6	PA_5	PA_4	PA_3	PA_2	PA_1	PA_0	Port B Output	Port A Output
R_1,R_2,G_3 and G_4	X	X	1	0	0	1	0	0	X	X	0	0	1	0	0	1	24H	09H
Y_1,Y_2,Y_3 and Y_4	X	X	0	1	0	0	1	0	X	X	0	1	0	0	1	0	12H	12H
R_3,R_4,G_1 and G_2	X	X	0	0	1	0	0	1	X	X	1	0	0	1	0	0	09H	24H

Source program

```
MVI A, 80H          : Initialize 8255, port A and port B
OUT 83H (CR)        : in output mode
START               : MVI A, 09H
OUT 80H (PA)        : Send data on PA to glow R1 and R2
MVI A, 24H
OUT 81H (PB)        : Send data on PB to glow G3 and G4
MVI C, 28H          : Load multiplier count (4010) for delay
CALL DELAY          : Call delay subroutine
MVI A, 12H
OUT (81H) PA        : Send data on Port A to glow Y1 and Y2
OUT (81H) PB        : Send data on port B to glow Y3 and Y4
MVI C, 0AH          : Load multiplier count (1010) for delay
CALL: DELAY         : Call delay subroutine
MVI A, 24H
OUT (80H) PA        : Send data on port A to glow G1 and G2
MVI A, 09H
OUT (81H) PB        : Send data on port B to glow R3 and R4
MVI C, 28H          : Load multiplier count (4010) for delay
CALL DELAY          : Call delay subroutine
MVI A, 12H
OUT PA              : Send data on port A to glow Y1 and Y2
OUT PB              : Send data on port B to glow Y3 and Y4
MVI C, 0AH          : Load multiplier count (1010) for delay
CALL DELAY          : Call delay subroutine
JMP START
```

Delay Subroutine

```
DELAY     : LXI D, Count : Load count to give 0.5 sec delay
BACK      : DCX D : Decrement counter
             MOV A, D
ORA E     : Check whether count is 0
JNZ BACK  : If not zero, repeat
DCR C     : Check if multiplier zero, otherwise repeat
             JNZ DELAY
RET       : Return to main program
```

UNIT III – 8086 MICROPROCESSOR

3.1 Introduction

The 8086 is a 16-bit microprocessor intended to be used as the CPU in a microcomputer. The term "16-bit" means that its arithmetic logic unit, internal registers, and most of its instructions are designed to work 16-bit binary words. It has 16-bit data bus and 20-bit address bus.

Words will be stored in two consecutive memory locations. If the first byte of a word is at an even address, the 8086 can read the entire word in one operation. If the first byte of the word is at an odd address, the 8086 will read the first byte in one operation, and the second byte in another operation.

Features

- 8086 is a 40 pin IC.
- It is a 16-bit processor.
- Its operating voltage is 5 volts.
- Its operating frequency is 5 MHz
- The total memory addressing capacity is 1MB (external).
- It has 16-bit data bus and 20-bit address bus.
- It has fourteen 16-bit registers.
- It has around 20000 transistors in its circuitry and it is made in HMOS technology.
- Pipelining improves the performance of the processor so that operation is faster.8086 uses two stage of pipelining.
- First is Fetch Stage and the second is Execute Stage.
- Fetch stage that prefetch upto 6 bytes of instructions stores them in the queue.
- Execute stage that executes these instructions.
- Operates in two modes: 8086 operates in two modes:

Minimum Mode: A system with only one microprocessor.
Maximum Mode: A system with multiprocessor.

- **8086 uses memory banks:**The 8086 uses a memory banking system. It means entire data is not stored sequentially in a single memory of 1 MB but memory is divided into two banks of 512KB.
- **Interrupts:**8086 has 256 vectored interrupts.
- **Multiplication and Division:**8086 has a powerful instruction set. So that it supports Multiply and Divide operation.

3.2 Pin Diagram

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

The following signal description are common for both the minimum and maximum modes.

AD15-AD0

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

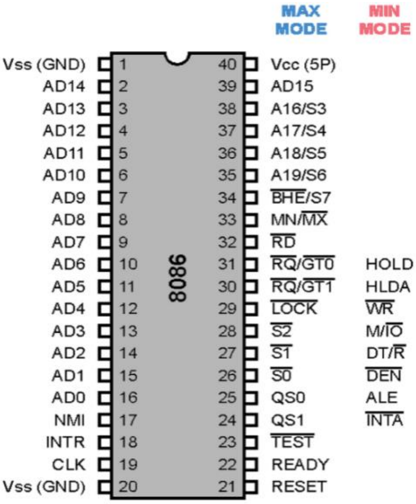


Fig. 3.1Pin Diagram

A19/S6, A18/S5, A17/S4, A16/S3: These are the time multiplexed address and status lines. During T1, these are the most significant address lines or memory operations. During I/O operations, these lines are low. During memory or I/O operations, status information is available on those lines for T2, T3, TW and T4.The status of the interrupt enable flag bit(displayed on S5) is updated at the beginning of each clock cycle. The S4 and S3 combined, indicate which segment register is presently being used for memory accesses as shown in the following table.These lines float to tri-state off during the local bus hold acknowledge. The status line S6 is always low(logical). The address bits are separated from the status bits using latches controlled by the ALE signal.

Table 3.1 Segment Register Indication

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

BHE/S7-Bus High Enable/Status: The bus high enable signal is used to indicate the transfer of data over the higher order (D15-D8) data bus as shown in the following table. It goes low for the data transfers 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, when- ever a byte is to be transferred on the higher byte of the data bus. The status information is available during T2, T3 and T4. The signal is active low and is high-impedance state during 'hold'. It is low during T1 for the first pulse of the interrupt acknowledge cycle.

Table 3.2 Bus High Enable Indication

BHE	A0	Indication
0	0	Whole word
0	1	Upper byte from or to odd address
1	0	Upper byte from or to even address
1	1	None

RD-Read: Read signal, when low, indicates the peripherals that the processor is performing a 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 in high-impedance during the 'hold acknowledge'.

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

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

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

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

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

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

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

MN/MX: The logic level at this pin decides whether the processor is to operate in either minimum (single processor) or maximum (multiprocessor) mode.

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

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

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

ALE-Address latch Enable: This output signal indicates the availability of the valid address on the address/data lines, and is connected to latch enable input of latches. This signal is active high and is never in high-impedance state.

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. Logically, this is equivalent to S1 in maximum mode. Its timing is the same as M/I/O.

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 DEN is in high-impedance state during 'hold acknowledge' cycle.

HOLD, HLDA-Hold/Hold Acknowledge: When the HOLD line goes high, it indicates to the processor that another master is requesting the bus access. The processor, after receiving the HOLD request, issues the hold acknowledge signal on HLDA pin, in the middle of the next clock cycle after completing the current bus (instruction) cycle. At the same time, the processor floats the local bus and control lines. When the processor detects the HOLD line low, it lowers the HLDA signal. HOLD is an asynchronous input, and it should be externally synchronized. If the DMA request is made while the CPU is performing a memory or I/O cycle, it will release the local bus during T 4 provided:

1. The request occurs on or before T 2 state of the current cycle.
2. The current cycle is not operating over the lower byte of a word (or operating on an odd address).
3. The current cycle is not the first acknowledge of an interrupt acknowledge sequence.
4. A Lock instruction is not being executed.

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

S2, S1, S0 -Status Lines: These are the status lines which reflect the type of operation, being carried out by the processor. These become active during T4 of the previous cycle and remain active during T1 and T2 of the current bus cycle. The status lines return to passive state during T3 of the current bus cycle so that they may again become active for the next bus cycle during T4. Any change in these lines during T3 indicates the starting of a new cycle, and return to passive state indicates end of the bus cycle. These status lines are encoded in the following table.

Table 3.3 Status Lines Indication

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

Lock

This output pin indicates that other system bus masters 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. This floats to tri-state off during "hold acknowledge". When the CPU is executing a critical instruction, which requires the system bus, the LOCK prefix instruction ensures that other processors connected in the system will not gain the control of the bus. The 8086, while executing the prefixed instruction, asserts the bus lock signal output, which may be connected to an external bus controller.

QS1, QS0-Queue Status

These lines give information about the status of the code prefetch queue. These are active during the CLK cycle after which the queue operation is performed. These are encoded as shown in the following table.

Table 3.4 Queue Status Indication

Qs1	Qs0	Indication
0	0	No operation
0	1	First byte of opcode from the queue
1	0	Empty queue
1	1	Subsequent byte from the queue

RQ/GT0, RQ/GT1-ReQuest/Grant: These pins are used by other local bus masters, in maximum mode, to force the processor to release the local bus at the end of the processor's current bus cycle. Each of the pins is bidirectional with RQ/GT0 having higher priority than RQ/ GT1, RQ/GT pins have internal pull-up resistors and may be left unconnected. The request! grant sequence is as follows:

1. A pulse one clock wide from another bus master requests the bus access to 8086.
2. During T4 (current) or T1 (next) clock cycle, a pulse one clock wide from 8086 to the requesting master, indicates that the 8086 has allowed the local bus to float and that it will enter the "hold acknowledge" state at next clock cycle. The CPU's bus interface unit is likely to be disconnected from the local bus of the system.
3. A one clock wide pulse from another master indicates to 8086 that the 'hold' request is about to end and the 8086 may regain control of the local bus at the next clock cycle. 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 requests those are received

while 8086 is performing memory or I/O cycle, the granting of the bus is governed by the rules of HOLD, and HLDA in minimum mode.

3.3 Internal Block Diagram

The 8086 CPU is divided into two independent functional parts, the bus interface unit or BIU, and the execution unit or EU.

The Bus Interface Unit

The BIU handles all data and addresses on the buses for the execution unit such as it sends out addresses, fetches instructions from memory, reads data from ports and memory as well as writes data to ports and memory. In BIU there are so many functional groups or parts these are as follows.

Instruction Queue

To increase the execution speed, BIU fetches as many as six instruction bytes ahead to time from memory. The prefetched instruction bytes are held for the EU in a first in first out group of registers called an instruction queue. When the EU is ready for its next instruction, it simply reads the instruction from this instruction queue. This is much faster than sending out an address to the system memory and to send back the next instruction byte. Fetching the next instruction while the current instruction executes is called pipelining.

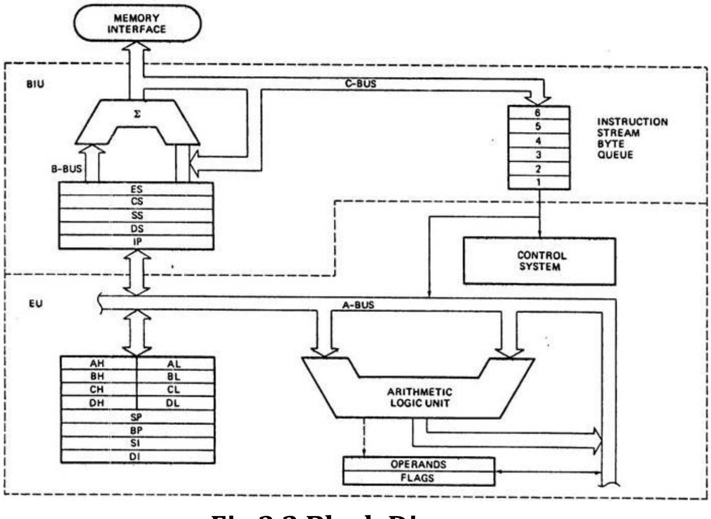


Fig 3.2 Block Diagram

Segment Registers

The BIU contains four 16-bit segment registers. They are: the extra segment (ES) register, the code segment (CS) registers, the data segment (DS) registers, and the stack segment (SS) registers. These segment registers are used to hold the upper 16 bits of the starting address for each of the segments. The part of a segment starting address stored in a segment register is often called the segment base.

1. **Code Segment (CS):** The CS register is used for addressing a memory location in the Code Segment of the memory, where the executable program is stored.
2. **Data Segment (DS):** The DS contains most data used by program. Data are accessed in the
   Data Segment by an offset address or the content of other register that holds the offset address.
3. **Stack Segment (SS):** SS defined a section of memory to store addresses and data while a subprogram executes.
4. **Extra Segment (ES):** ES is additional data segment that is used by some of the string to hold the extra destination data.

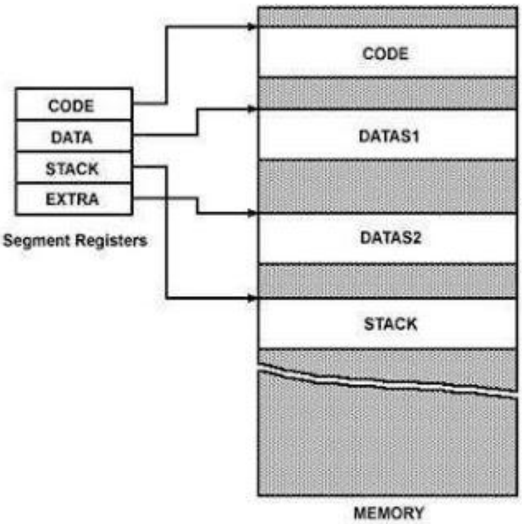


Fig 3.3 Segment Register

Instruction Pointer (IP)

In the BIU, the next register, below the segment register is instruction pointer. The instruction pointer (IP) holds the 16-bit address of the next code byte within this code segment.

The Execution Unit

The execution unit (EU) tells the BIU where to fetch instructions or data from, decodes instructions, and executes instructions. The functional parts of the execution unit are control circuitry or system, instruction decoder, and Arithmetic logic unit (ALU).Control circuitry to perform various internal operations. A decoder in the EU translates instructions fetched from memory to generate different internal or external control signals that required performing the operation. The EU has a 16-bit ALU, which can perform arithmetic operations such as add, subtract etc. and logical operations such as AND, OR, XOR, increment, decrement etc.

Flag Register

A 16-bit flag register is a flip-flop which indicates some condition produced by the execution of an instruction or controls certain operations of the EU. They are modified automatically by CPU after mathematical operations. It has 9 flags and they are divided into two categories:

1. Conditional Flags
2. Control Flags

Conditional Flags

Conditional flags represent result of last arithmetic or logical instructions.

- **Carry Flag (CF):** This flag will be set to one if the arithmetic operation produces the carry in MSB position. It is also used in multiple-precision arithmetic.
- **Auxiliary Flag (AF):** If an operation performed in ALU generates a carry/barrow from lower nibble (i.e. D0 – D3) to upper nibble (i.e. D4 – D7), the AF flag is set i.e. carry given by D3 bit to D4 is AF flag. This is not a general-purpose flag; it is used internally by the processor to perform Binary to BCD conversion.
- **Parity Flag (PF):** This flag is used to indicate the parity of result. If lower order 8-bits of the result contains even number of 1's, the Parity Flag is set to one and for odd number of 1's, the Parity Flag is reset i.e. zero.
- **Zero Flag (ZF):** It is set to one; if the result of arithmetic or logical operation is zero else it is reset.
- **Sign Flag (SF):** In sign magnitude format the sign of number is indicated by MSB bit. If the result of operation is negative, sign flag is set to one.

- **Overflow Flag (OF):** It occurs when signed numbers are added or subtracted. An OF indicates that the result has exceeded the capacity of machine.

Control Flags

Control flags are intentionally set or reset to control certain operations of the processor with specific instructions put in the program from the user. Control flags are as follows:

1. **Trap Flag (TF):** It is used for single step control. It allows user to execute one instruction of a program at a time for debugging. When trap flag is set, program can be run in single step mode.
2. **Interrupt Flag (IF):** It is an interrupt enable/disable flag, i.e. used to allow/prohibit the interruption of a program. If it is set, the maskable interrupt is enabled and if it is reset, the interrupt is disabled.
3. **Direction Flag (DF):** It is used in string operation. If it is set, string bytes are accessed from higher memory address to lower memory address. When it is reset, the string bytes are accessed from lower memory address to higher memory address.

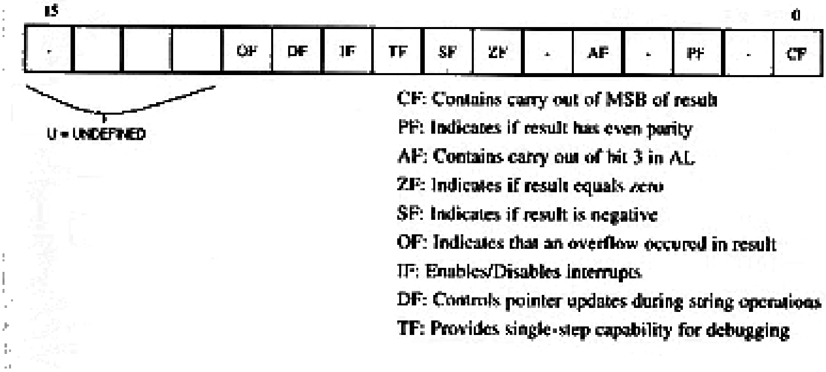

15
0
-
OF
DF
IF
TF
SF
ZF
-
AF
-
PF
-
CF
U = UNDEFINED
CF: Contains carry out of MSB of result
PF: Indicates if result has even parity
AF: Contains carry out of bit 3 in AL
ZF: Indicates if result equals zero
SF: Indicates if result is negative
OF: Indicates that an overflow occured in result
IF: Enables/Disables interrupts
DF: Controls pointer updates during string operations
TF: Provides single-step capability for debugging

Fig 3.4 Flag Register

3.4 General Purpose Registers

The EU has eight general purpose registers labeled AH, AL, BH, BL, CH, CL, DH, and DL. These registers can be used individually for temporary storage of 8-bit data. The AL register is also called the accumulator. Certain pairs of these general-purpose registers can be used together to store 16-bit data. The valid register pairs are AH and AL, BH and BL, CH and CL and DH and DL. These register pairs are referred to the AX, BX, CX, and DX resp.

1. **AX Register:** For 16-bit operations, AX is called the accumulator register that stores operands for arithmetic operations.
2. **BX Register:** This register is mainly used as a base register. It holds the starting base location of a memory region within a data segment.
3. **CX Register:** It is defined as a counter. It is primarily used in loop instruction to store loop counter.
4. **DX Register:** DX register is used to contain I/O port address for I/O instruction.

Stack Pointer Register

The stack pointer (SP) register contains the 16-bit offset from the start of the segment to the memory location where a word was most recently stored on the stack. The memory location where a word was most recently stored is called the top of stack.

Other Pointer and Index Registers

The EU also contains a 16-bit source index (SI) register, base pointer (BP) registers, and Destination Index (DI) registers. These three registers can be mainly used for temporary storage of 16-bit data just like a general-purpose register.

3.5 Addressing Modes of 8086

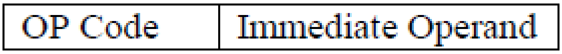
When 8086 executes an instruction, it performs the specified function on data. These data are called its operands and may be part of the instruction, reside in one of the internal registers of the microprocessor, stored at an address in memory or held at an I/O port, to access these different types of operands, the 8086 is provided with various addressing modes (Data Addressing Modes).

The 8086 has 12 addressing modes. The various 8086 addressing modes can be classified into five groups.

A. Addressing modes for accessing immediate and register data (register and immediate modes).
B. Addressing modes for accessing data in memory (memory modes)
C. Addressing modes for accessing I/O ports (I/O modes)
D. Relative addressing mode
E. Implied addressing mode

Immediate addressing mode

In this mode, 8- or 16-bit data can be specified as part of the instruction.



OP Code	Immediate Operand

Example 1

MOV CL, 03 H

Moves the 8-bit data 03 H into CL

Example 2

MOV DX, 0525 H

Moves the 16-bit data 0525 H into DX

In the above two examples, the source operand is in immediate mode and the destination operand is in register mode.

A constant such as “VALUE” can be defined by the assembler EQUATE directive such as VALUE EQU 35H

Example

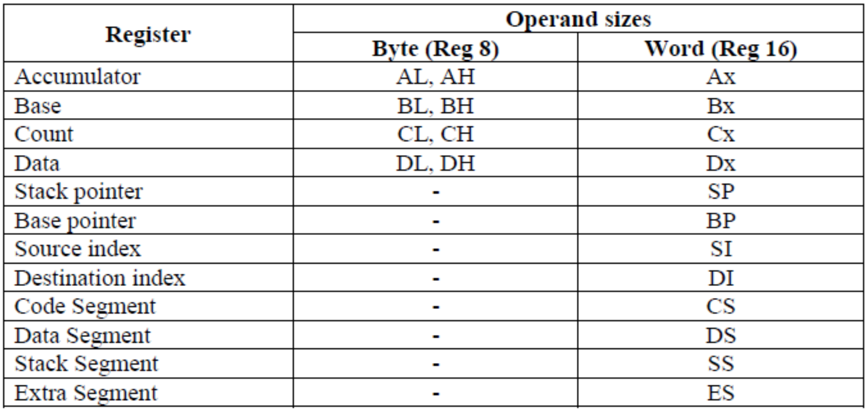
MOV BH, VALUE

Used to load 35 H into BH

Register addressing mode

The operand to be accessed is specified as residing in an internal register of 8086. The table below shows internal registers, one can be used as a source or destination operand, however only the data registers can be accessed as either a byte or word.

Table 3.5 Register with Operand Sizes



Register	Operand sizes	
	Byte (Reg 8)	Word (Reg 16)
Accumulator	AL, AH	Ax
Base	BL, BH	Bx
Count	CL, CH	Cx
Data	DL, DH	Dx
Stack pointer	-	SP
Base pointer	-	BP
Source index	-	SI
Destination index	-	DI
Code Segment	-	CS
Data Segment	-	DS
Stack Segment	-	SS
Extra Segment	-	ES

Example 1

MOV DX (Destination Register), CX (Source Register)

Which moves 16-bit content of CS into DX.

Example 2

MOV CL, DL

Moves 8-bit contents of DL into CL

MOV BX, CH is an illegal instruction.

* The register sizes must be the same.

Direct addressing mode

The instruction Opcode is followed by an affective address, this effective address is directly used as the 16-bit offset of the storage location of the operand from the location specified by the current value in the selected segment register.

The default segment is always DS.

The 20-bit physical address of the operand in memory is normally obtained as PA = DS: EA

But by using a segment override prefix (SOP) in the instruction, any of the four segment registers can be referenced,

$$PA = \left\{ \begin{array}{c} CS \\ DS \\ SS \\ ES \end{array} \right\} : \left\{ \text{Direct Address} \right\}$$

The Execution Unit (EU) has direct access to all registers and data for register and immediate operands. However, the EU cannot directly access the memory operands. It must use the BIU, in order to access memory operands. In the direct addressing mode, the 16-bit effective address (EA) is taken directly from the displacement field of the instruction.

Example 1

MOV CX, START

If the 16-bit value assigned to the offset START by the programmer using an assembler pseudo instruction such as DW is 0040 and [DS] = 3050.

Then BIU generates the 20-bit physical address 30540 H.

The content of 30540 is moved to CL

The content of 30541 is moved to CH

Example 2

MOV CH, START

If [DS] = 3050 and START = 0040

8-bit content of memory location 30540 is moved to CH.

Example 3

MOV START, BX

With [DS] = 3050, the value of START is 0040.

Physical address: 30540

MOV instruction moves (BL) and (BH) to locations 30540 and 30541 respectively.

Register indirect addressing mode

The EA is specified in either pointer (BX) register or an index (SI or DI) register. The 20-bit physical address is computed using DS and EA.

Example

MOV [DI], BX

register indirect

If [DS] = 5004, [DI] = 0020, [Bx] = 2456 PA=50060.

The content of BX (2456) is moved to memory locations 50060 H and 50061 H.

$$PA = \left\{\begin{matrix} CS \\ DS \\ SS \\ ES \end{matrix}\right\} = \left\{\begin{matrix} BX \\ SI \\ DI \end{matrix}\right\}$$

Based addressing mode

$$PA = \left\{\begin{matrix} CS \\ DS \\ SS \\ ES \end{matrix}\right\} : \left\{\begin{matrix} BX \\ \text{or} \\ BP \end{matrix}\right\} + \text{displacement}$$

when memory is accessed PA is computed from BX and DS when the stack is accessed PA is computed from BP and SS.

Example

MOV AL, START [BX]

or

MOV AL, [START + BX]

based mode

EA: [START] + [BX]

PA: [DS] + [EA]

The 8-bit content of this memory location is moved to AL.

Indexed addressing mode

$$PA = \left\{\begin{matrix} CS \\ DS \\ SS \\ ES \end{matrix}\right\} : \left\{\begin{matrix} SI \\ \text{or} \\ DI \end{matrix}\right\} + 8 \text{ or } 16\text{bit displacement}$$

Example

MOV BH, START [SI]

PA: [SART] + [SI] + [DS]

The content of this memory is moved into BH.

Based Indexed addressing mode

$$PA = \left\{\begin{matrix} CS \\ DS \\ SS \\ ES \end{matrix}\right\} : \left\{\begin{matrix} BX \\ \text{or} \\ BP \end{matrix}\right\} + \left\{\begin{matrix} SI \\ \text{or} \\ DI \end{matrix}\right\} + 8 \text{ or } 16\text{bit displacement}$$

Example

MOV ALPHA [SI] [BX], CL

If [BX] = 0200, ALPHA – 08, [SI] = 1000 H and [DS] = 3000

Physical address (PA) = 31208

8-bit content of CL is moved to 31208 memory address.

String addressing mode

The string instructions automatically assume SI to point to the first byte or word of the source operand and DI to point to the first byte or word of the destination operand. The contents of SI and DI are automatically incremented (by clearing DF to 0 by CLD instruction) to point to the next byte or word.

Example

MOV S BYTE

If [DF] = 0, [DS] = 2000 H, [SI] = 0500,

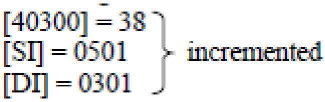
[ES] = 4000, [DI] = 0300

Source address: 20500, assume it contains 38

PA: [DS] + [SI]

Destination address: [ES] + [DI] = 40300, assume it contains 45

After executing MOV S BYTE,



$$\left.\begin{array}{l}[40300] = 38\\ [SI] = 0501\\ [DI] = 0301\end{array}\right\}\text{incremented}$$

I/O mode

Direct Mode

Port number is an 8-bit immediate operand.

Example: OUT 05H, AL

Outputs [AL] to 8-bit port 05 H

Indirect Mode

The port number is taken from DX.

Example 1

INAL, DX

If [DX] = 5040

8-bit content by port 5040 is moved into AL.

Example 2

IN AX, DX

Inputs 8-bit content of ports 5040 and 5041 into AL and AH respectively.

Relative addressing mode

Example: JNC START

If CY=O, then PC is loaded with current PC contents plus 8-bit signed value of START, otherwise the next instruction is executed.

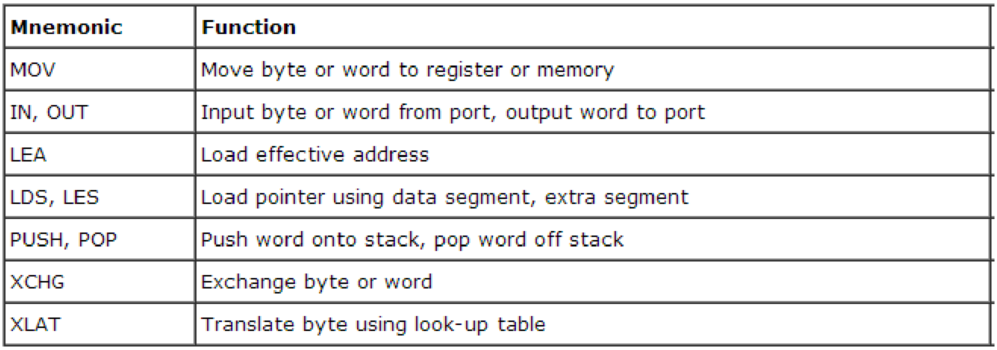
Implied addressing mode

Instruction using this mode have no operands.

Example: CLC which clears carry flag to zero.

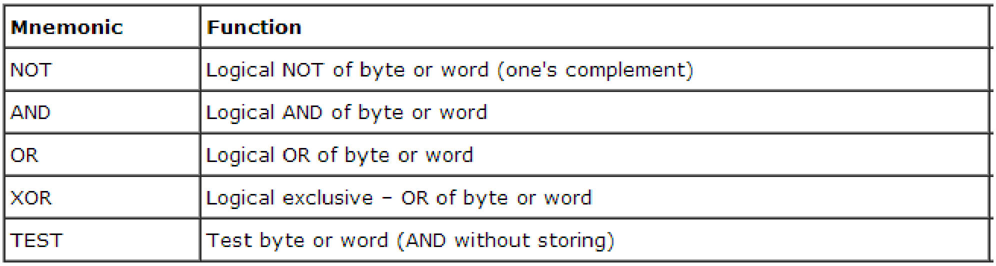
3.6 Instruction Set

Data Transfer Instructions



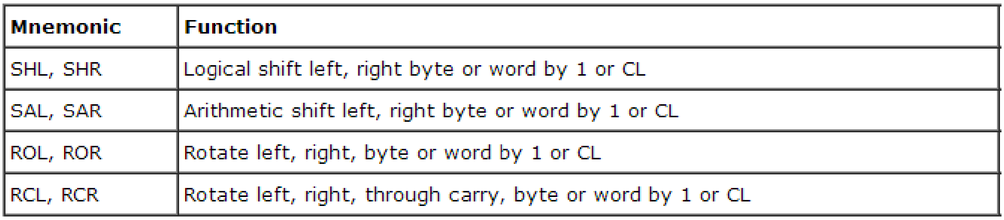
Mnemonic	Function
MOV	Move byte or word to register or memory
IN, OUT	Input byte or word from port, output word to port
LEA	Load effective address
LDS, LES	Load pointer using data segment, extra segment
PUSH, POP	Push word onto stack, pop word off stack
XCHG	Exchange byte or word
XLAT	Translate byte using look-up table

Logical Instructions



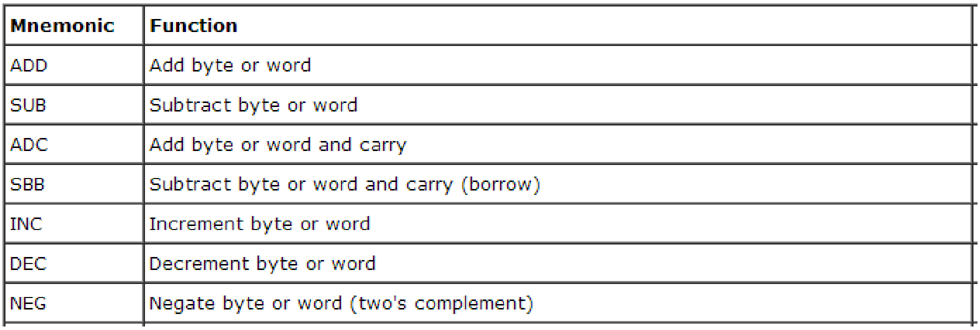
Mnemonic	Function
NOT	Logical NOT of byte or word (one's complement)
AND	Logical AND of byte or word
OR	Logical OR of byte or word
XOR	Logical exclusive – OR of byte or word
TEST	Test byte or word (AND without storing)

Shift and Rotate Instructions

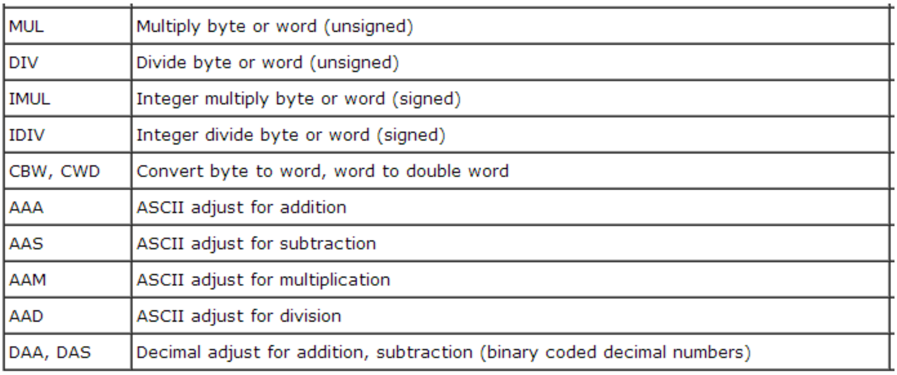


Mnemonic	Function
SHL, SHR	Logical shift left, right byte or word by 1 or CL
SAL, SAR	Arithmetic shift left, right byte or word by 1 or CL
ROL, ROR	Rotate left, right, byte or word by 1 or CL
RCL, RCR	Rotate left, right, through carry, byte or word by 1 or CL

Arithmetic Instructions

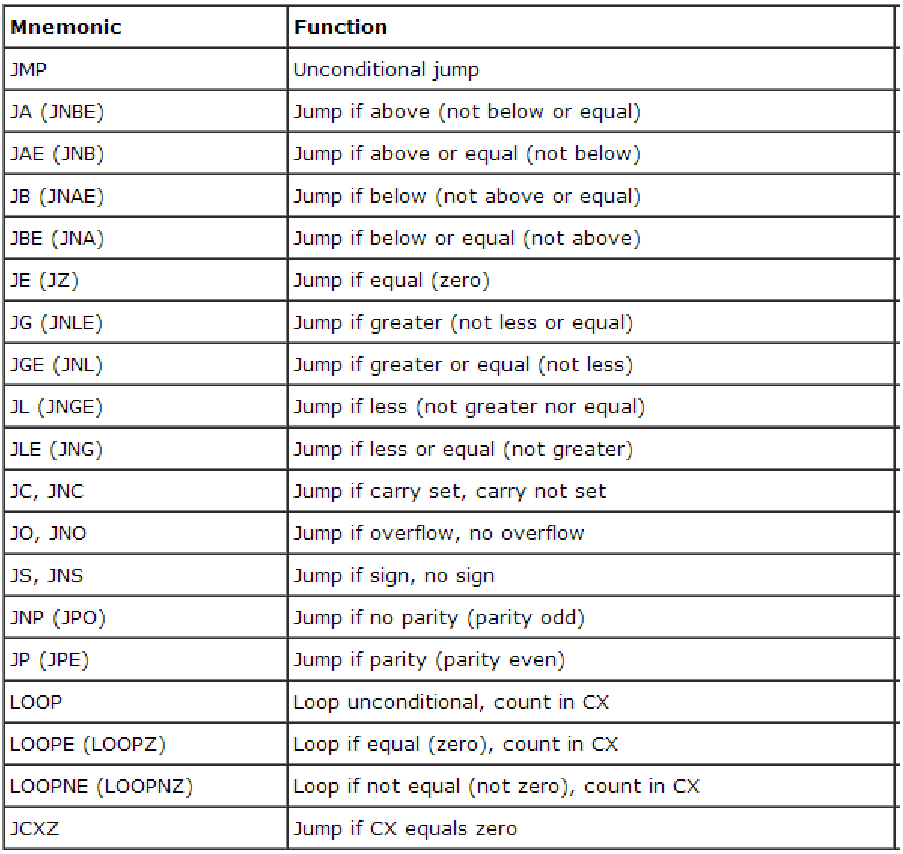


Mnemonic	Function
ADD	Add byte or word
SUB	Subtract byte or word
ADC	Add byte or word and carry
SBB	Subtract byte or word and carry (borrow)
INC	Increment byte or word
DEC	Decrement byte or word
NEG	Negate byte or word (two's complement)



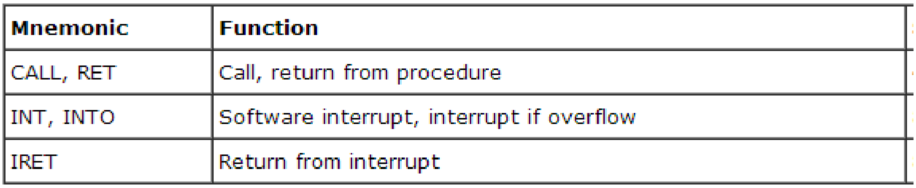
MUL	Multiply byte or word (unsigned)
DIV	Divide byte or word (unsigned)
IMUL	Integer multiply byte or word (signed)
IDIV	Integer divide byte or word (signed)
CBW, CWD	Convert byte to word, word to double word
AAA	ASCII adjust for addition
AAS	ASCII adjust for subtraction
AAM	ASCII adjust for multiplication
AAD	ASCII adjust for division
DAA, DAS	Decimal adjust for addition, subtraction (binary coded decimal numbers)

Jump and Loop Instructions



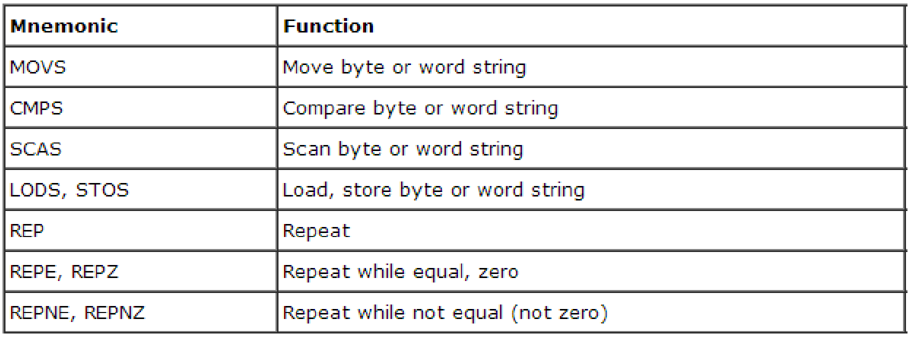
Mnemonic	Function
JMP	Unconditional jump
JA (JNBE)	Jump if above (not below or equal)
JAE (JNB)	Jump if above or equal (not below)
JB (JNAE)	Jump if below (not above or equal)
JBE (JNA)	Jump if below or equal (not above)
JE (JZ)	Jump if equal (zero)
JG (JNLE)	Jump if greater (not less or equal)
JGE (JNL)	Jump if greater or equal (not less)
JL (JNGE)	Jump if less (not greater nor equal)
JLE (JNG)	Jump if less or equal (not greater)
JC, JNC	Jump if carry set, carry not set
JO, JNO	Jump if overflow, no overflow
JS, JNS	Jump if sign, no sign
JNP (JPO)	Jump if no parity (parity odd)
JP (JPE)	Jump if parity (parity even)
LOOP	Loop unconditional, count in CX
LOOPE (LOOPZ)	Loop if equal (zero), count in CX
LOOPNE (LOOPNZ)	Loop if not equal (not zero), count in CX
JCXZ	Jump if CX equals zero

Call and Return Instructions



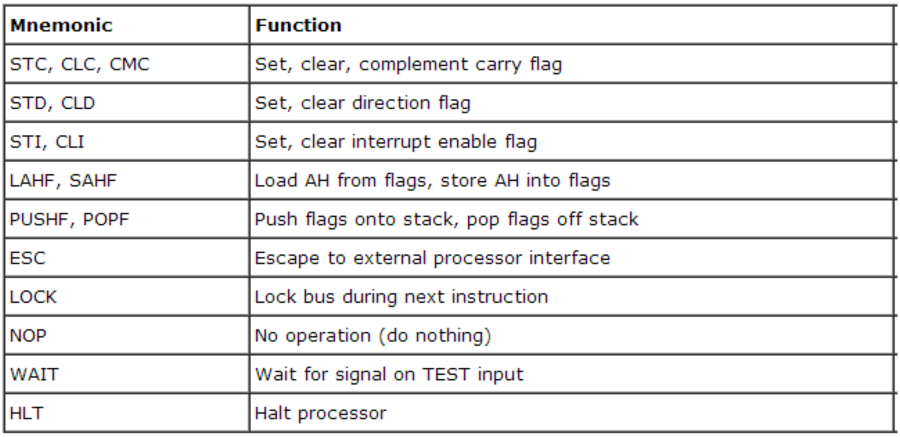
Mnemonic	Function
CALL, RET	Call, return from procedure
INT, INTO	Software interrupt, interrupt if overflow
IRET	Return from interrupt

String Instructions



Mnemonic	Function
MOVS	Move byte or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS, STOS	Load, store byte or word string
REP	Repeat
REPE, REPZ	Repeat while equal, zero
REPNE, REPNZ	Repeat while not equal (not zero)

Processor and Flag Control Instructions



Mnemonic	Function
STC, CLC, CMC	Set, clear, complement carry flag
STD, CLD	Set, clear direction flag
STI, CLI	Set, clear interrupt enable flag
LAHF, SAHF	Load AH from flags, store AH into flags
PUSHF, POPF	Push flags onto stack, pop flags off stack
ESC	Escape to external processor interface
LOCK	Lock bus during next instruction
NOP	No operation (do nothing)
WAIT	Wait for signal on TEST input
HLT	Halt processor

3.7 Min/Max Mode of Operation

The microprocessors 8086 and 8088 can be configured to work in two modes: The Minimum mode and the Maximum mode.

The Minimum mode is used for single processor system, where 8086/8088 directly generates all the necessary control signals.

The Maximum mode is designed for multiprocessor systems, where an additional "Bus-controller" IC is required to generate the control signals. The processors control the Bus controller using status-codes.

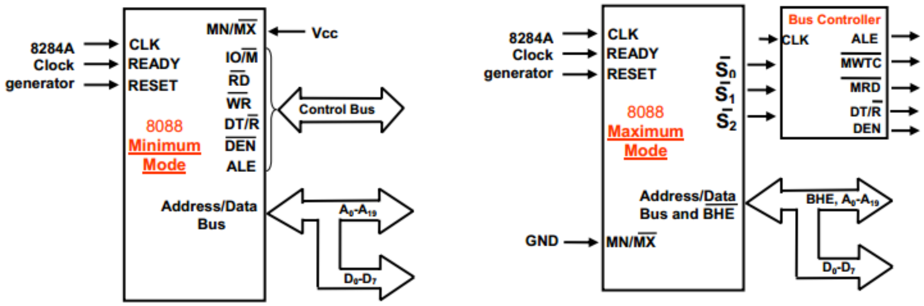


Fig 3.5 Min/Max mode signals

3.8 Interrupts

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

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

Interrupt can divide to five groups:

1. Hardware interrupt
2. Non-maskable interrupt
3. Maskable interrupt
4. Software interrupt
5. Reset

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

The 8086 chips allow up to 256 vectored interrupts. This means that you can have up to 256 different sources for an interrupt and the 80x86 will directly call the service routine for that interrupt without any software processing. This is in contrast to non-vectored interrupts that transfer control directly to a single interrupt service routine, regardless of the interrupt source.

The 8086 provides a 256-entry interrupt vector table beginning at address 0:0 in memory. This is a 1K table containing 256 4-byte entries. Each entry in this table contains a segmented address that points at the interrupt service routine in memory. The lowest five types are dedicated to specific interrupts such as the divide by zero interrupt and the non-maskable interrupt. The next 27 interrupt types, from 5 to 31 are reserved by Intel for use in future microprocessors. The upper 224 interrupt types, from32 to 255, are available to use for hardware and software interrupts.

Hardware Interrupt

The primary sources of interrupts, however, are the PCs timer chip, keyboard, serial ports, parallel ports, disk drives, CMOS real-time clock, 4 mouse, sound cards, and other peripheral devices. These devices connect to an Intel 8259A programmable interrupt controller (PIC) that prioritizes the interrupts and interfaces with the 80x86 CPU. The 8259A chip adds considerable complexity to the software that processes interrupts.

Non-Maskable Interrupt (NMI)

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

Maskable Interrupt

Whenever an external signal activates the INTR pin, the microprocessor will be interrupted only if interrupts are enabled using set interrupt Flag instruction. If the interrupts are disabled using clear interrupt Flag instruction, the microprocessor will not get interrupted even if INTR is activated. That is, INTR can be masked. INTR is a non-vectored interrupt, which means, the 8086 does not know where to branch to service the interrupt. The 8086 has to be told by an external device like a Programmable Interrupt controller regarding the branch. Whenever the INTR pin is activated by an I/O port, if Interrupts are enabled and NMI is not active at that time, the microprocessor finishes the current instruction that is being executed and gives out a '0' on INTA pin twice. When INTA pin goes low for the first time, it asks the external device to get ready. In response to the second INTA the microprocessor receives the 8 bits, say N, from a programmable Interrupt controller. The action taken is as follows.

1. Complete the current instruction.
2. Activates INTA output, and receives type Number, say N
3. Flag register value, CS value of the return address & IP value of their turn address are pushed on to the stack.
4. IP value is loaded from contents of word location N x 4.
5. CS is loaded from contents of the next word location.
6. Interrupt Flag and trap Flag are reset to 0.

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

Software Interrupt

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

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

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

INT-Interrupt Instruction with Type number Specified

The mnemonic for this is INT. It is a 2-byte instruction. The first byte provides the op-code and the second byte the Interrupt type number. Op-code for this instruction is CD_H. The execution of an INT instruction, say INTN, when N is the value in the range 00H to FFH, results in the following:

1. Flag register value is pushed on to the stack.
2. CS value of the Return address and IP value of the Return address are pushedon to the stack.
3. IP is loaded from the contents of the word location N x 4.
4. CS is loaded from the contents of the next word location.
5. Interrupt Flag and Trap Flag are reset to 0.

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

INT 3-Break Point Interrupt Instruction

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

The execution of INT3 instruction results in the following.

1. Flag register value is pushed on to the Stack.
2. CS value of the return address and IP value of the return address are pushed onto the Stack.
3. IP is loaded from the contents of the word location 3x4 = 0000CH.
4. CS is loaded from the contents of the next word location.
5. Interrupt Flag and Trap Flag are reset to 0.

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

INTO - Interrupt on overflow instruction

The overflow flag, OF, will be set if the signed result of an arithmetic operation on two signed numbers is too large to be represented in the destination register or memory location. For example, if we add the 8-bit signed number 01101100 and the 8- bit signed number 01010001, the signed result will be 10111101. This is correct if we add unsigned binary numbers, but it is not the correct signed result. There are two ways to detect and respond to an overflow error in a program. One way is to put the jump if overflow instruction, JO, immediately after the arithmetic instruction. If the overflow flag is Set, execution will jump to the address specified in the JO instruction. At this address an error routine may be put which respond to the overflow. The second way is to put them INTO instruction immediately after the arithmetic Instruction in the program. The mnemonic for the instruction is INTO. It is a 1-byte instruction. The op-code for this is CEH. It is a conditional interrupt instruction. Only if the overflow flag is Set, a branch takes place to an interrupt handler whose interrupt type number is 4. If the overflow flag is reset, the execution continues with the next instruction. The execution of INTO results in the following.

1. Flag register values are pushed on to the Stack.
2. CS value of the return address and IP value of the return address and IP value of the return address are pushed on to the stack.
3. IP is loaded from the contents of word location 4x4 = 00010H.
4. CS is loaded from the contents of next word location.
5. Interrupt flag and Trap flag are reset to 0.

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

Reset

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

3.9 System Clock

- To synchronize the internal and external operations of the microprocessor a clock (CLK) input signal is used. The CLK can be generated by the 8284 clock generator IC.
- The 8086 is manufactured in three speeds: 5 MHz, 8 MHz and 10 MHz
- For 8086, we connect either a 15-, 24- or 30-MHz crystal between inputs X1 and X2 inputs of the clock chip. The fundamental crystal frequency is divided by 3 within the 8284 to give either a 5-, 8- or 10-MHz clock signal, which is directly connected to the CLK input of the 8086.

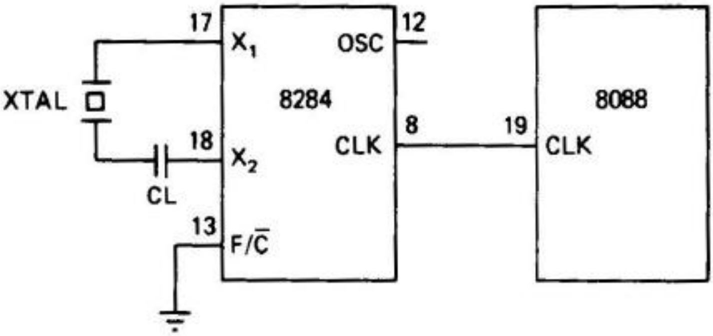


Fig 3.6 Clock circuit

Bus Cycle and Time States

A bus cycle defines the sequence of events when the MPU communicates with an external device, which starts with an address being output on the system bus followed by a read or write data transfer.

3.10 Types of bus cycles

- Memory Read Bus Cycle
- Memory Write Bus Cycle
- Input/Output Read Bus Cycle
- Input/Output Write Bus Cycle

The bus cycle of the 8086 microprocessor consists of at least four clock periods. These four time-states are called T1, T2, T3 and T4.

Memory Read and Write Bus Cycles

During period T1,

- The 8086 outputs the 20-bit address of the memory location to be accessed on its multiplexed address/data bus. BHE is also output along with the address during T1.
- At the same time a pulse is also produced at ALE. The trailing edge or the high level of this pulse is used to latch the address in external circuitry.
- Signal M/IO is set to logic 1 and signal DT/R is set to the 0-logic level and both are maintained throughout all four periods of the bus cycle.
- Beginning with period T2,
- Status bits S3 through S6 are output on the upper four address bus lines. This status information is maintained through periods T3 and T4. On the other hand, address/data bus lines AD0 through AD7 are put in the high-Z state during T2.
- Late in period T2, RD is switched to logic 0. This indicates to the memory subsystem that a read cycle is in progress. DEN is switched to logic 0 to enable external circuitry to allow the data to move from memory onto the microprocessor's data bus.

During period T3,

- The memory must provide valid data during T3 and maintain it until after the processor terminates the read operation. The data read by the 8086 microprocessor can be carried over all 16 data bus lines 8.

During T4,

- The 8086 switches RD to the inactive 1 logic level to terminate the read operation. DEN returns to its inactive logic level late during T4 to disable the external circuitry.

During period T1,

- The address along with BHE are output and latched with the ALE pulse.
- M/IO is set to logic 1 to indicate a memory cycle.
- However, this time DT/R is switched to logic 1. This signals external circuits that the 8086 is going to transmit data over the bus.

Beginning with period T2,

- WR is switched to logic 0 telling the memory subsystem that a write operation is to follow.
- The 8086 puts the data on the bus late in T2 and maintains the data valid through T4. Data will be carried over all 16 data bus lines.
- DEN enables the external circuitry to provide a path for data from the processor to the memory.

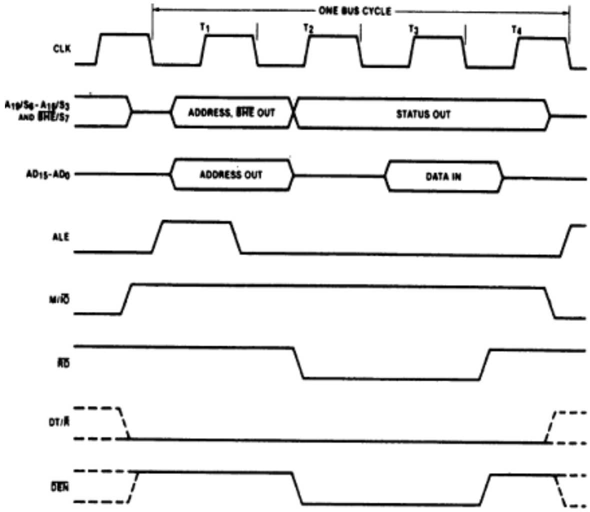


Fig 3.7 Min mode memory read bus cycle

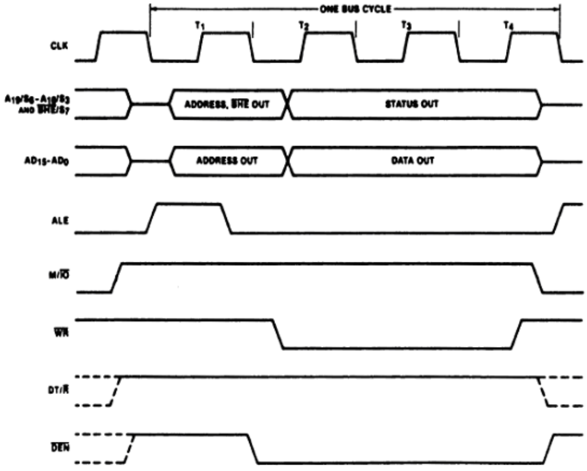


Fig 3.8 Min mode memory write bus cycle

3.11 Methods of Interfacing I/O Devices

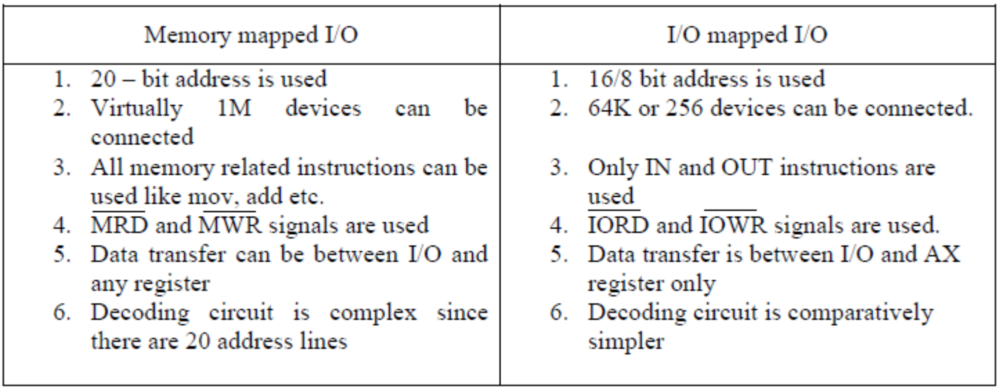
There are two ways by which one can interface I/O devices to a microprocessor.

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

In memory mapped I/O, the 1 Mb memory that can be interfaced to 8086/8088 itself is used to address I/O devices. But in I/O mapped I/O there is a separate address space for I/O devices.

Some important differences between I/O mapped I/O and memory mapped I/O are listed below:

Table 3.6 Difference between memory and I/O mapped I/O



Memory mapped I/O	I/O mapped I/O
1. 20 – bit address is used	1. 16/8 bit address is used
2. Virtually 1M devices can be connected	2. 64K or 256 devices can be connected.
3. All memory related instructions can be used like mov, add etc.	3. Only IN and OUT instructions are used
4. $\overline{MRD}$ and $\overline{MWR}$ signals are used	4. $\overline{IORD}$ and $\overline{IOWR}$ signals are used.
5. Data transfer can be between I/O and any register	5. Data transfer is between I/O and AX register only
6. Decoding circuit is complex since there are 20 address lines	6. Decoding circuit is comparatively simpler

Memory mapped I/O is seldom used. Address of the port is 0740H.

Design Examples

1. Interface 8 switches to 8086 microprocessor and read their status (1 – ON, 0 – OFF)

Store this information in a register.

The Interface diagram is shown below.

The code to read and store the status of switches is as follows:

MOV DX, 0740H; Variable port addressing mode, hence keep port address in DX register

IN AL, DX; AL will receive the status of switches.

MOV BL, AL

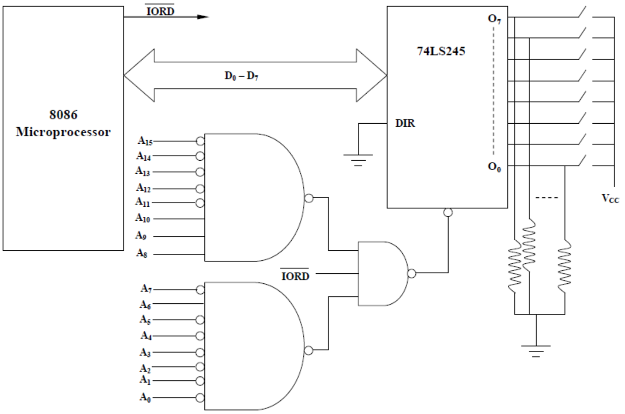


Fig 3.9 Interface diagram

2 Interface 8 switches to 8086 microprocessor and read their status (1 – ON, 0 – OFF)

Count the number of 1's in the byte received. Store this information in a register.

The interface diagram is as same as above.

The code is as follows

```
MOV DX, 0740H
IN AL, DX
MOV AH, 00; TO HOLD BIT COOUNT
MOV CX, 00; NUMBER OF TIMES THE INFORMATION TO BE CHECKED
```

Repeat

```
ROR AL,1
JNC NEXT; IF THERE IS A CARRY, INCREMENT THE COUNTER
INC AH
```

Next

```
LOOP REPEAT
MOV BH, AH; STORE BIT COUNT IN A REGISTER
```

3. Interface 8 switches to 8086 microprocessor and read their status. Count the number of 1's in the byte received. Display it on a seven-segment display. Input port address is 0008H and output port address is 000AH.

The Interface diagram is shown below.

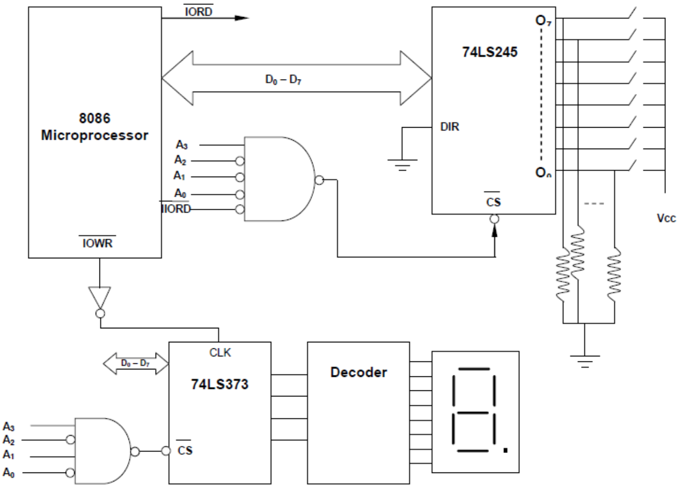


Fig. 3.10 Interface diagram

Here we are not decoding the entire address bus. Instead, we used only least significant 4 bits to decode the input and output devices. It creates a problem called foldback where a device has multiple addresses. Since we are not using this interface in any other application, we can sometimes use linear decoding also like this.

Sample code is as follows

```
MOV BL, 00
MOV CX, 08H
IN AL, 08H; fixed port addressing mode
```

Repeat

```
ROR AL, 1; to check for 1
JNC NEXT
INC BL
```

Next

```
LOOP REPEAT
MOV AL, BL
OUT 0AH, AL; sending information to 7 – segment display connected to port 0AH
```

3.12 Assembly Language Programming

1.To write 8086 Assembly Language Program to add two 32-bit signed & unsigned number.

Code

```
MOV AX,5000H; Initialize DATA SEGMENT
MOV DS, AX; to 5000H
MOV AX, [1000H]; take lower 16-bit of NUM1 in AX
MOV BX, [2000H]; take lower 16-bit of NUM2 in BX
ADD AX, BX; AX = AX + BX
MOV [3000H], AX; Store lower 16-bit result at NUM3
MOV AX, [1002H]; take higher 16-bit of NUM1 in AX
MOV BX, [2002H]; take higher 16-bit of NUM2 in BX
ADC AX, BX; AX = AX + BX + CF (add with carry)
MOV [3002H], AX; Store higher 16-bit result at NUM3
HLT; Halt 8086
```

Given inputs

NUM1 Unsigned Signed
[5000:1000] = 78H (LSB) = 88 H (LSB)
[5000:1001] = 56H = A9 H
[5000:1002] = 34H = CB H
[5000:1003] = 12H (MSB) =ED H (MSB)

NUM2
[5000:2000] = 34H (LSB) = CC H (LSB)
[5000:2001] = 12H = ED H
[5000:2002] = 34H = CB H
[5000:2003] = 12H (MSB) = ED H (MSB)

and the output is this

NUM3
[5000:3000] = ACH (LSB) = 54 H (LSB)

```
[5000:3001] = 68H = 97 H
[5000:3002] = 68H = 97 H
[5000:3003] = 24H (MSB) = DB H (MSB)
```

2. To write 8086 Assembly Language Program to Add two ASCII & BCD number.; ALP to add two ASCII numbers

```
MOV AX,5000H; Initialize DATA SEGMENT
MOV DS, AX; to 5000H
MOV AX,0000H; Clear AX register
MOV AL, [1000H]; take first ASCII number NUM1 in AL
MOV BL, [2000H]; take second ASCII number NUM2 in BL
ADD AL, BL; AL = AL + BL
AAA; ASCII Adjust After Addition
OR AX,3030H; logical OR with contents of AX & 3030H
MOV [3000H], AX; Store ASCII result at NUM3
HLT; Halt 8086
```

; ALP to add two BCD numbers

```
MOV AX,5000H; Initialize DATA SEGMENT
MOV DS, AX; to 5000H
MOV AX,0000H; Clear AX register
MOV AL, [1000H]; take first BCD number NUM1 in AL
MOV BL, [2000H]; take second BCD number NUM2 in BL
ADD AL, BL; AL = AL + BL
DAA; Decimal Adjust After Addition
MOV [3000H], AL; Store BCD result at NUM3
HLT; Halt 8086
```

Input

```
NUM1 ASCII BCD NUM2 ASCII BCD
[5000:1000] = 39H = 45H [5000:2000] = 39H = 49H
```

Output

```
NUM3 ASCII BCD
[5000:3000] = 38H (LSB) = 94H
[5000:3001] = 31H (MSB) = 00H
```

3. To write 8086 Assembly Language Program to Subtract two 32-bit signed & unsigned number

```
    MOV AX,5000H; Initialize DATA SEGMENT
    MOV DS, AX; to 5000H
    MOV AX, [1000H]; take lower 16-bit of NUM1 in AX
    MOV BX, [2000H]; take lower 16-bit of NUM2 in BX
    SUB AX, BX; AX = AX - BX
    MOV [3000H], AX; Store lower 16-bit result at NUM3
    MOV AX, [1002H]; take higher 16-bit of NUM1 in AX
    MOV BX, [2002H]; take higher 16-bit of NUM2 in BX
    SBB AX, BX; AX = AX - BX - CF (subtract with barrow)
    MOV [3002H], AX; Store higher 16-bit result at NUM3
    HLT; Halt 8086
```

Input

NUM1 Unsigned Signed
[5000:1000] = 78H (LSB) = 88 H (LSB)
[5000:1001] = 56H = A9 H
[5000:1002] = 78H = 87 H
[5000:1003] = 56H (MSB) =A9 H (MSB)

NUM2

[5000:2000] = 34H (LSB) = CC H (LSB)
[5000:2001] = 12H = ED H
[5000:2002] = 34H = CB H
[5000:2003] = 12H (MSB) = ED H (MSB)

Output

NUM3
[5000:3000] = 44H (LSB) = BCH (LSB)
[5000:3001] = 44H = BB H
[5000:3002] = 44H = BB H
[5000:3003] = 44H (MSB) = BB H (MSB)

UNIT IV MICROCONTROLLER 8051

4.1 Introduction

Microprocessors are playing an important role in various industries and influencing our day-to-day life more strongly than one can imagine. Basically, microprocessor is a single chip Very Large Scale Integrated (VLSI) digital circuit. The term microprocessor comes from the merger of micro-electronics technology. Microprocessor is defined as Arithmetic Logic Unit (ALU) and Control Unit (CU) on a single chip. The Central Processor Unit (CPU) consists of an Arithmetic Logic Unit (ALU) and Control Unit (CU), So microprocessor can act as CPU. To have a minimum system configuration, microprocessor should be connected with external memory, I/O ports etc.

Similarly, the microcontroller can be defined as a complete microprocessor system on a single chip which includes the CPU (microprocessor), Memory (RAM, ROM), I/O ports, Serial interface etc. Microcontrollers are developed to replace the microprocessor in low cost products.

4.2 Block Diagram of Microcontroller

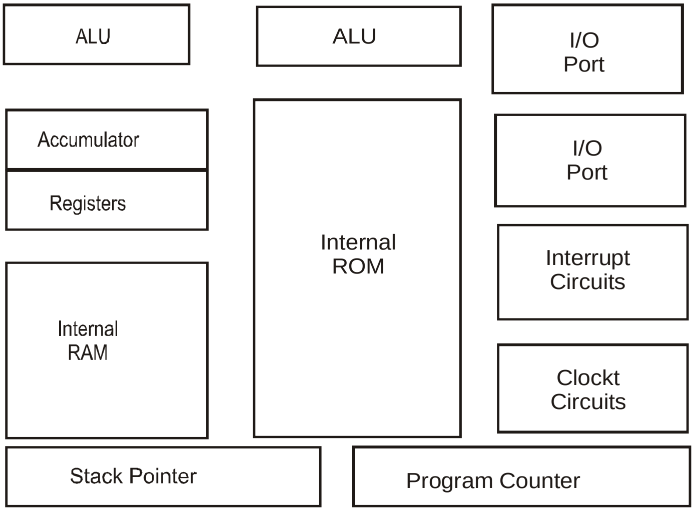


Fig. 4.1 Block diagram of Micro Controller

Fig. 4.1 shows the block diagram of a typical microcontroller, which is a true computer on a single ship. It has all features found in a microprocessor CPU, ALU, Program Counter, Stack Pointer and Registers. In added to the above feature, it has internal ROM, RAM, Parallel I/O ports, Serial I/O port and timers. Like microprocessor it is a general-purpose device, which reads the data, perform limited calculations and logical operations and control the environment based on the result of calculations/logical operations.

The important use of a microcontroller is to control the operation of machines using a program stored in ROM memory and does not change its operation over the life time of the system.

Many of the instructions are coupled with pins on the integrated circuit package. Thus, the pins are programmable, which means pins of microcontroller are accessible by the programmer and can be set as per the requirement.

Architecture of Microcontroller 8051

The Intel 8051 is an 8-bit single chip 40 pin Dual in Package (DIP) IC. It is available in N channel metal oxide (CMOS) silicon construction. It is designed for complicated real time instrumentation and industrial controls. It operates on 12MHz clock and single +5V supply.

Fig. 1.4 shows the architecture of 8051 microcontroller. It shows all the features such as memories, I/O ports, Timers, Serial data communication, Interrupt control, registers etc.

CPU

Microcontroller 8051 has an 8-bit CPU. It consists of Accumulator (A Register), B register, Arithmetic Logic Unit (ALU) and Control Unit (CU). It is used to perform basic mathematical calculations and Logical operations. 8051 CPU can process 8-bit data only.

Memory

The 8051 has internal RAM and ROM memories. 128 bytes of RAM and 4Kbytes of ROM are available with 8051. Internal RAM has 4 register banks each having 8 registers.

I/O Ports

8051 consist of four 8-bit ports. They are port 0, port 1, port 2 and port 3. These ports can be programmed either input or output port. These ports are arranged in thirty-two input/output pins.

Timer/Counter

There are two timers timer 0 and timer 1 provided in 8051. Both timers are 16 bits wide. Timers are programmed to generate desired time delays.

Registers

In the CPU of 8051, registers are used to store information temporarily. The majority of 8051 registers are 8-bit registers. Some of the widely used general purpose registers are A (Accumulator), B, R0, R1, R2, R3, R4, R5, R6, R7, Data Pointer and Program Counter. Only two registers Data pointer (DPTR) register and Program Counter (PC) are 16-bit wide.

Special Function Registers or Control registers

A group of specific internal registers use for some special functions is called Special Function Registers (SFR). SFRs hold the control bits of ALU operation, Port operation, Timer operation, Serial communication, Interrupt Control, Memory management etc.

Addresses of SFRs are between 80h and FFh. Typical SFRs are TCON, TMOD, SCON, PCON, IP and IE.

Oscillator and Clock Circuit

The 8051 has an on-chip oscillator but requires an external resonant tank circuit to determine the clock frequency. Normally the external resonant tank circuit is a quartz crystal and two 30PF capacitors. The crystal circuit is connected to pins XTAL1 and XTAL2. The crystal frequency is the basic clock frequency i.e. 12MHz. Clock pulses generated are used to synchronise the internal operations.

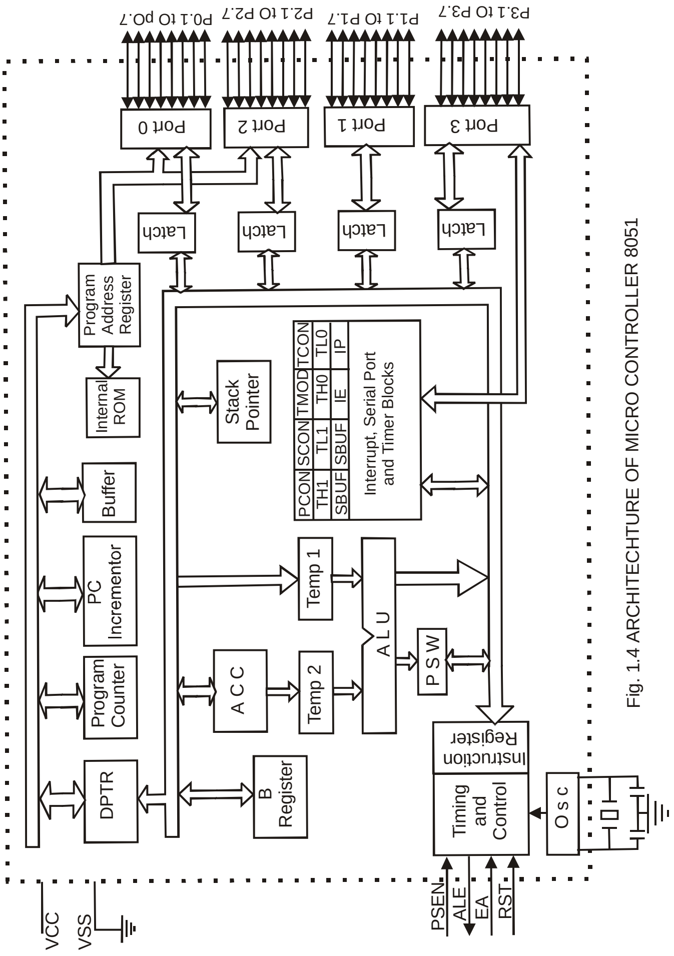


Fig. 4. 1 Architecture of 8051

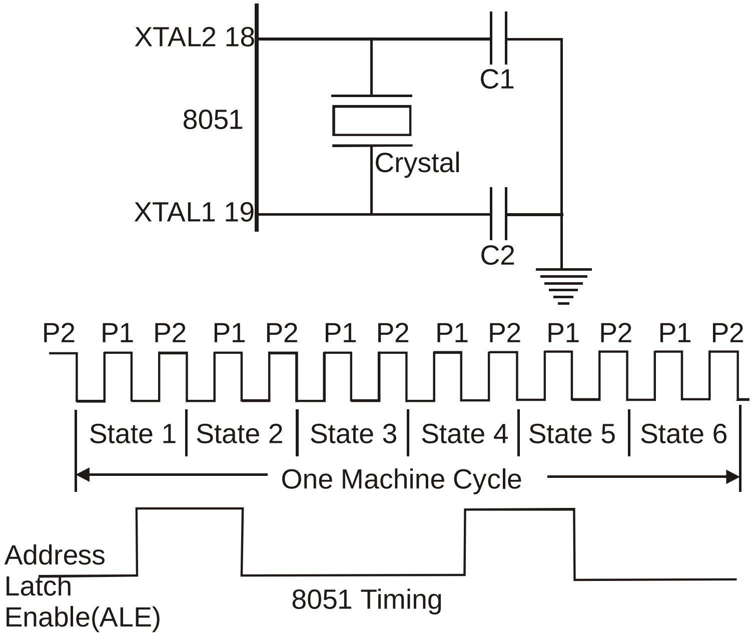


Fig. 4.2 Crystal Oscillator Circuit

Interrupts

There are five interrupts available in 8051, in which three are internal interrupts and two are external interrupts. Three internal interrupts are timer 0(TF0), timer 1(TF1) and serial I/O port (RI or TI) interrupts. Two external interrupts are INT0 and INT1.

Serial Data Communication

The 8051 transfers and receives data serially at many different baud rate (Data transfer of bits per second). In 8051 the Serial communication is in full duplex mode. If the data are transmitted and received simultaneously is called full duplex mode. Two 8-bit SBUF registers are used to have full duplex mode. One SBUF is used for Transmission and another SBUF is used for reception.

4.3 8051 Pin Configuration

The 8051 microcontroller is packaged in 40 pin dual in line package (DIP). Out of 40 pins, 16 pins are single function pins and 24 pins may be used for one or two entirely different function.

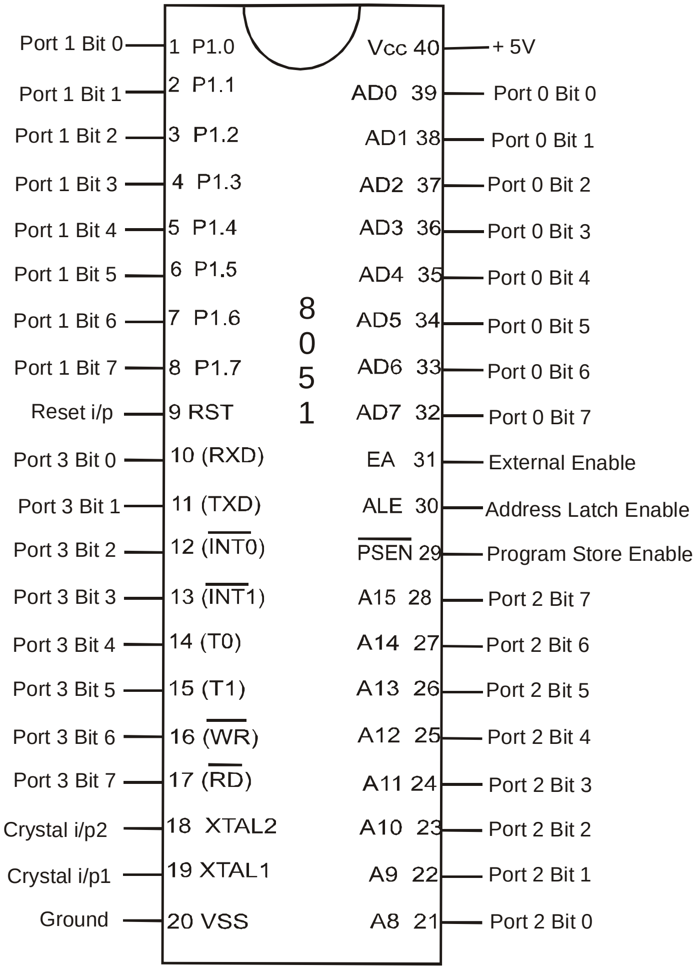


Fig. 4.3 Pin Diagram of 8051

Single Function Pins (16 Pins)

Pin No.	Designation	Function
20	Vss	Supply Pin, Ground
40	Vcc	Supply Pin, +5V
9	RST	Reset Input
18, 19	Xtal1, Xtal2	Crystal Input Pins
29	PSEN	Program Store Enable
30	ALE	Address Latch Enable
31	EA	External Enable
1 to 8	P1.0 to P1.7	I/O pins Port 1

Dual Function Pins (24 Pins)

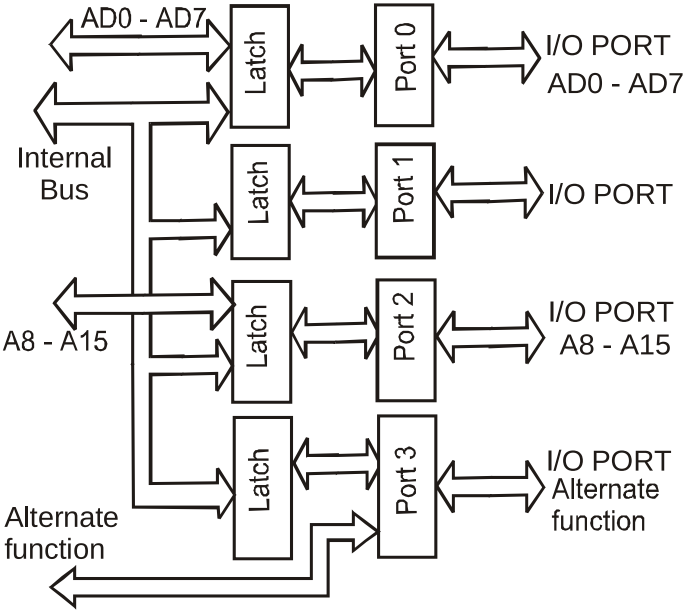


Fig. 4.4 Dual Function Pins

Out of 32 I/O lines of four ports, 8 lines of port 1 are used as single I/O function only. The remaining 24 pins of port 0, port 2 and port 3 are used for dual functions. Port 0 pins used for I/O function and lower address and data multiplexed port for external memory. Port 2 pins used for I/O function and higher address for external memory. Port 3 pins used for I/O functions and Alternative uses.

Vcc

Pin 40 provides supply to the chip. The voltage source is +5V.

GND

Pin 20 is Ground.

Xtal1 and Xtal2

External crystal with two 30PF capacitor is connected to these pins. The crystal frequency determines the basic clock frequency of 8051.

RST

Pin 9 is RESET pin. It is an active high input pin and normally low. Upon applying a high pulse to this pin, 8051 terminate all activities and go to reset condition.

EA

Pin 31 is External enable pin (EA). When this pin is connected to Vcc, the CPU access internal memory. When EA pin connected to ground, the CPU access the program/data from external memory.

PSEN

This is an output pin. PSEN stands for Program Store Enable. An active low pulse in this pin enables the external ROM. This pin is connected to OE of external ROM chip.

ALE

This is an active high output pin. ALE stands for Address Latch Enable. ALE used to demultiplex the lower byte address of external memory and data from port 0. When ALE is high, port 0 is connected to external memory address latch. When ALE is low, port 0 carries the data.

Program Counter (PC)

The Program Counter (PC) is a 16-bit pre-settable up counter used as a memory pointer. The Program Counter (PC) keeps track of program execution. To execute a program, the starting address of the program is loaded in Program Counter (PC). The instruction in an address pointed by the PC is fetched and executed by the CPU. When an instruction is fetched from memory, the Program Counter (PC) is automatically incremented by one to point the next location of memory where next instruction/data is stored.

Data Pointer (DPTR)

Data pointer is a 16-bit register made up of two 8-bit registers. They are Data Pointer Higher order byte (DPh) and Data Pointer Lower byte (DPl) registers. DPTR holds a 16-bit address of external memory. This register is under control of program instructions and can be specified by its 16-bit name (DPTR) of 8-bit name (DPh or DPl). DPTR does not have single internal address, DPh and DPl are assigned separate address.

General Purpose Registers

8051 has 34 general purpose working registers.

Accumulator (A register)

The Accumulator (A register) is used for many operations like Addition, Subtraction, Multiplication, Division and Boolean bit logical operations. It is also used for data transfer between 8051 and external memories, I/O devices.

B Register

The B register is used to do arithmetic operations with A register. B register may be used to store the data and it has no other functions.

Flags register, Program Status Word (PSW)

Flags are single bit register and used to store the result of certain function after executing instruction. Flags are grouped inside PSW and PCON registers. PSW register contains math flags and PCON contains general user flags. Math flags are grouped in PSW and they are Carry(C), Auxiliary Carry (AC), Over flow (OV), and Parity (P). The general user flags are GF0 and GF1 which are grouped in PCON register.

7	6	5	4	3	2	1	0
CY	AC	F0	RS1	RS0	0V	----	P

Program Status Word(PSW)

Fig. 4.5

Bit	Symbol	Function
0	P	Parity Flag
1	--	User Definable Bit
2	OV	Overflow Flag
3	RS0	Register Bank Select Bit 0
4	RS1	Register Bank Select Bit 1
5	F0	User Flag
6	AS	Auxiliary Carry Flag
7	CY	Carry Flag

Carry Flag (CY)

This flag bit is used in arithmetic, jump, rotate and Boolean instructions. This flag is set whenever a carry is obtained.

Auxiliary Carry Flag (AC)

This flag bit is used in BCD arithmetic operations. This flag is set whenever a carry is obtained in bit 4. This flag is considered only for BCD addition, otherwise the status of flag is ignored.

Parity Flag (P)

This parity flag bit is used to show the number of 1s in the accumulator only. If the accumulator register contains an odd number of 1s, then this flag set to 1. If accumulator contains even number of 1s, then this flag cleared to 0.

Overflow Flag (OV)

This flag bit is used to detect errors in signed arithmetic operations. This flag is set whenever the result of signed number operation exceeds 7 bits, causing the higher order bit to overflow into the sign bit.

4.4 Internal Memory Organisation

For the CPU to process information, the data must be stored in RAM or ROM. 8051 has internal 128 bytes RAM and 4Kbytes ROM for these functions. RAM is called data memory and ROM is called program memory. Although both RAM and ROM uses same address, the selection of memory depends upon the instruction used. This type of architecture is called Harvard Architecture.

Internal RAM

8051 has 128-byte internal RAM and called data memory. RAM is a read-write memory and processor can read data or write data to this memory space. These 128 bytes are divided into three groups and they are Register Banks, Bir addressable registers and General purpose. Internal organisation of RAM is given in fig. 1.9

There are four register banks available in 8051 are Bank 0, Bank 1, Bank 2 and Bank 3. By default, or on reset Bank 0 is selected. To select other Banks RS0 and RS1 in PSW is to set accordingly. Unselected register banks are used for general purpose. The address of four register banks occupies from 00h to 1fh.

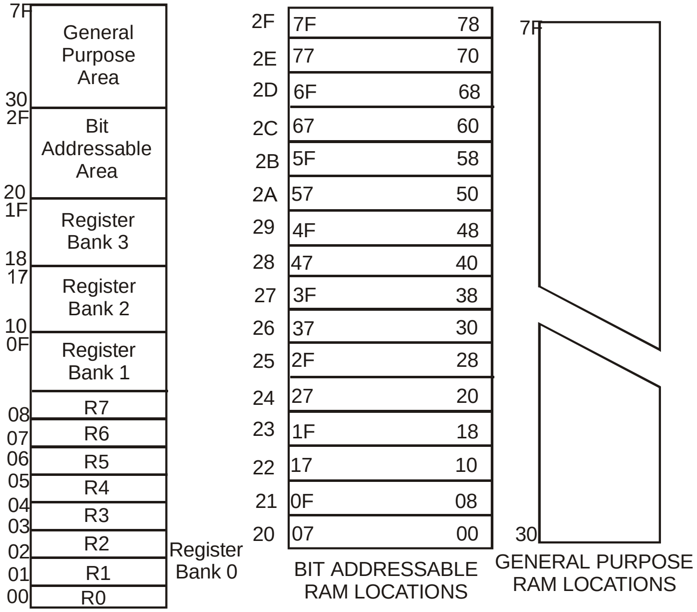


Fig. 4.6 Internal RAM Organisation

Addresses from 20h to 2Fh are bit addressable area. A bit in this area may specified by its bit address. Addressable bits are useful when the program need to set or reset a single function (On or OFF).

Address from 30h to FFh are used as general-purpose area. Locations in this area can be addressed byte wise.

Internal ROM

8051 has 4Kbyte of ROM. Program memory (ROM) is read only memory. The CPU reads program from this memory and execute it. CPU cannot write data inside this memory. Address of internal ROM is from 000h to FFFh. Above these address the CPU will access external memory.

4.5 The Stack and Stack Pointer

The stack is a section of internal RAM used by the CPU to store information temporarily. This information could be data or an address. The CPU needs this storage area since there are only limited number of registers. The register to access the stack is called Stack Pointer. It is an 8-bit register and it can point an address location between 00h to FFh. Stack pointer is used to hold the address of Top of the Stack. The address held in the SP is the location in internal RAM where the last byte of data was stored by a stack operation.

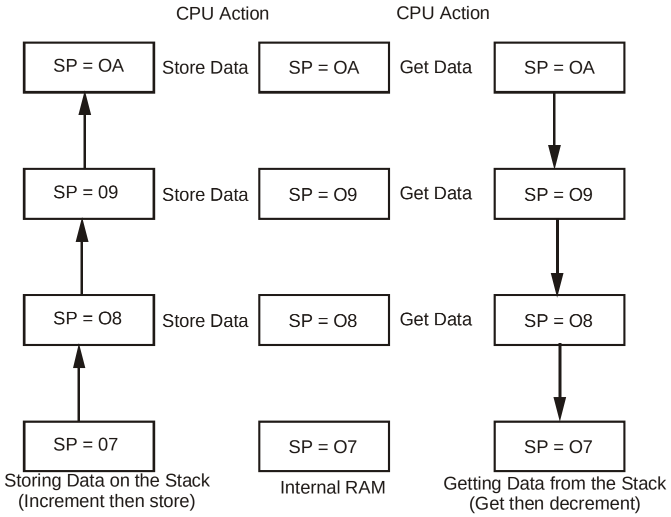


Fig. 4.7 Stack Operation

When data is to be placed on the stack, the SP increments one before storing the data on the stack. When the data is retrieved from the stack, the SP decrements one after retrieving the data. Detailed stack operation shown in fig. 4.7

4.6 Special Function Registers (SFR)

There are 21 Special Function Registers used in 8051. Addresses of these SFRs are between 80h and FFh. Not all addresses from 80h to FFh are not used. The unused addresses are reserved for future use.

Out of 21 Special function registers, eleven are bit addressable registers and ten are byte addressable registers. Bit addressable SFRs are TCON, SCON, IE, IP, PSW, ACC, B, P0, P1, P2 and P3. Byte addressable registers are SP, DPh, DPl, PCON, TMOD, TL0, TH0, TL1, TH1 and SBUF.

Special function register can be either accessed by its name or by its address. The addresses of the Special function registers are shown in fig.1.8

Accumulator	0E0
B Register	0F0
DPH	83
DPL	82
IE	0A8
IP	0B8
P0 (Port latch)	80
P1 (Port latch)	90
P2 (Port latch)	A0
P3 (Port latch)	0B0
PCON	87
PSW	0D0
SCON	98
SBUF	99
Stack Pointer	81
TMOD	89
TCON	88
TL0	8A
TH0	8C
TL1	8B
TH1	8D

Fig. 4.8 Special Function Register

4.7 External Memory

In many systems the internal ROM and RAM is not sufficient. In this case external RAM and ROM may be used. Since the Program Counter (PC) and Data Pointer (DP) are 16-bit, maximum of 64Kbytes of external memory can be added to 8051 system.

The CPU of 8051 address the external memory on two occasions. Firstly, when the address bus exceeds FFFh and secondly the pin EA is grounded. When pin EA is grounded internal memory is not addressed. Since same address is used for both external RAM and ROM the selection of RAM or ROM is done by instruction method. The control bit PSEN goes low to fetch program from external ROM. Data cannot be written in this memory. Control bits RD and WR goes low to enable external RAM. RD signal used to read the data from RAM and WR used to write data from RAM.

4.8 Input/Output Pins and Ports

The Input / Output sections of 8051 includes four I/O ports. The four ports are port 0, port 1, port 2 and port 3. Each port is 8-bit port having 8 pins and has D-type o/p latch for each pin. The Special Function Register (SFR) for each port is made up of these eight latches, which can be address for that port. The port latches should not be confused with the port pins. The data in the latches does not have to be the same as that on port pins. The data in latch buffer and the data of ports pins can be read independently. To make any port as input or output it must be programmed. To make a port as input port, the port latch should be written with all 1s. By default, the port is defined as output port.

I/O operation of ports (P0, P1, P2, and P3)

I/O operation of all ports i.e. port 0, port1, port 2 and port 3 are same. Except port 1 all other ports are having alternate function. Each port figures shows the operation of both I/O operation and alternate function also. Common figure of all port I/O function are shown in fig. 1.9

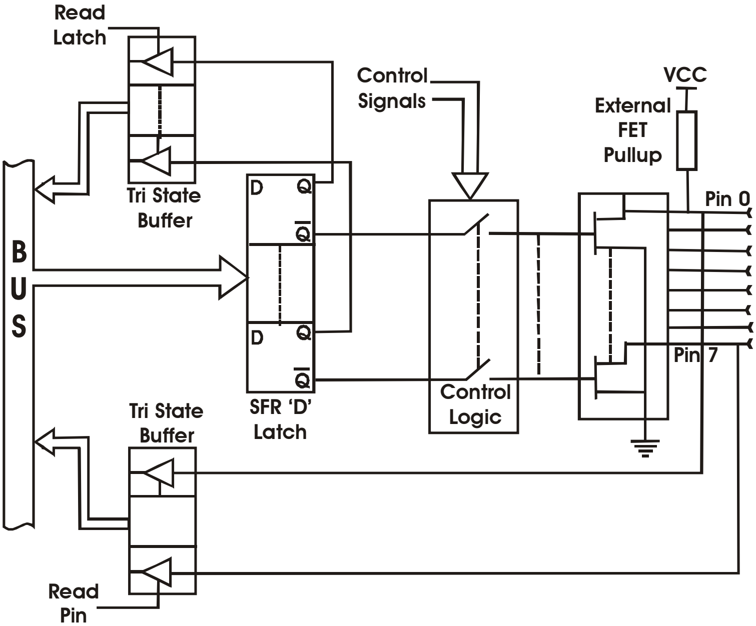


Fig. 4.9 I/O operation of 8051 Ports

Output Operation

Except port 0 other ports have pull up resistors. Pull up resistor are connected to Vcc to make the pin high. To make port 0 as output port, external pull up resistors of 10K are to be connected to port pins. Each port has eight D type output latch for each pin. These eight latches are called as port SFR latch. The data to be sent out is loaded into port SFR latch buffer. Where ever 0s are loaded in latch, the high output from Q drives the concerned FET drivers through control circuit and makes the corresponding pins low. Where ever 1s are loaded in latch, output of Q is low and the control circuit cut off the FET driver. The port pins are made 1 by external pull up resistors. Thus, the 0s and 1s written in port latch are transferred to port pins.

Note: It may be noted that the data in latch buffer and port pins may not be the same always. There are two sets of tri-state buffers used to read the latch or pin. When upper set of tri-state buffer is enabled by read latch signal, the latch data are transferred to internal bus. When the lower set of tri-state buffer is enabled by read pin data signal, the data of pin are transferred to internal bus.

Input Operation

When a port to be used as an input port, all 8-bits of port latch are to be programmed with 1s. If a pin is to be set as input pin then corresponding latch bit is to be set to1. When latch is set to 1, the output of Q is low and FET drivers are in cut-off state. The pin is in float condition. The data in port pins are connected to input buffers. When control signal read pin data goes high, the contents of port are transferred to internal bus.

Port 0

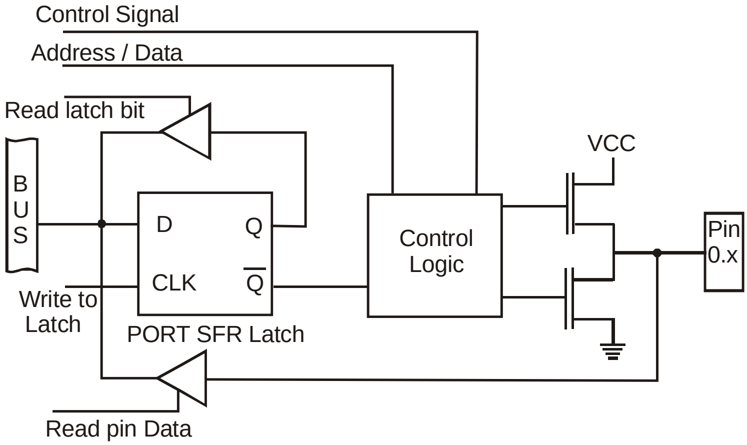


Fig.4.10 I/O operation of Port 0

Port 0 has dual function as I/O port and Address/Data bus for external memory. When external memory is not used, port 0 is used as simple I/O port. When external memory is used, port 0 is used as address bus and data bus. Lower byte address and data are multiplexed in this port. ALE signal is used to demultiplex the address from this port. ALE goes high, when this port contains address. After addressing ALE goes low and the port become data bus. To read the data from external memory logic 1 is written to all port SFR latches.

Port 1

Port 1 is not dual function port as port 0. It is used only for simple I/O operation of data. So, control circuit is not used and the output of SFR latch is directly connected to FET drivers. The Pin configuration diagram of port 1 is shown in fig. 1.11

It has internal pull up resistors to drive the output device and no need to connect external pull up resistors like port 0. The data to be send out is loaded to the SFR latch of port1.

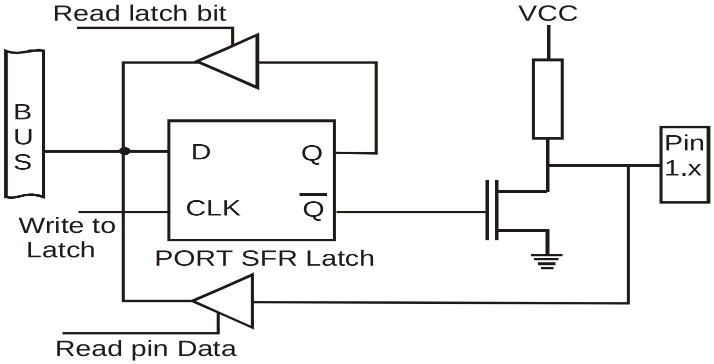


Fig.4.11 Pin Configuration of port 1

Where ever thedata are 0 the Q output of latch is high and drives the FET driver. The output pin goes low. Where ever data are 1 the Q output of latch is low and the FET driver is in cut-off condition. The pull up resistor make the output pin high.

To program the port 1 as input port, the SFR latch buffer should be set to 1. The low output from the Q of latch isolates the FET drivers and pull up resistors. The pins of port 1 are connected to input buffer. When read pin signal enabled, data available in port pins are transferred to internal bus.

Port 2

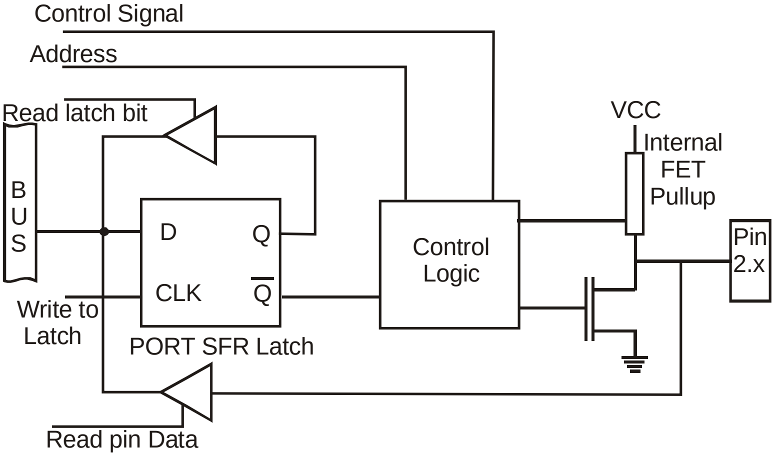


Fig. 4.12 Pin Configuration of Port 2

Port 2 is similar to port 0 having dual function. In addition to I/O operation it can address external memory also. When 64K external memory is connected, 16-bit address is required. Lower byte address is carried by port 0 and higher byte address is carried by port 2. Since data is 8-bit, only address is carried in this port. I/O operation of this port is like port 0. By default, this port is an output port. To make this port as input port the port SFR latch buffer should be set to all 1s.

Port 3

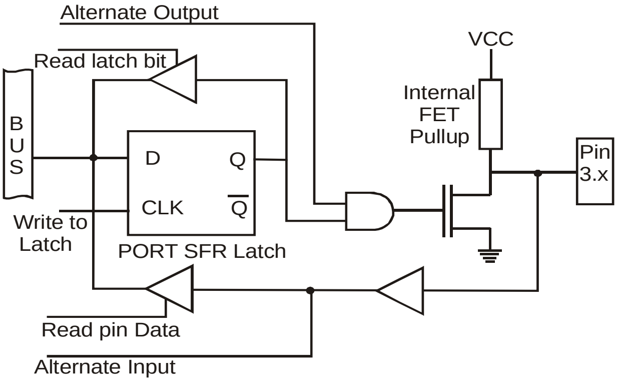


Fig. 4.13 Pin Configuration of Port 3

Port 3 is also having dual function. In addition to I/O operation it has alternate function also. I/O operation of this port is like other ports. Each pin of port 3 may be individually programmed either I/O pin or alternate function. The alternate uses of port 3 is as follows,

Pin no.	Pin name	Alternate function	SFR register
P3.0	RXD	Serial input data	SBUF(RD)
P3.1	TXD	Serial output data	SBUF(WR)
P3.2	INT0	External Interrupt 0	TCON.1
P3.3	INT1	External Interrupt 1	TCON.3
P3.4	T0	External Timer 0 clock	TMOD
P3.5	T1	External Timer 1 clock	TMOD
P3.6	WR	External RAM write enable	----
P3.7	RD	External RAM read enable	----

4.9 Addressing Modes

The CPU of 8051 can access data in various ways. The data could be in a register, or in memory, or to be provided as an immediate value. These various ways of accessing data are called addressing modes.

The 8051 provides a total of five distinct addressing modes. They are as follows:

1. Immediate addressing mode
2. Register addressing mode
3. Direct addressing mode
4. Register indirect addressing mode
5. Indexed addressing mode

Immediate addressing mode

In immediate addressing mode the source operand is part of the instruction. The data stored in the immediate next location where the op-code is stored is transferred to destination. The data must be preceded by the # sign. This addressing mode can be used to load information into any of the register, including the DPTR register.

Simple Example

```
i)   MOV        A, #35h; Load the 8-bit hexadecimal number 35
                ; intoAccumulator (A Register)
ii)  MOV        R4, #25h; Load the 8-bit hexadecimal number 25
                 ; into register R4
```

Note

In the above instruction, 25h is loaded into register R4. We do have four register banks in the internal RAM memory of 8051, starting from Bank 0 to Bank 3. Each bank consists of 8 register namely R0 - R7. But the default register bank is bank 0. Hence the 8-bit hex number 25h is loaded into register R4 of Bank 0.

```
iii) MOV    B, #40h; Load the 8-bit hex. number 40h into
            ; B register
iv)  MOV    DPTR, #4521h; Load the 16-bit hex. number
             ; 4521 into Data Pointer Register (DPTR)
```

Although the DPTR register is a 16-bit register, it can also be accessed as two 8-bit registers namely DPh and DPl.

DPL is the low byte register and DPH is the high byte register.

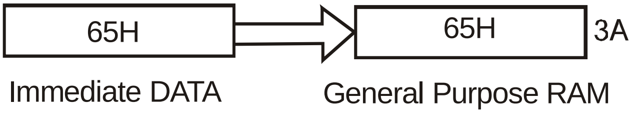
Hence 4521h is loaded into DPTR register as DPL = 21h and DPh = 45h

DPH(8 BIT)	DPL(8 bit)
45H	21H

DPTR Register (16 Bit)

The above instruction can also be expressed as

```
MOV    DPL,    #21h
MOV    DPH,    #45h
MOV    3Ah,    #65h; The 8-bit data 65h is loaded into
               ; RAM location 3Ah.
```



In the above instruction the data is send immediate to the RAM location having address as 3Ah.

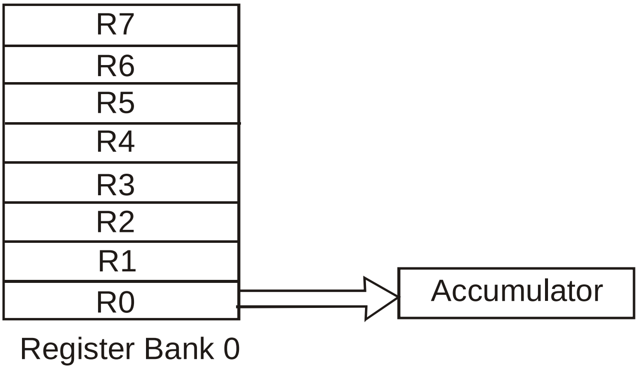
```
MOV    P1,     #55h    ; Load the 8-bit hex number 55
               ; into port P1.
```

Register addressing mode

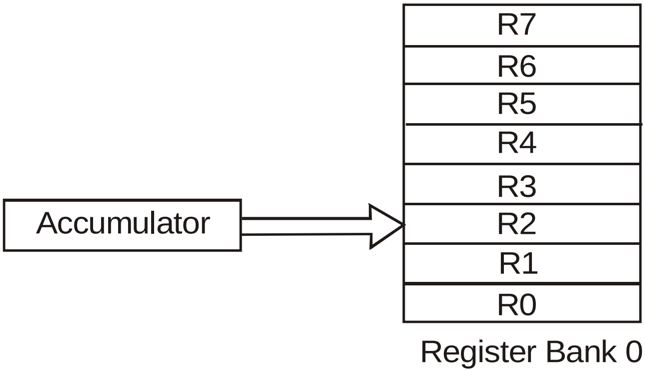
In register addressing mode data transfer takes place between a register to a destination. The destination can be a register or any RAM memory location.

Simple Example

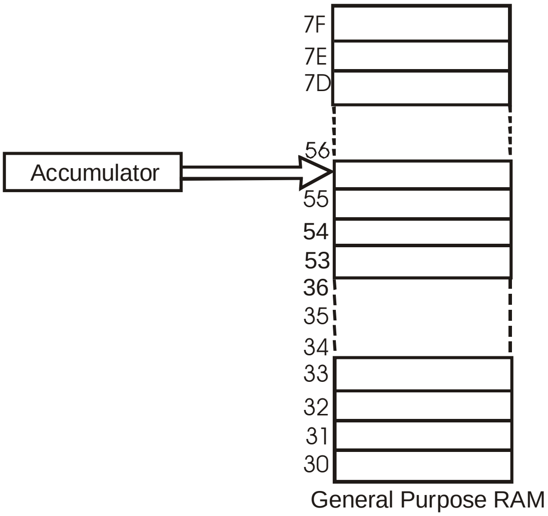
```
    MOV      A,      R0      ; Copy the contents of register R0
                             ; into Accumulator.
```



```
MOV   R2,    A       ; Copy the contents of Accumulator
                     ; (A register) into register R2.
```



```
MOV   56h,   A       ; Copy the contents of the A register
                                 ; in to RAM location 56 h
```



In register addressing mode, Data can only be moved between register of same size.

Example

MOV DPTR, A is invalid. Because DPTR is a16 bit register and Accumulator is an8-bit register. Hence the source and destination registers must match in size. But this instruction,

```
MOV     DPTR,  #3542h
MOV     R7,    DPL
MOV     R6,    DPH    are valid.
```

In register addressing mode, data can only be moved between accumulator and register Rn (where n = 0 to 7). But the movement of data between Rn register is not allowed.

For Example

MOV R4, R7 is invalid.

In the immediate addressing mode and register addressing mode, the operands are either tagged along with the instruction itself or inside one of the registers.

Direct Addressing Mode

Direct addressing mode is used to access the data stored either in RAM or in register of the 8051 by specifying its address.

In this mode, the data is in a RAM location whose address is known, and this address is given as a part of the instruction.

Simple Example

Accessing the contents of Register Bank

The registers R0 to R7 of register bank 0 can be accessed in direct addressing mode as,

```
        MOV    A, 04h          ; Copy the contents of RAM
                                                ; location 04h (R4 of register
bank                                                          ; 0) in to
accumulator
        MOV    A, 02h          ; Copy the contents of RAM
                                                ; location 02h (R2 of register
bank                                                          ; 0) into
accumulator
```

Accessing the contents of general-purpose RAM

The memory locations of general-purpose RAM ranges from 30h - 7Fh. Any data present in these locations can be accessed as

```
        MOV    R2, 40h         ; Copy the contents of RAM
                                                ; location 40h into register R2.
        MOV    R0, 7Fh         ; Move the contents of RAM
                                                ; location 7Fh into register R0.
```

Accessing the contents of special function register

The special function register has addresses between 80h and FFh. The data in any of the special function register can be accessed by specifying its address.

Example

```
        MOV    A, 8Ah          ; Copy the contents of TL0(SFR)
                                                ; into Accumulator
        8AH -  Address of TL0(Special function register)
```

Register indirect addressing mode

In this type of addressing mode, a register is used as a pointer to the data. Only the registers R0 and R1 are used for this purpose. The registers R2 to R7 cannot be used in this type of addressing mode.

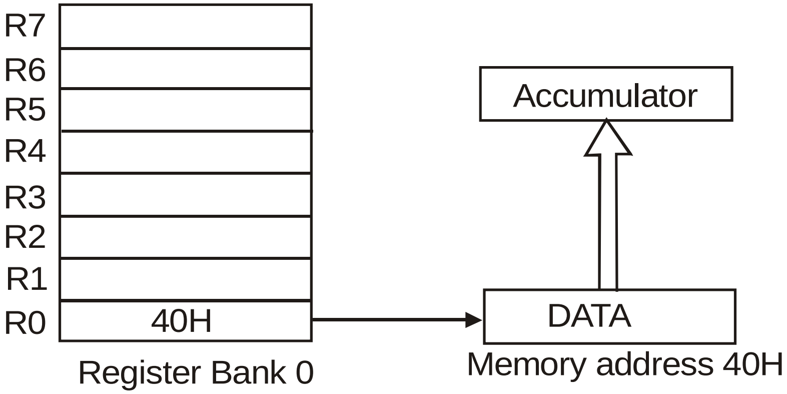
The registers R0 and R1 holds the data addresses of RAM locations from where the data is to be moved. These registers should be preceded by @ sign.

Simple Examples

MOV A, @R0 ; Copy the contents of RAM

; location whose address is held by

; register R0 into Accumulator.



ii) MOV B, @R1 ; Copy the contents of RAM

; location whose address is held by

; register R1 in to B Register.

Advantages

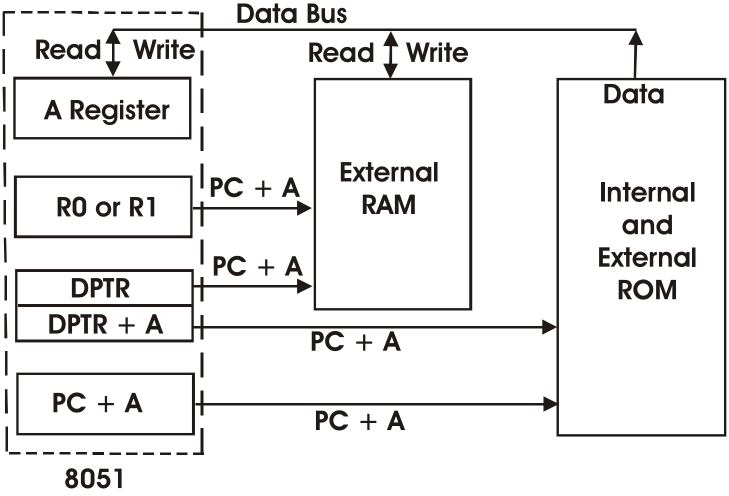
Looping is possible in register indirect addressing mode.

Limitations

The registers R0 and R1 which are used in this mode are 8 bits wide, therefore their use is limited to access any information in internal RAM (Scratch pad memory 30h to 7Fh or special function register). The data in external RAM or in code space of on-chip ROM cannot be accessed.

4.10 External data moves (Indexed addressing mode)

Indexed addressing mode can be used to access data from external memory (i.e. external RAM or ROM). Normally we use a MOV instruction to access data from internal RAM. The external data moves (i.e. from external RAM or from external ROM) uses MOVX and MOVC instructions.



Accessing data (or) moving data from external RAM

The data moves from external RAM uses MOVX instruction. All the external data move must involve the Accumulator, or A register. Only the registers R0 and R1 are used to access data from external RAM addresses ranging from 00h to FFh. For accessing data from higher order addresses ranging from 0000h to FFFFh, data pointer register (DPTR) can be used.

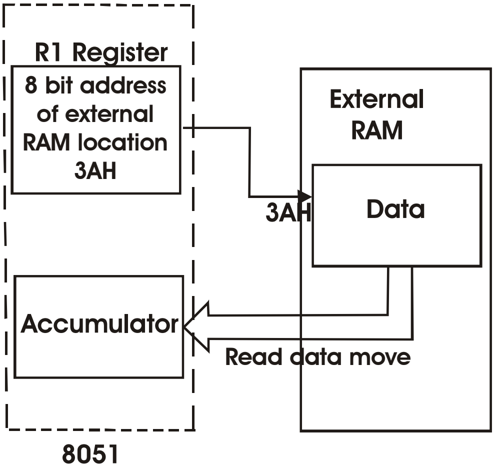
Simple Example

MOVX A, @R1; Copy the contents of data from external

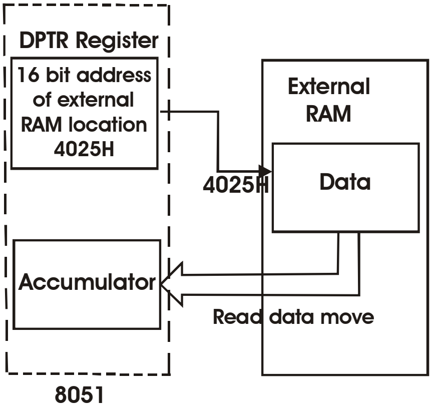
; RAM location whose 8-bit address is

held ; in register R1 to

A Register.

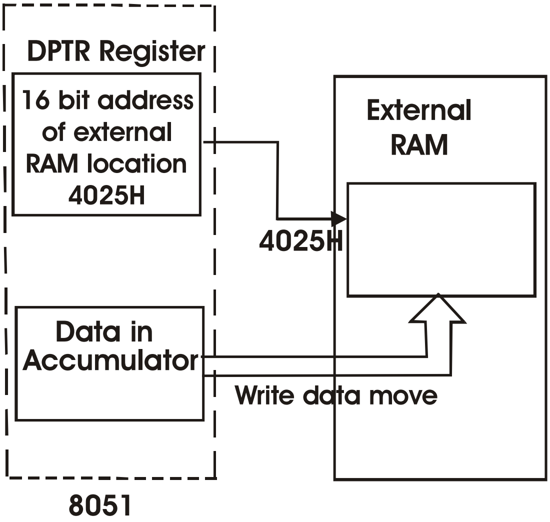


MOVX A, @DPTR ; Copy the contents of data from ; external RAM location whose ; 16-bit address is held in DPTR to ; Accumulator

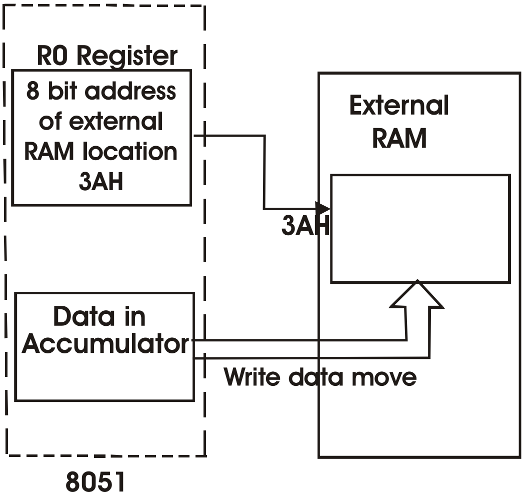


Both the above instructions are referred to as read Data moves i.e., data from external RAM is moved to accumulator of 8051.

MOVX @DPTR, A ; Copy the contents A register to ; external RAM location whose ; address is stored in Data pointer ; register DPTR.



```
MOVX @R0, A        ; Copy the contents of Accumulator to
                   ; external RAM location whose address
                   ; is stored in register R0.
```



Both the above instructions are referred to as write data moves. i.e., the data from accumulator of 8051 is moved to external RAM.

4.11 Code memory read only data moves

Data move from external ROM to accumulator (A Register) uses MOVC instruction. The address of the ROM location from where the data has to be read is given by the instruction, @ A+DPTR, @ A + PC. MOVC instruction is normally used with internal or external ROM and can address 4K of internal or 64K of external ROM.

Accessing data from external ROM

Use of data pointer DPTR in read only data move

Instruction MOVC A, @A + DPTR

DPTR - Data pointer holds a 16-bit address or a 16-bit data

Accumulator - Holds an8-bit data

@ A + DPTR - Addition of 16 bits (address or data) in DPTR and 8-bit data in Accumulator results in another 16-bit address. This newly formed 16-bit number is the address of the external ROM location from where the data is to be read.

Example

```
MOV     DPTR, #2234h            ; DPTR = 2234h
MOV     A, #66h                 ; A = 66h
MOVC A, @A + DPTR               ; A = 2234h + 66h
                                            : = 229Ah
```

The contents of ROM location 229Ah is moved to Accumulator.

Use of PC (Program Counter) in read only data move.

Instruction

MOVC A, @A + PC

PC - Program counter holds a 16-bit address. The program counter is incremented by 1 before it is added to Accumulator's content.

A - Holds an8-bit data

@ A + PC - Addition of 16-bit address in the program counter incremented by 1- and 8-bit data in Accumulator results in another 16-bit address. This newly formed 16-bit address is the address of the external ROM location from where the data is to be read.

Example

```
MOV      A,        #60h         ; A = 60h
MOVC   A, @A + PC
```

IF the program counter holds an address 4000H then first it is incremented by 1 before it is added with the 8-bit data in accumulator.

i.e., PC = 4000h + 1 = 4001h

@ A + PC - 4001h + 60h = 4061h

The contents of the external ROM location 4061h is moved to Accumulator.

4.12 Stack in 8051

The stack is a section of RAM used by the CPU to store information temporarily. This information could be Data or address. The CPU needs this storage area since there are only a limited number of registers.

Accessing of stack in 8051

The register used to access the stack is called the stack pointer register. The SP register is only 8-bit wide, which means that it can take values of 00h to FFh. When 8051 is powered up, the SP register contains value 07. This means that RAM location 08 is the first location being used for the stack by the 8051. The storing of a CPU register in the stack is called a PUSH. The retrieving of the contents of the stack back into a CPU register is called POP.

In 8051, RAM locations 08H to 1FH can be used for stack. The locations 20H - 2FH of RAM are reserved for bit addressable registers and must not be used by the stack. That means the stack in 8051 has only 24 bytes, but by using a special instruction,

```
MOV         SP, #xx (xx Ranges from 30h - 7Fh)
```

i.e., the RAM locations from 30h to 7Fh can be also used as stack.

Pushing on to the stack

A PUSH instruction copies data from any of the registers to the stack. The stack pointer is incremented by 1 before the data is copied to the stack.

Popping of the stack

A POP instruction copies data from the stack location to any of the registers in the CPU of 8051. The stack pointer is decremented by 1 after the data is copied to any of the registers.

Example

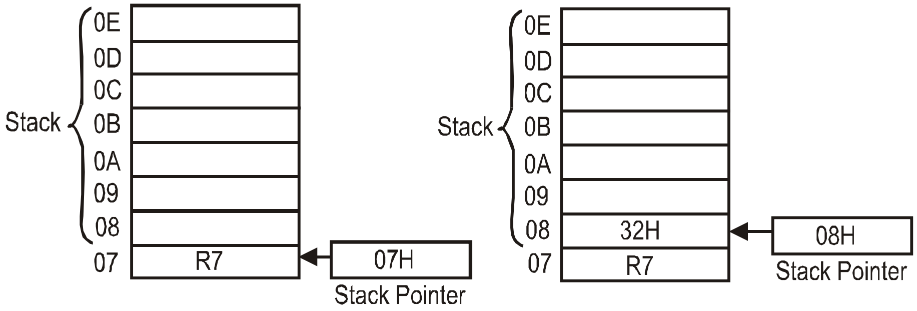
```
MOV         R6, #32h
MOV         R1, #25h
MOV         R4, #0A2h
PUSH 6
PUSH 1
PUSH 4
```

When 8051 is powered up the stack pointer (SP) points to the location 07h (SP = 07h).

PUSH 6 ; Push the contents of register R6 to stack. Whenever 8051 executes a PUSH instruction the first step involved is the stack pointer (SP) incremented by 1. Therefore, stack pointer becomes 08H. The 8-bit data 32H is stored in memory location 08H.

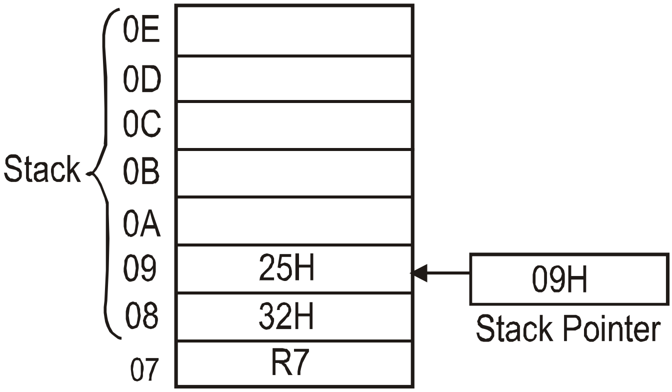
Before PUSH operation

After PUSH 6 operation



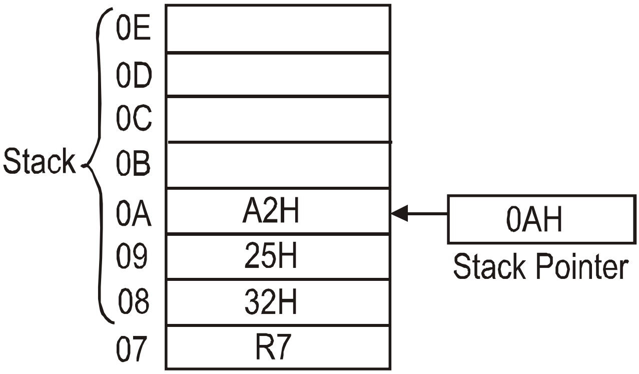
PUSH 1 ; Push the contents of register R1 to stack. Before pushing it to the stack, Stack pointer (SP) incremented by 1 and becomes 09h. The 8-bit data 25h is stored in stack location 09h.

After PUSH 1 Operation



PUSH 4; Push the contents of register R4 to stack. Before pushing it to the stack, Stack pointer (SP) incremented by 1 and becomes 0Ah. The 8-bit data A2h is stored in stack location 0Ah.

After PUSH 4 Operation



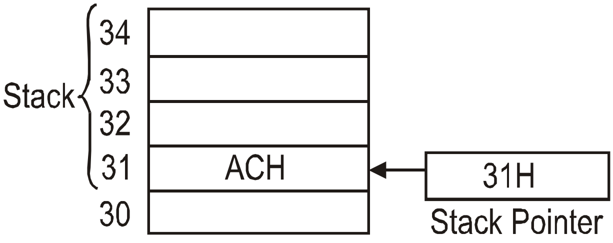
```
MOV 81h, #30h
MOV     R0, #0ACh
PUSH    00h
PUSH    00h
POP     01h
POP     80h
```

First instruction MOV 81H, #30H loads the 8-bit hex. number 30h into stack pointer (Address of stack pointer is 81H). i.e., the stack pointer is loaded with 30h. By this instruction addresses above 30h are used as stack.

MOV R0, # 0ACh ; Load the data Ache into register R0

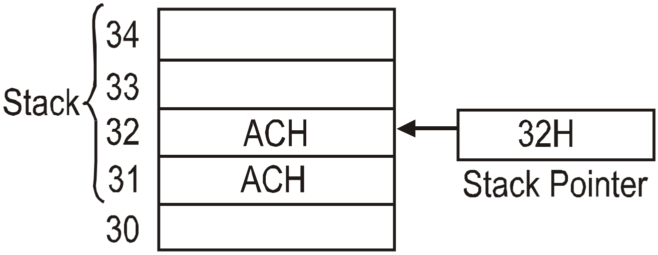
PUSH 00h ; Push the contents of R0(address of R0 is 00H) into stack. First the stack pointer is incremented by 1 and becomes 31h. After incrementing by 1, the contents of register R0 i.e., ACH is moved to stack location 31h.

After Push 00h



The next instruction also PUSH 00h. When this instruction executed, the stack pointer incremented by one and become 32h. The Contents of register R0 i.e. ACh is move to stack location 32h.

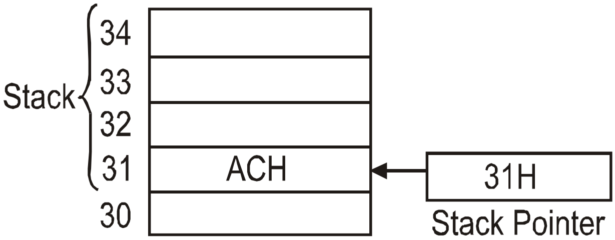
After PUSH 00h



POP 01h ; Whenever 8051 executes POP instruction, the contents from the stack location where the Stack Pointer points (Stack Pointer points to the top of the stack) is moved. i.e., Data ACh in stack location 32h is moved to R1(01h is address of register R1).

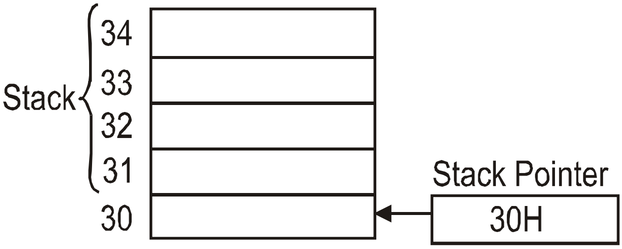
R1 = ACh. After moving the data, the stack pointer decremented by 1 and becomes 31h.

After POP 01h



POP 80h ; Stack pointer points to the location 31h. The contents of this location are moved to Port0 (80H is memory address of Port0) Port 0 = ACh. After moving the data ACh, the stack pointer decremented by 1 and become 30h.

After POP 80h.



Simple program containing both PUSH and POP instructions.

```
MOV        R0, #66h        ; R0 = 66h
```

```
    MOV     R3, #7Fh        ; R3 = 7Fh
    MOV     R7, #5Dh        ; R7 = 5Dh
    PUSH    0               ; SP = 07 + 1 = 08, Data 66h in R0 moved
                                        ; to Stack location 08H. Stack 08h
= 66h
    PUSH    3,              ; SP = 08 + 1 = 09, Data 7Fh in R3 moved
                                        ; to stack location 09h. Stack 09h
= 7Fh
    PUSH    7,              ; SP = 09 + 1 = 0A, Data 5Dh in R7 moved
                                        ; to stack location 0Ah.

    POP     0,              ; SP = 0Ah. Data 5Dh in stack 0Ah is
                                        ; moved to register R0. SP = 0A -
1 = 09h                                           ; and R0 = 5Dh.
    POP     7,              ; SP = 09h. Data 7Fh in stack 09h is moved to
                                        ; register R7. SP = 09 - 1 = 08h
and R7 = 7Fh.
    POP     3,              ; SP = 08h. Data 66h in stack 08h is moved to
                                        ; register R3. SP = 08 - 1 = 07h.
```

Before executing the program

R0 = 66h
R3 = 7Fh
R7 = 5Dh

After executing the program

R0 = 5d
R3 = 66h
R7 = 7Fh

4.13 Data Exchanges

There is some specific instruction set in 8051which facilitates the exchange of data between the Accumulator (Register A) and any other register or any other memory location. These instructions can only exchange the data internal to 8051 and not externally.

Instruction

```
    XCH     A, Source Byte  ; The source byte can be any
                                        ; register or internal RAM
location.
```

Examples

XCH A, R3 ; Exchanges the contents of R3 register with ; Accumulator

XCH A, 40h ; Exchanges the data in memory location 40H ; with Accumulator

XCH A, @R0 ; Exchanges the contents of a memory location

whose address stored in register R0 with Accumulator

Illustrative Example

```
MOV A, #65h     ; A = 65h
MOV R2, #97h    ; R2 = 97h
XCH A, R2   ; Now A= 97h and R2 = 65h
```

Instruction

XCHD A, @Rn

The XCHD instruction exchanges only the lower nibble of A with the lower nibble of the data present in RAM location pointed by Rn while leaving the upper nibble in both places intact.

A nibble consists of 4 bits. Normally a Byte is represented by 8 bits or 2 nibbles as lower nibble and higher nibble.

```
MOV     40H, #97h; RAM location 40h = 1001 0111(97h)
MOV     A, #12h; Accumulator = 0001 0010(12h)
MOV R1, #40h; R1 = 40h (RAM Location)
XCHD A, @R1
```

R1 points the memory location 40H where the data is stored. The lower nibble in memory location 40H is exchanged with the lower nibble of Accumulator. i.e., before the program executed

40H	=	(97h) 1001 0111
Accumulator	=	(12h) 0001 0010

After the execution of program,

40H	=	(92h) 1001 0010
Accumulator	=	(17h) 0001 0111

4.14 Interrupts and Returns

An interrupt is a hardware or program generated call. Just as an ACALL and LCALL instructions are used to access the subroutine, similarly an interrupt sets interrupt flag. This interrupt flag calls a subroutine programmed in memory location called interrupt vector. There are five interrupt flags and their corresponding interrupt vector addresses are as follows;

Interrupt Flags	Flag Name	Addresses of interrupt subroutine vectors.
IE0	External interrupt 0	0003h
TF0	Timer 0 overflow flag	000Bh
IE1	External interrupt 1	0013h
TF1	Timer 1 overflow flag	001Bh
R1 or T1	Serial Interrupt flag	0023h

The interrupt subroutine is placed at these memory locations. Whenever an interrupt flag is set to 1, the Program Counter (PC) suspends the main program and jumps automatically to the interrupt subroutine vector address without separate subroutine call. The last instruction of an interrupt subroutine is RETI. When the interrupt subroutine is executed the RETI instruction clears the interrupt flag to 0 and resumes the main program where the interrupt occurs.

Interrupt enabled subroutine has the following events:

i) Internal generated or external hardware generated interrupt signal set the interrupt flags to 1.

ii) When interrupt flag set to 1 the processor suspends the main program and stores the next address of main program to stack.

iii) The program Counter jumps to interrupt subroutine vector and activates the interrupt subroutine.

iv) After execution of interrupt subroutine, the RETI instruction reset the interrupt flag to 0, which was set by the interrupt signal.

v) The processor resumes the main program where the interrupt suspends the program. The address is taken from the stack.

Sample Program

```
SJMP      OVER
.ORG      0003h  ; IE 0 interrupt vector
LJMP      XINT   ; Jump to label of XINT
.ORG      000Bh  ; TF0 interrupt vector
LJMP      TINT   ; Jump to label TINT
OVER:     MOV    SP,     #40h   ; Staring address of stack is 40h
- - - - - - -
- - - - - - -
XINT:     - - - - - - -        ; IE 0 Interrupt Subroutine
          - - - - - - -
```

```
RETI
TINT:       - - - - - - -        ; TF 0 Interrupt Subroutine
            - - - - - - -
RETI
END
```

Assembler directives of 8051

Some widely used 8051 directives are as followers:

ORG (origin)

The ORG directive is used to indicate the beginning of the address. The number that comes after ORG can be either in hex or in decimal. Some assemblers use " .ORG " instead of "ORG" for the origin directive.

EQU(Equate)

This is used to define a constant without occupying a memory location. The EQU directive does not set aside storage for a data item but associates a constant with a data label so that when the label appears in the program, its constant value will be substituted for the label.

For example

```
count EQU 25
- - - - - - -
MOV R3, # count
```

whenever the label count appears in the program segment it is replaced by a constant value 25.

End

This indicates the assembler the end of the source (ASM) file. The end directive is the last line of an 8051 program, meaning that in the source code anything the end directive is ignored by the assembler. Some assembler used " END" instead of " END".

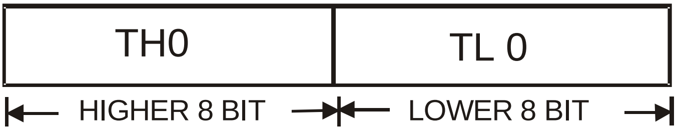
4.15 Timer / Counter

In earlier days the control circuit of electrical machines used to have pneumatic timers. These pneumatic timers are two types as ON delay timer and OFF delay timer. Present days electronic timers are replaced pneumatic timers. Electronic timers are more accurate, reliable and efficient than pneumatic timers. PLC based control circuit, Microprocessor based control circuit and

Microcontroller based control circuits are using electronic timers. Microcontroller 8051 based control circuit is much cheaper and easy to handle than PLC based control circuit and Microprocessor based control circuit.

Microcontroller 8051 has two timer/counters namely timer 0 and timer 1. Size of each timer is 16 bits wide. Since the architecture of 8051 is 8 bits, the timers are separated into two 8bit registers. The timer 0 is divided into low byte of timer 0 (TL0) and high byte of timer 0 (TH0). Same way timer 1 as TL1 and TH1.

Timer 0 divided as



Timer 1 divided as

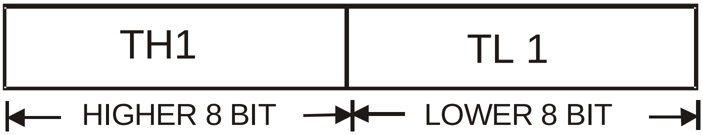


Fig 4.14 Timer Registers

The timer registers TL0, TL1, TH0 and TH1 can be addressed by its location like any other registers A, B, Rn etc. The addresses of timer registers TL0, TH0, TL1, TH1 and Special Function Register related to timer TMOD and TCON are given in fig. 1.15.

TH1	8DH
TH0	8CH
TL1	8BH
TL0	8AH
TMOD	89H
TCON	88H

TIMER SPECIAL FUNCTION REGISTER
AND ADDRESSES

Fig. 4.15 Timer SFRS and Addresses

Each timer clock or by external clock. When internal clock is used, it is called Timer and when external clock is used it is called as Counter.

Timer / Counter Operation

The control logic diagram of Timer shown in figure.Timer 0 and Timer 1 are presettable and any 16-bit can be loaded. If the timer is loaded with any 16-bit, then the timer will start counting from that 16-bit to FFFFh. The 16-bit should be loaded in to two bytes as higher byte and lower byte. The lower byte should be loaded into TL0/1and higher byte to TH0/1. For example, to load 3820h in to timer, 20h should be loaded into TL0/1 and 38h should be loaded into TH0/1.

```
MOV      TL0,   #20H   ;20H loaded to TL0 of timer 0
MOV      TH0,   #38H   ;38H loaded to TH0 of timer 0
```

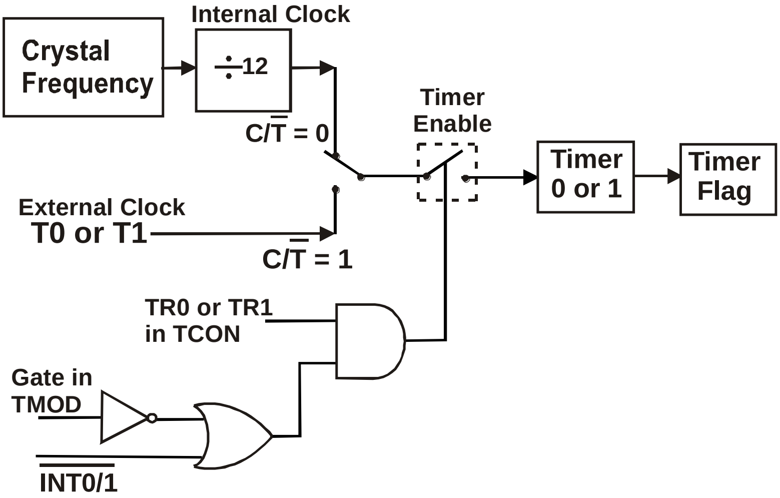


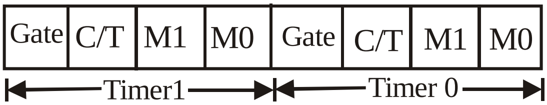
Fig. 4.16 Logic diagram of Timer

Timer can be addressed by its location also. The above instruction can be written as follows,

```
MOV      8AH,   #20H   ;8AH is address of TL0
MOV      8CH,   #38H   ;8CH is address of TH0
```

When timer 0 enabled it will start counting from 2038h to FFFFh. Both timers are controlled by two special function registers TMOD and TCON registers. The TMOD register is a byte addressable and only used to set the modes of both timers. TCON register has control bits and over flow flags for both timers.

TMOD (Timer Mode) Special Function Register



Gate	C/T	M1	M0	Gate	C/T	M1	M0
Timer1				Timer 0			

Fig.4.17 TMOD Special Function Register

The lower nibble of TMOD register is used to set the mode of Timer0 and the higher nibble is used to set the mode of Timer1.

Gate: (Timer enable bit internal/external)

By referring fig. 4.3 the Gate bit in TMOD register is set to enable timer. When Gate is set to 0 by the programmer, the timer is start and stop by control bit TR1/0 in TCON register. Gate set to 0, the timer is start and stop by external control signal INT0/1.

C/T: (Counter/ Timer Bit)

This bit is used to select the clock pulse to timer registers. C/T cleared by 0. When C/T is set to 1, the timer is receiving clock frequency from external pins T0/1(P3.5&P3.4) and act as counter.

M0 & M1:(Mode selection Bits)

These bits are used to set the operation modes of timers. Lower nibble M0 and M1 are for timer0 and Higher nibble M0 and M1 for timer1.

M1	M0	Mode
0	0	Mode 0 (13-bit timer mode)
0	1	Mode 1(16-bit timer mode)
1	0	Mode 2 (8bit - auto-reload)
1	1	Mode 3 (8-bit Split timer)

To set the timer mode all 8 bits are to be set by an instruction. With a single instruction, single timer or both timers can be set.

Example

```
MOV TMOD, #01H ;0000 0001 - mode 1 of timer 0 is set
MOV TMOD, #20H ;0010 0000 - mode 2 of timer 1 is set
MOV TMOD, #21h ;0010 0001 - mode 1 of timer 0 and
               ; mode 2 of timer1 are set.
```

TCON (Timer Control) Special Function Register

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

Fig. 4.18 TCON Special Function Register

The lower nibble bits IT1, IE0, IT1 and IE1 are control bits of interrupts. Higher nibble bits TR0, TF0, TR1 and TF1 are control bits and flags of timers.

TCON.0 (IT0)

External interrupt 0 signal type control bit. When set to 1, external interrupt flag IE0 raised by low active INT0 (Level triggering). When it is cleared by 0, external interrupt flag IE0 raised by high to low level INT0(Edge transient triggering).

TCON.1 (IE0)

External interrupt 0 edge flags. When external interrupt INT0 received on port P3.2, this flag is raised to 1. It is cleared when Interrupt Service Routine (ISR) of external interrupt 0 is executed.

TCON.2 (IT1)

External interrupt 1 signal type control bit. When set to 1, external interrupt flag IE1 raised by low active INT1 (Level triggering). When it is cleared by 0, external interrupt flag IE1 raised by high to low level INT1(Edge transient triggering).

TCON.3 (IE1)

External interrupt 1 edge flags. When external interrupt INT1 received on port P3.3, this flag is raised to 1. It is cleared when Interrupt Service Routine (ISR) of external interrupt 1 is executed.

TCON.4 (TR0)

Timer 0 start/stop bit. When set to 1, the timer 0 is start counting. On clear by 0 the timer stops. The content of timer 0 does not cleared by this bit.

TCON.5 (TF0)

Timer 0 over flow flag. When timer 0 is overflow with FFFFh and goes to 0000H with next pulse, this flag TF0 is raised by the timer 0. These flags are cleared when Interrupt Service Routine of timer 0 is executed.

TCON.6 (TR1)

Timer 1 start/stop bit. When set to 1, the timer1 is start counting. On clear by 0 the timer 1 stops. The content of timer 1 does not cleared by this bit.

TCON.7 (TF1)

Timer 1over flow flag. When timer 1 is overflow with FFFFH and goes to 0000H with next pulse, this flag TF1 is raised by the timer 1. These flags are cleared when Interrupt Service Routine of timer 1 is executed.

4.16 Modes of Timer / Counter operations

Both timer 0 and timer 1 of 8051 can be operated in 4 modes mode 0, mode 1, mode 2 and mode 3. These modes are set by M0 and M1 bits in TMOD register. The detail functions of all modes are as follows,

Mode 0: (16-bit Timer mode)

M0 and M1 are set to 0. In this mode all 16 bits of timer are not used. It is a 13-bit timer mode. 5 bits of TLx and 8 bits of THx are used to form 13-bit timer. When the timer is started in this mode, it is counting from loaded value to 1FFFh.

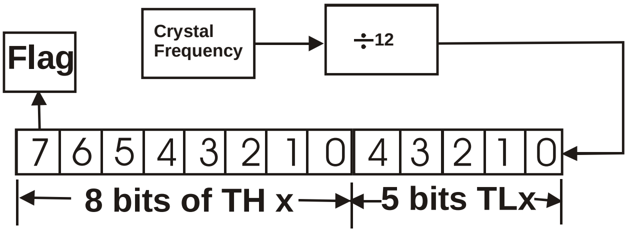


Fig. 4.19 Mode 0 of timer

Mode 1:(13-bit Timer mode)

M0 set to 1 and M1 is set to 0. In this mode the timer is working as 16-bit register. TLx and THx are connected serially to form 16-bit timer. When the timer is started, it counts from loaded value to FFFFh.

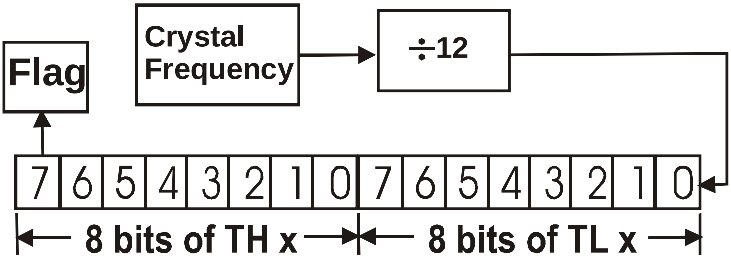


Fig. 4.20 - Mode 1 of Timer (16 bit)

Mode 2:(8-bit Auto reload timer mode)

M0 set to 0 and M1 set to 1. In this mode the timer is divided into two 8-bit registers as TLx and THx. The TLx is used as 8bit timer and THx is used to load the preset value for TLx register. THx loading the value automatically when TLx timer flag is set. When the timer started it counts from loaded value to FFH.

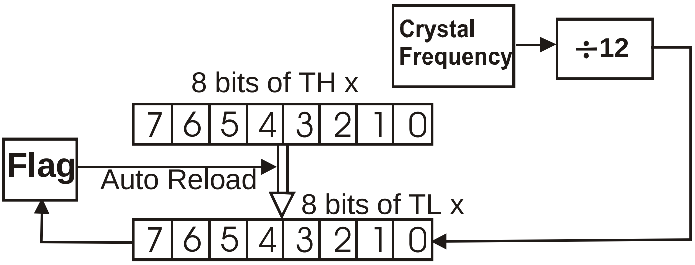


Fig. 4.21 Mode 2 of Timer (Auto Reload)

Mode 3:(8-bit separate timer mode)

Both M0 and M1 are set to 1. In this mode Timer0 is split into two separate 8-bit timers. TL0 as a separate timer and TH0 as separate timer. The timer overflow flag of timer1 (TF1) is used by TH0. So, in this mode timer1 will not have any Flag.

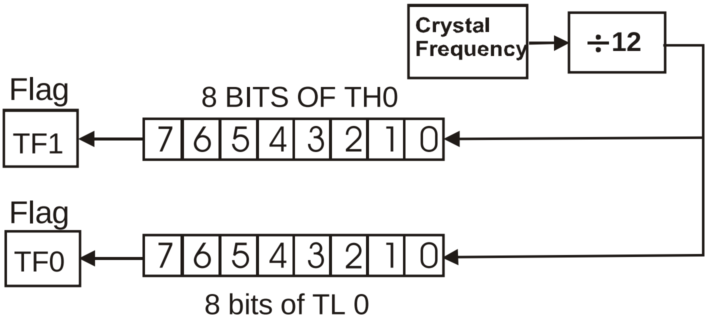


Fig. 4.22 Mode 3 of timer

Simple program using Timer/Counter

Programming of Timer

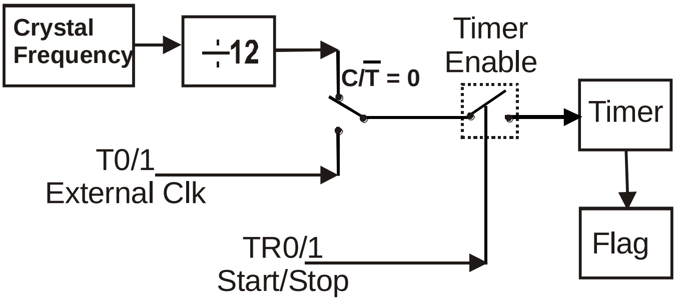


Fig 4.23 Timer logical circuit

The logical circuit of timer is shown in fig. 4.23. Timers are used to get desired time delays. Both timer0 and timer1 can be programmed at the same time to get two different time delays. Before programming, the preset value to be loaded in timer to be calculated for the delay required.

For example, if 0000h is loaded in timer and started with internal clock, it requires 65535 clock-pulses to reach FFFFh (Decimal equivalent to FFFFh is 65535). When time rolls over from FFFFh to 0000h the timer flag set to 1. So, the maximum delay is the time period of 65535 pulses. Time period for one pulse is 1micro-sec when 12MHz crystal is used.

Total time delay is 65535 x 1 = 65535 micro-sec

Example

Find Maximum delay achieved when 6MHz crystal is used.

Maximum delay achieved when a 6 MHz crystal used can be calculated as follows,

Time period for one clock T = 1/ F

F = Crystal frequency / 12

= 6 / 12

= 500 KHz

Time period for one pulse = 1/ 500 x 10 3

= .000002Secs or 2⍰ Sec

If one clock time period is 2⍰ Secs, then time period for 65535 clock are 2 x 65535 =1.131mSec

Example Program

A program to get largest delay using internal clock. Output should be available at port1 pin 3. A 12MHz crystal is used. Timer 0 to be used.

```
CLR     P1.3          ; Clear port 1 pin 3
MOV     TMOD, #01H    ; Timer 0, mode 1
                                        ; Timer 1 off
MOV     TL0,   #00H   ; 00H loaded in TL0 register
MOV     TH0,   #00H   ; 00H loaded in TH0 register
SETB    P1.3          ; Port1 pin 3 made high
SETB    TR0           ; Timer started
```

Repeat:

```
    JNB     TF0, REPEAT; Keep track timer flag TF0
    CLR     TR0           ; When TF0 set stop timer
    CLR     P1.3          ; Port 1 pin 3 to low
    CLR     TF0           ; Clear timer flag TF0
```

When the above program is executed, port 1 pin 3 high for 65535□ secs and goes to low.

The first instruction makes the output pin port1.3 to low. TMOD register is loaded with 01H. It is 0000 0001b. M0 of Lower nibble is set to 1, so the timer 0 in mode 1. Gate bit is set to 0 to start/stop the timer by TR0 in TCON register. Since C/T of timer 0 is set to 0, the internal clock is routed to timer. Next instruction loads 0000H in both timer registers TL0 and TH0.

SETB P1.3 set the port 1 pin 3 to high and SETB TR0 starts the timer to count from the loaded value 0000H. When timer is counting the flag TF0 is under track by the instruction JNB TF0, REPEAT. This instruction is repeated till the flag TF0 set to 1. When the timer 0 rolls over from FFFFH to 0000H the flag TF0 set to 1 and the processor jump to next instruction. Then CLR TR0 stops the timer, and the flag TF0 and pin p1.3 are cleared by CLR P1.3.

Port1 pin 3 goes to high and remains in high for a duration, when timer counts from 0000H to FFFFH. There are 65535 clock pulses used to count up to FFFFH. Since 12Mhz crystal is used, time duration of one pulse is 1□ sec. Total time duration for 65535 is 65535□ sec or 65.535mSec.So, the port 1 pin 3 was high for 65.535msec.

4.17 Counter programming

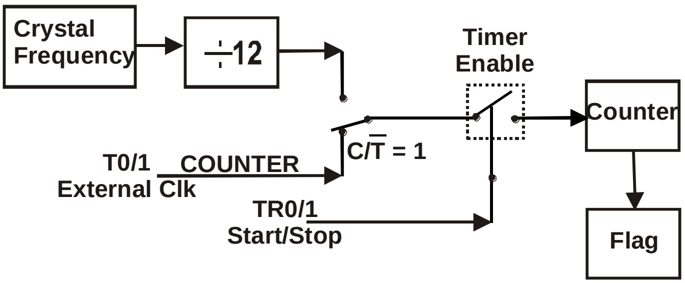


Fig. 4.24 Counter logic diagram

When the timer receives external clock frequency, then it is called as counter. To activate timer with external clock, the C/T bit in TMOD register required to be 1.

Counter Programming

```
    MOV        TMOD, #05H    ; Timer 0 selected, C/T is set to 1
    MOV        TL0,    #00H  ;
    MOV        TH0,    #00H  ; 0000h is loaded in timer registers
    SETB       P3.4          ; P3.4 set as I/p for ext. clock.
    SETB       TR0           ; Timer 0 started
```

Repeat

```
    MOV        A,      TL0   ; Contents of TL0 moved to A reg.
    MOV        P2,     A     ; Contents of A reg. moved to P2
    JNBTF0, REPEAT           ; Keep load TL0 to P2 till TF0 = 1
    CLR        TR0
    CLR        TF0
```

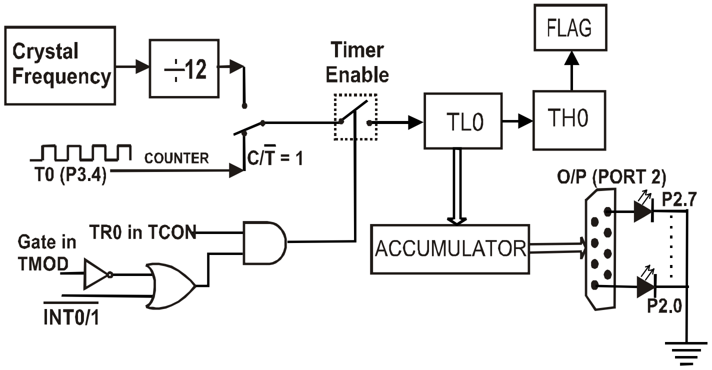


Fig.4.25

The fig. 4.25 shows the counter action when the program is executed. 8 LEDs are connected to port 2 to visualise the counting of TL0. When one clock pulse is available at pin P3.4, TL0 counts up 01h. The contents of TL0 is moved to P2. Thus, the LEDs connected to P2 shows the status of TL0. Let us analyse the program,

TMOD register is set to timer 0, mode 1 and C/T = 1, so that the timer will run on external clock from pin P3.4. Timer register is loaded with 0000h to start from beginning. The timer is started by the instruction SETB TR0.

When the first clock pulse is available at pin P3.4, the TL0 counts up one to 01H. Now 01H is moved to accumulator and then to port 2 by MOV A, TL0 and MOV P2, A instructions respectively. The corresponding LED glows when 01h available at port 2. This action is repeated till timer flag TF0 is set to 1.

Time delay Error

When a program is executed to get a desired delay, the actual delay available at output pin is more than the desired delay. This extra delay is due to instructions in loop. To get accurate delay overhead delay should be taken into account while calculating initial load value of timer.

Actual delay - Desired delay
Error = Desired delay

Example

Program to create a continuous square wave of 50% duty cycle on port P3.1 using timer 1. Desired time delay is 110µ Sec. Crystal frequency is 12MHz

Solution

Calculation of initial loading number for timer.
Timer Clock is 12 MHz / 12
= 1 MHz

Time period for each clock is 1/1 x 10-6
= 1µ Sec

Time delay required is 110µ Sec. To get 110µ Sec delay timer should be loaded with -110. The hex number for -110 is FF92.

Program

```
MOV          TMOD, #10h              ; mode 1 of timer is set
MAIN: MOV          TH1,     #FFh; Initial value FF is set
```

```
MOV TL1,   #92h                    ; Initial value 92 is set

CPL        P1.3                    ; Toggle P1.3 pin
ACALL DLAY                         ; Subroutine DELAY call

SJMP MAIN                          ; Keep continue
DELAY:
SETB       TR1                     ; Timer 1 started
REPEAT:
JNB TF1 REPEAT             ; Repeat till TF1 is set
CLR        TR0                     ; Stop Timer 1

CLR        TF1                     ; Flag bit TF1 cleared

RET
```

Calculation of actual delay by including overhead instructions. No. of machine cycles used in loop to be calculated.

Instruction	No. of Machine Cycles	Delay in μ Sec
Main:		
MOV TH1, #FFH	02	02
MOV TL1, #92H	02	02
CPL P1.3	01	01
ACALL DLAY	02	02
SJMP MAIN	02	02
Daly: SETB TR1	01	01
Repeat:		
JNB TF1 REPEAT	110	110
CLR TR0	01	01
CLR TF1	01	01
RET	01	01
Total delay is 123		

From the above calculation, the actual delay is 123 Sec. The program is made to for 110 Sec delay. 13 Sec more than the desired delay is due to overhead instruction in loop.

4.18 Interrupt

When 8051 executes a program, a signal which suspend the program is called interrupt. When an interrupt occurs, the Program Counter (PC) loads with interrupt service routine (ISR) vector. Then the 8051 processor starts to execute Interrupt Service Routine. On completion of ISR the processor come back to main program where the break occurred. Interrupt may be generated from inside 8051 or may come from outside of 8051. Interrupt generated inside 8051 is called internal interrupt and interrupt from outside is called external interrupt.

There are five interrupts provided in 8051. Three interrupts are internal interrupts and two are external interrupts. Internal interrupts are Timer0over flow flag (TF0), Timer1 over flow flag (TF1) and serial port interrupt(R1orT1). Two external interrupts are INT0 (port 3.2) and INT1(port 3.3).

Each interrupt source causes the program to jump to Interrupt Service Routine in one of the absolute address of program memory. When an interrupt occurs, a call is then done to appropriate memory location called ISR vectors. The list of ISR vectors are as follows,

Interrupt Name	ISR Vector Address (in Hex.)
External Interrupt 0(IE0)	0003H
Timer Overflow Flag 0(TF0)	000BH
External Interrupt 1(IE1)	0013H
Timer Overflow Flag 1(TF1)	001BH
Serial Interrupt (T1 or R1)	002BH

Three special function registers IE (Interrupt Enable), IP (Interrupt Priority) and TCON (Timer Control) are used in 8051 to have a complete interrupt function.

IE (Interrupt Enable) Special Function Register

7	6	5	4	3	2	1	0
EA	----	----	ES	ET1	EX1	ET0	Ex0

Fig. 4.26 IE Special Function Register

This register is a bit addressable and is used to enable concern interrupt. If the interrupt bit in this register is not set, then the 8051 CPU will not execute the Interrupt Service routine even any of the interrupt occurs. The description of each bit of this register is as follows,

IE.0 EX0 (Enable external interrupt 0). Programming this bit to 1, external interrupt INT0 raises the external edge flag IE0 in TCON register. External Interrupt edge flag IE0 causes the CPU to execute Interrupt service routine.

IE.1 ET0 (Enable timer0 over flow flag). When this bit is set to 1 by program, it enables the timer over flow flag TF0.

IE.2 EX1 (Enable external interrupt 1). Programming this bit to 1, external interrupt INT1 raises the external edge flag IE1 in TCON register. External Interrupt edge flag

IE1 causes the CPU to execute Interrupt service routine.

IE.3 ET1 (Enable timer1 over flow flag). When this bit is set to 1 by program, it enables the timer over flow flag TF1.

IE.4 ES (Enable serial port interrupt). When this bit is set to 1 by program, serial reception flag (R1) or serial transmission flag(T1) in SCON register are enabled.

IE.5, IE.6 They are kept for future use.

IE.7 EA (Enable interrupt bits). Set to 0 by program disables all interrupts. Should be set to 1 to enable one or all interrupts. IP (Interrupt Priority) Special Function Register

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

Fig. 4.27 IP Special Function Register

IP.0 PX0 - External interrupt 0 INT0 priority bit.

IP.1 PT0 - Timer 0 over flow flag (TF0) priority bit.

IP.2 PX1 - External interrupt 1 INT1 priority bit.

IP.3 PT1 - Timer 1 over flow flag (TF1) priority bit.

IP.4 PS - Serial port interrupt priority bit.

Any one of these bits is set to 1 given more priority than set to 0. If more than one bit is set to 1, then the following priority is used in 8051,

1. EX0 External Interrupt 0 (INT0)
2. ET0 Timer 0 over flow flag (TF0)
3. EX1 External Interrupt 1 (INT1)
4. ET1 Timer 1 over flow flag (TF1)
5. ES Serial interrupt RI or TI

External Interrupts (INT0, INT1)

External interrupt pins P3.2(INT0) and P3.3(INT1) are used by i/o devices or other external circuit. These pins are active low. When INT0 or INT1 goes low, interruptflags IE0 or IE1 in TCON register are set to 1 respectively in two different ways. Refer the fig. 1.28

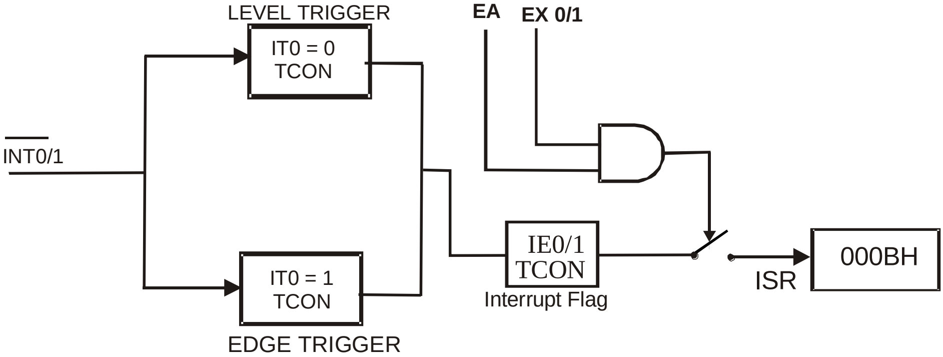


Fig 4.28 External Interrupt logical Diagram

Low level signal in INT0/1 set the IE0/1 interrupt flag to 1 when IT0/1 bit in TCON register is set to 0. If the IT0/1 bit in TCON register is set to 1, a high to low transition of INT 0/1 set the IE0/1 flag to 1. When the interrupt flag IE0/1 is set, CPU executes interrupt service routine. On completion of ISR the interrupt flags are cleared and the main program resumed.

Timer flag interrupt TF0 and TF1

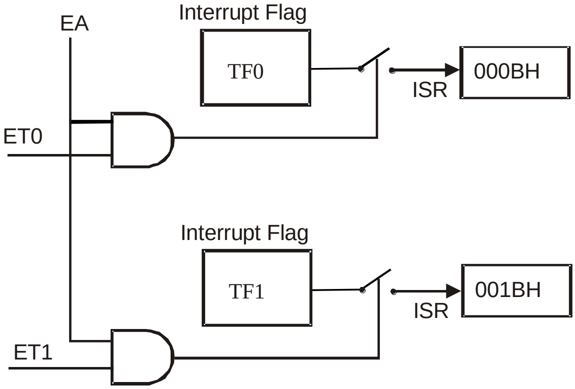


Fig. 4.29 Timer interrupt diagram

When timer/counter over flows from FFFFH to 0000H, the corresponding timer over flow flag TF0/1 is set to 1. Bits EA and Enable timer overflow flag (ET0/1) in IE register are set to 1 by program causes a timer interrupt. The CPU of 8051 executes Interrupt Service Routine. On completion of ISR timer over flags TF0/1 are cleared. The logical diagram of timer interrupt is shown in fig.1.16

Serial interrupt (RI or TI)

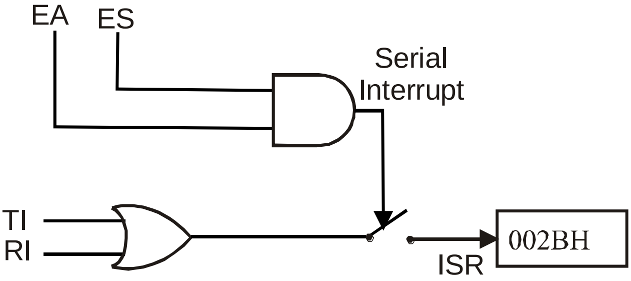


Fig. 4.30 Serial Interrupt logical diagram

Serial communication is slow compared to parallel communication. Serial data flags T1 and R1 are in SCON special function registers. These flags are ORed and serial data interrupt is produced. This interrupt is enabled only when serial interrupt bit ES and EA bit in IE register are set to 1. Although interrupt service routine is executed by the CPU 0f 8051, the T1 and R1 flags are not cleared by the ISR. These flags are to be cleared by the programmer in program.

Reset

Reset is also considered as interrupt because whenever active high is available at (RST)pin 9 of 8051, the CPU stops executing the program and goes to reset condition. The program counter jumps to 0000H. On reset of CPU, the special function registers have their reset value. Unlike another interrupt the program counter cannot resume the main program but the program restarted. So, it is called non maskable interrupt.

Including reset, there are six interrupts available in 8051.

Program using interrupts

Program to get continuously 8-bit data from port 0 and send to port2, while creating square wave of 200 micro-sec on port1 pin 1. Crystal frequency is 12Mhz.

```
ORG      0000H
LJMP     HERE
.ORG     000BH ; timer interrupt vector

CPL      P1.1
RETI
.ORG     0030H; Main program started from this address
```

```
    HERE:     MOV    TMOD, # 02H   ; Timer 0, mode 2(Auto reload)

    MOV       P0,       #0FFH; Port 0 made as input
    MOV       TH0,      #38H; To get 200msec the TH0
              ; loaded with 38H
    MOV       IE,       #82H  ; EA & timer interrupt ET0 are set
    SETB      TR0             ; timer 0 started

Back:
    MOV       A,       P0     ; port 0 data load to A register
    MOV       P2,      A      ; contents of A reg. moved to P1
```

SJMP Back

From the above program, the main program stored in ROM location 0030 onwards. Timer interrupt vector is 000BH. Timer 0 is set to mode 2(Auto reload). MOV P0, #0FFH makes Port P0 as input port by writing all 1s to Port 0 latch buffers.

Since 12MHz crystal is used, one count required 1micro-sec.To get a 200 micro-secdelay 200 counts required by timer. The initial value to load the timer is FFH - C8H (C8H is equivalent to 200) = 37. One extra clock pulse to be added for roll over to FFH to 00H.

i.e., 37 + 1 = 38.

Interrupt enable register IE is loaded with 82H to enable EA bit and ET0 bit. Master interrupt enable bit and timer interrupt enable bits are set.

SETB TR0 starts the timer and the subroutine BACK keep transferring data from P0 to P2 via Accumulator.

Whenever the timer 0 rolls over, the timer flag TF0 set to 1 and enable timer interrupt. The CPU suspend the BACK subroutine and execute timer ISR loaded at 000BH. The CPL P1.1 toggles the previous state of P1.1. RETI clears the timer flag and the program is resumed.

Subroutine HERE keep generating 200□ Sec square pulses and subroutine BACK keep transferring data from P0 to P2 till reset is pressed.

4.19 Serial Communication

8051 connected to other circuits and i/o devices via four ports P0, P1, P2 and P3. Each port having 8 pins and totally 32 pins used for data transfer. Data transferred via these ports are byte wise (8-bit at a time.) This type of communication called parallel communication.

Apart from parallel communication, 8051 provides serial communication also. In serial communication data transfer takes place bit wise. i.e. one bit at a time. Since single bit is transferred from and to, only one wire is sufficient to have serial communication.

The advantages of serial communication are,

- Single data line is used instead of wire strips
- Serial communication is much cheaper than parallel
- Since single line is used the data can be transferred to
- very large distance.
- Telephone wire can be used for transfer of data

The disadvantage of serial communication is, it is a slow process since the data are transferred bit wise. To increase the speed of the serial communication baud rate should be increased.

Baud rate

Baud rate means number of bits per second. The common baud rates used by digital serial communication are 1200, 2400, 4800, 9600 and 19200. To achieve these baud rate 11.0592MHz crystal is used.

Baud rate setting

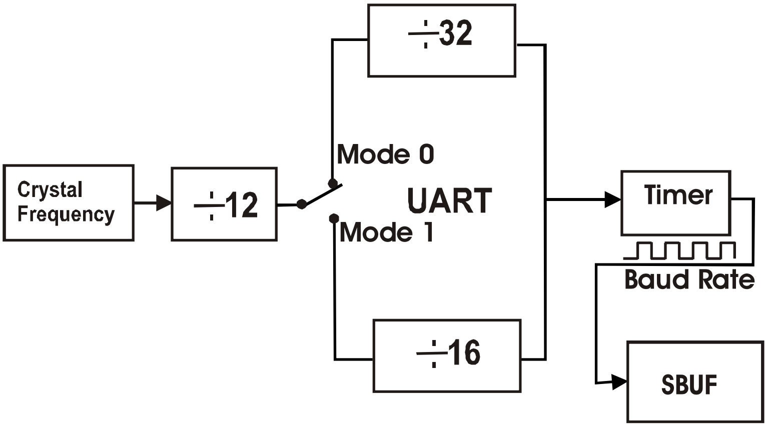


Fig. 4.31 Logical diagram of Baud rate

Since we required different baud rate depend upon the circuit connected to 8051, system clock frequency alone cannot be used. To achieve different baud rate timers are used. 8051 employs UART (Universal Asynchronous Receive and Transmit) to determine the baud rate. The clock frequency in 8051 is 1/12 of crystal frequency. i.e. 11.0592/12 = 921.58Khz. The UART divides this frequency as per mode is selected. The mode of serial communication is set by SM0 and SM1 bit in SCON special function register. When mode 0 is set, the UART divides the clock by 32 and in mode 1 by 16. Due to this the baud rate of mode 1 is double than the baud rate of mode 0. With the crystal having 11.059Mhz, the preset value to be loaded in timer to get different baud rate at different mode is as follows,

Baud Rate		Preset Value in Timer	
SMOD 0	Smode1	HEX	Decimal
1200	2400	E8	-24
2400	4800	F4	-12
4800	9600	FA	-6
9600	19200	FD	-3

SBUF (Serial Buffer) Register

8051 uses register SBUF to hold the data to be either transferred or received. SBUF is physically two 8-bit registers as Write only SBUF and Read only SBUF. The data to be transmitted is loaded to SBUF write only and transmitted out via pin TXD(P3.1) of 8051. During reception, the data available at pin RXD(P3.0) is loaded to SBUF read only.

SCON (Serial Control) Special Function Register.

This register SCON controls the serial data communication. It is an 8 bit register and bit addressable. The address of this register is 98H.

7	6	5	4	3	2	1	0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SCON Special Function Register

Fig 4.32

SCON.0 (RI)Receive interrupt flag. This interrupt flag is set to 1 when SBUF receives the stop bit. RI flag should be cleared by the programmer.

SCON.1 TI (Transmit interrupt flag). This flag is raised to 1 when SBUF transmits stop bit. This flag also not cleared by ISR and to be cleared by the programmer.

SCON.2 RB8(Received bit 8). This bit is not used in mode0. in mode 1 the stop bit is set this bit and in mode 2 & 3 bit 8.

SCON.3 TB8(Transmit bit 8). This bit is not used in mode 0&1. In mode 2&3 this bit is set/cleared by the programmer when the bit 8 is transmitted.

SCON.4 REN (Receive Enable). Set to 1 this bit enables the serial reception. To disable reception, set to 0.

SCON.5 SM2 (Multi processor communication bit): This bit is used for multi-processor communications in mode 2 & 3.

SCON.6 & SCON.7 SM0 & SM1(Serial mode bits): Set/Cleared to select serial mode. The selection of modes are as follows,

SM0	**SM1**	**Mode**	**Description**
0	0	0	SBUF used as shift register
0	1	1	Variable baud, 8-bit UART
1	0	2	baud = f/32 or f/64, 9-bit UART
1	1	3	baud rate variable, 9-bit UART

The Power Mode Control (PCON) Special Function Register

7	6	5	4	3	2	1	0
SMOD	----	----	----	GF1	GF0	PD	IDL

Fig.4.33 PCON Register

PCON Special function register is not bit addressable. The functions of each bit in PCON register is as follows,

PCON.0 - (IDL)Idle mode bit.

PCON.1- (PD)Power down bit. This bit is used as handshaking signal to other processors connected to 8051. Set to 1 by program to enter power down configuration for CHMOS processors.

PCON.2 - (GFO) General purpose user flag 0. As per requirement of the programmer, this bit can be either set or cleared in program.

PCON.3 - (GF1) General purpose user flag 1. As per requirement of the programmer, this bit can be either set or cleared in program.

PCON.4 - 6 Not used

PCON.7 - (SMOD) Serial mode baud rate bit. Set to 0 by program to use the baud rate set by timer 1. Set to 1 by program the timer 1 baud rate is doubled for mode 1,2 and 3.

Modes of Serial Communication

Mode 0:

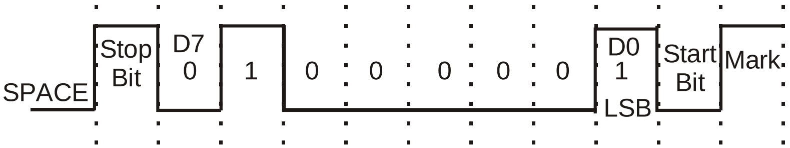


Fig. 4.34 Mode 0 timing

Mode 0 uses SBUF as an 8-bit shift register that transmits and receives data on port 3.0, while using port 3.1 to output the shift clock. Fig. 5. shows the timing for the rate transmission and reception of data of a character. When data is written into SBUF, the LSB is transmitted first. Data is transmitted with the speed of baud rate.

The receiver is enabled when bit 5 in SCON(REN) is enabled. At the end of reception interrupt flat RI will set. Mode 0 is the only mode that controls when reception can take place. Enabling reception also enables the clock pulses that shift the received data into the receiver.

MODE 1 (Standard 8-bit UART mode)

This mode is used normally to communicate in 8-bit ASCII code. Only 7-bits are needed to encode the entire set of ASCII characters. The 8th bit can be used for even or odd parity or ignored completely. asynchronous data transmission requires a start and stop bit to enable the receiving circuitry to detect the start and finish of a complete character. A total of 10 bits needed to transmit the 7bit ASCII character, as shown if fig. 4.21.

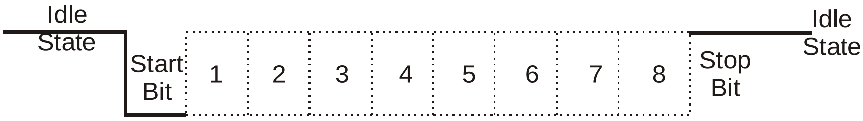


Fig. 4.35 Mode 1 Asynchronous 8-bit mode1

MODE 2 and 3

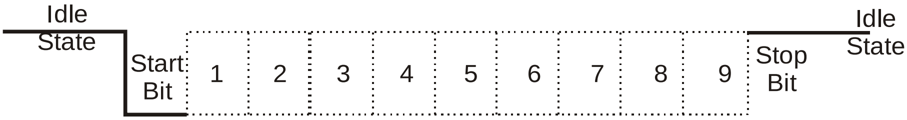


Fig. 4.36 Mode 1 Asynchronous 9-bit mode2 & 3

These modes are identical except for the baud rate. Mode 2 uses a baud rate of f/32 if SMOD is cleared or f/64 if SMOD is set. Baud rate for mode 3 are programmable using the overflows of timer1 exactly as for data mode1.

Data transmission using mode 2 and 3 features 11 bits per character as shown fig. 4. A data begins with a start bit, and the 10th bit of data is programmable bit followed by stop bit. The 10th bit is programmed in SCON.3.

Serial Data Transmission

Before the data transmitted, the baud rate and mode should be set in SCON special function register. The data of a byte to be transferred is loaded in write only SBUF. Depend upon the mode selected in SCON register SBUF frames the data with start and stop bits. Baud rate from timer is clocking SBUF to transmit loaded data bit wise. The data transmission is via port3 pin1(TXD). During transmission the TI flag is 0. When last stop bit is transmitted the TI flag in SCON is set. The program has to track the TI flag and when it is set, the next data of a byte should be loaded to SBUF. If the data is loaded before the TI flag raised, the previous data in SBUF will be over written by the new data. The programmer should clear the TI flag after loading SBUF.

Serial Data Reception

In serial data reception the port3 pin0 (RXD) is set as input pin by SETB P3.0 instruction. The REN bit in SCON is also set to 1. The data available in P3.0 is loaded in to SBUF. After receiving last stop bit the flag RI in SCON register is raised. After receiving the stop bit the SBUF register reframe the data. As soon as the RI flag is raised the data in SBUF should be shifted to Accumulator immediately for further process. Otherwise, next data available in P3.0 will overwrite SBUF and the old data will be lost. The RI flag has to be cleared by the program after moving the data from SBUF to Accumulator.

UNIT V MICROCONTROLLER PROGRAMMING& APPLICATIONS

5.1 Introduction

The function of the microprocessor or a micro controller is to accept data from input devices such as keyboards, A/D converters, etc and send the results to output devices such as LEDs, Printers and Video Monitors. These input and output devices are called either peripherals or I/O devices. Designing 108C circuits and writing instructions to enable the processor to communicate with these peripherals is called interfacing. And the logic circuits are called I/O ports or interfacing devices.

5.2 Data Transfer Instructions

The MOV – op-code involve data transfers within the 8051 memory.

The MOVX – op-code involve data transfer from external RAM

The MOVC – op-code involve data transfer from external ROM

PUSH & POP – op-code involves data transfer from stack and memory locations.

XCH – op-code exchanges data between Accumulator and Registers

Data Transfer Instructions of 8051

Sl.	Mnemonic	Byte	Machine Cycle
1.	MOV A, Rn	1	1
2.	MOV A, Direct	2	1
3.	MOV A, @Ri	1	1
4.	MOV A, #Data	2	1
5.	MOV Rn, A	1	1
6.	MOV Rn, Direct	2	2
7.	MOV Rn, #Data	2	1
8.	MOV Direct, A	2	1
9.	MOV Direct, Rn	2	2
10.	MOV Direct, Direct	3	2
11.	MOV Direct, @Ri	2	2

12.	MOV Direct, #Data	3	2
13.	MOV @Ri, A	1	1
14.	MOV @Ri, Direct	2	2
15.	MOV @Ri, #Data	2	1
16.	MOV DPTR, #Data (16)	3	2
17.	MOVX A, @Ri	1	2
18.	MOVX A, @DPTR	1	2
19.	MOVX @Ri, A	1	2
20.	MOVX @DPTR, A	1	2
21.	PUSH Direct	2	2
22.	POP Direct	2	2
23.	XCH A, Rn	1	1
24.	XCH A, Direct	2	1
25.	XCH A, @Ri	1	1
26.	XCHD A, @Ri	1	1

5.3 Logical Operations

The main application of 8051 microcontroller is that of machine control. A large part of machine controls, making decisions based on the switch states, and then turning external circuits ON or OFF is based on Boolean operators. These operations are termed as Logical operations.

Byte Level Logical Operations

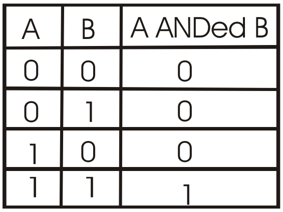
AND Operation

ANL Destination, Source

This instruction will perform a logical AND on the two operands and place the result in destination. Normally the destination is Accumulator. The source operand can be a register, location in memory or immediate data. Both the source and the destination values are of single byte size.

Examples

```
I)  MOV   A, #39h; A = 39h
    ANL   A, #43h; 39H ANDed with 43h, A = 01h
          Truth table of AND operation
```



A	B	A ANDed B
0	0	0
0	1	0
1	0	0
1	1	1

```
    39h       =       0 0 1 1 1 0 0 1
    43h       =       0 1 0 0 0 0 1 1
    ANDed     =       0 0 0 0 0 0 0 1
    Now A     =       01h
ii) MOV       A, #32h; A = 32h
    MOV       R4, #50h       ; R4 = 50h
    ANL       A, R4          ; A Anded with R4. A = 10h
    32h       =       0 0 1 1 0 0 1 0
    50h       =       0 1 0 1 0 0 0 0
    ANDed =   0 0 0 1 0 0 0 0  Now A = 10h
```

OR Operation

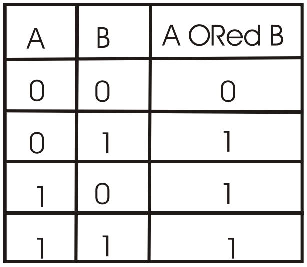
Instruction: ORL Destination, Source

This instruction performs a logical OR on the two operands and place the result in the destination. The destination is normally the Accumulator. The source operand can be a register, any location in memory or immediate data. Both the source and the destination values are of single byte size.

Example

```
I) MOV     A,      #39h    ; Accumulator = 39h
ORL        A,      #43h    ; 39h ORed with 43h, A = 7Bh
```

Truth table of OR operation



A	B	A ORed B
0	0	0
0	1	1
1	0	1
1	1	1

```
    39H      =      0 0 1 1 1 0 0 1
    43H      =      0 1 0 0 0 0 1 1
    ORed            0 1 1 1 1 0 1 1
    Now A    = 7Bh

ii) MOV      A,     #32h; Accumulator = 32h
    MOV      R4,    #50h; Register R4 = 50h
    ORL      A,     R4; 32H ORed withe 50h, A = 72h
    32H      =      0 0 1 1 0 0 1 0
    50H      =      0 1 0 1 0 0 0 0
    ORed            0 1 1 1 0 0 1 0   Now A = 72h
```

EXOR Operation

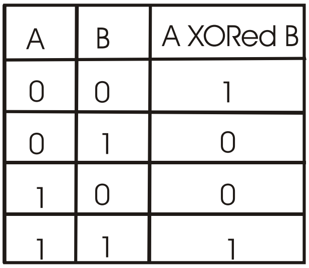
Instruction: XRL Destination, Source

This instruction performs a logical XOR on the two operands and places the result in the destination. The destination is normally the Accumulator. The source operand can be a register, any location in memory or immediate data. Both the source and the destination values are of single byte size.

Example

```
    I) MOV   A,     #39h    ; Accumulator = 39h
    XRL      A,     #43h    ; 39h ORed with 43h, A = 7Ah
```

Truth table of XOR operation



A	B	A XORed B
0	0	1
0	1	0
1	0	0
1	1	1

```
    39h      =      0 0 1 1 1 0 0 1
    43h      =      0 1 0 0 0 0 1 1
    XORed    0 1 1 1 1 0 1 0
    Now A    = 7Ah

ii) MOV      A,     #32h; Accumulator = 32h
    MOV      R4,    #50h; Register R4 = 50h
```

```
XRL       A,      R4; 32h ORed with 50h, A = 62h
32h       =       0 0 1 1 0 0 1 0
50h       =       0 1 0 1 0 0 0 0
XORed     =       0 1 1 0 0 0 1 0
Now A     = 62h
```

Complement Operation

Instruction CPL A

This instruction complements the contents of register A the Accumulator. The complementation changes the 0s to 1s and the 1s to 0s. To perform the complement operation, the data should be in Accumulator.

Example

```
MOV       A, #55h        ; Accumulator = 55h

CPL       A              ; 55hcomplemented to AAH, A = AAh
A         = 0 1 0 1 0 1 0 1 A = 55h
CPL A     = 1 0 1 0 1 0 1 0 A = AAh
```

5.4 Rotate and Swap Operations

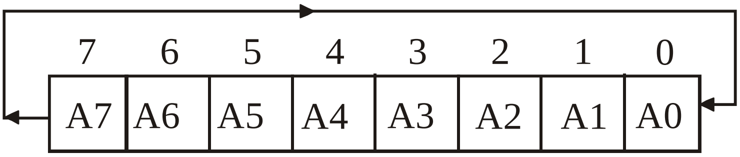
The data in Accumulator can be rotated one bit in left or right by rotate instruction. The carry flag CY also can be included with accumulator for rotate instruction. When Accumulator alone is rotated, only 8 bits in A register is rotated. When CY flag also included, then 9 bits are rotated.

The swap instruction is used to interchange the lower and upper nibble of Accumulator. When swap instruction is executed the lower nibble of A register becomes upper nibble, and upper nibble becomes lower nibble.

Instructions

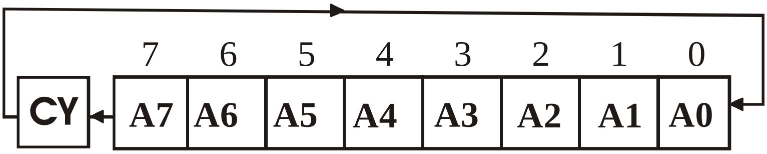
RL A

Accumulator contents are shifted on bit to left and A7 to A0, A0 to A1, A1 to A2 and so on...



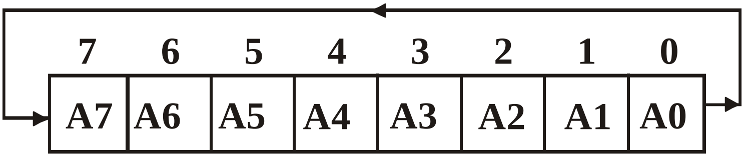
RLC A

Contents of Accumulator with CY flag shifted one bit left. A7 to CY, CY to A0, A0 to A1 and so on...



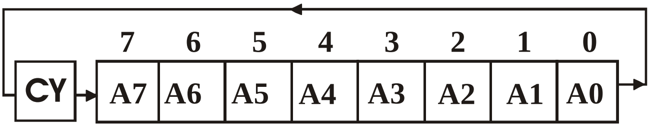
RR A

Accumulator contents shifted one bit to Right. A0 to A7, A7 to A6, A6 to A5 and so on...

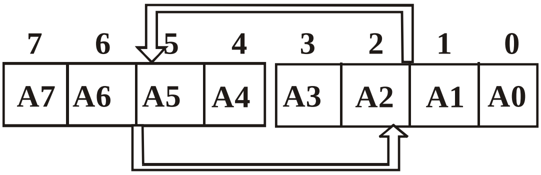


RRC A

Contents of Accumulator with CY flag Shifted one bit right. A0 to CY, CY to A7, A7 to A6 and so on...



SWAP A ; Interchange the nibbles of Accumulator.



5.5 Bit Level Logical Operations

A Unique and powerful operation feature of the 8051 microcontroller is single bit operation. Single bit instructions allow programmer to Set, Clear, Move and Complement individual bits of a port, memory or register. Bit operators yield compact program code that enhances program execution speed. In 8051 some registers, a portion of RAM and I/O Ports are bit addressable.

Instruction		Function
SETB	Bit Address	Addressed bit set to 1
CLR	Bit Address	Addressed bit cleared to 0
CPL	Bit Address	Addressed bit complemented

Internal RAM bit Addressable

The availability of individual bit addresses in internal RAM makes the use of RAM very efficient when storing bit information. Whole bytes do not have to be used up to store one or two bits of data.

Of the 128-byte Internal RAM of the 8051, only 16 bytes of it addressable. The bit addressable locations are 20h to 2Fh.

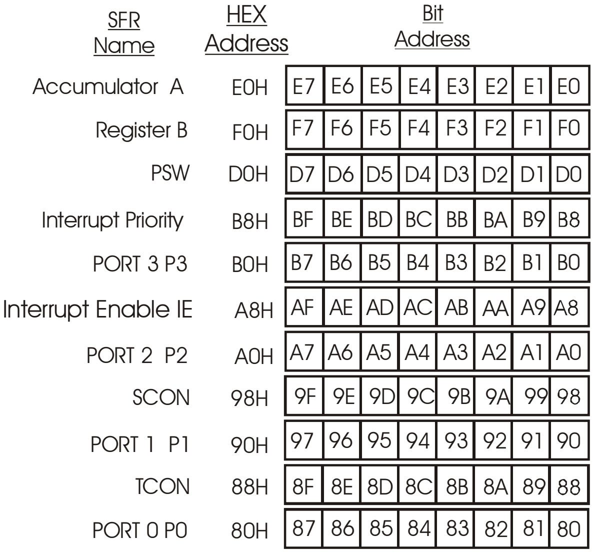
Example

SETB 05

The bit addressable RAM locations start from 20H. The 6th bit of RAM location 20H is set to 1.

SFR Bit Addresses

All the special function registers (SFRs) may be addressed at the byte level by using the direct address assigned to it, but some of the SFRs are addressable at bit level. The bit addressable SFR and the corresponding bit addresses are as follows;



SFR Name	HEX Address	Bit Address							
Accumulator A	E0H	E7	E6	E5	E4	E3	E2	E1	E0
Register B	F0H	F7	F6	F5	F4	F3	F2	F1	F0
PSW	D0H	D7	D6	D5	D4	D3	D2	D1	D0
Interrupt Priority	B8H	BF	BE	BD	BC	BB	BA	B9	B8
PORT 3 P3	B0H	B7	B6	B5	B4	B3	B2	B1	B0
Interrupt Enable IE	A8H	AF	AE	AD	AC	AB	AA	A9	A8
PORT 2 P2	A0H	A7	A6	A5	A4	A3	A2	A1	A0
SCON	98H	9F	9E	9D	9C	9B	9A	99	98
PORT 1 P1	90H	97	96	95	94	93	92	91	90
TCON	88H	8F	8E	8D	8C	8B	8A	89	88
PORT 0 P0	80H	87	86	85	84	83	82	81	80

Bit Addressable Accumulator

```
SETB      ACC.6       ; 7th bit of Accumulator is set to 1
CLR       ACC.7       ; 8th bit of Accumulator is set to 0
```

Bit Addressable I/O Ports

The 8051 has 4 I/O ports P0, P1, P2 and P3, each of 8 bits.

We can access either the entire 8 bits on any single bit without altering the rest.

To access a port in a single bit, the syntax is

```
SETB      Port number. Pin number
```

Port numbers are P0 for Port 0, P1 for Port 1, P2 for Port 2 and P3 for Port 3. Port pin numbers are 0 to 7.

Port's Bit	P0	P1	P2	P3
D0	P0.0	P1.0	P2.0	P3.0
D1	P0.1	P1.1	P2.1	P3.1
D2	P0.2	P1.2	P2.	P3.2
D3	P0.3	P1.3	P2.3	P3.3
D4	P0.4	P1.4	P2.4	P3.4
D5	P0.5	P1.5	P2.5	P3.5
D6	P0.6	P1.6	P2.6	P3.6
D7	P0.7	P1.7	P2.7	P3.7

Example

i) SETB P1.3; Port 1 Pin 3 is set to 1

ii) CLR P2.6; Port 2 Pin 6 cleared to 0

Bit Addressable Program Status Word

7	6	5	4	3	2	1	0
CY	AC	F0	RS1	RS0	0V	----	P

Program Status Word(PSW)

PSW.0 - P (Parity Flag) Shows the parity status of Accumulator

PSW.1 - Reserved for future use

PSW.2 - 0V (Over flow flag) Used in arithmetic function. When the result of an arithmetic function is more than 8 bits in size, the OV flag set to 1.

PSW.3 - RS0 Register Bank select Bit 0

PSW.4 - RS1 Register Bank select Bit 1

RS0	RS1	Selected Register Bank
0	0	Register Bank 0
0	1	Register Bank 1
1	0	Register Bank 2
1	1	Register Bank 3

By default, RS0 and RS1 are set to 0 to select Register Bank0.

PSW.5 - F0 User Flag 0.

PSW.6 - AC Auxiliary Flag use in BCD arithmetic

PSW.7 - CY Carry Flag use in arithmetic functions.

Example

To select Register Bank 2, RS0 = 0 and RS1 = 1

```
SETB       PSW.4
CLR        PSW.3
```

Carry Bit related instructions

Instruction		**Function**
SETB	C	Make CY = 1
car C		Clear carry bit. CY = 0
CPL	C	Complement Carry bit
MOV	bit, C	Copy carry to bit status
MOV	C, bit	Copy bit status to carry
JNCRel		Jump to Relative address if C = 0
JC Rel		Jump to Relative address if C = 1
ANL	C, bit	AND CY with Bit and result in C
ANL	C, /bit	AND CY with complement of Bit and store result in C
ORL	C, bit	OR CY with Bit and result in C
ORL	C, /bit	OR CY with complement of Bit and store result in C

ANL C, Source bit

In this instruction the carry flag bit is ANDed with a source bit and the result is placed in carry.

Example 1

Write a program to clear the accumulator if both bits P2.1 and P2.2 are high, otherwise make accumulator FFh.

```
MOV        A, #0FFh; A = FFh
MOV        C, P2.1; CY = P2.1
ANL        C, P2.2; P2.2 and CY are ANDed. If both are 1 then
                    ; Carry C is 1, otherwise 0
JNC OVER            ; Jump to B1 if carry is not 1
CLR        A        ; If carry is 0 clear accumulator.
OVER:
```

Example 2

Write a program to clear the accumulator if P2.1 is high and P2.2 is low, otherwise make accumulator FFh.

```
MOV        A, #0FFh; A = FFh
MOV        C, P2.1; CY = P2.1
ANL        C, /P2.2; P2.1 ANDed with complement of P2.2
JNC OVER            ; Jump to B1 if carry is 0
CLR        A        ; If carry is 0 clear accumulator.
OVER:
```

MOV destination bit, Source bit

This move instruction copies the source bit to the destination bit. In this instruction one of the operands must be the Carry 'C'.

Example

```
MOV        P1.2, C; State of carry bit is loaded to port 1 pin 2.
MOV        C, P2.5; State of port 2 pin 5 is loaded to carry bit.
```

ORL C, Source bit

In this instruction the carry flag bit is ORed with a source bit and the result is placed in the carry flag. Therefore, the carry flag bit is set to 1 if the source bit is 1 otherwise carry flag is unchanged.

Write a program to clear the accumulator if P2.1 or P2.2 is high, otherwise make accumulator FFh.

```
MOV        A, #0FFh; A = FFh
MOV        C, P2.1; CY = P2.1
ORL        C, P2.2; P2.2 and CY are ANDed. If both are 1 then
                    ; Carry C is 1, otherwise 0
```

```
JNCOVER           ; Jump to B1 if carry is not 1
CLR        A      ; If carry is 0 clear accumulator.
OVER:
```

5.6 Arithmetic Instruction Flags

There are four arithmetic instruction flags available in 8051 microcontrollers. There are Carry flag (CY)

1. Auxiliary carry flag (AC)
2. Overflow flag (OV)
3. Parity flag (PF)

The CY, AC and OV flags are Arithmetic flags. They are set to 1 or cleared to 0 automatically depending on the result of the Arithmetic Instructions.

The parity flag (pf) is affected by every instruction executed. The Parity Flag will be set to1 if the number of 1s in the A register is odd and will be set to 0 if the number of 1s in the A register is even.

5.7 Addition

Addition of two 8-bit numbers in 8051 involves the Accumulator (A Register) to store the result of Addition. i.e., A register is used as a destination Register. The source can be an immediate number, a Register, a direct address and an indirect address.

There are two types of addition

i) Unsigned addition and

ii) Signed addition

Unsigned Addition

Unsigned numbers are defined as data in which all the bits are used to represent data, and no bits are set aside for the positive or negative sign. This means that the operand or data can be between 00h and FFh (i.e. 0 to 255 decimal).

Adding two unsigned numbers generates a carry flag (CY) when the sum exceeds FFh. Auxiliary carry flag (AC) is set when there is a carry from bit D3 i.e. carry from lower nibble to higher nibble. Overflow flag (OV) is not used in unsigned addition.

Example

Decimal	Hex.	Binary
95	5Fh	0 1 0 1 1 1 1 1
189	BDh	1 0 1 1 1 1 0 1
284	1Ch	

ALU of 8051 adds the above number as follows

0 1 0 1 1 1 1 1
1 0 1 1 1 1 0 1
carry 1 0 0 0 1 1 1 0 0 ==> 284 d

In the above example, there is carry from bit D6 to bit D7. Therefore, carry flag (CY) set to 1. This shows that the sum is more than 255d and the carry is to be adjusted as with the sum as 9-bit result. Since the 9-bit result cannot be stored in single byte, the sum with carry is stored in two registers as 16-bit data. The sum should be stored in one register as lower byte result and the carry in another byte as higher byte result.

Signed Addition

Signed numbers are defined as data in which the MSB of the 8-bit data i.e. D7 bit is used to represent the sign of the number. This means that the numbers of size 0 to +127d can be represented as positive numbers. Numbers of size 0 to -127d can be represented as negative numbers.

Representation of +ve numbers in signed form

i) + 5 ==> 0 0 0 00 1 0 1 D7-bit is 0 = +ve number
ii) + 32 ==>0 0 0 10 0 0 0 D7-bit is 0 = +ve number

Representation of -ve numbers in signed form

The magnitude of negative number is represented in its 2's Complement form.

-5 can be represented as follows,
5 ==>0 0 0 00 1 0 1
1 1 1 11 0 1 0 (1's Complement of 5)
1 (+)
2' Complement 1 1 1 11 0 1 1 ==> FBh

In hex number -5 decimal is represented as FBh.

Addition of singed numbers involves the overflow flag (OV) and carry flag (CY). **Overflow**

Flag (OV) is set to 1 if either of the following two conditions occurs:

i) There is a carry from D6 to D7 but no carry from D7.

ii) There is a carry from D7 but no carry from D6 to D7.

Logically OV is expressed as follows. Overflow Flag is set when carry 6(C6) XOR carry 7(C7).

OV = C6 XOR C7.

When OV set to 1, it indicates that the result of addition exceeds the largest positive or negative number i.e. +127 to -127, and the result is not in true form. The result should be in 9-bit. The result in A register is 8-bit only. To adjust the sum to true form, the following action are to be taken by the programmer depending on the status of Carry and Overflow flags.

Carry	**Overflow**	**Action**
Flag (CY)	Flag (OV)	To taken
0	0	None
0	1	Complement the sign.
1	0	None
1	1	Complement the sign.

Example

Add -1d with +27d.

-01(+)

+27

+26

-1d Should be loaded into 8051 in 2's complement form.

2' Complement of -1 is

1 1 1 1 1 1 1 0 1's complement of 1

1 (+)

1 1 1 1 1 1 1 1 2's complement of 1

-1d = 1 1 1 1 1 1 1 1 (2's Complement form of 1)

+27d = 0 0 0 1 1 0 1 1(+)

0 0 0 1 1 0 1 0 ==> 1Ah = +26

Carry flag (AC) = 1 (Carry from D7)

Overflow Flag (OV) = 0(Carry from D7, No carry from D6)

From the above table 3.1, it shows that the sum is true value and no adjustment required.

Example 2

Add +100d to +50d.

+100d ==> 64h

+ 50d ==> 32h

+150d ==> 96h

1(Carry from D7)

+100d = 0 1 1 00 1 0 0 (+)

+ 50d = 0 0 1 10 0 1 0

0 1 0 0 10 1 1 0

Carry flag (CY) = 0(No Carry from D7)

Overflow Flag (OV) = 1(Carry from D6, No carry from D7)

The Overflow flag is set to 1 indicates that the sum is not in true form. From the result we can see that the D7 is 1. It shows that it is negative number which is not true. The sum should be adjusted as per the table 3.1. The whole 8-bit is taken as sum and the sign bit is complement of D7.

So, the adjusted result is 0 1 0 0 1 0 1 1 0 ==> +150 = 96h

Sign Data

Example 3

Add -30d with -50d.

30d = 0 0 0 1 1 1 1 0

1's complement of 30d is 1 1 1 0 0 0 0 1

1(+)

2's Complement of 30d = 1 1 1 0 0 0 1 0 = E2h Same way,

2's Complement of 50d = 1 1 0 0 1 1 1 0 = ECh

-30d ==> E2h (2's complement of -30d)

-50d ==> CEh (2's complement of -50d)

-80d ==> 1B0h

1 1 1 1

-30d = 1 1 1 0 0 0 1 0 (2's complement of 30d)

-50d = 1 1 0 0 1 1 1 0(+) (2's complement of 50d)

1 1 0 1 1 0 0 0 0 ==> B0h = -80d

Carry flag (CY) = 1(Carry from D7)

Overflow Flag (OV) = 0(Carry from D6 and Carry from D7)

From the table 3.1, the overflow flag OV is 0 and carry flag CY is 1 the sum is true value and no adjustment is needed. The accumulator content is in 2's complement form of 80d.

Example 4

Add -70d with -70d.

30d = 0 1 0 0 0 1 1 0

1's complement of 70d is 1 0 1 1 1 0 0 1

1(+)

2's Complement of 70d = 1 0 1 1 1 0 1 0 = BAh

-70d ==> BAh (2's complement of -70d)

-70d ==> BAh (2's complement of -70d)

-140d ==> 17Ah

1 1 1

-70d = 1 0 1 1 1 0 1 0 (2's complement of 30d)

-70d = 1 0 1 1 1 0 1 0(+) (2's complement of 50d)

1 0 1 1 1 0 1 0 0 ==> 74h

Carry flag (CY) = 1(Carry from D7)

Overflow Flag (OV) = 1(No Carry from D6 and Carry from D7)

From the table 3.1, The Overflow flag is set to 1 indicates that the sum is not in true form. From the result we can see that the D7 is 0. It shows that it is positive number which is not true. The sum should be adjusted as per the table 3.1. The whole 8-bit is taken as sum and the sign bit is complement of D7.

So, the adjusted result is 1 0 1 1 1 0 1 0 0 ==> 74h = -140d

Sign Data

The instruction used for addition of two 8-bit data can be given as,

```
ADD     A, Source
```

Source - Immediate data or data in register.

Program of adding two 8-bit numbers:

```
MOV     A, #0F5h; A = 75h
ADD     A, #0Bh; A = 75h + 0Bh = 80h
```

75h ==> 0 1 1 1 0 1 0 1

0Bh ==> 0 0 0 0 1 0 1 1(+)

1 0 0 0 0 0 0 0 = 80h

CY = 0 OV = 0

After addition of two 8-bit data A = 80h.

5.8 Addition two 16-bit data

Since the architecture of 8051 is 8-bit, a 16- bit data required two bytes. Lower byte is stored in one register/memory location and higher byte is stored in next register/memory location. Same way two 16-bit data are stored in four register/memory location.

Initially the carry is to be cleared. To add 16-bit data first the lower bytes are be added and the result is stored in a register/memory location. Then the higher bytes are be added with the carry from first result. The sum in accumulator is moved next to the lower byte result.

Example

A program to add two 16-bit numbers. The numbers are 3CE7h and 3B8Dh. Place the sum in R6 and R7.

Higher Byte	Lower Byte
1	
3C	E7
3B	8D (+)
78	74

Program of adding two 16-bit numbers

```
CLR        C              ; Carry flag is cleared
MOV        A, #0E7h       ; A = E7h
ADD        A, #8Dh        ; A + 8Dh = 74h and CY = 1
MOV        R6,    A       ; R6 = 74h
MOV        A, #3Ch        ; A = 3Ch
ADDC       A, #3Bh        ; A + 3Bh + CY = 78h
MOV        R7,    A       ; R7 = 78h
```

ADDC instruction is used to add two data bytes along with the status of Carry flag.

5.9 Subtraction

In microprocessor systems, subtraction is done by taking the 2's complement of the number to be subtracted the subtrahend, and adding it to another number the minuend. But in 8051 direct subtraction of two numbers is possible. 8051 uses SUBB instruction for direct subtraction. Register A is the destination address for subtraction, the source can be an immediate data, a data in register, a data in direct address and data in an indirect address.

There are two types of subtraction.

i) Unsigned Subtraction

ii) Signed Subtraction

Unsigned Subtraction

In unsigned subtraction all 8-bits are used for magnitude. The carry flag CY is used as a borrow flag in subtraction. The Barrow flag CY should be set to 0 before starting subtraction of two 8-bit numbers. Overflow flag is not taken in to account while doing unsigned subtraction.

If the source number is smaller than the number in A, the result will be in true form with barrow flag is 0. If the source number is more than the number in A, the result is a negative number. The barrow flag is set to 1 indicating that the result is not in true form but in 2's complement form.

Example

Subtract 100d from 15d (15d - 100d).

15d (-)	==>	0 0 0 0 1 1 1 1	==>	0Fh (-)
100d	==>	0 1 1 0 0 1 0 0	==>	64h
- 85d	==> 1	1 0 1 0 1 0 1 1	==> 1	ABh
Borrow flag		= 1		

Since Barrow flag is set to 1, the answer is a negative number. Negative number is always not in true form but in 2's complement form. Result ABh is 2's complement of 85d or -85d = ABh.

Signed Subtraction

When numbers of like signs are subtracted, the result will not exceed positive or negative magnitude limits of +127d or -128d and so the magnitude and the sign of the result do not need to be adjusted. When unlike sign numbers are subtracted the result may exceed the limit of 7-bit. When the result is more than the limit, the over flow flag set to 1. Like signed addition, same action to be taken. Refer table 3.1.

Example 1

Subtract +126d from 100d (100d - 126d).

Sign.		Data
+100d (-)	==>	0 1 1 0 0 1 0 0 ==> 64h (-)
+126d	==>	0 1 1 1 1 1 1 0 ==> 7Eh
- 26d	==> 1	1 1 1 0 0 1 1 0 ==> 1 ABh

Borrow flag = 1

Overflow flag = 0 (Borrow to D7 and Borrow to D6)

Since the overflow flag is not set, the result is in true form. Result is -26d == ABh.

Example 2

Subtract -116d from -061d (-061d - (-116d)).

-061d (-) ==> C3h (-) (2's Complement of 61)

-116d ==> 8Ch (2's Complement of 116)

+ 55d ==> 037h

-06d is represented in hex. as follows,

61d ==> 0 0 1 1 1 1 0 1

1's Complement of 61d ==> 1 1 0 0 0 0 1 0

1(+)

2's Complement of 61d 1 1 0 0 0 0 1 1 ==> C3h

-116d is represented in hex. as follows,

116d ==> 0 1 1 1 0 1 0 0

1's Complement of 116d ==> 1 0 0 0 1 0 1 1

1(+)

2's Complement of 116d 1 0 0 0 1 1 0 0 ==> 8Ch

Sign. Data

- 61d (-) ==> 1 1 0 0 0 0 1 1 ==> C3h (-)

-116d ==> 1 0 0 0 1 1 0 0 ==> 7Eh

+ 37d ==> 0 0 0 1 1 0 1 1 1 ==> 0 37h

Borrow flag = 0

Overflow flag = 0 (Borrow to D7 and Borrow to D6)

Since the overflow flag is not set, the result is in true form. Result is +37d == 37h.

Example 3

Subtract +100d from -099d (-099d - (+100d)).

-99d (-) ==> 9dh (-) (2's Complement of 91)

+100d ==> 64h

-199d ==> 0 39h

-99d is represented in hex. as follows,

99d ==> 0 1 1 0 0 0 1 1

1's Complement of 99d ==> 1 0 0 1 1 1 0 0

1(+)

2's Complement of 99d 1 0 0 1 1 1 0 1 ==> 9Dh

Sign. Data

- 99d (-) ==> 1 0 0 1 1 1 0 1(-) ==> 9Dh (-)

+100d ==> 0 1 1 0 0 1 0 0 ==> 64h

-199d ==> 0 0 0 1 1 1 0 0 1 ==>0 39h

Borrow flag = 0

OV flag = 1(No Borrow to D7 and Borrow to D6)

Since the overflow flag is set, the result is in not true form (See the result. The Sign bit is 0, indicating + 39, which is not true). The sum has to be adjusted as per table 3.1. The sign bit D7 should be complemented. The sum is an 8-bit data.

So, the adjusted result is 1 0 0 1 1 1 0 0 1 ==> 39h = -199d

Sign Data

Example 4

Subtract -52d from +87d (87d - (-52d)).

+87d (-) ==> 57h (-)

-52d ==> CCh (2's Complement of 51)

+139d ==> 1 8Bh

-52d is represented in hex. as follows,

52d ==> 0 0 1 1 0 1 0 0

1's Complement of 52d ==> 1 1 0 0 1 0 1 1

1(+)

2's Complement of 52d 1 1 0 0 1 1 0 0 ==> CCh

Sign. Data

+87d (-) ==> 0 1 0 1 0 1 1 1(-) ==> 57h (-)

-52d ==> 1 1 0 0 1 1 0 0 ==> CCh

+139d ==> 1 1 0 0 0 1 0 1 1 ==> 1 8Bh

Borrow flag = 0

Overflow flag = 1 (Borrow to D7 and No Borrow to D6)

Since the overflow flag is set, the result is in not true form (See the result Sign bit is 1 indicating the sum is negative, which is not true.). The sum has to be adjusted as per table 3.1. The sign bit D7 should be complemented. The sum is an 8-bit data.

So, the adjusted result is 0 1 0 0 0 1 0 1 1 ==> 8Bh = +139d

Sign Data

Sample Program

```
CLR     C               ; CY = 0
MOV     A,      #3Fh    ; A = 3Fh
MOV     R3,     #23h    ; R3 = 23h
SUBB    A,      R3      ; A = 3Fh - 23h = 1Ch
                        ; Flags CY = 0 AC = 0
```

```
3Fh (-)     ==>     0 0 1 1 1 1 1 1
23h         ==>     0 0 1 0 0 1 1 0
1Ch         ==> 0   0 0 0 1 1 1 0 0
```

5.10 Multiplication

Multiplication operation use registers A and B as both source and destination address for the operation. The 8051 supports byte by byte multiplication only. The bytes are assumed to be unsigned data. The instruction used for multiplication operation is as follows:

```
MUL         AB        ; Multiplies A by B and place the 16-bit
                      ; result in registers B and A.
```

In byte by byte multiplication, one of the operands must be in register A and the second operand must be in register B. After multiplication, the 16-bit result is stored in registers A and B. The lower byte is in A and upper byte in B.

Before start of multiplication operation, carry flag is always cleared to 0. The largest possible product is FFh x FFh. So, the largest possible result is FE01h (FFh x FFh = FE01h). If the result of product is less than FFh, the result is stored in register A only. If the result of product is more than FFh, the lower byte result is stored in register A and upper byte result in register B. This occasion the overflow flag is set to 1. So whenever overflow flag is set to 1, the programmer should check the register B for upper byte result.

Sample Program 1

```
MOV         A,        #7Bh    ; A = 7Bh
MOV         b,        #02h    ; B = 02h
MUL         AB                ; 7Bh x 02h = F6h
    7Bh x    ==> 123d x
    02h      ==> 2d
    F6h      ==> 246d
```

Since the result F6h is less than FFh (246 < 255), the result is stored in register A only and the overflow flag is 0.

Sample Program 2

```
MOV         A,        #7Bh            ; A = 7Bh
MOV         b,        #0FE2h          ; B = FEh
MUL         AB                        ; 7Bh x FEh = 7A0Ah
    7Bh x             ==>      123d x
    FEh               ==>      254d
```

```
7A0Ah           ==> 31242d
```

Since the result 7A0Ah is more than FFh (31242 >255), the lower byte result 0A is stored in register A and upper byte result 7Ah is stored in register B and the overflow flag is 1.

5.11 Division

Division operation use registers A and B as both source and destination address for the operation. The 8051 supports byte by byte division only. The bytes are assumed to be unsigned data. The instruction used for division operation is as follows:

```
DIVAB       ; Divide A by B
```

In byte by byte multiplication, the numerator must be in register A and the denominator in register B. After division the quotient is in the register A and the remainder in register B. The denominator should not be 0. If the denominator is 0, the overflow flag is set to 1 indicating that the division is undefined. If the denominator is not 0, the overflow flag is 0 and CY is also 0.

Sample Program

```
MOV     A,      #95     ; A = 95d
MOV     B,      #10     ; B = 10d
DIVAB                   ; 95 / 10 , A = 9 and B = 5.
```

In the above program decimal 95 is loaded in A and decimal 10 is loaded in B. After the instruction DIV AB is executed the quotient is stored in A and the remainder is stored in B.

5.12 Binary coded decimal (BCD) Arithmetic

In BCD four bits are required to represent a decimal number from 0 to 9(0000 to 1001). Such numbers are called Binary Coded Decimal (BCD) numbers. There are two types of BCD numbers.

i) Unpacked BCD

ii) Packed BCD

Unpacked BCD

In unpacked BCD, a decimal number is represented in an 8-bit binary. The lower 4 bits shows the decimal number and the upper 4-bits are 0s.

Example 9 ==> 0 0 0 0 1 0 0 1

5 ==> 0 0 0 0 0 1 0 1

Packed BCD

In packed BCD two decimal numbers are represented in an 8-bit binary number or in a byte. The lower 4-bits shows the unit digit and the upper 4-bits shows the ten digits.

Example

59h ==> 0 1 0 1 1 0 0 1   5 9

Addition of BCD numbers

8051 Adds two BCD number like normal addition. So, the Adding two BCD numbers results in a non-BCD number. To make the result to a BCD number 6 must be added to lower nibble or higher nibble. The instruction used to adjust the result to BCD number is,

```
DAA     ; Adjust for BCD addition
```

The added result should be placed in A register. Then only DA instruction will work on A register. This instruction always assumes the numbers were in BCD before the addition was done.

There are two flags are affected and they are Auxiliary carry flag (AC) and Carry flag (CY). When lower nibble is more than 9, the Auxiliary flag (AC) set to 1 and when the higher nibble is more than 9, the Carry flag (CY) set to 1.

Simple program 1

Write a program to add two BCD numbers 17 and 28. Result should be in BCD form.

Packed BCD number for 17 is 17h

Packed BCD number for 28 is 28h

```
MOV     A,      #17h    ; A = 17h
ADD     A,      #28h    ; A = 17h + 28h
DA A                    ; Adjust the result to BCD.
17h     ==>     0 0 0 1 0 1 1 1 (+)
28h     ==>     0 0 1 0 1 0 0 0
45h     ==>     0 0 1 1 1 1 1 1
```

Flag AC is 1 (Lower nibble is more than 9)

Flag CY is 0 (Higher nibble is less than 9)

Since the AC flag is set to 1, the DAA instruction adds 6 to the lower nibble and stores the result in A register.

```
0 0 1 1 1 1 1 1(+)
0 0 0 0 0 1 1 0
```

```
        0 1 0 0 0 1 0 1 ==> 45h
```

The result 45 is stored in A register in BCD form as 45h.

Simple program 2

Write a program to add two BCD numbers 52 and 87. Result should be in BCD form.

Packed BCD number for 52 is 52h

Packed BCD number for 87 is 87h

```
MOV     A,      #52h   ; A = 52h
ADD     A,      #87h   ; A = 52h + 87h
DA  A                  ; Adjust the result to BCD.
52h     ==>     0 1 0 1 0 0 1 0 (+)
87h     ==>     1 0 0 0 0 1 1 1
139h            1 1 0 1 1 0 0 1 ==> D9h
```

Flag AC is 0 (Lower nibble is 9)

Flag CY is 1 (Higher nibble is more than 9)

Since the CY flag is set to 1, the DA A instruction adds 6 to the higher nibble and stores the result in A register.

```
    1 1 0 1 1 0 0 1(+)
    0 1 1 0 0 0 0 0
  1 0 0 1 1 1 0 0 1 ==> 139h
```

The result 139 is stored in A register and B register. 39h in A as packed BCD form and 1h in B as unpacked BCD.

The DA A instruction to be used only with BCD addition. When DA A instruction is used, 8051 assumes the numbers added are in BCD. This instruction will work only with ADD and ADDC instructions. It will not work with other Subtraction, Multiplication and Division instructions

5.13 Arithmetic Operation Used by 8051.

Mnemonic	Byte	Machine Cycle
ADDA, Rn	1	1
ADDA, direct	2	1
ADDA, @Ri	1	1
ADDA, #Data	2	1
ADDC A, Rn	1	1
ADDC A, Rn	1	1
ADDC A, direct	2	1
ADDC A, @Ri	1	1
ADDC A, #Data	2	1

SUBB A, Rn	1	1
SUBB A, direct	2	1
SUBB A, @Ri	1	1
SUBB A, #Data	2	1
INC A,	1	1
INC Rn	1	1
INC direct	2	1
INC @Ri	1	1
DEC A,	1	1
DEC Rn	1	1
DEC direct	2	1
DEC @Ri	1	1
INC DPTR,	1	2
MUL AB	1	4
DIV AB	1	4
DA A	1	1

5.14 LOGICAL Operation Used by 8051.

Mnemonic	Byte	Machine Cycle
ANL A, Rn	1	1
ANL A, direct	2	1
ANL A, @Ri	1	1
ANL A, #data	2	1
ANL direct, A	2	1
ANL direct, #data	3	2
ORL A, Rn	1	1
ORL A, direct	2	1
ORL A, @Ri	1	1
ORL A, #data	2	1
ORL direct, A	2	1
ORL direct, #data	3	2
XRL A, Rn	1	1
XRL A, direct	2	1
XRL A, @Ri	1	1
XRL A, #data	2	1
XRL direct, A	2	1
XRL direct, #data	3	2
CLR A	1	1
CPL A	1	1

RL A		1
RLC A	1	1
RR A		1
RRC A	1	1
SWAP		1

5.15 Data Transfer Instructions Used by 8051

Mnemonic	Byte	Machine Cycle
MOV A, Rn	1	1
MOV A, direct	2	1
MOV A, @Ri	1	1
MOV A, #data	2	1
MOV Rn, A	1	1
MOV Rn, direct	2	2
MOV Rn, #data	2	1
MOV direct, @Ri	2	2
MOV direct, #data	3	2
MOV @Ri, A	1	1
MOV @Ri, direct	2	2
MOV @Ri, #data	2	1
MOV DPTR, #data(16)	3	2
MOVX A, @Ri	1	2
MOVX A, @DPTR	1	2
MOVX @Ri, A	1	2
MOVC @DPTR, A	1	2
PUSH direct	2	2
POP direct	2	1
XCH A, Rn	1	1
XCH A, direct	2	1
XCH A, @Ri	1	1
XCHD A, @Ri	1	1

5.16 Boolean Variable Instructions Used by 8051

Mnemonic	Byte	Machine Cycle
CLR C	1	1
CLR bit	2	1

SETB C	1	1
SETB bit	2	1
CPL C	1	1
CPL bit	2	1
ANL C, bit	2	2
ANL C, /bit	2	2
ORL C, bit	2	2
C, /bit	2	2
MOV C, bit	2	1
MOV bit, C	2	2
JC rel	2	2
JNC rel	2	2
JB bit, rel	3	2
JNB bit, rel	3	2
JBC bit, rel	3	2

5.17 Program Branching Instruction

ACALL	addr11	2	2
LCALL	addr16	3	2
RET		1	2
RETI		1	2
AJMP	addr 11	2	2
LJMP	addr 16	3	2
SJMP	rel	2	2
JMP	@A+DPTR	1	2
JZ	rel	2	2
JNZ	rel	2	2
CJNE	A, direct, rel	3	2
CJNE	A, #data, rel	3	2
CJNE	Rn, #data, rel	3	2
CJNE	@Ri, #data, rel	3	2
DJNZ	Rn, rel	2	2
DJNZ	direct, rel	3	2
NOP		1	2

5.18 Loop Instructions

Repeating a sequence of instructions, a certain number of times is called a loop. In 8051, the loop action is performed by the instruction,

```
DJNZ      reg,    Label
```

In this instruction, the register is decremented one from a loaded number. If the result is not 0, the program jumps to the target address referred by the label.

Sample Program

```
    MOV     a,      #0      ; A = 0 Clear the accumulator
    MOV     R2,     #10     ; R2 = 10d
Again:
    ADD     A,      #03     ; A = A + 03d = 03d
    DJNZ    R2,     AGAIN
    MOV     R5,     A       ; Store the result in R5. R5 = 30
```

In the above program R3 register is used as counter. R2 is set to 10 initially. In labelAGAIN, the accumulator gets add with 3. The contents of register are decremented one by the instruction DJNZ R2. The loop is repeated till R2 becomes 0. There are 10 loops because 10 is loaded in R2. So, 10 times 3 is added in Accumulator. Hence the result 30.

Jump Instructions

There are two types of Jump instructions used in 8051. They are,

i) Conditional jump instructions
ii) Unconditional jump instructions

Conditional Jump Instructions

These instructions check a condition and depending on the status of the condition control is transferred to some label. Al conditional jumps are short jumps, which means that the address of the target label must be within -128 to +127 bytes of the contents of the program counter.

Some conditional jump instructions used in 8051 are:

Instruction	**Action**
JZ label	Jump to label address if A = 0
JNZ label	Jump to label address if A is not equal to 0
DJNZ Rn, label	Decrement Rn and jump to label if Rn # 0
JC label	Jump to label address if Carry flag = 1

Instruction	Action
JNC label	Jump to label address if Carry flag = 0
JB bit, label	Jump to label if bit = 1
JNB bit, label	Jump to label if bit = 0
JBC bit, label	Jump to label if bit = 1 and clear bit to 0

Instruction 1

```
JZ   label     ; Jump to label address if A = 0
```

In this instruction the content of register is checked. If it is 0, it jumps to the address of label. Otherwise it goes to next instruction.

```
MOV    A,      R0      ; A= R0
JZ     OVER            ; Jump to label OVER if A = 0
MOV    a,      R1      ; A = R1
JZ     OVER            ; Jump to label OVER if A = 0
```

Over

In the above program, if either R0 or R1 is 0, it jumps to the label OVER. This instruction is only to check the contents of register A.

Instruction 2

```
JNClabel     ; Jump to label address if Carry CY = 0
```

In this instruction the Carry bit CY in Program Status Word (PSW) is checked. If it is 0, it jumps to the address of label. Otherwise it goes to next instruction. This instruction is mainly used in arithmetic operations.

```
MOV    A,      #0      ; A= 0
MOV    R5,     A       ; A= R5
ADD    A,      #79h    ; A = 0 + 79h = 79h
JNC    OVER            ; Jump to label OVER if C = 0
INC    R5              ; Increment R5, R5 = R5 +1
OVER: ADD      A,      #0F5h A = A + F5h
```

In the above program, The Carry flag CY is cleared to 0. It jumps to the label OVER and adds next data F5h. Incise if carry is set to 1 the register R5 will incremented by one.

Unconditional Jump Instructions

The unconditional jump is a jump in which control is transferred unconditionally to the target location. In 8051 there are two unconditional jump instructions. They are

1. LJMP (Long Jump)
2. SJMP (Short Jump)

LJMP (Long Jump)

Ljmp is an unconditional long jump. It is a 3-byte instruction in which the first byte is the op-code and the other two bytes represent the 16-bit address of the target location. The two-byte target address allows a jump to any memory location from 0000h to FFFFh.

SJMP (Short Jump)

Sjmp is an unconditional short jump. It is a 2-byte instruction in which the first byte is the opcode and the other byte represent the 8-bit address of the target location. The 8-bit target address allows a jump to any memory location from 00h to FFh.

Forward and Backward Jump

Forward jump refers to the transfer of control from the current address of the Program Counter (PC) to a positive displacement. Address 0 to +127 bytes.

Backward jump refers to transfer of control from the current address of the Program Counter(PC) to a negative displacement. Address 0 to -128 bytes.

5.19 Call and Subroutine

Subroutines are separate program segments used to perform tasks that are need to be performed frequently. Subroutines are used to make the program more structured and to save memory space. It avoids repeating of same program. There are some instructions in 8051 which are used to call a subroutine and they are,

1. LCALL (Long Call)
2. ACALL (Absolute Call)

LCALL (Long Call)

LCALL is a 3-byte instruction in which the first byte is the opcode and the other two bytes represent the 16-bit address of the target label of subroutine. Therefore, LCALL can be used to call subroutines located from 64K bytes of memory space. (0000h to FFFFh). This instruction transfers the control to the subroutine program and after the execution of subroutine it come back to the instruction immediately below the LCALL. Whenever the processor executed the instruction LCALL, it automatically saves the address of the instruction immediately below LCALL to the Stack.

Every subroutine has the last instruction RET. This instruction transfers the control back to the address specified by the stack memory location. Address of the stack memory location is hold by the stack pointer.

Sample Program

Write a program to toggle all the bits of port 1 by sending it the values 55h and AA continuously. Add time delay between each toggle operation.

```
.org 0
```

Back

```
MOV      A,     #55h   ; A = 55h = 0 1 0 1 0 1 0 1
MOV      P1,    A      ; P1 = A = 0 1 0 1 0 1 0 1
LCALL DELAY            ; DELAY subroutine is called
MOV      A,     #0AAh  ; A = AAh = 1 0 1 0 1 0 1 0
MOV      P1,    A      ; P1 = A = 1 0 1 0 1 0 10
LCALL DELAY            ; DELAY subroutine is called
SJMP     BACK
ORG      3000h
```

Delay

```
MOV      R5,    #0FFh  ; R5 = FFh = 1 1 1 1 1 1 1 1
AGAIN: DJNZ R5, AGAIN ; Decrement R5. Repeat till R5 = 0
RET
END
```

In the above program the alternative pins of port 1 is set to 1 by loading 55h. After loading to port 1 a subroutine delay is called which is in address of 3000h. This subroutine is used to generate a time delay. During this delay the port 1 pins hold the data. In this DELAY subroutine the register R5 is loaded with FFh. DNJZ R5 makes a loop. This instruction decrement R5 and the loop is repeated till R5 becomes 0. When R5 becomes 0, The next instruction is executed where LCALL subroutine is called. It loads AAh to port 1, which means toggle all port 1 pins. Again, the subroutine DELAY is called. The same action is repeated again and again by the jump instruction SJMP BACK.

ACALL (Absolute Call)

ACALL is a 2-byte instruction in which the first byte is the opcode and the second byte represent the 8-bit address of the target label of subroutine. Therefore, aCALL can be used to call subroutines located from 2K bytes of memory space. (00h to FFh). Like LCALL instruction, this instruction also transfers the control to the subroutine program and after the execution of subroutine it come back to the instruction immediately below the ACALL. Whenever the processor executed the instruction ACALL, it automatically saves the address of the instruction immediately below ACALL to the Stack.

ACALL subroutine has the last instruction RET. This instruction transfers the control back to the address specified by the stack memory location. Address of the stack memory location is hold by the stack pointer.

5.20 8279 Programmable Key Board and Display Interface

The 8279 is a general-purpose programmable keyboard / display interface specially developed for processors. The 8279 has two sections namely keyboard section and display section. The function of the keyboard section is to interface the keyboard which is used as input device for the processor (CPU). The purpose of the display section is to drive LED's some important features are simultaneous keyboard display operations, scanned sensor mode, scanned keyboard mode, 8 – character keyboard FIFO, strobed input entry mode, 2 – key lock out or N – key roll over with contact debounce, single 16 – character display, dual 8 or 16 numerical display, interrupt output on key entry, programmable scan timing and mode programmable from CPU.

Signal description of 8279

The 8279 is a 40 pin DIP IC. The pin diagram and signal flow diagram are shown in figure 5.16a

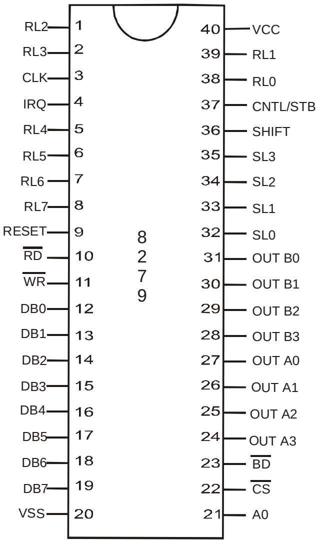


Fig 5.1a - Pin diagram of 8279

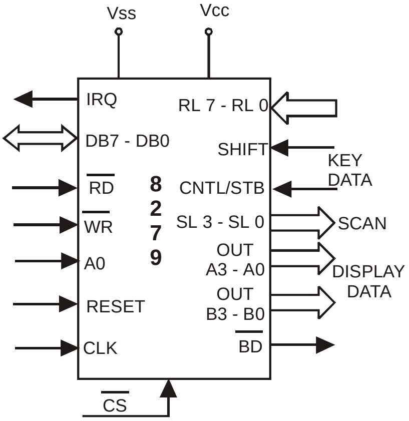


Fig 5.1b - Signal flow diagram of 8279.

Bi-directional data bus (DB7 – DB0)

These are 8-bit lines used for data transfer between 8279 and the CPU. Common words are also transmitted through these lines.

Read (RD)

It is an active low signal which enables the read operation.

Write (WR)

It is an active low signal which enables the write operation.

Reset

When this pin is set in high it resets the 8279.

Clock (CLK)

The clock signal is used to generate internal timing for the system.

Interrupt Request (IRQ)

When this line goes to high indicates that a valid key data is available in the FIFO RAM.

Return Lines (RL7 – RL0)

These lines are inputs to 8279. They are connected to the columns of the matrix keyboard. These lines have active internal pull-up resistors to keep high until a switch closure pulls one of the lines low.

Buffer Address (A0)

When this line goes low it indicates data transfer takes place. a high on this line indicates command word.

Chip Select (CS)

It is an active low signal, which enables the communication between 8279 and processor (CPU).

Shift

The shift line reads the status of shift key of the key board.

The control line reads the status of control key. It is high until a switch closure pulls it low. In the strobed input mode this line acts as a strobe line that enters the data into the FIFO RAM.

Scan Lines (SL3 – SL0)

These four scan lines are used to scan the key switch or sensor matrix and the display digits.

Display outputs Out A0 – Out A3 and Out B0 – Out B3

These 8 output lines (two 4-bit output ports) provide the 8 – bit segment information for the display digits.

5.21 Block diagram of 8279

The block diagram of 8279 is shown in figures. It consists of four major sections. They are

1) Key board section
2) Display section
3) Scan section
4) CPU interface section

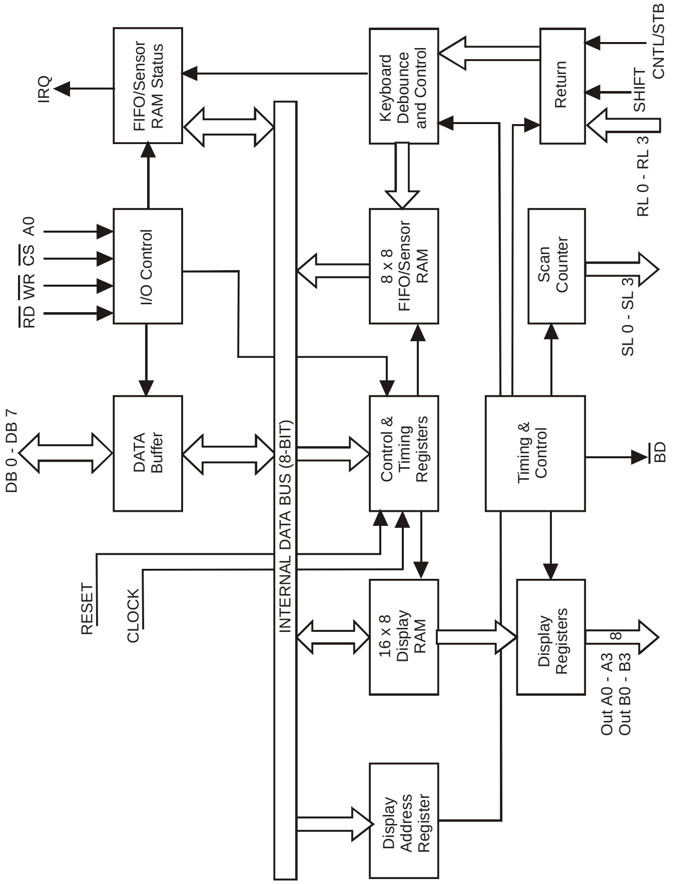


Fig. 5. 2 Block Diagram of 8279

Key board section

The Key board section includes return buffers, key board debounce and control and 8 x 8 FIFO / Sensor RAM. The return lines RL0 – RL7 are used to form the columns of a key board matrix. It has two additional inputs shift and control / strobe. The key board debounce and control unit debounce the key board entries. The two operating modes of key board section and 2 keys lock out and N – key roll over. In the 2 – key lock out mode, if two keys are pressed simultaneously, only the first key is re-cognized. In the N – key roll over mode simultaneously keys are recognized and their codes are stored in FIFO RAM.

The key board section also hasan 8 x 8 FIFO (First In first out) RAM. The 8 x 8 FIFO RAM can store eight key board entries. The status of the shift key and control key are also stored along with key code. When a key is pressed, the corresponding 8 – bit key code is loaded into the first location of the FIFO RAM.

Then the interrupt request line (IRQ) is made high to indicate that the FIFO RAM is not empty.

Display section

The display section consists of 16 x 8 display RAM and display address registers. The display address registers hold the address of the word currently being writer or read by the CPU and two 4 – bit nibbles being displayed.

The 8279 can drive a maximum of 16 display digits. the 16 x 8 (16 numbers of 8-bit buffer RAM) display RAM holds the 8 – bit data for 16 display digits. The 16 display digits are multiplexed.

The display section of 8279 has eight output lines divided into two groups Out A0 – A3 and Out B0 – B3. These lines provide the 8 – bit segment information for the display digits. The blank line BD is used to blank the display.

Scan section

The scan section consists of a scan counter and four scan lines (SL0 – SL3). This scan counter can be operated in two modes namely encoded mode and Decoded mode. In encoded mode, the counter provides a binary count which can be decoded to provide the scan lines for the key board or display. (i.e.). The scan lines periodically output 0 to 15).

In the decoded mode the scan counter itself is a decoder. (That is the least significant two bits are decoded and decoded outputs are available in the scan lines SL3 – SL0).

The four scan lines are common to key board and display. These lines are used to form the digit drivers of a multiplexed display and the rows of a matrix key board.

CPU Interface section

The CPU interface section consists of Data buffer, I/O control, control and timing registers, One Interrupt Request line (IRQ). The data buffer has 8 – bit bi – directional data lines DB7 – DB0 to transfer data between 8279 and the CPU of the processor. The I/O control signals RD, WR, CS, and A0 are used for read / write data to 8279. The 8279 has an interrupt request line (IRQ). this signal is used to interrupt the processor to indicate the availability of data.

Programming of 8279

The 8279 can be programmed to perform various functions through eight command words. D5, D6 and D7 are the code of command word.

Key board / Display mode set(Code 000)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	D	D	K	K	K

Code

Where DD specifies the display mode and
KKK specifies the keyboard mode.

D D
0 0 8 No. of 8-bit character display mode - left entry
0 1 16 No. of 8-bit character display mode - left entry
1 0 8 No. of 8-bit character display mode - right entry
1 1 16 No. of 8-bit character display mode - right entry

K K K	
0 0 0	- Encoded Scan keyboard - 2 key lockout
0 0 1	- Decoded Scan keyboard - 2 key lockout
0 1 0	- Encoded Scan keyboard - N key Roll over
0 1 1	- Decoded Scan keyboard - N key Roll over
1 0 0	- Encoded Scan sensor matrix
1 0 1	- Decoded Scan sensor matrix
1 1 0	- Strobed input, Encoded display scan
1 1 1	- Strobed input, Decoded display scan

Program Clock(Code 001)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	P	P	P	P	P

Code

Where P P P P P are the prescaler value from 2 to 31. The programmable prescaler divides the input clock to get basic internal frequency around 100KHz, so that the specified scan and debounce times are obtained.

Read FIFO / Sensor RAM(Code 010)

D7	D6	D5	D4	D3	D2	D1	D0
0	1	0	AI	X	A	A	A

Code

Where X is don’t care, AI stands for auto increment and AAA is sensor RAM address. 3-bit of AAA is used to address eight location of FIFO/Sensor RAM. This command sets up the 8279 to read the FIFO/Sensor RAM and used in the case of sensor matrix mode.

Read display RAM(Code 011)

D7	D6	D5	D4	D3	D2	D1	D0
0	1	1	AI	A	A	A	A

Code

Where AI is for auto increment flag and AAAA is the address of display RAM. 4-bit of AAAA is used to address 16 locations of display RAM. This command sets up the 8279 to read the display RAM.

Write Display RAM(Code 100)

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	AI	A	A	A	A

Code

Where AI is for auto increment flag and AAAA is the address of display RAM. 4-bit of AAAA is used to address 16 locations of display RAM. This command sets up the 8279 to write the display RAM.

Display write inhibit / blanking(Code 101)

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	X	IW	IW	BL	BL

CODE

Where IW is for inhibit write and BL stands for blank. This command is used to inhibit writing in A or B nibble of the display RAM and to blank the display selectively.

Write Display RAM(Code 110)

D7	D6	D5	D4	D3	D2	D1	D0
1	1	0	CD	CD	CF	CF	CA

CODE

Where CD is clear display, CF is Clear FIFO Status and CA is clear all. This command is used to clear the display RAM and the FIFO status at the time of initialisation. The display RAM can be set to all ones, all zeros are hex 20 as follows,

CD	CD	CD	
1	0	X	All zeros
1	1	0	AB = 20h (0010 0000b)
1	1	1	All ones
0	X	X	Inhibit clear display

End Interrupt(Code 111)

D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	E	X	X	X	X

CODE

Where X is don't care. In the sensor matrix mode, this command lowers the INT line and enables further writing into RAM. In the N key roll over mode, this command forces the chip in a special error mode if the E bit is set to 1.

5.22 Interfacing Stepper Motor

Stepper motor is a special type of DC motor. For every electrical pulse given stepper motor rotates certain no. of degrees. It is used in digital control systems such as robotics, dot matrix printers, disk drives etc. The stepper motor has a permanent magnet rotor shaft surrounded by a stator. The stator windings are energised by electrical pulses. The most common stepper motors have four stator windings. Series of pulses are applied to the input of stator windings. The axis of air gap magnetic field setup around according to the input pulses. The axis of rotor magnetic field follows the stator axis of air gap magnetic field.

The direction of rotation is formed by the stator poles. When the polarity of stator windings is changed the direction of the current also changed and causing the reverse rotation of rotor. The sequence of pulses applied to each stator winding, the direction of rotating is determined. Degree of rotation is determined by the no. of stator poles and no. of rotor poles.

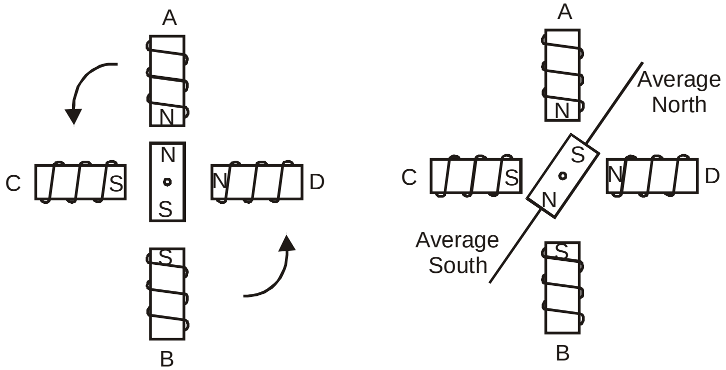


Fig. 5.3 Rotor alignment in stepper motor

From the figure the switching is called 4-step switching sequence. After completing a sequence, the same windings will be ON and rotation continuous. No. of steps per revolution is depend upon the degree of rotation. degree of rotation = 360 degree / no. of rotor pole x no. of stator poles

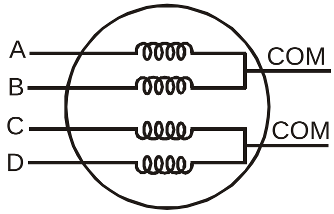


Fig. 5.4 Stator winding in stepper motor

Example

No. of rotor poles = 5

No of stator poles = 4

Degree of rotation = 360o/ 5 x 4

= 18o

No. of steps/revolution = 360/18

= 20 steps

Interfacing Stepper Motor

Since the 8051 lacks sufficient current to drive the stepper motor windings, we must use a driver to energize the stator. Instead we could have used transistors as drivers. The port of 8051 is buffered and connected to stator winding of stepper motor through separate drivers. The sequence of pulses is programmed to determine the direction of rotation. The clockwise and anti-clockwise rotations are done by changing the steps sequence of the stator windings.

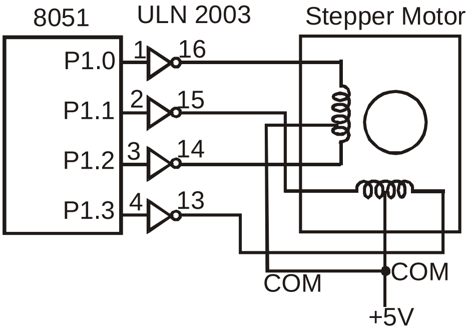


Fig 5.5 Interfacing Stepper motor to 89C51

Software Design

Program to rotate the stepper motor continuously. Stepper motor connected to 89C51.

```
    MOV         A, #66h ; Load step sequence in Accumulator
```

Back

```
    MOV P1, A             ; Issue sequence to motor
    RR  A                 ; Rotate right clockwise
    ACALL STEP            ; Goto subroutine called STEP
    SJMP        BACK      ; Continue the rotation
    STEP:                 ; This subroutine is to determine the delay between steps.
    MOV R2, #64h          ; 100 decimal loaded in R2
    S1:  MOV    R3, #FFh ; 255 decimal loaded in R3
    S2:  DJNZ R3, S2       ; Keep decrement R3 to 0
    DJNZ R2, S1            ; Keep decrement R2 to 0
    RET
```

5.23 Interfacing DC Motor

The microcontroller-based DC motor speed control system is shown in fig. To measure the speed of the motor, a circular disk with several slots is attached to the spindle of the motor. An LED and photo transistor assembly are used to generate pulse train with frequency related to the speed of the motor. Each rotation of the spindle, the photo transistor is obstructed to the light emitted by the LED n times. Where n is the number of slots on the circular disk. By using a suitable pulse shaper, it is converted into a TTL compatible square wave. The square wave is applied to 89C51 through RXD pin. No. of square waves are counted to measure the speed of motor.

To alter the speed a variable DC voltage is applied to the armature of motor. Variable DC voltage is generated with a help of DAC. Input to the DAC should be programmed in 89C51. The desired speed is set by program. The 89C51 keep varying input to DAC till measured speed is equal to desired speed. Measured speed and desired speed are displayed in LED display.

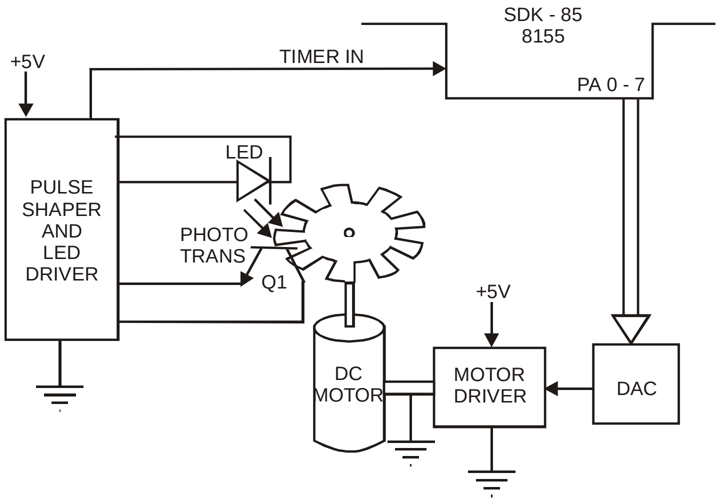


Fig. 5.6 Speed Control of DC motor