The various techniques to specify data for instructions are:

(i) 8-bit or 16-bit data may be directly given in the instruction itself.
(ii) The address of the memory location, I/O port or I/O device, where data resides, may be given in the instruction itself.
(iii) In some instructions only one register is specified. The content of the specified register is one of the operands. It is understood that the other operand is in the accumulator.
(iv) Some instructions specify two registers. The contents of the registers are the required data.
(v) In some instructions data is implied. The most instructions of this type operate on the content of the accumulator.

Due to different ways of specifying data for instructions, the machine codes of all instructions are not of the same length.

There are three types of the Intel 8085 instructions as described below:

(1) Single-Byte Instruction
(2) Two-Byte Instruction
(3) Three-Byte Instruction

This has already been discussed in Chapter 3; see details.

4.3 ADDRESSING MODES

Each instruction requires certain data on which it has to operate. It has already been explained that there are various techniques to specify data for instructions. These techniques are called *addressing modes*. Intel 8085 uses the following addressing modes:

1. Direct addressing
2. Register addressing
3. Register indirect addressing
4. Immediate addressing.

4.3.1 Direct Addressing

In this mode of addressing the address of the operand (data) is given in the instruction itself.

Examples are:

(1) STA 2400 H — Store the content of the accumulator in the memory location 2400 H.

32, 00, 24 — The above instruction in the code form.

In this instruction 2400H is the memory address where data is to be stored. It is given in the instruction itself. The 2nd and 3rd bytes of the instruction specify the address of the memory location. Here, it is understood that the source of the data is accumulator.

(2) IN 02 — Read data from the port C.

DB, 02 — Instruction in the code form.

In this instruction 02 is the address of the port C of an I/O port from where the data is to be read. Here, it is implied that the destination is the accumulator. The 2nd byte of the instruction specifies the address of the port.

4.3.2 Register Addressing

In register addressing mode the operand is in one of the general purpose registers. The opcode specifies the address of the register(s) in addition to the operation to be performed. Examples are:

(1) MOV A, B	Move the content of register B to register A.
78	The instruction in the code form.
(2) ADD B	Add the content of register B to the content of register A.
80	The instruction in the code form.

In Example 1 the opcode for MOV A, B is 78H. Besides the operation to be performed the opcode also specifies source and destination registers. The opcode 78H can be written in binary form as 01111000. The first two bits, i.e. 01 are for MOV operation, the next three bits 111 are the binary code for register A, and the last three bits 000 are the binary code for register B.

In Example 2 the opcode for ADD B is 80H. In this instruction one of the operands is register B (its content is one of the data) which is indicated in the instruction itself. In this type of instruction (arithmetic group) it is understood that the other operand is in the accumulator. The opcode 80H in the binary form is 10000000. The first five bits, *i.e.* 10000 specify the operation to be performed, i.e. ADD. The last three bits 000 are the binary code for register B for 8085 microprocessor.

4.3.3 Register Indirect Addressing

In this mode of addressing the address of the operand is specified by a register pair. Examples are:

(1) LXI H, 2500 H	Load H-L pair with 2500 H.
MOV A, M	Move the content of the memory location, whose address is in H-L pair (*i.e.* 2500 H) to the accumulator
HLT	Halt.

In the above program the instruction MOV A, M is an example of register indirect addressing. For this instruction the operand is in the memory. The address of the memory is not directly given in the instruction. The address of the memory resides in H-L pair and this has already been specified by an earlier instruction in the program, i.e. LXI H, 2500 H.

(2) LXI H, 2500 H	Load the H-L pair with 2500 H.
ADD M	Add the content of the memory location, whose address is in H-L pair (i.e. 2500 H), to the content of the accumulator.
HLT	Halt.

In this program the instruction ADD M is an example of register indirect addressing.

4.3.4 Immediate Addressing

In immediate addressing mode the operand is specified within the instruction itself, examples are:

(1) MVI A, 05	Move 05 in register A.
3E, 05	The instruction in the code form.
(2) ADI 06	Add 06 to the content of the accumulator.
C6, 06	The instruction in the code form.

In these instructions the 2nd byte specifies data.

(3) LXI H, 2500 is an example of immediate addressing. 2500 is 16-bit data which is given in the instruction itself. It is to be loaded into H-L pair.

4.3.5 Implicit Addressing

There are certain instructions which operate on the content of the accumulator. Such instructions do not require the address of the operand. Examples are: CMA, RAL, RAR etc.

4.4 STATUS FLAGS

There is a set of five flip-flops which indicate status (conditions) arising after the execution of arithmetic and logic instructions. It has already been discussed in Section 3.1.3.

4.5 SYMBOLS AND ABBREVIATIONS

The symbols and abbreviations which have been used while explaining Intel 8085 instructions are as follows:

Symbol/Abbreviations	*Meaning*
addr	16-bit address of the memory location.
data	8-bit data.
data 16	16-bit data.
r, r_1, r_2	One of the registers A, B, C, D, E, H or L,
A, B, C, D, H, L	8-bit register
A	Accumulator
H-L	Register pair H-L
B-C	Register pair B-C
D-E	Register pair D-E
PSW	Program Status Word
M	Memory whose address is in H-L pair
H	Appearing at the end of a group of digits specifies hexadecimal, e.g. 2500H.
rp	One of the register pairs. The representation of a register pair is made as described below: B represents B-C pair, B is high order register and C low order register. D represents D-E pair, D is high order register and E is low order register. H represents H-L pair, H is high order register and L low order register. SP represents 16-bit stack pointer, SPH is high order 8 bits and SPL low order 8 bits of register SP.
rh	The high order register of a register pair.
rl	The low order register of a register pair.
PC	16-bit program counter, PCH is high order 8 bits and PCL low order 8 bits of register PC.
CS	Carry status.
[ ]	The contents of a register identified within bracket.
[[ ]]	The content of the memory location whose address is in the register pair identified within brackets.
^	AND operation
v	OR operation

∀ or ⊕	EXCLUSIVE-OR
←	Move data in the direction of arrow.
⇔	Exchange contents.

4.6 INTEL 8085 INSTRUCTIONS

Some of Intel 8085 instructions are frequently, some occasionally and some seldom used by the programmer. It is not necessary that one should learn all the instructions to understand simple programs. The beginner can learn about 15 to 20 important instructions such as MOV, MVI, LXI, LDA, LHLD, STA, SHLD, ADD, ADC, SUB, JMP JC, JNC, JZ, JNZ, INX, DCR, CMP etc., and start to understand simple programs given in Chapter 6. While learning programs he can understand new instructions which he has not learnt earlier.

The operation codes (opcodes) are given in Appendix II. The explanations of the most instructions are given in the subsequent subsections.

4.6.1 Data Transfer Group MOV r_1, r_2 (Move data; Move the content of the one register to another)

$[r_1] \leftarrow [r_2]$. States: 4. Flags: none. Addressing: register.Machine cycle: 1.

The content of register r_2 is moved to register r_1. For example, the instruction MOV A, B moves the content of register B to register A. The instruction MOV B, A moves the content of register A to register B. The time for the execution of this instruction is 4 clock period. One clock period is called *State*. No flag is affected.

MOV r, M. (Move the content of memory to register).

$[r] \leftarrow [[H - L]]$. States: 7. Flag none. Addressing: register indirect. Machine cycles: 2.

The content of the memory location, whose address is in H-L pair, is moved to register r.

Example

LXI H, 2000 H	Load H-L pair by 2000H.
MOV B, M	Move the content of the memory location 2000H to register B.
HLT	Halt.

In this example the instruction LXI H, 2000 H loads H-L pair with 2000 H which is the address of a memory location. Then the instruction MOV B, M will move the content of the memory location 2000H to register B.

MOV M, r. (Move the content of register to memory).

$[[H-L]] \leftarrow [r]$. States: 7. Flags: none. Addressing: reg. indirect. Machine cycles: 2.

The content of register r is moved to the memory location addressed by H-L pair. For example, MOV M, C moves the content of register C to the memory location whose address is in H-L pair.

MVI r, data. (Move immediate data to register).

$[r] \leftarrow$ data. States: 7. Flags: none. Addressing: immediate. Machine cycle: 2.

The 1st byte of the instruction is its opcode, The 2nd byte of the instruction is the data which is moved to register r. For example, the instruction MVI A, 05 moves 05 to register A. In the code form it is written as 3E, 05. The opcode for MVI A is 3E and 05 is the data which is to be moved to regiser A.

MVI M, data. (Move immediate data to memory).

[[H - L]] ← data. States: 10. Flags: none. Addressing: immediate/reg. indirect. Machine cycle: 3.

The data is moved to memory location whose address is in H-L pair.

Example

LXI H, 2400H	Load H-L pair with 2400H.
MVI M, 08	Move 08 to the memory location 2400H.
HLT	Halt.

In the above example the instruction LXI H, 2400 H loads H-L pair with 2400 H which is the address of a memory location. Then the instruction MVI M, 08 will move 08 to memory location 2400H. In the code form it is written as 36, 08. The opcode for MVI M is 36 and 08 is the data which is to be moved to the memory location 2400H.

LXI rp, data 16. (Load register pair immediate).

[*rp*] ← data 16 bits, [*rh*] ← MSBs, [*rl*] ← 8 LSBs of data.

States: 10. Flags: none. Addressing: immediate. Machine cycles: 3.

This instruction loads 16-bit immediate data into register pair *rp*. This instruction is for register pair; only high order register is mentioned after the instruction. For example, H in the instruction LXI H stands for H-L pair. Similarly, LXI B is for B-C pair. LXI H, 2500H loads 2500H into H-L pair. H with 2500H denotes that the data 2500 is in hexadecimal. In the code form it is written as 21, 00, 25. The 1st byte of the instruction 21 is the opcode for LXI H. The 2nd byte 00 is 8 LSBs of the data and it is loaded into register L. The 3rd byte 25 is 8 MSBs of the data and it is loaded into register H.

LDA addr. (Load Accumulator direct).

[A] ← [*addr*]. States: 13. Flags: none. Addressing: direct. Machine cycles: 4.

The content of the memory location, whose address is specified by the 2nd and 3rd bytes of the instruction; is loaded into the accumulator. The instruction LDA 2400 H will load the content of the memory location 2400 H into the accumulator. In the code form it is written as 3A, 00, 24. The 1st byte 3A is the opcode of the instruction. The 2nd byte 00 is of 8 LSBs of the memory address. The 3rd byte 24 is 8 MSBs of the memory address.

STA Addr. (Store accumulator direct).

[*addr*] ← [A]. States: 13. Flags: none. Addressing: direct. Machine cycles: 4.

The content of the accumulator is stored in the memory location whose address is specified by the 2nd and 3rd byte of the instruction. STA 2000H will store the content of the accumulator in the memory location 2000H.

LHLD addr. (Load H-L pair direct).

[L] ← [*addr*], [H] ← [*addr* + 1]. States: 16. Flags: none. Addressing: direct. Machine cycles: 5.

The content of the memory location, whose address is specified by the 2nd and 3rd bytes of the instruction, is loaded into register L. The content of the next memory location is loaded into register H. For example, LHLD 2500H will load the content of the memory location 2500 H into register L. The content of the memory location 2501H is loaded into register H.

SHLD addr. (Store H-L pair direct)

[*addr*] ← [L], [*addr* + 1] ← [H]. States : 16. Flags: none. Addressing: direct. Machine cycles: 5.

The content of register L is stored in the memory location whose address is specified by the 2nd and 3rd bytes of the instruction. The content of register H is stored in the next memory location. For example, SHLD 2500H will store the content of register L in the memory location 2500H. The content of register H is stored in the memory location 2501H.

LDAX rp. (LOAD accumulator indirect)

[A] ← [[*rp*]]. States; 7. Flags; none. Addressing: register indirect. Machine cycles: 2.

The content of the memory location, whose address is in the register pair *rp*, is loaded into the accumulator. For example, LDAX B will load the content of the memory location, whose address is in the B-C pair, into the accumulator. This instruction is used only for B-C and D-E register pairs.

STAX rp. (Store accumulator indirect)

[[*rp*]] ← [A]. States: 7. Flags: none. Addressing: register indirect. Machine cycles: 2.

The content of the accumulator is stored in the memory location whose address is in the register pair *rp*. For example, STAX D will store the content of the accumulator in the memory location whose address is in D-E pair. This instruction is true only for register pairs B-C and D-E.

XCHG. (Exchange the contents of H-L with D-E pair)

[H-L]↔ [D-E]. States: 4. Flags: none. Addressing: register. Machine cycles: 1.

The contents of H-L pair are exchanged with contents of D-E pair.

4.6.2 Arithmetic Group

ADD r. (Add register to accumulator)

[A] ← [A] + [*r*]. States: 4. Flags: all. Addressing: register. Machine cycle: 1.

The content of register *r* is added to the content of the accumulator, and the sum is placed in the accumulator.

ADD M. (Add memory to accumulator)

[A] ← [A] + [[H-L]]. States: 7. Flags: all. Addressing: reg. indirect, Machine cycles: 2.

The content of the memory location addressed by H-L pair is added to the content of the accumulator. The sum is placed in the accumulator.

ADC r. (Add register with carry to accumulator.)

[A] ← [A] + [*r*] + [CS]. States: 4. Flags: all. Addressing: register. Machine cycles: 1.

The content of register *r* and carry status are added to the content of the accumulator. The sum is placed in the accumulator.

ADC M. (Add memory with carry to accumulator)

[A] ← [A] + [[H-L]] [CS]. States: 7. Flags: all. Addressing: reg. indirect. Machine cycles: 2.

The content of the memory location addressed by H-L pair and carry status are added to the content of the accumulator. The sum is placed in the accumulator.

ADI data. (Add immediate data to accumulator)

[A] ← [A] + data. States: 7. Flags: all. Addressing : immediate. Machine cycles: 2.

The immediate data is added to the content of the accumulator. The 1st byte of the instruction is its opcode. The 2nd byte of the instruction is data, and it is added to content of the accumulator. The sum is placed in the accumulator. For example, the

instruction ADI 08 will add 08 to the content of the accumulator and place the result in the accumulator. In code form the instruction is written as C6, 08.

ACI data. (Add with carry immediate data to accumulator)

[A] ← [A] + data + [CS]. States: 7. Flags: all. Addressing: immediate. Machine cycles: 2

The 2nd byte of the instruction (which is data) and the carry status are added to the content of the accumulator. The sum is placed in the accumulator.

DAD rp. (Add register paid to H-L pair)

[H-L] ← [H-L] + [*rp*]. States: 10. Flags: CS. Addressing: register. Machine cycles: 3.

The contents of register pair *rp* are added to the contents of H-L pair and the result is placed in H-L pair. Only carry flag is affected.

SUB r. (Subtract register from accumulator)

[A] ← [A] – [*r*]. States: 4 Flags: all. Addressing: register. Machine cycles: 1.

The content of register *r* is subtracted from the content of the accumulator, and the result is placed in the accumulator.

SUB M. (Subtract memory from accumulator).

[A] ← [A] – [[H-L]]. States: 7. Flags: all. Addressing: reg. indirect. Machine cycles: 2

The content of the memory location addressed by H-L pair is subtracted from the content of the accumulator. The result is placed in the accumulator.

SBB r. (Subtract register from accumulator with borrow).

[A] ← [A] – [*r*] – [CS]. States: 4. Flags: all. Addressing: register. Machine cycles: 1.

The content of register *r* and carry status are subtracted from the content of the accumulator. The result is placed in the accumulator.

SBB M. (Subtract memory from accumulator with borrow).

[A] ← [A] – [[H-L]] – [CS]. States: 7. Flags: all. Addressing: reg. indirect. Machine cycles: 2.

The content of the memory location addressed by H-L pair and carry status are subtracted from the content of the accumulator. The result is placed in the accumulator.

SUI data. (Subtract immediate data from accumulator)

[A] ← [A] – data. States: 7. Flags: all. Addressing: immediate. Machine cycles: 2.

The 2nd byte of the instruction is data. It is subtracted from the content of the accumulator. The result is placed in the accumulator. For example, the instruction SUI 05 will subtract 05 from the content of the accumulator and place the result in the accumulator. In the code form the above instruction is written as D6, 05.

SBI data. (Subtract immediate data from accumulator with borrow).

[A] ← [A] – data – [CS]. States: 7. Flags: all. Addressing: immediate. Machine cycles: 2

The data and carry status are subtracted from the content of the accumulator. The result is placed in the accumulator

INR r. (Increment register content)

[*r*] ← [*r*] + 1. States : 4. Flags: all except carry flag. Addressing : register. Machine cycle : 1.

The content of register *r* is incremented by one. All flags except CS are affected.

INR M. (Increment memory content)

[[H-L]] ← [[H-L]] + 1. States: 10. Flags: all except carry flag. Addressing: reg. indirect.

Machine cycles: 3.

The content of the memory location addressed by H-L pair is incremented by one. All flags except CS are affected.

DCR r. (Decrement register content) [r] ← εε [r] − 1. States: 4. Flags: all except carry flag, Addressing: register. Machine cycles: 1.

The content of register *r* is decremented by one. All flags except CS are affected.

DCR M. (Decrement memory content)

[[H-L]] ← [[H-L]] − 1. States: 10. Flags: all except carry flag. Addressing: reg. indirect. Machine cycles: 3.

The content of the memory location addressed by H-L pair is decremented by one. All flags except CS are affected.

INX rp. (increment register pair)

[*rp*] ← [*rp*] + 1. States: 6. Flags: none. Addressing: register. Machine cycles : 1.

The content of the register pair *rp* is incremented by one. No flag is affected.

DCX rp (Decrement register pair)

[*rp*] ← [*rp*] − 1. States: 6. Flags: none. Addressing: register. Machine cycles: 1.

The content of the register pair *rp* is decremented by one. No flag is affected.

DAA. (Decimal adjust accumulator)

States: 4. Flags: all. Machine cycle : 1.

The instruction DAA is used in the program after ADD, ADI, ACI, ADC, etc instructions. After the execution of ADD, ADC, etc instructions the result is in hexadecimal and it is placed in the accumulator. The DAA instruction operates on this result and gives the final result in decimal system. It uses carry and auxiliary carry for decimal adjustment. 6 is added to 4 LSBs of the content of the accumulator if their value lies in between A and F or the AC flag is set to 1. Similarly, 6 is also added to 4 MSBs of the content of the accumulator if their value lies in between A and F or the CS flag is set to 1. All status flags are affected. When DAA is used data should be in decimal numbers.

4.6.3 Logical Group

The instructions of this group perform AND, OR, EXCLUSIVE-OR operations; compare, rotate or take complement of data in register or memory.

ANA r. (AND register with accumulator)

[A] ← [A] <185 > [*r*]. States: 4. Flags : all. Addressing: register. Machine cycles: 1.

The content of register *r* is ANDed with the content of the accumulator, and the result is placed in the accumulator. All status flags are affected. The flag CS is cleared, *i.e.* it is set to 0. Auxiliary carry flag AC is set to 1.

ANA M. (AND memory with accumulator)

[A] ← [A] ∧ [[H-L]]. States: 7. Flags: all. Addressing: reg. indirect. Machine cycles: 2.

The content of the memory location addressed by H-L pair is ANDed with the accumulator. The result is placed in the accumulator. All flags are affected. The CS flag is set to 0 and AC to 1.

ANI data. (AND immediate data with accumulator)

[A] ← [A] ∧ data. States: 7. Flags: all. Addressing: immediate. Machine cycles: 2.

The 2nd byte of the instruction is data, and it is ANDed with the content of the accumulator. The result is placed in the accumulator. The CS flag is set to 0 and AC to 1.

ORA r. (OR register with accumuiator)

[A] ← [A] ∨ [r] States: 4. Flags: all. Addressing: register. Machine cycles: 1.

The content of register *r* is ORed with the content of the accumulator. The result is placed in the accumulator. All status flags are affected. Carry and auxilary carry are cleared *i.e.* the CS and AC flags are set to 0.

ORA M. (OR memory with accumulator)

[A] ← [A] ∨ [[H-L]]. States: 7. Flags: all. Addressing: reg. indirect. Machine cycles: 2.

The content of the memory location addressed by H-L pair is ORed with the content of the accumulator. The result is placed in the accumulator. The CS and AC flags are set to 0.

ORI data. (OR immediate data with accumulator)

[A] ← [A] ∨ data. States: 7. Flags: all. Addressing: immediate. Machine cycles: 2.

The 2nd byte of the instruction is data, and it is ORed with the content of the accumulator. The result is placed in the accumulator. All status flags are affected. The CS and AC flags are set 0.

XRA r. (EXCLUSIVE - OR register with accumulator)

[A] ← [A] ∀□ [r] States: 4. Flags: all. Addressing: register. Machine cycles: 1.

The content of register *r* is EXCLUSIVE - ORed with the content of the accumulator. The result is placed in the accumulator. All status flags are affected. The CS and AC flags are set to 0.

XRA M. (EXCLUSIVE - OR memory with accumulator)

[A] ← [A] ∀ [[H-L]]. States: 7. Flags: all. Addressing: reg. indirect. Machine cycles: 2.

The content of the memory location addressed by H-L pair is EXCLUSIVE-ORed with the content of the accumulator. The result is placed in the accumulator. All status flags are affected. The CS and AC flags are set to 0.

XRI data. (EXCLUSIVE - OR immediate data with accumulator)

[A] ← [A] ∀ data. States: 7. Flags: all. Addressing: immediate. Machine cycles: 2.

The 2nd byte of the instruction is data, and it is EXCLUSIVE-ORed with the content of the accumulator. The result is placed in the accumulator. All flags are affected. The CS and AC flags are set to 0.

CMA. (Complement the accumulator)

[A] ← [$\overline{A}$]. States: 4. Flags: none. Machine cycles: 1. Addressing: implicit.

1's complement of the content of the accumulator is obtained, and the result is placed in the accumulator. To obtain the 1's complement of a binary number 0 is replaced by 1, and 1 by 0. For example, one's complement of 1100 is 0011.

CMC. (Complement the carry status)

[CS] ← [$\overline{CS}$]. States: 4. Flags: CS, Machine cycle: 1.

The CS flag is complemented. Other flags are not affected.

STC. (Set carry status)

[CS] ← 1. States: 4. Flags: CS. Machine cycles: 1.

The status flag CS is set to 1. Other flags are not affected.

CMP r. (Compare register with accumulator)

[A] − [*r*]. States 4. Flags: all. Addressing: register. Machine cycles: 1.

The content of register *r* is subtracted from the content of the accumulator and status flags are set according to the result of the subtraction. But the result is discarded. The content of the accumulator remains unchanged.

CMP M. (Compare memory with accumulator)

[A] − [[H-L]]. States: 7. Flags: all. Addressing: reg. indirect. Machine cycles: 2.

The content of the memory location addressed by H-L pair is subtracted from the content of the accumulator, and status flags are set according to the result of the subtraction. But the result is discarded. The content of the accumulator remains uncharged.

CPI data. (Compare immediate data with accumulator)

[A] − data. States: 7. Flags: all. Addressing: immediate. Machine cycles: 2.

The 2nd byte of the instruction is data, and it is subtracted from the content of the accumulator. The status flags are set according to the result of subtraction. But the result is discarded. The content of the accumulator remains unchanged.

RLC. (Rotate accumulator left)

$[A_{n+1}] \leftarrow [A_n]$, $[A_0] \leftarrow [A_7]$, $[CS] \leftarrow [A_7]$.

States: 4. Flags: CS. Machine cycles : 1. Addressing: implicit.

The content of the accumulator is rotated left by one bit. The seventh bit of the accumulator is moved to carry bit as well as to the zero bit of the accumulator. Only CS flag is affected. See Fig. 4.1

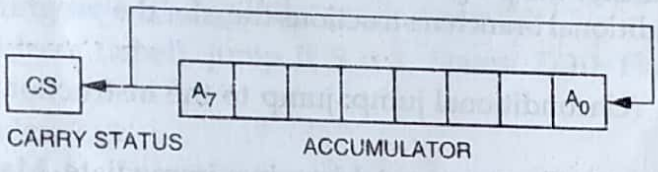

Fig. 4.1 Schematic Diagram for RLC

RRC. (Rotate accumulator right)

$[A_7] \leftarrow [A_0]$, $[CS] \leftarrow [A_0]$, $[A_n] \leftarrow [A_{n+1}]$.

States: 4. Flags: CS. Machine cycles : 1. Addressing: implicit.

The content of the accumulator is rotated right by one bit. The zero bit of the accumulator is moved to the seventh bit as well as to carry bit. Only CS flag is affected. See Fig. 4.2.

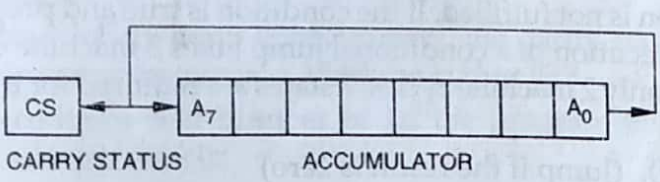

Fig. 4.2 Schematic Diagram for RRC

RAL. (Rotate accumulator left through carry)

$[A_{n+1}] \leftarrow [A_n]$, $[CS] \leftarrow [A_7]$, $[A_0] \leftarrow [CS]$.

States: 4. Flags: CS. Machine cycles: 1. Addressing: implicit.

The content of the accumulator is rotated left one bit through carry. The seventh bit of the accumulator is moved to carry, and the carry bit is moved to the zero bit of the accumulator. Only carry flag is affected. See Fig. 4.3.

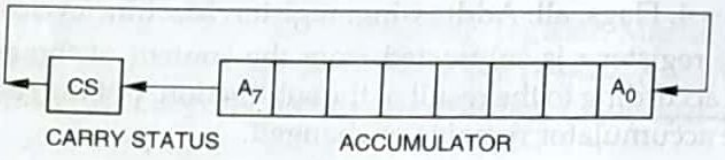

Fig. 4.3 Schematic Diagram for RAL

RAR. (Rotate accumulator right through carry)

$[A_n] \leftarrow [A_{n+1}], [CS] \leftarrow [A_0], [A_7] \leftarrow [CS]$

States: 4 Flags : CS. Machine cycle: 1. Addressing implicit.

The content of the accumulator is rotated right one bit through carry. The zero bit of the accumulator is moved to carry, and the carry bit to the seventh bit of the accumulator. Only CS flag is affected. See Fig. 4.4.

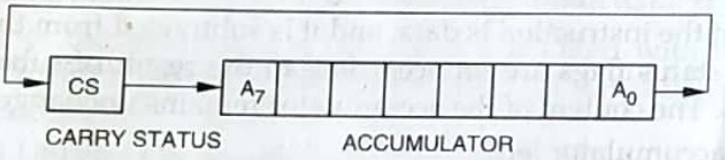

Fig. 4.4 Schematic Diagram for RAR

4.6.4 Branch Group

The instructions of this group change the normal sequence of the program. There are two types of branch instructions: conditional and unconditional. The conditional branch instructions transfer the program to the specified label when certain condition is satisfied. The unconditional branch instructions transfer the program to the specified label unconditionally.

JMP addr (label). (Unconditional jump: jump to the instruction specified by the address).

[PC] ← Label. States: 10. Flags: none. Addressing: immediate. Machine cycles: 3.

Byte 2nd and byte 3rd of the instruction give the address of the label where the program jumps. The address of the label is the address of the memory location for next instruction to be executed. The program jumps to the instruction specified by the address (label) unconditionally.

Conditional Jump addr (label). After the execution of the conditional jump instruction the program jumps to the instruction specified by the address (label) if the specified condition is fulfilled. The program proceeds further in the normal sequence if the specified condition is not fulfilled. If the condition is true and program jumps to the specified label, the execution of a conditional jump takes 3 machine cycles: 10 states. If condition is not true, only 2 machine cycles; 7 states are required for the execution of the instruction.

(i) **JZ addr (label).** (Jump if the result is zero)

[PC] ← address (label), jump if $Z = 1$. States: 7/10. Flags: none. Addressing: immediate.

Machine cycles: 2/3.

The program jumps to the instruction specified by the address (label) if the result is zero (*i.e.* the zero status Z = 1). Here the result after the execution of the preceding instruction is under consideration.

(ii) **JNZ addr (label).** (Jump if the result is not zero)

[PC] ← address (label), jump if Z = 0. States: 7/10. Flags: none. Addressing : immediate.

Machine cycles: 2/3.

The program jumps to the instruction specified by the address (label) if the result is non-zero (*i.e* the zero status Z = 0).

(iii) **JC addr (label).** (Jump if there is a carry)

[PC] ← address (label), jump if CS = 1. States: 7/10. Flags: none. Addressing : immediate. Machine cycles: 2/3.

The program jumps to the instruction specified by the address (label) if there is a carry (*i.e.* the carry status: CS = 1). Here the carry after the execution of the preceding instruction is under consideration.

(iv) **JNC addr (label).** (Jump if there is no carry)

[PC] ← address (label), jump if CS = 0. States: 7/10. Flags: none. Addressing : immediate. Machine cycles: 2/3.

The program jumps to the instruction specified by the address (label) if there is no carry (*i.e.* the carry states CS = 0).

(v) **JP addr (label).** (Jump if the result is plus)

[PC] ← address (label), jump if S = 0. States: 7/10. Flags: none. Addressing: immediate.

Machine cycles: 2/3.

The program jumps to the instruction specified by the address (label) if the result is plus.

(vi) **JM addr (label).** (Jump if the result is minus)

[PC] ← address (label), jump if S = 1. States: 7/10. Flags: none. Addressing: immediate.

Machine cycles: 2/3.

If the result is minus the program jumps to the instruction specified by the address (label).

(vii) **JPE addr (label).** (Jump if even parity)

[PC] ← address (label), jump if even parity: the parity status P = 1, States: 7/10. Flags: none. Addressing: immediate. Machine cycles: 2/3.

If the result contains even number of 1s, the program jumps to the instruction specified by the address (label).

(viii) **JPO addr (label).** (Jump if odd parity)

[PC] ← address (label), jump if odd parity ; the parity status P = 0, States: 7/10, Flags: none. Addressing: immediate, Machine cycles: 2/3.

If the result contains odd number of 1s, the program jumps to the instruction specified by the address (label).

CALL addr (label). (Unconditional CALL: call the subroutine identified by the address)

[[SP] − 1] ← [PCH], Save the address of the next instruction of the program in the stack.

[[SP] – 2] ← [PCL],
[SP] ← ([SP] – 2)
[PC] ← addr (label)

States: 18. Flags: none, Addressing: immediate/reg. indirect. Machine cycles: 5.

CALL instruction is used to call a subroutine. Before the control is transferred to the subroutine, the address of the next instruction of the main program is saved in the stack. The content of the stack pointer is decremented by two to indicate the new stacktop. Then the program jumps to subroutine starting at address specified by the label.

Conditional CALL addr (label)

[[SP] – 1] ← [PCH], [[SP] – 2] ← [PCL],
[PC] ← addr (label), [SP] ← ([SP– 2]).

States: 9/18. Flags: none. Addressing: immediate/reg. indirect. Machine cycles: 2/5. If the condition is true and program calls the specified subroutine, the execution of a conditional call instruction takes 5 machine cycles; 18 states. If condition is not true, only 2 machine cycles; 9 states are required for the execution of the instruction.

(i) CC addr (label) Call subroutine if carry status CS = 1.
(ii) CNC addr (label) Call subroutine if carry status CS = 0.
(iii) CZ addr (label) Call subroutine if the result is zero; the zero status Z = 1.
(iv) CNZ addr (label) Call subroutine if the result is not zero; the zero status Z = 0.
(v) CP addr (label) Call subroutine if the result is plus; the sign status S = 0.
(vi) CM addr (label) Call subroutine if the result is minus, the sign status S = 1.
(vii) CPE addr (label) Call subroutine if even parity; the parity status P = 1.
(viii) CPO addr (label) Call subroutine if odd parity; the parity status P = 0.

RET. (Return from subroutine)

[PCL] ← [[SP]],
[PCH] ← [[SP] + 1],
[SP] ← ([SP] + 2).

States: 10. Flags: none. Addressing: reg. indirect. Machine cycles: 3.

RET instruction is used at the end of a subroutine. Before the execution of a subroutine the address of the next instruction of the main program is saved in the stack. The execution of RET instruction brings back the saved address from the stack to the program counter. The content of the stack pointer is incremented by 2 to indicate the new stack top. Then the program jumps to the instruction of the main program next to CALL instruction which called the subroutine.

Conditional Return

[PCL] ← [[SP]], [PCH] ← [[SP] + 1],
[SP] ← ([SP] + 2).

States: 6/12. Flags: none. Addressing: reg. indirect. Machine cycles: 1/3. If the condition is true and the program returns from the subroutine, the execution of a conditional return instruction takes 3 machine cycles, 12 states. If condition is not true only one machine cycle, 6 states are required.

(*i*) RC Return from subroutine if carry status CS = 1.

(*ii*) RNC Return from subroutine if carry status CS = 0.

(*iii*) RZ Return from subroutine if the result is zero; the zero status Z = 1.

(*iv*) RNZ Return from subroutine if the result is not zero; the zero status Z = 0.

(*v*) RP Return from subroutine if the result is plus, the sign status S = 0.

(*vi*) RM Return from subroutine if the result is minus, the sign status S = 1.

(*vii*) RPE Return from subroutine if even parity, the parity status P = 1.

(*viii*) RPO Return from subroutine if odd parity, the parity status P = 0.

RST n (Restart).

[[SP] − 1] ← [PCH], [[SP] − 2] ← [PCL],

[SP] ← ([SP] − 2), [PC] ← 8 times *n*.

States: 12. Flags: none, Addressing: reg. indirect. Machine cycles : 3.

Restart is a one-word CALL instruction. The content of the program counter is saved in the stack. The program jumps to the instruction starting at restart location. The address of the restart location is 8 times *n*. The restart instruction and locations are as follows:

Instruction	*Opcode*	*Restart Locations*
RST 0	C7	0000
RST 1	CF	0008
RST 2	D7	0010
RST 3	DF	0018
RST 4	E7	0020
RST 5	EF	0028
RST 6	F7	0030
RST 7	FF	0038

PCHL. (Jump to address specified by H-L pair)

[PC] ← [H-L], [PCH] ← [H], [PCL] ← [L]

States: 6. Flags: none. Addressing: register. Machine cycle: 1.

The contents of H-L pair are transferred to program counter. The contents of register H are moved to high order 8 bits of register PC. The contents of register L are transferred to low order 8 bits of register PC.

4.6.5 Stack, I/O and Machine Control Group

IN port-address. (Input to accumulator from I/O port)

[A] ← [Port]. States: 10. Flags: none. Addressing: direct. Machine cycles: 3.

The data available on the port is moved to the accumulator. After instruction IN, the address of the port is specified. The 2nd byte of the instruction contains the address of the port. The address of a port is an 8-bit address. For example, IN 01. The address of the port B of an I/O port 8255.1 of a microprocessor kit is 01.

OUT port-address. (Output from accumulator to I/O port)

[Port] ← [A]. States: 10. Flags: none. Addressing: direct. Machine cycles: 3.

The content of the accumulator is moved to the port specified by its address. After the OUT instruction, the port address is specified. The 2nd byte of the instruction contains the address of the port. For example, OUT 00. The address of the port A of an I/O port 8255.1 of a microprocessor kit is 00.

PUSH rp. (Push the content of register pair to stack)

[[SP] − 1] ← [*rh*],

[[SP] − 2] ← [*rl*],

[SP] ← ([SP] − 2).

States: 12. Flags: none. Addressing: register(source)/reg. indirect(destination), Machine cycles: 3.

The content of the register pair *rp* is pushed into the stack.

PUSH PSW. (PUSH program status word to the stack)

[[SP] − 1] ← [A]

[[SP] − 2] ← PSW (Program Status Word)

[SP] ← ([SP] − 2).

States: 12, Flags: none. Addressing: register(source)/reg.indirect(destination), Machine cycles: 3.

The content of the accumulator is pushed into the stack. The contents of status flags are also pushed into the stack. The content of the register SP is decremented by 2 to indicate new stacktop.

POP rp. (Copy two bytes from the top of the stack into the specified register)

[*rl*] ← [[SP]]

[*rh*] ← [[SP] + 1]

[SP] ← ([SP] + 2).

States: 10. Flags: none. Addressing: register(destination)/reg.indirect (source), Machine cycles: 3.

The content of the register pair, which was saved earlier is moved from the stack to the register pair.

POP PSW. (Copy two bytes from the top of the stack into PSW and Accumulator)

PSW ← [[SP]]

[A] ← [[SP] + 1]

[SP] ← ([SP] + 2).

States: 10. Flags: all. Addressing: reg. indirect. Machine cycles: 3.

The processor status word which was saved earlier during the execution of the program is moved from the stack to PSW. The content of the accumulator which was also saved is moved from the stack to the accumulator.

HLT (Halt)

States: 5. Flags: none. Machine cycle: 1.

When this instruction is executed, any further program execution is stopped. The microprocessor remains in Halt state. An interrupt or reset is required to exit from Halt state. Registers and status flags remain unaffected.

XTHL. (Exchange stack-top with H-L)

[L] ← [[SP]]

[H] ← [[SP] + 1].

States: 16. Flags: none. Addressing: register indirect. Machine cycles: 5.

The contents of the register L are exchanged with the byte of the stack-top. The contents of the H register exchanged with the byte below the stack-top.

SPHL (Move the contents of H-L pair to stack pointer)

[H-L] ← [SP].

States: 6. Flags: none. Addressing: register. Machine cycle: 1.

The contents of H-L pair are transferred to the SP register.

EI. (Enable interrupts)

States: 4. Flags: none, Machine cycle: 1.

When this instruction is executed the interrupts are enabled.

DI (Disable Interrupts)

States: 4. Flags : none, Machine cycle: 1

When this instruction is executed interrupt are disabled.

SIM (Set Interrupt Masks)

States: 4. Flags: none, Machine cycle: 1.

When this instruction is executed bits 0-5 of the accumulator are used in programming the restart interrupt masks. Bits 6-7 of the accumulator are used in making serial output on SOD line. See details in Chapter 7, Section 7.5: Interrupts of Intel 8085.

RIM (Read Interrupt Mask)

States: 4. Flags: none. Machine cycle: 1.

When this instruction is executed, the accumulator is loaded with pending interrupts, the restart interrupt masks and the contents of SID. See details in Chapter 7, Section 7.5: Interrupts of Intel 8085.

NOP (No operation)

States: 4. Flags: none. Machine Cycle: 1.

No operation is performed when this instruction is executed. The registers and flags remain unaffected. The content of the Program Counter is incremented by one.

PROBLEMS

1. Classify 8085 instructions in various groups. Give examples of instructions for each group.
2. What are the various types of data formats for Intel 8085 instructions? Give examples for each type of data format.
3. Discuss various types of addressing modes of Intel 8085 with suitable examples.
4. Explain what operation will take place when the following instructions are executed:
   LXI rp, data; LDA addr, LHLD addr, STA addr, and SHLD addr.
5. Explain what operation is performed when the following instructions are executed:
   DAD rp, DAA, CMP r, CMP M, CMA, RAL, RAR, PUSH rp and POP rp.

CHAPTER 7

PERIPHERAL DEVICES AND INTERFACING

7.1 INTRODUCTION

A microprocessor combined with memory and input/output devices, forms a microcomputer. The microprocessor is the heart of a microcomputer. Memories and input/output devices are interfaced to microprocessor to form a microcomputer. In case of large and minicomputers the memories and input/output devices are interfaced to CPU by the manufacturer. In a microprocessor-based system the designer has to select suitable memories and input/output devices for his task and interface them to the microprocessor. The selected memories and input/output devices should be compatible with microprocessor. If a particular device is not compatible, an additional electronic circuit has to be designed through which the device may be interfaced to the CPU.

7.2 ADDRESS SPACE PARTITIONING

The Intel 8085 uses a 16-bit wide address bus for addressing memories and I/O devices. Using 16-bit wide address bus it can access $2^{16} = 64$ K bytes of memory and I/O devices. The 64 K addresses are to be assigned to memories and I/O devices for their addressing. There are two schemes for the allocation of addresses to memories and input/output devices:

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

7.2.1 Memory Mapped I/O Scheme

In memory mapped I/O scheme there is only one address space. Address space is defined as the set of all possible addresses that a microprocessor can generate. Some addresses are assigned to memories and some addresses to I/O devices. An I/O device is also treated as a memory location and one address is assigned to it. Suppose that memory locations are assigned the addresses 2000 to 24FF. One address is assigned to each memory location. Any one of these addresses cannot be assigned to an I/O device. The addresses for I/O devices are different from the addresses which have been assigned to memories. The addresses which have not been assigned to memories can be assigned to I/O devices. For example, 2500, 2501, 2502 etc. may be assigned to I/O devices. One address is assigned to each I/O device.

In this scheme all the data transfer instructions of the microprocessor can be used for both memory as well as I/O devices. For example, MOV A, M will be valid for data transfer from the memory location or I/O device whose address is in H-L pair. If the H-L pair contains the address of a memory location, data will be transferred from the memory location to the accumulator. If the H-L pair contains the address of an I/O device, data will be moved from the I/O device to the accumulator. The memory mapped I/O scheme is suitable for a small system.

7.2.2 I/O Mapped I/O Scheme

In this scheme the addresses assigned to memory locations can also be assigned to I/O devices. Since, the same address may be assigned to a memory location or an I/O device, the microprocessor must issue a signal to distinguish whether the address on the address bus is for a memory location or an I/O device. The Intel 8085 issues an IO/$\overline{M}$ signal for this purpose. When this signal is high the address on the address bus is for an I/O device. When this signal is low, the address on the address bus is for a memory location. Two extra instructions IN and OUT are used to address I/O devices. The IN instruction is used to read to data of an input device. The OUT instruction is used to send data to an output device. This scheme is suitable for a large system.

7.3 MEMORY AND I/O INTERFACING

Several memory chips and I/O devices are connected to a microprocessor. Figure 7.1 shows a schematic diagram to interface memory chips or I/O devices to a microprocessor. An address decoding circuit is employed to select the required I/O device or a memory chip. Figure 7.2 shows a schematic diagram of a decoding circuit. If IO/$\overline{M}$ is high the decoder 2 is activated and the required I/O device is selected. If IO/$\overline{M}$ is low, the decoder 1 is activated and the required memory chip is selected. A few MSBs of the address lines are applied to the decoder to select a memory chip or an I/O device.

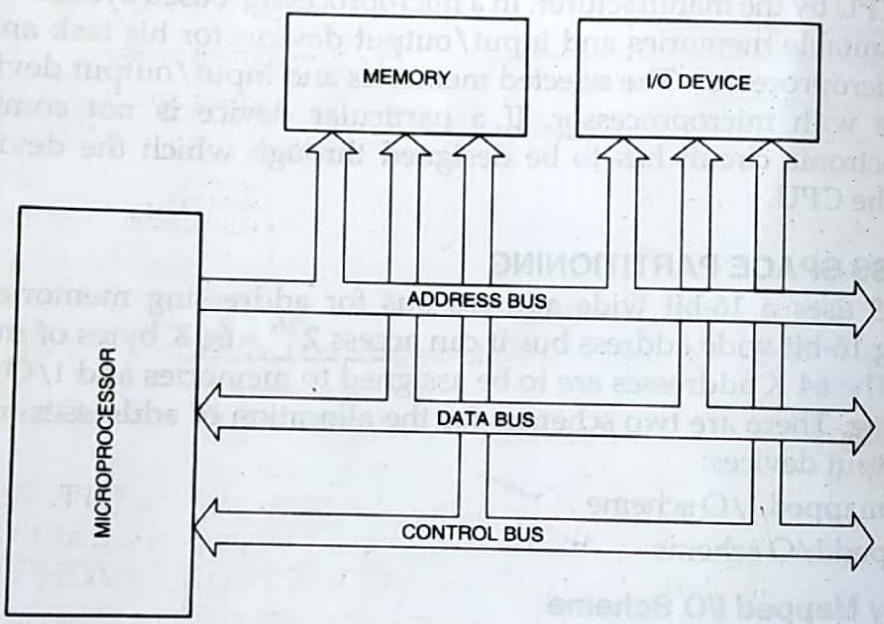

Fig. 7.1 Schematic Diagram for Memory and I/O Interfacing

7.3.1 Memory Interfacing

The address of a memory location or an I/O device is sent out by the microprocessor. The corresponding memory chip or I/O device is selected by a decoding circuit. The decoding task can be performed by a decoder, a comparator, a bipolar PROM or PLA (Programmed logic array). In this section the application of 74LS138, a 1 to 8 lines decoder will be illustrated. Figure 7.3 shows the interface of memory chips through 74LS138. G1, G2A and G2B are enable signals. To enable 74LS138, G1 should be high, and G2A and G2B should be low. A, B and C are select lines. By applying proper logic to select lines any one of the outputs can be selected. Y_0, Y_1..... Y_7 are 8 output lines. An output lines goes low when it is selected. Other output lines remain high. Table 7.1 shows the truth table for 74LS138. When G1 is low or G2A is high or G2B is high, all

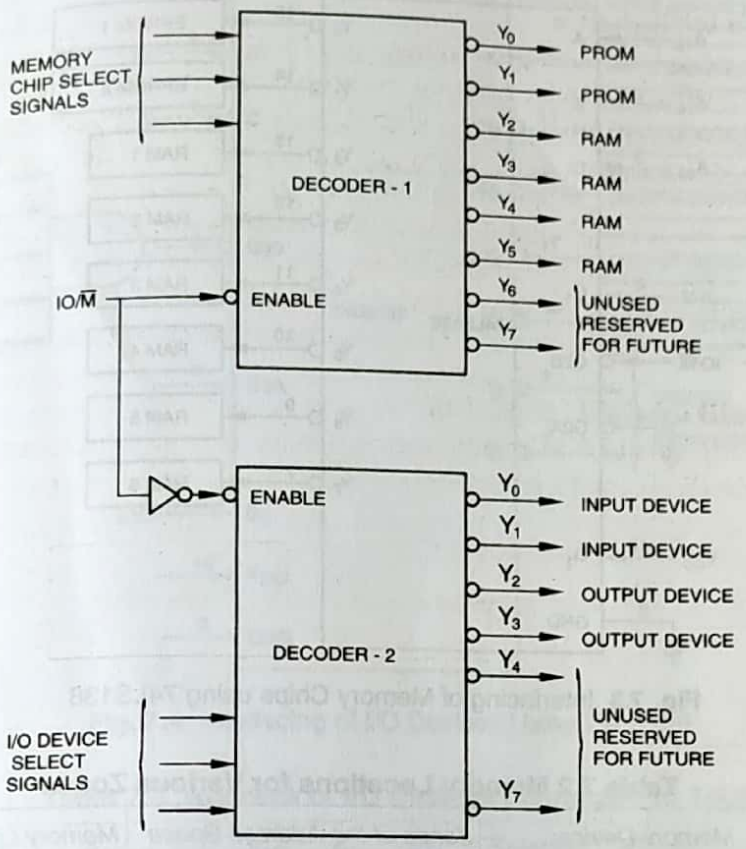

Fig. 7.2 Interfacing of Memory and I/O Devices

output lines become high. Thus, 74LS138 acts as decoder only when G1 is high, and G2A and G2B are low.

The memory locations for EPROM 1 will lie in the range 0000 to 1FFF. These are the memory locations for ZONE 0 for the memory chip which is connected to the output line Y_0 of the decoder. Similarly, for ZONE 1 is 2000 to 3FFF and for ZONE 7 is E000 to FFFF. Table 7.2 shows the memory locations for various zones.

Table 7.1 Truth table for 74LS138

INPUTS						*OUTPUTS*							
ENABLE			*SELECT*										
G1	G2A	G2B	C	B	A	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
X	H	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	L	H	H	H	H	H	L	H	H	H	H
H	L	L	H	L	L	H	H	H	H	L	H	H	H
H	L	L	H	L	H	H	H	H	H	H	L	H	H
H	L	L	H	H	L	H	H	H	H	H	H	L	H
H	L	L	H	H	H	H	H	H	H	H	H	H	L

X Denotes irrelevant

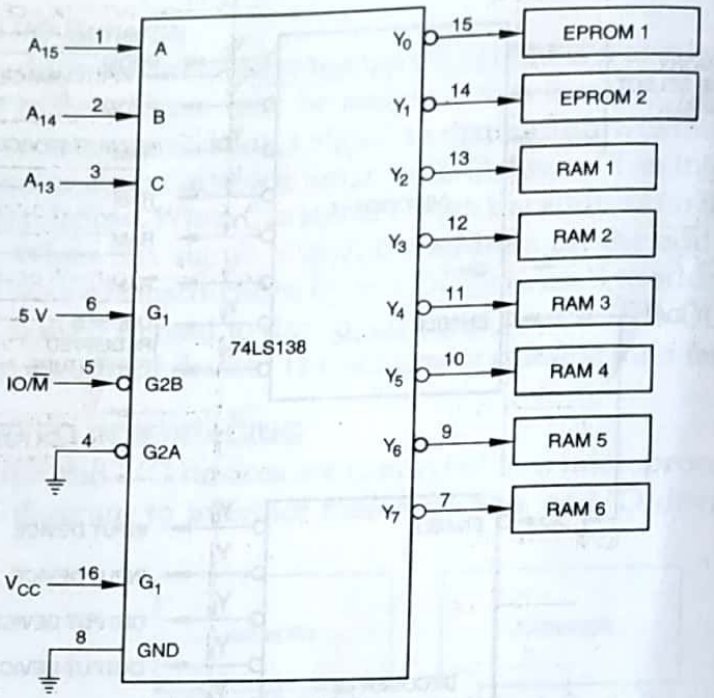

Fig. 7.3 Interfacing of Memory Chips using 74LS138

Table 7.2 Memory Locations for Various Zones

Decoder Output	*Memory Device*	*Zones of the Address Space*	*Memory Locations Address*
Y_0	EPROM 1	ZONE 0	0000 to 1FFF
Y_1	EPROM 2	ZONE 1	2000 to 3FFF
Y_2	RAM 1	ZONE 2	4000 to 5FFF
Y_3	RAM 2	ZONE 3	6000 to 7FFF
Y_4	RAM 3	ZONE 4	8000 to 9FFF
Y_5	RAM 4	ZONE 5	A000 to BFFF
Y_6	RAM 5	ZONE 6	C000 to DFFF
Y_7	RAM 6	ZONE 7	E000 to FFFF

The entire memory address (64K for 8085) has been divided into 8 zones. Address lines A_{15}, A_{14} and A_{13} have been applied to the select lines A, B and C of the 74LS138. The logic applied to these lines selects a particular memory device, an EPROM or a RAM. Other address lines A_0, A_1, A_2, ... and A_{12} go directly to memory chip. They decide the address of the memory location within a selected memory chip. IO/$\overline{M}$ is connected to G2B. When IO/$\overline{M}$ goes low for memory read/write operation, G2B goes low. G1 is connected to + 5 $V_{d.c}$. supply and G2A is grounded.

7.3.2 I/O Interfacing

Figure 7.4 shows the interface of I/O devices through decoder 74LS138. As the address of an I/O device is of 8 bits, only A_8 - A_{15} lines of address bus are used for I/O addressing. The address lines A_8, A_9 and A_{10} have been applied to select lines A, B and C of the 74LS138. The address lines A_{11} - A_{15} are applied to G2B through a NAND gate. G2B becomes low only when all address lines A_{11} - A_{15} are high. G2A is grounded. IO/$\overline{M}$ is connected to G1. When IO/$\overline{M}$ goes high for I/O read/ write operation, G1 goes high. Table 7.3 shows the addresses of I/O devices connected to 74LS138.

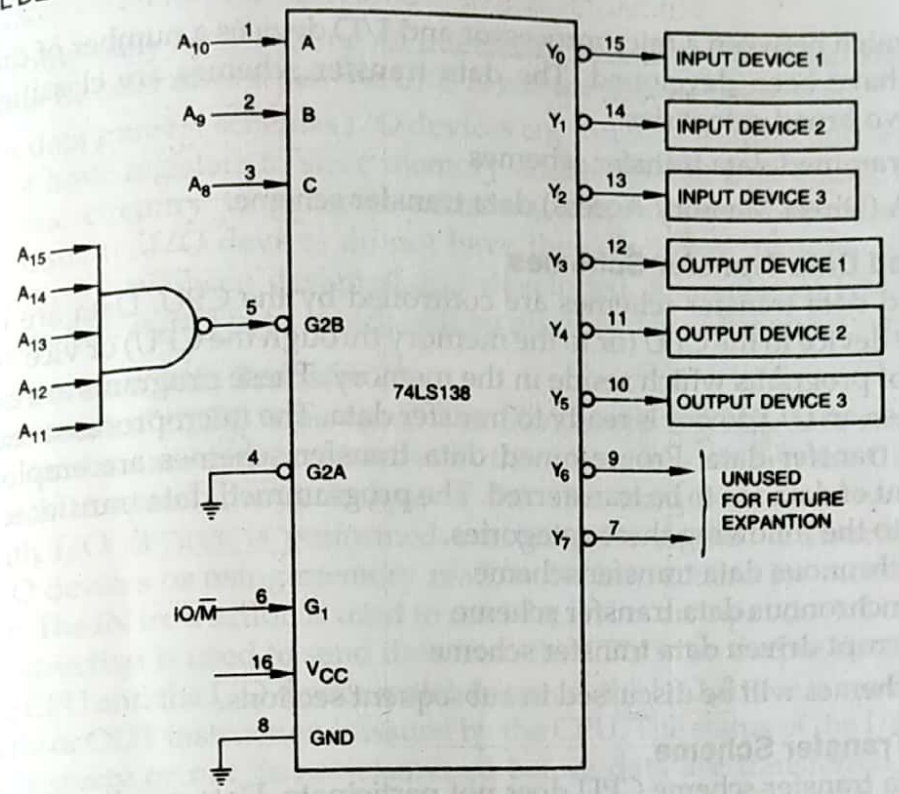

Fig. 7.4 Interfacing of I/O Devices Using 74LS138.

Table 7.3 Address of I/O Devices connected to 74LS138

A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	*Selected Output Lines*	*Corresponding Address*	*I/O Device*
1	1	1	1	1	0	0	0	Y_0	F8	Input Device 1
1	1	1	1	1	0	0	1	Y_1	F9	Input Device 2
1	1	1	1	1	0	1	0	Y_2	FA	Input Device 3
1	1	1	1	1	0	1	1	Y_3	FB	Output Device 1
1	1	1	1	1	1	0	0	Y_4	FC	Output Device 2
1	1	1	1	1	1	0	1	Y_5	FD	Output Device 3
1	1	1	1	1	1	1	0	Y_6	FE	Unused
1	1	1	1	1	1	1	1	Y_7	FF	Unused

7.4 DATA TRANSFER SCHEMES

In a microprocessor-based system or in a computer data transfer takes place between two devices such as microprocessor and memory, microprocessor and I/O devices, and memory and I/O device. Usually, semiconductor memories are compatible with microprocessor because the same technology is employed in the manufacturing of both semiconductor memories and microprocessors. Hence, there is less problem associated with the interfacing of memory. A wide variety of I/O devices having wide range of speed and other different characteristics are available. They use different manufacturing technology such as electronic, electrical, mechanical, electro-mechanical, optical etc. Due to these reasons designers face difficulties in interfacing I/O devices with microprocessor. Special interfacing circuitries have to be designed for the purpose.

A microprocessor-based system or a computer may have several I/O devices of different speed. A slow I/O device can not transfer data when microprocessor issues instruction for the same because it takes some time to get ready. To solve the problem of

speed mismatch between a microprocessor and I/O devices a number of data transfer techniques have been developed. The data transfer schemes are classified into the following two broad categories.

1. Programmed data transfer schemes
2. DMA (Direct Memory Access) data transfer scheme.

Programmed Data Transfer Schemes

Programmed data transfer schemes are controlled by the CPU. Data are transferred from an I/O device to the CPU (or to the memory through the CPU) or vice versa under the control of programs which reside in the memory. These programs are executed by the CPU when an I/O device is ready to transfer data. The microprocessor executes the program to transfer data. Programmed data transfer schemes are employed when small amount of data are to be transferred. The programmed data transfer schemes are classified into the following three categories.

(i) Synchronous data transfer scheme
(ii) Asynchronous data transfer scheme
(iii) Interrupt driven data transfer scheme

These schemes will be discussed in subsequent sections.

DMA Data Transfer Scheme

In DMA data transfer scheme CPU does not participate. Data are directly transferred from an I/O device to the memory or vice versa. The data transfer is controlled by the I/O device or a DMA controller. This scheme is employed when large amount of data are to be transferred. If bulk data are transferred through the CPU, it takes appreciable time and the process becomes slow. An I/O device which wants to send data using DMA technique, sends the HOLD signal to the CPU. On receiving a HOLD signal from an I/O device the CPU gives up the control of buses as soon as the current machine cycle is completed. The CPU sends a hold acknowledge signal to the I/O device to indicate that it has received the HOLD request and it has released the buses. The I/O device takes over the control of buses and directly transfers data to the memory or reads data from the memory.

DMA data transfer scheme is a faster scheme as compared to programmed data transfer scheme. It is used to transfer data from mass storage devices such as hard disks, optical disks etc. It is also used for high-speed printers. When data transfer is over, the CPU regains the control over the buses.

DMA data transfer schemes are of the following two types:

(i) Burst mode of DMA data transfer
(ii) Cycle stealing techniques of DMA data transfer.

Burst Mode of DMA Data Transfer

A scheme of DMA data transfer, in which the I/O device withdraws the DMA request only after all the data bytes have been transferred, is called burst mode of data transfer. By this technique a block of data is transferred. This technique is employed by magnetic disks drives. In case of magnetic disks data transfer can not be stopped or slowed down without loss of data. Hence, block transfer is a must.

Cycle Stealing Technique

In this technique a long block of data is transferred by a sequence of DMA cycles. In this method after transferring one byte or several bytes the I/O device withdraws DMA request. This method reduces interference in CPU's activities. The interference can be

eliminated completely by designing an interfacing circuitry which can steal bus cycle for DMA data transfer only when the CPU is not using the system bus.

In DMA data transfer schemes I/O devices control data transfer and hence the I/O devices must have registers to store memory addresses and byte count. It must also have electronic circuitry to generate necessary control signals required for DMA operations. Usually, I/O devices do not have these facilities. To solve this problem DMA controllers have been designed and developed. Examples of DMA controller chips are: Intel 8237A, 8257 etc. which will be discussed in subsequent sections.

7.4.1 Synchronous Data Transfer

Synchronous means "at the same time." The device which sends data and the device which receives data are synchronised with the same clock. When the CPU and I/O devices match in speed, this technique of the data transfer is employed. The data transfer with I/O devices is performed executing IN or OUT instructions for I/O mapped I/O devices or using memory read/write instructions for memory mapped I/O devices. The IN instruction is used to read data from an input device or input port. The OUT instruction is used to send data from the CPU to an output device or output port. As the CPU and the I/O device match in speed, the I/O device is ready to transfer data when IN or OUT instruction is issued by the CPU. The status of the I/O device *i.e.*, whether it is ready or not, is not examined before data are transferred, as it is not needed.

The I/O devices compatible with microprocessors in speed are usually not available. Hence, this technique of data transfer is rarely used for I/O devices. However, memories compatible with microprocessors are available, and therefore, this technique is invariably used with compatible memory devices.

7.4.2 Asynchronous Data Transfer

Asynchronous means "at irregular intervals". In this method data transfer is not based on perdetermined timing pattern. This technique of data transfer is used when the speed of an I/O device does not match the speed of the microprocessor, and the timing characteristic of I/O device is not predictable. In this technique the status of the I/O device *i.e.* whether the device is ready or not, is checked by the microprocessor before the data are transferred. The microprocessor initiates the I/O device to get ready and then continuously checks the status of the I/O device till the I/O device becomes ready to transfer data. When I/O device becomes ready, the microprocessor sends instructions to transfer data. This mode of data transfer is also called *handshaking mode* of data transfer because some signals are exchanged between the I/O device and microprocessor before the actual data transfer takes place. The microprocessor issues an initiating signal to the I/O device to get ready (or to start). When I/O device becomes ready it sends signals to the processor to indicate that it is ready. Such signals are called *handshake signals*.

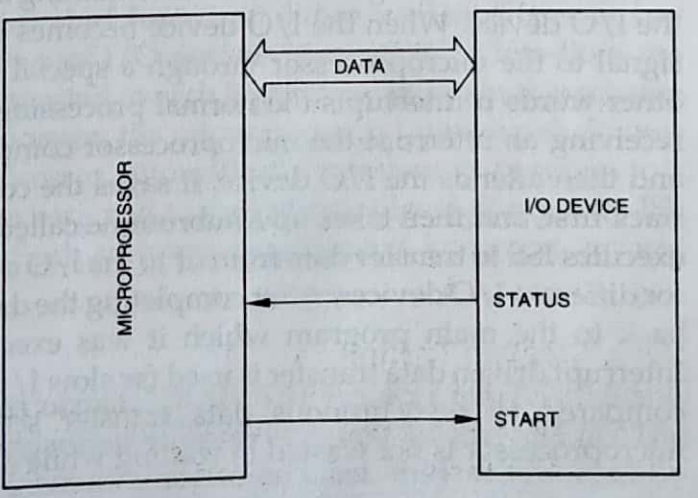

Fig. 7.5 Asynchronous Data Transfer

Figure 7.5 shows a schematic diagram for asynchronous data

transfer. Asynchronous data transfer is used for slow I/O devices. This technique is an inefficient technique because the precious time of the microprocessor is wasted in waiting.

Figure 7.6 shows a simple example of asynchronous data transfer. An A/D converter has been interfaced to the microprocessor to transfer data in asynchronous mode. The microprocessor sends a start of conversion signal, S/C to the A/D converter. The A/D converter being a slow device as compared to a microprocessor, takes some time to convert analog signal to digital signal. When conversion is over the A/D converter makes end of conversion signal, E/C high. The microprocessor goes on checking E/C till it becomes high. When E/C becomes high, the microprocessor issues instructions for data transfer.

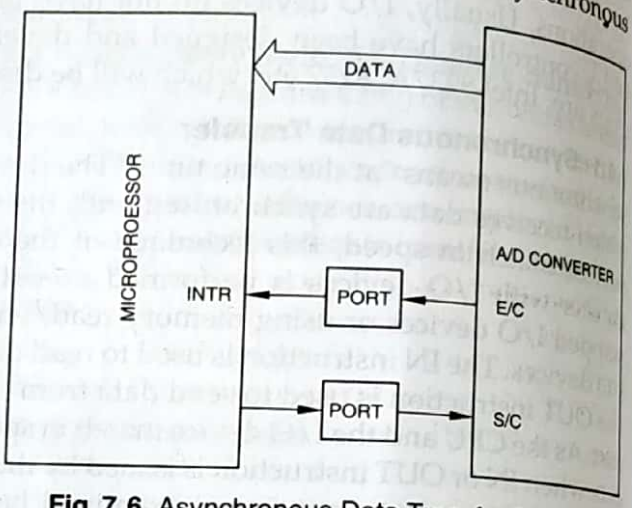

Fig. 7.6 Asynchronous Data Transfer Scheme for an A/D Converter

Some simple I/O devices may not have status signal. In such a case the microprocessor goes on checking whether data are available on the port or not. A keyboard interfaced to a microprocessor through a port is an example of this type of data transfer scheme.

Asynchronous data transfer discussed so far use software approach. It can also be implemented by hardware approach employing READY signal. An I/O device is interfaced to the microprocessor through READY signal. I/O device (or memory chip) and microprocessor have READY pins. When an I/O device or memory chip becomes ready to transfer data, it makes READY signal high. The microprocessor checks READY signal before data are transferred. If READY is low the microprocessor enters a wait state. The status of the READY signal is sensed by the microprocessor in the T_2 state of a machine cycle. The microprocessor remains in wait state till READY becomes high. This technique is commonly used by slow memory devices.

7.4.3 Interrupt Driven Data Transfer

In this scheme the microprocessor initiates an I/O device to get ready, and then it executes its main program instead of remaining in a program loop to check the status of the I/O device. When the I/O device becomes ready to transfer data, it sends a high signal to the microprocessor through a special input line called an interrupt line. In other words it interrupts the normal processing sequence of the microrpocessor. On receiving an interrupt the microprocessor completes the current instruction at hand, and then attends the I/O device. It saves the contents of the program counter on the stack first, and then takes up a subroutine called ISS (Interrupt Service Subroutine). It executes ISS to transfer data from or to the I/O device. Different ISS are to be provided for different I/O devices. After completing the data transfer the microprocessor returns back to the main program which it was executing before the interrupt occurred. Interrupt driven data transfer is used for slow I/O devices. It is an efficient technique as compared to asynchronous data transfer scheme because precious time of the microprocessor is not wasted in waiting while an I/O device is getting ready.

Figure 7.7 show the interfacing of an A/D converter to transfer data employing interrupt driven data transfer scheme. The microprocessor sends first the start of conversion signal, S/C to the A/D converter. Thereafter, the microprocessor executes its main program. A/D converter is a slow device compared to a microprocessor. It takes some time to convert analog signal to its equivalent digital quantity. When A/D converter completes the task of conversion, it makes an end of conversion signal, E/C high. The E/C signal is connected to an interrupt line of the microprocessor. When interrupt line goes high, the microprocessor takes all necessary steps to transfer data from the A/D converter. After completing the data transfer the micro-processor returns back to execute the main program that it was executing prior to the interrupt.

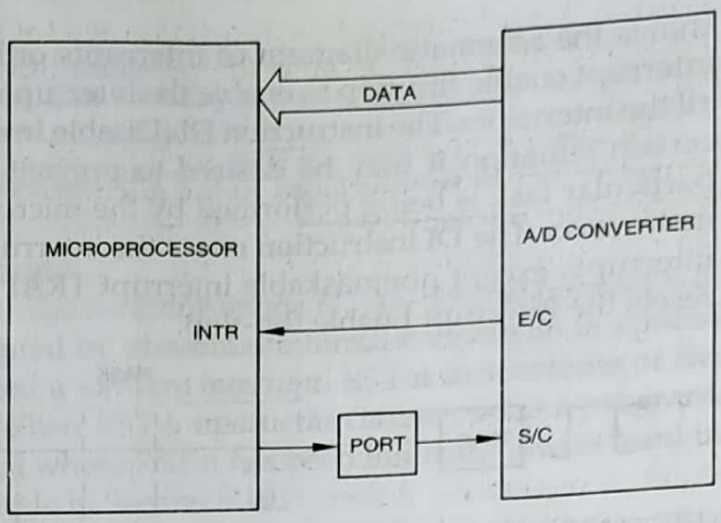

Fig. 7.7 Interrupt Driven Data Transfer Scheme for an A/D Converter

7.4.4 Multiple Interrupts

In a microcomputer system several I/O devices can use interrupt driven data transfer scheme. While interfacing I/O devices using interrupts the following situations may arise:

1. A microprocessor may have only one interrupt level and several I/O devices are to be connected to it.
2. A microprocessor may have several interrupt levels and one I/O device is to be connected to each interrupt level.

The schemes which are used to tackle such situations are described in the following subsections:

Several I/O Devices Connected to a Single Interrupt Level. When several I/O devices are to be connected to a single interrupt level, they are connected through interrupt controller, 8259. Up to 8 I/O devices can be connected to the microprocessor through an8259. If more than 8 I/O devices are to be connected, more 8259 ICs are used in series. The 8259 has been described in detail in Section 7.9.

One Device Connected to Each Level of Interrupt. When a microprocessor has several interrupt levels and the number of I/O devices is equal to or less than the interrupt levels, one device may be connected to each level of interrupt. In this scheme when a device interrupts the microprocessor, the microprocessor immediately knows which device has interrupted. The processor automatically transfers its program to a specific memory location that has been assigned to the interrupt level. It executes ISS for the device which has interrupted. Such an interrupt scheme is known as *vectored interrupt*.

7.5 INTERRUPTS OF INTEL 8085

The Intel 8085 has five interrupt inputs namely TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR. The TRAP has the highest priority, followed by RST 7.5, RST 6.5 and RST 5.5. The INTR has the lowest priority. When interrupts are to be used, they are enabled by software using the instruction EI (Enable Interrupt) in the main program. Figure 7.8

shows the schematic diagram of interrupts of Intel 8085. The instruction EI sets the interrupt enable flip-flop to enable the interrupts. The use of the instruction EI enables all the interrupts. The instruction DI (Disable Interrupt) is used to disable interrupts. In certain situation it may be desired to prevent the occurrence of interrupts while a particular task is being performed by the microprocessor. This can be done using DI instruction. The DI instruction resets the interrupt enable flip-flop and disables all the interrupts except nonmaskable interrupt TRAP, see Fig. 7.8. The system RESET also resets the Interrupt Enable flip-flop.

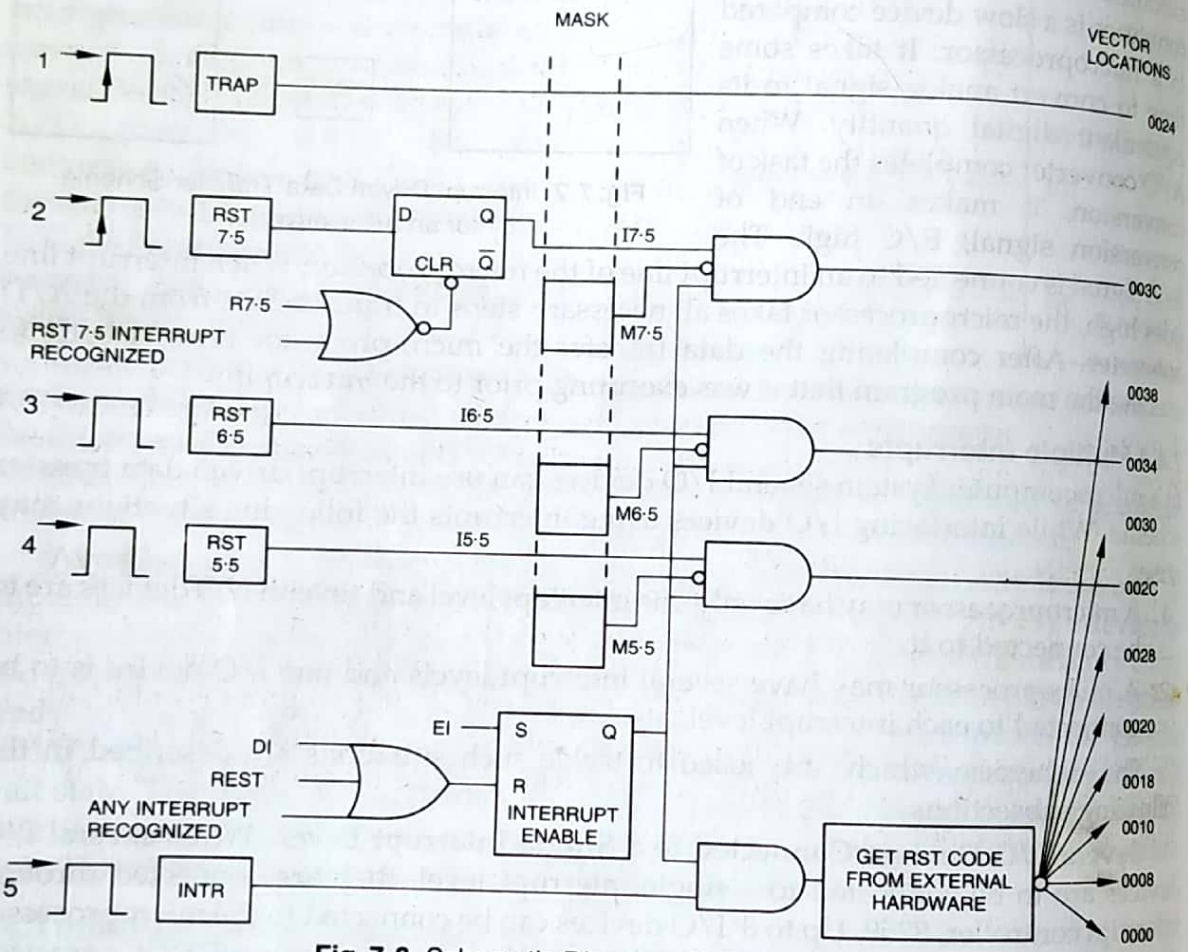

Fig. 7.8 Schematic Diagram of 8085 Interrupts

When an interrupt line goes high processor completes its current instruction and saves program counter on the stack. It also resets Interrupt Enable flip-flop before taking up ISS so that the occurrence of further interrupts by other devices is prevented during the execution of ISS. All the interrupts except TRAP are disabled by resetting the Interrupt Enable flip-flop.

The resetting of this flip-flop can be done in one of the three ways : by software using instruction DI, system reset or by recognition of an interrupt request.

Before the program returns back from ISS to the main program all the interrupts are to be enabled again. This is done using instruction EI in ISS before using the instruction RET.

At many occasions the programmer may like to prevent the occurrence of a few of several interrupts while microprocessor is performing certain tasks. This is done by masking off those interrupts which are not required to occur when certain task is being performed. The interrupts which can be masked off (*i.e.*, made ineffective) are called

maskable interrupts. Masking is done by software. The Intel 8085 has two categories of interrupts : maskable and nonmaskable. The TRAP is a nonmaskable interrupt. It need not be enabled. It cannot be disabled. It is not accessible to user. It is used for emergency situation such as power failure and energy shut-off. RST 7.5, RST 6.5 and RST 5.5 are maskable interrupts.

7.5.1 Hardware and Software Interrupts

Interrupts caused by I/O devices are called *hardware interrupt*. The normal operation of a microprocessor can also be interrupted by abnormal internal conditions or special instructions. Such an interrupt is called a *software interrupt*. RST n instructions of the 8085 are used for software interrupt. When RST n instruction is inserted in a program, the program is executed upto the point where RST n has been inserted. This is used in debugging of a program. See an example in Section 5.10.

The internal abnormal or unusual conditions which prevent the normal processing sequence of a microprocessor are also called exceptions. For example, divide by zero will cause an exception. Intel literatures do not use the term exception, whereas Motorola literatures use the term exception. Intel includes exception in software interrupt.

When several I/O devices are connected to INTR interrupt line, an external hardware is used to interface I/O devices. The external hardware circuit generates RST n codes to implement the multiple interrupt scheme. This will be discussed later on in this chapter.

7.5.2 Interrupts Call-Locations

When an interrupt occurs, the program is transferred to a specific memory location. Then the monitor transfers the program from the specific memory location to a memory location in RAM, from where the user can write the program for interrupt service sub-routine (ISS). For TRAP, RST 7.5, 6.5 and 5.5 the program is automatically transferred to specific memory locations without any external hardware. The necessary hardware is already provided within 8085. The specific memory locations for these interrupts are as follows:

Interrupt	*Call-Location in Hex*
TRAP	0024
RST 7.5	003C
RST 6.5	0034
RST 5.5	002C

An interrupt for which hardware automatically transfers the program to a specific memory location is known as *vectored interrupt*.

INTR CALL Locations. There are 8 numbers of CALL-locations for INTR interrupt. Table 7.4 shows CALL-locations, RST n instructions and corresponding hex-code. For INTR, external hardware is used to transfer program to specific CALL-location. The hardware circuit generates RST codes for this purpose. The INTR line is sampled by the microprocessor in the last state of the last machine cycle of each instruction. When INTR is high, the microprocessor saves the contents of the program counter on the stack and then sends an interrupt acknowledge signal, INTA to the external hardware. In response to INTA the external hardware generates a RST n code. When microprocessor receives this code, it transfers program to the corresponding CALL-location. Upto 8 number of I/O devices can be connected to INTR through an

external hardware. Priority can be assigned to I/O devices connected to INTR through external hardware or interrupt controller. The external hardware recognizes which I/O device has interrupted and it generates proper RST code that causes microprocessor to take up ISS for that particular I/O device. An external hardware can be built employing prority encoder and a tri-state buffer, see Ref. 9. Alternatively, Priority Interrupt Controller 8214 can be used. Programmable interrupt controller 8259 is more versatile, and persent trend is to use programmable interrupt controller. INTA is used to activate 8259 or some other hardware. The 8259 has been described later on in this chapter.

7.5.3 RST 7.5, 6.5 and 5.5

RST 7.5, RST 6.5 and RST 5.5 are maskable interrupts. These interrupts are enabled by software using instructions EI and SIM (Set Interrupt Mask). The execution of the instruction SIM enables/disables interrupts according to the bit pattern of the accumulator. Figure 7.9 shows accumulator contents for SIM instruction.

Bits 0 - 2 set/rest the mask bits of interrupt mask register for RST 5.5, 6.7 and 7.5. Bit 0 is for RST 5.5 mask, bit 1 for RST 6.5 mask and bit 2 for RST 7.5 mask. If a bit is set to 1 the corresponding interrupt is masked off (disabled). If it is set to 0, the corresponding interrupt is enabled. Bit 3 is set to 1 to make bits 0 - 2 effective. Bit 4 is an additional control for RST 7.5. If it is set to 1, the flip-flop for RST 7.5 is reset. Thus RST 7.5 is disabled regardless of whether bit 2 for RST 7.5 is 0 or 1.

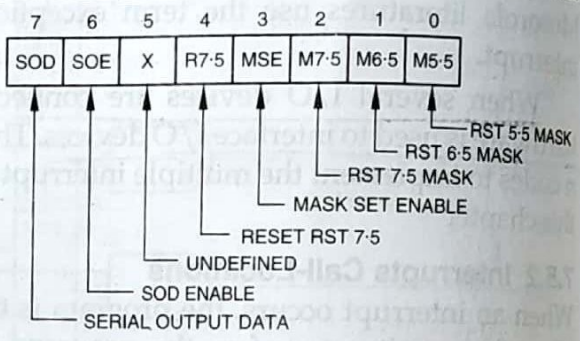

Fig. 7.9 Accumulator Content for SIM

Table 7.4 CALL-Locations and Hex-Codes for RST n

RST n	*Hex-Code*	*CALL-Locations*
RST 0	C7	0000
RST 1	CF	0008
RST 2	D7	0010
RST 3	DF	0018
RST 4	E7	0020
RST 5	EF	0028
RST 6	F7	0030
RST 7	FF	0038

Bits 6 and 7 are for serial data output. The instruction SIM is also used for serial data transmission. Bit 6 is to enable SOD. If SIM instruction is executed the content of the 7th bit of the accumulator is output on SOD line of the microprocessor. The content of bit 7 may be either high (1) or low (0).

The instruction DI disables all the interrupts. This is not always desired. When the processor is performing a particular task, it may be desired to prevent the occurrence of a few of the several interrupts. In such a situation the interrupts which are not desired to occur may be masked off using SIM instruction.

Triggering Levels. TRAP is edge as well as level triggered. This means that TRAP should go high and stay high until it is acknowledged. In this way false triggering caused by noise and transients is avoided. TRAP has also a flip-flop which is not shown in Fig. 7.8. The flip-flop is cleared when interrupt is acknowledged so that future interrupt may be entertained.

RST 7.5 is positive edge triggered interrupt. It can be triggered with a short duration pluse. RST 6.5 and 5.5 are level triggered interrupts. Triggering level for RST 6.5 and 5.5 has to be kept high until the microprocessor recognises these interrupts. If the processor does not recognise these interrupts immediately, their triggering level should be held by external hardware.

Example 1. Enable all the interrupts of Intel 8085.

The content of the accumulator for instructions SIM to enable RST 7.5, 6.5 and 5.5 are programmed as follows.

7	6	5	4	3	2	1	0	
SOD	SOE	X	R7.5	MSE	M7.5	M6.5	M5.5	
0	0	0	0	1	0	0		0 = 08

Bits 0, 1 and 2 are set to 0 to enable RST 7.5, 6.5 and 5.5. Bit 3 is set to make bits 0, 1 and 2 effective. Bit 4 is set to 0 to enable RST 7.5 as it is an additional control for RST 7.5.

Instructions

Mnemonic	*Operands*	*Comments*
EI		Enable all interrupts.
MVI	A, 08	Get accumulator bit pattern to enable RST 7.5, 6.5 and 5.5.
SIM		Enable RST 7.5, 6.5 and 5.5.

Instruction EI and SIM both are essential to enable RST 7.5, 6.5 and 5.5. In addition to RST 7.5, 6.5 and 5.5, the instruction EI also enables INTR.

Example 2. Enable RST 6.5 and disable RST 7.5 and 5.5.

The contents of the accumulator to enable RST 6.5 and disable RST 7.5 and 5.5 are:

7	6	5	4	3	2	1	0	
SOD	SOE	X	R7.5	MSE	M7.5	M6.5	M5.5	
0	0	0	1	1	1	0		1 = 1D

Bit 1 is set to 0 to enable RST 6.5.

Bits 0 and 2 are set to 1 to mask off (disable) RST 5.5 and 7.5 respec ively. Bit 4 which is an addition control for RST 7.5 is set to 1 to disable RST 7.5.

Bit 3 is set to 1 to make bits 0, 1 and 2 effective.

Instructions

EI	Enable interrupts.
MVI A, 1D	Accumulator bit pattern to enable RST 6.5 and mask off RST 7.5 and 5.5.
SIM	Enable RST 6.5 and disable RST 7.5 and 5.5.

Example 3. Use RST 7.5 to interrupt Intel 8085.

For laboratory testing a very simple program, in which microprocessor is moving in a loop, has been considered. The interruption is made by applying a rising pulse at RST 7.5 terminal manually.

MAIN PROGRAM

Memory address	*Machine Codes*	*Labels*	*Mnemonics*	*Operands*	*Comments*
FC00	FB		EI		Enable all interrupts.
FC01	3E, 08		MVI	A, 08	Get accumulator bit pattern to enable RST 7.5, 6.5 and 5.5.

FC03	30		SIM		
FC04	21, 50, FC	LOOP	LXI	H, FC50	Memory address of 1st number
FC07	7E		MOV	A, M	Get 1st number.
FC08	23		INX	H	
FC09	86		ADD	M	Add 1st and 2nd number.
FC0A	32, 52, FC		STA	FC52	Sum in FC52
FC0D	C3, 04, FC		JMP	LOOP	Jump to LOOP

DATA
FC50 — 05
FC01 — 02
SUM
FC52 — 07

In the beginning of the program all interrupts have been enabled as explained in Example 1. There is a simple program from FC04 to FC0C, to add two numbers and to store their sum in the memory location FC03. The program moves in a loop and repeats the same task again and again.

Now, interrupt the microprocessor by applying a pulse to RST 7.5 line. For this purpose make contact of RST 7.5 line to ground terminal, then to 5 V supply. Now, the program will jump to the memory location 003C which is the specific memory location for RST 7.5. The manufacturer of the Vinytics microprocessor kit has given the following monitor program:

003C C3, BD, FF JMP FFBD

Therefore, the monitor transfers the program from 003C to the memory location FFBD. Now, the user has to transfer the program from FFBD to a memory location in RAM from which service subroutine for RST 7.5 has been written.

User's JMP Instruction

Transfer program from FFBD to FC20.

FFBD C3, 20, FC JMP FC20 ; Transfer program from FFBD to FC20.

FC20 is the starting address of the interrupt service subroutine.

Interrupt Service subroutine

Memory address	Machine Codes	*Mnemonics*	*Operands*	Comments
FC20	3A, 00, FD	LDA	FD00	Get content of FD00.
FC23	32, 30, FC	STA	FC30	Store it in FC30.
FC26	2A, 01, FD	LHLD	FD01	Get contents of FD01 and FD02.
FC29	22, 31, FC	SHLD	FC31	Store them in FC31 and FC32.
FC2C	FB	EI		Enable interrupts.
FC2D	C9	RET		Return to main program.

DATA
FD00 — 01
FD02 — 02
FD03 — 05

Result
FC30 — 01
FC31 — 02
FC32 — 05

The service subroutine is a simple program to transfer the contents of memory locations FD00, FD01 and FD02 to memory locations FC30, FC31 and FC32 respectively. When interrupt occurs the microprocessor transfers the program from main program to memory location 003C. Then the program is transferred to FFBD by the monitor. From FFBD the program is transferred to service subroutine. After performing the task of service subroutine the program goes back to the main program. The processor disables all the interrupts before it enters service subroutine. Therefore, at the end of service subroutine instruction EI is used to enable interrupts so that further interrupts will be recognised. After making interrupt see the result in FC30 – FC32.

Similarly, user can try with RST 6.5 and 5.5. The monitor program for these interrupts for Vinytics kit are:

0034 C3, B7, FF,	JMP	FFB7	It is for RST 6.5
002C C3, 09, 05	JMP	0509	It if for RST 5.5.

As RST 6.5 and 5.5 are level triggered, to make an interrupt the interrupt line has to be kept simply high till the processor recognises the interrupt. For laboratory testing as explained above the user may simply make contact of interrupt line to $5V_{d.c}$ terminal on the board.

Pending Interrupts. When one interrupt request is being served, other interrupt may occur resulting in a pending request. When more than one interrupts occur simultaneously, the interrupt having higher priority is served and the interrupts with lower priority remain pending. As 8085 contains several interrupt lines; an interrupt may occur while other interrupt is being served resulting in a pending interrupt. The 8085 has an instruction RIM using which the programmer can know the current status of the pending interrupts. This instruction gives the current status of only maskable interrupts. In case of INTR the processor reads its status in the last state of the last machine cycle of an instruction. When the processor returns to the main program after serving interrupt service subroutine it knows the status of INTR line as soon as it makes further execution of the main program.

Figure 7.10 shows accumulator contents after the execution of instruction RIM. Bits 0-2 are for interrupt masks; 1 = masked. Bit 3 indicates interrupt enable flag; 1 = enabled. Bits 4-6 indicate pending interrupts; 1 = pending. Bit 7-serial input data, if any.

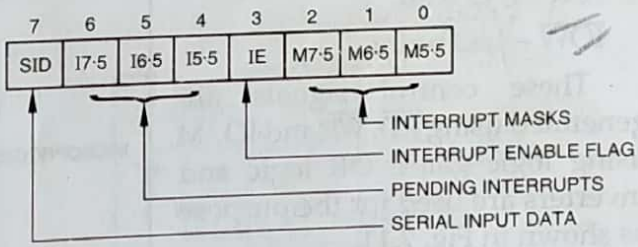

Fig. 7.10 Accumulator Content after the Execution of RIM

After executing the interrupt service subroutine the processor checks whether any other interrupt is pending using RIM instruction. If an interrupt is pending the processor executes its interrupt service subroutine before it returns to the main program.

7.6 INTERFACING DEVICES AND I/O DEVICES

To communicate with the outside world microcomputers use peripherals (I/O devices). Commonly used peripherals are: A/D converter, D/A converter, CRT, printers, hard disks, floppy disks, magnetic tapes etc. Peripherals are connected to the microcomputer through electronic circuits known as interfacing circuits. Generally, each I/O device requires a separate interfacing circuit. The interfacing circuit converts

the data available from an input device into compatible format for the computer. The interface associated with the output device converts the output of the microcomputers into the desired peripheral format. To simplify the work of the designer microprocessor manufacturers have developed a number of general purpose and special purpose single chip interfacing devices. Some of the general purpose devices are:

(i) I/O Port
(ii) Programmable Peripheral Interface (PPI)
(iii) DMA Controller
(iv) Interrupt Controller
(v) Communication Interface

Special purpose interfacing devices are designed to interface a particular type of I/O device to the microprocessor. Examples of such devices are:

(i) CRT Controller.
(ii) Optical Disk Controller.
(iii) Key Board and Display Interface.

A large number of interfacing devices have been developed by various manufacturers. The detailed description of these devices is beyond the scope of the book. Some important interfacing devices developed by INTEL Corporation are described in the subsequent subsections.

7.6.1 Generation of Control Signals for Memory and I/O Devices

Intel 8085 issues control signals $\overline{RD}$, $\overline{WR}$ for read and write operation of memory and I/O devices. It also issues a status signal IO/$\overline{M}$ to distinguish whether read/write operation is to be performed by memory or I/O device. Memory and I/O devices require control signals in modified form shown below:

$\overline{MEMR}$ – Memory read.

$\overline{MEMW}$ – Memory write.

$\overline{IOR}$ – I/O read.

$\overline{IOW}$ – I/O write.

These control signals are generated using $\overline{RD}$, $\overline{WR}$ and IO/$\overline{M}$ using logic gates. OR logic and inverters are used for the purpose as shown in Fig. 7.11.

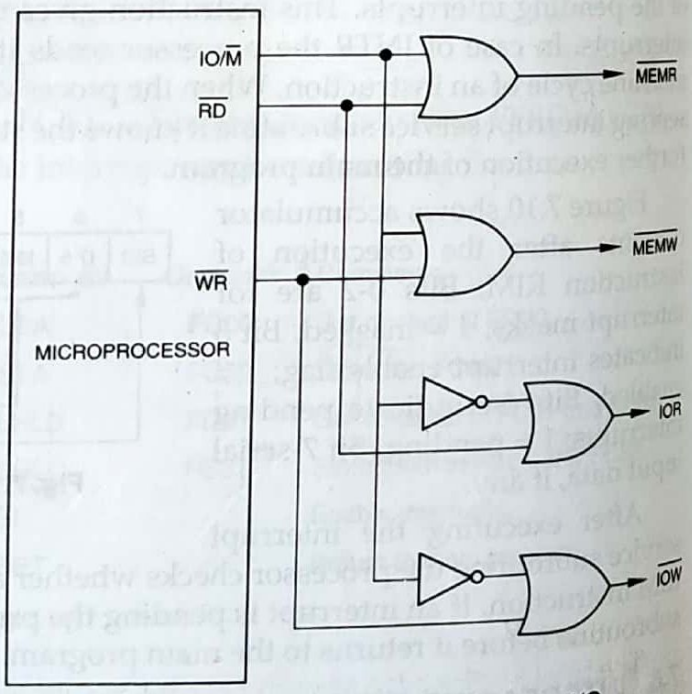

Fig. 7.11 Control Signals for Memory I/O Read/Write Operation

To get $\overline{MEMR}$, use IO/$\overline{M}$ ∨ $\overline{RD}$ ∨ is a symbol for logical OR operation. Memory read operation takes place when IO/$\overline{M}$ and $\overline{RD}$ both are low. IO/$\overline{M}$ and $\overline{RD}$ are applied to an OR gate. The output of the OR gate is $\overline{MEMR}$. When both IO/$\overline{M}$ and $\overline{RD}$ are low, $\overline{MEMR}$ goes low and it activates memory for read operation. Similarly, other control signals are obtained as shown below:

To get $\overline{MEMW}$, use IO/$\overline{M}$ ∨ $\overline{WR}$.

I/O read/write operation takes place when IO/$\overline{M}$ is high. To get $\overline{IOR}$ and $\overline{IOW}$ signals, IO/$\overline{M}$ is inverted and then applied to OR gates.

Get $\overline{IOR}$ using inverted IO/$\overline{M} \vee \overline{RD}$.

Get $\overline{IOW}$ using inverted IO/$\overline{M} \vee \overline{WR}$.

7.7 I/O PORTS

An input device is connected to the microprocessor through an input port. An input port is a place for unloading data. An input device sends data to the input part. The microprocessor reads data from the input port. Thus data are transferred from the input device to the accumulator via input port. Similarly, an output device is connected to the microprocessor through an output port. The microprocessor sends data to an output port. As the output port is connected to the output device, data are transferred to the output device. Figure 7.12 shows a schematic connection of the CPU, I/O port and I/O devices. An I/O port may be programmable or non-programmable. A non-programmable port behaves as an input port if it has been designed so and connected in input mode. Similarly, a non-programmable port designed so and connected in output mode, acts as an output port. But a programmable I/O port can be programmed to act either as an input port or output port; the electrical connections remain same. This will be clear when the method of programming of such devices is discussed in the subsequent subsections.

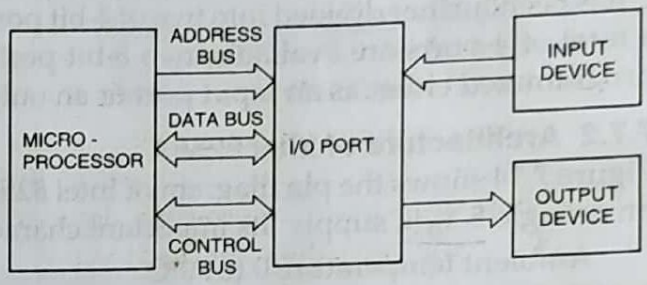

Fig. 7.12 Interfacing of I/O Device through I/O Port

The Intel 8212 is an 8-bit non-programmable I/O port. It can be connected to the microprocessor either as an input port or an output port. If we require one input port and one output port, two units of 8212 will be required. One of them will be connected in input mode and the other in the output mode. Figure 7.13 shows the connections of 8212 in input mode and output mode.

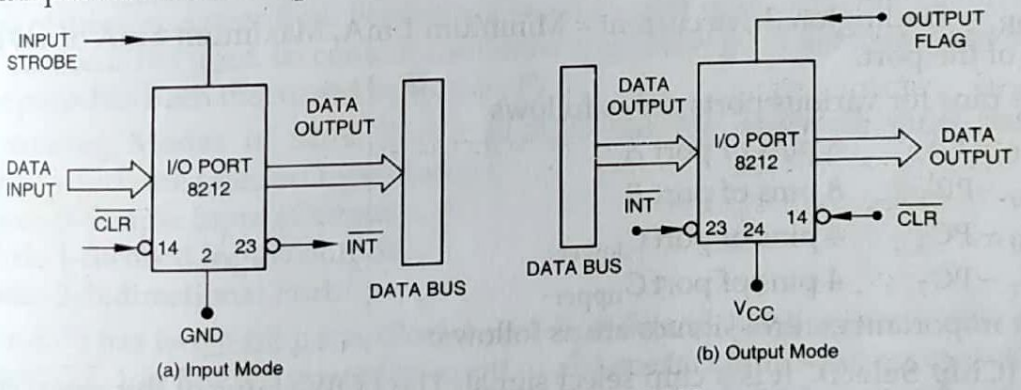

Fig. 7.13 Interfacing of Intel 8212

The Intel 8155 is a RAM with I/O ports. It contains a 256 byte RAM, 3 I/O ports and a 14-bit timer/counter. There are three ports: A, B and C. The port A and port B are of 8-bits and port C of 6 bits. Each port can be programmed either as an input port or output port. The port C may also be programmed as a control port for the port A and port B. The details of 8155 can be seen in the Intel's Microprocessor and Peripheral Handbook. The 8155 is not widely used in Indian microprocessor kits. Indian kits

generally use Intel 8255 which is a programmable peripheral interface (PPI). The Intel 8255 is described in detail in the subsequent subsections. It is more powerful than Intel 8155.

7.7.1 Programmable Peripheral Interface (PPI)

A programmable peripheral interface is a multiport device. The ports may be programmed in a variety of ways as required by the programmer. The device is very useful for interfacing peripheral devices. The term PIA, Peripheral Interface Adapter is also used by some manufacturer.

INTEL 8255. The Intel 8255 is a programmable peripheral interface (PPI). It has two versions, namely the Intel 8255A and the Intel 8255A-5. General descriptions for both are same. There are some differences in their electrical characteristics. Hereafter, they will be referred to as 8255. Its main functions are to interface peripheral devices to the microcomputer. It has three 8-bit ports, namely Port A, Port B and Port C. The port C has been further divided into two of 4-bit ports, port C upper and Port C lower. Thus, a total of 4 ports are available, two 8-bit ports and two 4-bit ports. Each port can be programmed either as an input port or an output port.

7.7.2 Architecture of Intel 8255A

Figure 7.14 shows the pin diagram of Intel 8255A. It is a 40 pin I.C. Package. It operates on a single 5 $V_{d.c}$. supply. Its important characteristics are as follows:

Ambient temperature 0 to 70°C.

Voltage on any pin : 0.5 V to 7 V.

Power dissipation 1 Watt.

V_{IL} = Input low voltage
= Minimum 0.5 V, Maximum 0.8 V.

V_{IH} = Input high voltage
= Minimum 2 V, Maximum V_{CC}.

V_{OL} = Output low voltage = 0.45 V.

V_{OH} = Output high voltage = 2.4 V.

I_{DR} = Darlington drive current = Minimum 1 mA, Maximum 4 mA of any 8 pins of the port.

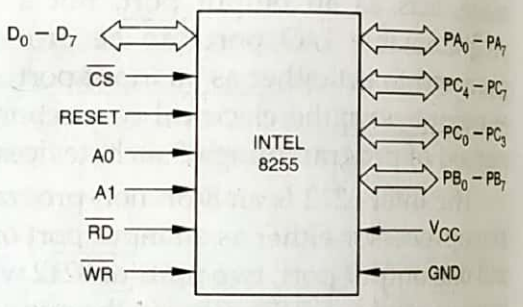

Fig. 7.14 Schematic Diagram of Intel 8255 A

The pins for various ports are as follows:

$PA_0 - PA_7$	8 pins of port *A*
$PB_0 - PB_7$	8 pins of port *B*
$PC_0 - PC_3$	4 pins of port C_{lower}.
$PC_4 - PC_7$	4 pins of port C_{upper}.

The important control signals are as follows:

$\overline{CS}$ (Chip Select). It is a chip select signal. The LOW status of this signal enables communication between the CPU and 8255.

$\overline{RD}$ (Read). When $\overline{RD}$ goes LOW, the 8255 sends out data or status information to the CPU on the data bus. In other words, it allows the CPU to read data or status information from 8255.

$\overline{WR}$ (Write). When $\overline{WR}$ goes LOW, the CPU writes data or control word into 8255. The CPU writes data into the output port of 8255 and the control word into the control word register.

A_0 and A_1. The selection of ports and control word register is done using A_0 and A_1 in conjunction with $\overline{RD}$ and $\overline{WR}$. A_0 and A_1 are normally connected to the least significant bits of the address bus. If two 8255 units are used the address of ports are as follows:

For the 1st unit of 8255, *i.e.* 8255.1

Port/Control word register	*Port/Control word register Address*
Port A	00
Port B	01
Port C	02
Control word register	03

For the 2nd unit of 8255, *i.e.* 8255.2

Port/Control Word Register	*Port/Control Word Register Address*
Port A	08
Port B	09
Port C	0A
Control word register	0B

If we write the instruction IN 00, it means that it is for the Port A of 8255.1. When this instruction is executed data are transferred from the Port A to the accumulator. The instruction OUT 03 will transfer the content of the accumulator to the control word register of 8255.1. The instruction IN 09 transfers the data from Port B of 8255.2 to the accumulator. OUT 0A transfers the content of the accumulator to the Port C of 8255.2. The instruction OUT 0B transfers the content of the accumulator to the control word register of 8255.2. The IN instruction is used for the port which has been defined as input port. After IN the address of the port is specified. Similarly, OUT instruction is used for the ports which have been defined as output ports. After OUT instruction the address of the port is specified. For control word register only OUT instruction is used, IN instruction is not used. Its content can not be read. How ports are defined as input or output ports has been discussed in section 7.7.4.

Operating Modes of 8255. The Intel 8255 has the following three modes of operation which are selected by software:

Mode 0–Simple Input/Output
Mode 1–Strobed Input/output
Mode 2–Bidirectional Port.

The 8255 has two 8-bit ports (Port A and Port B) and two 4-bit ports (Port C_{upper} and Port C_{lower}). In Mode 0 operation a port can be operated as a simple input or output port. Each of the four ports of 8255 can be programmed to be either an input or output port.

Mode 1 is strobed input/output mode of operation. The Port A and Port B both are designed to operate in this mode of operation. When Port A and Port B are programmed in Mode 1, six pins of port C are used for their control. PC_0, PC_1 and PC_2 are used for the control of the Port B which can be used either as input or output port. If the Port A is operated as an input port, PC_3, PC_4 and PC_5 are used for its control. The remaining pins of Port C *i.e.* PC_6 and PC_7 can be used as either input or output. When

Port A is operated as an output port, pins PC_3, PC_6 and PC_7 are used for its control. The pins PC_4 and PC_5 can be used either as input or output.

The combination of Mode 1 and Mode 0 operation is also possible. For example, when Port A is programmed to operate in Mode 1, the Port B can be operated in Mode 0.

Mode 2 is strobed bidirectional mode of operation. In this mode Port A can be programmed to operate as a bidirectional port. The Mode 2 operation is only for Port A. When Port A is programmed in Mode 2, the Port B can be used in either Mode 1 or Mode 0. For Mode 2 operation PC_3 to PC_7 are used for the control of Port A.

7.7.3 Control Groups

For the control purpose 24 lines of I/O ports are divided into two groups, namely Group A and Group B. The group A contains the Port A and the Port C_{upper}. The Group B contains the Port B and the Port C_{lower}. The function of each port is programmed as desired. A control word is formed which contains the information regarding the function and modes of the ports. The CPU outputs the control word to 8255. The control word initialises the ports. Each of control blocks (*i.e.* Group A and Group B) accepts commands from Read/Write Control Logic which is within 8255. The control blocks receive control words from internal data bus of 8255 and issue the proper commands to their associated ports.

7.7.4 Control Word

According to the requirement a port can be programmed to act either as an input port or an output port. For programming the ports of 8255 a control word is formed. The bits of the control word are as shown in Fig. 7.15. Control word is written into the control word register which is within 8255. No read operation of the control word register is allowed. The control word bit corresponding to a particular port is set to either 1 or 0 depending upon the definition of the port, whether it is to be made an input port or output port. If a particular port is to be made an input port, the bit corresponding to that port is set to 1. For making a port an output port, the corresponding bit for the port is set to 0. The detailed description of the bits of the control word is as follows:

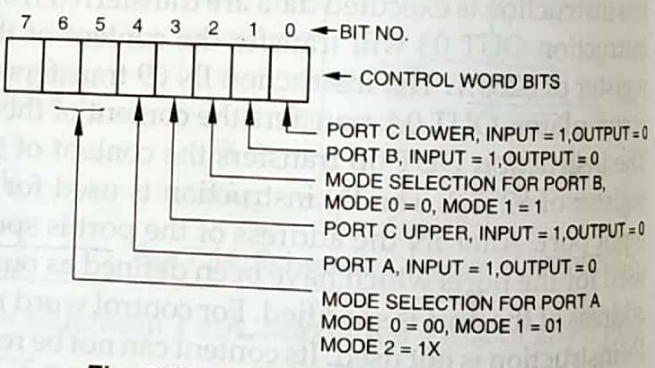

Fig. 7.15 Control Word Bits for Intel 8255

Bit No. 0. It is for Port C_{lower}.
To make Port C_{lower} an input port, the bit is set to 1.
To make Port C_{lower} an output port, the bit is set to 0.

Bit No. 1. It is for Port B.
To make Port B an input port, the bit is set to 1.
To make Port B an output port, the bit is set to 0.

Bit No. 2. It is for the selection of the mode for the Port B. If the Port B has to operate in Mode 0, the bit is set to 0. For Mode 1 operation of the port B, it is set to 1.

Bit No. 3. It is for the Port C_{upper}.
To make Port C_{upper} an input port, the bit is set to 1.
To make Port C_{upper} an output port, the bit is set to 0.

Bit No. 4. It is for Port A.
To make Port A an input port, the bit is set to 1.
To make Port A an output Port, the bit is set to 0.

Bit No. 5 and 6. These bits are to define the operating mode of the Port A. For the various modes of Port A these bits are defined as follows:

Mode of Port A	*Bit No. 6*	*Bit No. 5*
Mode 0	0	0
Mode 1	0	1
Mode 2	1	0 or 1

For mode 2 bit No. 5 is set to either 0 or 1; it is immaterial.

Bit No. 7. It is set to 1 if Port A, B and C are defined as input/output port. It is set to 0 if the individual pins of the Port C are to be set or reset.

Table 7.5 shows control words for various configurations of the ports of 8255 for Mode 0 operation. The following examples will illustrate how to make control words:

Table 7.5 Control Words for 8255A for Mode 0 Operation

Control Word Bits								*Control word*	*Port A*	*Port C_{upper}*	*Port B*	*Port C_{lower}*
7	6	5	4	3	2	1	0					
1	0	0	0	0	0	0	0	80	Output	Output	Output	Output
1	0	0	0	0	0	0	1	81	Output	Output	Output	Input
1	0	0	0	0	0	1	0	82	Output	Output	Input	Output
1	0	0	0	0	0	1	1	83	Output	Output	Input	Input
1	0	0	0	1	0	0	0	88	Output	Input	Output	Output
1	0	0	0	1	0	0	1	89	Output	Input	Output	Input
1	0	0	0	1	0	1	0	8A	Output	Input	Input	Output
1	0	0	0	1	0	1	1	8B	Output	Input	Input	Input
1	0	0	1	0	0	0	0	90	Input	Output	Output	Output
1	0	0	1	0	0	0	1	91	Input	Output	Output	Input
1	0	0	1	0	0	1	0	92	Input	Output	Input	Output
1	0	0	1	0	0	1	1	93	Input	Output	Input	Input
1	0	0	1	1	0	0	0	98	Input	Input	Output	Output
1	0	0	1	1	0	0	1	99	Input	Input	Output	Input
1	0	0	1	1	0	1	0	9A	Input	Input	Input	Output
1	0	0	1	1	0	1	1	9B	Input	Input	Input	Input

Example 1. Make control word when the ports of Intel 8255 are defined as follows:
Port A as an input port.
Mode of the Port A—Mode 0.
Port B as an output port.
Mode of the Port B—Mode 0.
Port C_{upper} as an input port.
Port C_{lower} as an output port.

If we assume all undefined bits 0, the control word is C2. The computer ignores the undefined bits. For making the control word they may be assumed either 1 or 0, it is immaterial.

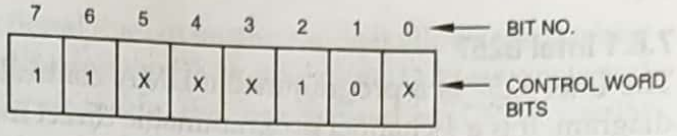

Fig. 7.23

Example 10. Write the control word for the following configuration of the ports of Intel 8255 for Mode 2 operation:

Port A — bidirectional

Mode of the Port A — Mode 2

Port B — output

Mode of the Port B — Mode 1

As the Port B is operated in Mode 1, PC_0, PC_1 and PC_2 are used for control. The bits of the control word for above configuration of the ports are shown in Fig. 7.23.

As the Port C_{lower} is used for control signals, the Bit No. 0 is undefined.

As the Port C_{upper} is also used for control signals, the Bit No. 3 is undefined.

The Bit No. 4 is undefined because the Port A operates in Mode 2.

For Mode 2 operation of the Port A, the Bit No. 6 is set to 1 and the Bit No. 5 is undefined.

If all undefined bits are assumed 0, the control word is C4.

7.8 PROGRAMMABLE DMA CONTROLLER

The bulk data transfer from fast I/O devices to the memory or from the memory to I/O devices through the accumulator is a time consuming process. For such a situation the direct memory access (DMA) technique is preferred. In DMA data transfer scheme, data are directly transferred from an I/O device to RAM or from RAM to an I/O device. For DMA data transfer, the data and address buses come under the control of the peripheral device which wants DMA data transfer. The microprocessor has to relinquish the control of the address and data buses for DMA operation on the request of the I/O device. For DMA data transfer the I/O device must have its own registers to store byte count and memory address. It must also be able to generate control signals required for DMA data transfer. Generally, such facilities are not available with I/O devices. Single chip programmable DMA controllers have been developed by several manufactures for the interfacing of I/O devices to the microprocessor for DMA data transfer. Such controllers provide all the facilities for DMA data transfer. Intel 8257 and 8237 are described in the subsequent sections.

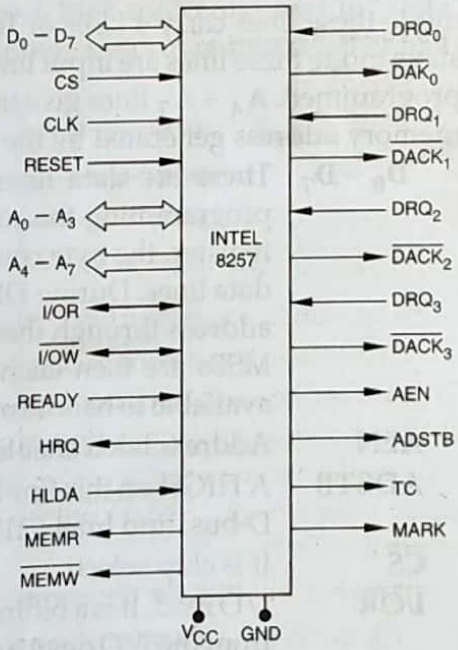

Fig. 7.24 Schematic Diagram of Intel 8257

7.8.1 Intel 8257

The Intel 8257 is a programmable DMA controller. Figure 7.24 shows its schematic diagram. It is a 4-channel programmable direct memory access (DMA) controller. It is in a 40 pin I.C. package and requires a single + 5 V supply for its operation. Four I/O devices can be interfaced to the microprocessor through this device. It is capable of performing three operations, namely read, write and verify. During the read operation data are directly transferred from the memory to the I/O device. During the write operation data are transferred from the I/O device to the memory. On receiving a request from an I/O device, the 8257 generates a sequential memory address which allows the I/O device to read or write directly to or from the memory. Each channel incorporates two 16-bit registers, namely (i) DMA address register and (ii) byte count register. These registers are initialised before a channel is enabled. Initially, the DMA address register is loaded with the address of the first memory location to be accessed. During DMA operation it stores the next memory location to be accessed in the next DMA cycle. 14-LSBs of the byte count register store the number of bytes to be transferred. 2^{14} (16384) bytes of data can directly be transferred to the memory from the I/O device or from the memory to the I/O device. 2 MSBs of the byte count register indicate the operation which will be performed by the controller on that channel. Besides these registers the 8257 also includes a mode set register and a status register.

Important pins of Intel 8257 are as follows:

DRQ0 – DRQ3. These are DMA request lines. An I/O device sends its DMA request on one of these lines. A HIGH status of the line generates a DMA request.

$\overline{\text{DACK0}}$ – $\overline{\text{DACK3}}$. These are DMA acknowledge lines. The Intel 8257 sends an acknowledge signal through one of these lines informing an I/O device that it has been selected for DMA data transfer. A LOW on the line acknowledges the I/O device.

$A_0 - A_7$. These are address lines. $A_0 - A_3$ are bidirectional lines. In the master mode these lines carry 4 LSBs of 16-bit memory address generated by the 8257. In the slave mode these lines are input lines. The inputs select one of the registers to be read or programmed. $A_4 - A_7$ lines give tristated outputs which carry 4 through 7 of the 16-bit memory address generated by the 8257.

$D_0 - D_7$ These are data lines. These are bidirectional three-state lines. While programming the controller the CPU sends data for the DMA address register, the byte count register and the mode set register through these data lines. During DMA cycle, the 8257 sends the 8 MSBs of the memory address through these lines at the beginning of the DMA cycle. These 8 MSBs are then latched in 8212 latch. Thereafter, the data bus is made available to handle memory data transfer during rest of the DMA cycle.

AEN Address latch enable.

ADSTB A HIGH on this line latches the 8 MSBs of the address, which are sent on D-bus, into Intel 8212 connected for this purpose.

$\overline{\text{CS}}$ It is chip select.

$\overline{\text{I/OR}}$ I/O read. It is a bidirectional line. In output mode it is used to access data from the I/O device during the DMA write cycle. In input mode this line is used by CPU to read status register of 8259.

fixed priority mode or rotating mode of operation. READY line is used by slow memory or I/O devices.

7.8.2 Intel 8237A

The 8237A is a high performance DMA controller. It is compatible with 8086, 8088 and 8085 microprocessors. It is used in PC, PC/XT and PC/AT. It contains four independent DMA channels. All channels have autoinitialisation capability. After the end of a data transfer process each channel can autoinitialize to its original condition. It is accomplished by software. It has memory to memory transfer, and automatic address increment and decrement features. Data transfer rate up to 1.6 MB/second is possible with 8237A-5.

Figure 7.25 shows the pin diagram of 8237A. Pins are similar to those of 8257. Descriptions of most of the pins can be seen in the preceding section. Some pins which are new or have different nomenclature are as follows:

$\overline{\text{EOP}}$	End of process. $\overline{\text{EOP}}$ goes low when data transfer is complete. The 8237 also allows an external signal to terminate an active DMA service through this line.
$\text{DB}_0 - \text{DB}_7$	Bidirectional data bus.
$\overline{\text{IOR}}$	I/O read.
$\overline{\text{IOW}}$	I/O write.
DREQ	These are DMA request lines.

Registers of 8237A

Current Address Register. Each channel contains a 16-bit current address register. After each transfer the word count is decremented. It holds the memory address during DMA transfer. The address can be automatically incremented or decremented following each data transfer. It can be reinitialised back to original value by an autoinitialization which can take place only after an $\overline{\text{EOP}}$ has occurred.

Current Word Count Register. Each channel contains a 16-bit current word count register. The number of transfers is determined by this register. It can also be reinitialised by an autoinitialisation back to its original value after the end of DMA service. Autoinitialisation can take place only when an $\overline{\text{EOP}}$ occurs.

Base Address and Base Word Count Registers. Each channel is provided with these registers. These 16-bit registers hold the original value of the address and word count respectively. During autoinitialisation these values are required to restore the current registers to their original values.

In addition to above registers there are some more registers such as command register, mode register, request register, mask register, status register and temporary register. The details can be seen in Intel's handbook.

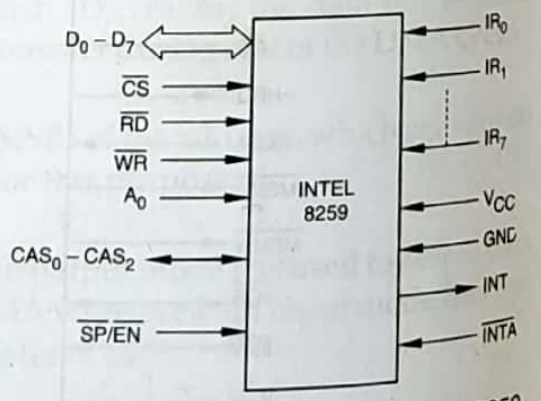

Fig. 7.26 Schematic Diagram of Intel 8259

For DMA read cycle in which data are transferred from memory to an I/O device, the 8237 makes both the $\overline{\text{MEMR}}$ and $\overline{\text{IOW}}$ low. The $\overline{\text{MEMR}}$ enables the addressed memory for reading data from it. The $\overline{\text{IOW}}$ enables the I/O device to accept data. Similarly, for DMA write cycle in which data are

transferred from the I/O device to the memory, the 8237 makes both the $\overline{MEMW}$ and $\overline{IOR}$ low. The $\overline{MEMW}$ enables memory for writing data to it. The $\overline{IOR}$ enables I/O device to output data.

7.9 PROGRAMMABLE INTERRUPT CONTROLLER (PIC) INTEL 8259

The programmable interrupt controller is used when several I/O devices transfer data using interrupt and they are to be connected to the same interrupt line of the microprocessor. When the number of I/O devices is less than the number of interrupt lines of the microprocessor, such controllers are not required. The Intel 8259 is a single chip programmable interrupt controller. It is compatible with 8086, 8088 and 8085 microprocessors. It is a 28-pin DIP I.C. package and uses N-MOS technology. It requires a single +5 V supply for its operation. Figure 7.26 shows the schematic diagram of Intel 8259. The details of its pins are as follows:

$\overline{CS}$ Chip select.

$\overline{WR}$ Write. A LOW on this pin enables Intel 8259 to accept command word from CPU.

$\overline{RD}$ Read. A LOW on this pin enables Intel 8259 to send various status signals on the data bus for CPU.

$D_0 - D_7$ Bidirectional data bus. Control, status and interrupt vector information are transferred via this bus.

$CAS_0 - CAS_2$ Cascade lines.

$\overline{SP}/\overline{EN}$ Slave program/Enable buffer. It is related to cascade control.

INT Interrupt. It is used to interrupt CPU.

$\overline{INTA}$ Interrupt acknowledge.

$IR_0 - IR_7$ Interrupt requests. I/O devices send interrupt request through these lines.

A_0 Address line. It acts in conjunction with $\overline{RD}$, $\overline{WR}$ and $\overline{CS}$. The Intel 8259 uses it to interpret command words the CPU writes and status the CPU wants to read.

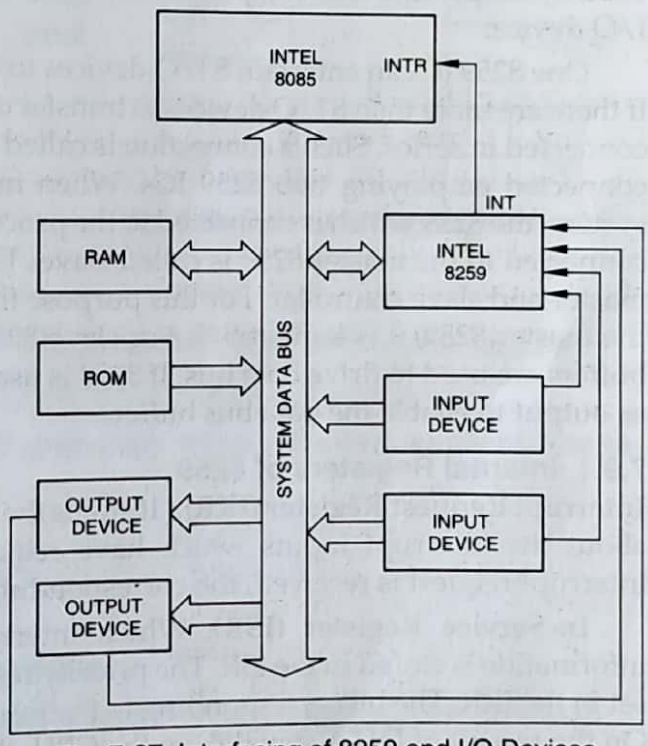

Fig. 7.27 Interfacing of 8259 and I/O Devices

Figure 7.27 shows the block diagram of connections of PIC and I/O devices to the microcomputer. The priority can be assigned to the I/O devices connected to PIC. 8 I/O devices can be connected to 8259 through $IR_0 - IR_7$ lines. The interrupt controller functions as overall manager in an interrupt driven system. One or more I/O devices can send interrupt request to the interrupt controller at the same time through IR_0-IR_7 lines. If more than one I/O devices send interrupt requests at the same time, the interrupt controller determines their priority. It first entertains the I/O device of higher priority. It sends an interrupt request to the microprocessor through INT line. The microprocessor sends an acknowledgment through the $\overline{INTA}$ line. On the receipt of $\overline{INTA}$ signal all the interrupts of lower priority are inhibited. In case of 8085 system, the 8259 sends a binary code to the processor through the data bus. After receiving this code (which is a CALL instruction code) the processor sends two more $\overline{INTA}$ pulses. These two $\overline{INTA}$ pulses allow the 8259 to send the address of the ISS for the concerned I/O device which is to be served. The 8 LSBs of the address is released at the first $\overline{INTA}$ pulse and the 8 MSBs of the address is released at the second $\overline{INTA}$ pulse.

In case of 8086-based system, the processor sends two consecutive $\overline{INTA}$ pulses after receiving the interrupt request from the 8259. On the receipt of the first $\overline{INTA}$ pulse the 8259 makes necessary setting of its internal registers. On the receipt of the second $\overline{INTA}$ pulse, the 8259 sends an 8-bit pointer to the processor. The 8-bit pointer is used by the processor to compute the starting memory address of ISS for the concerned I/O device.

One 8259 IC can entertain 8 I/O devices to transfer data using interrupt technique. If there are more than 8 I/O devices to transfer data using interrupt, two 8259 ICs can be connected in series. Such a connection is called **cascading**. Up to 64 I/O devices can be connected employing two 8259 ICs. When more than one 8259 are employed in a system, the 8259 which is connected to the processor is called master. The 8259 which is connected to the master 8259 is called slave. The $\overline{SP}/\overline{EN}$ signal is used to distinguish master and slave controller. For this purpose this signal is used as an input signal. For the master 8259, it is kept at 5 V. For slave 8259 it is kept at 0 volt. In a large system buffers are used to drive data bus. If 8259 is used in buffered mode, the $\overline{SP}/\overline{EN}$ is used as output to enable the data bus buffer.

7.9.1 Internal Registers of 8259

Interrupt Request Register (IRR). It stores the interrupt requests. It keeps information about the interrupt inputs which have requested for interrupt service. When an interrupt request is received, the corresponding bit in IRR is set.

In-Service Register (ISR). Which interrupt is currently being serviced; this information is stored in the ISR. The priority resolver determines the priority of the bits set in the IRR. The bit corresponding to the interrupt of the highest priority is selected. On the receipt of $\overline{INTA}$ signal from the CPU, the highest priority ISR bit is set and the corresponding IRR bit is reset.

Interrupt Mask Register (IMR). It contains a specific bit for each interrupt line. It is used to mask (disable) or enable (unmask) individual interrupt input. An interrupt input can be masked by setting the corresponding bit to 1 in IMR. An interrupt which is masked by software (by programming IMR) is not recognised and serviced even if the corresponding bit is set in the IRR.

7.9.2 The 82C59

The 82C59 is the CHMOS version of NMOS 8259. It consumes less power.

7.10 PROGRAMMABLE COMMUNICATION INTERFACE (INTEL 8251)

Programmable communication interface is used for serial data transmission. The Intel 8251 is a programmable communication interface. It is Universal Synchronous/ Asynchronous Receiver/ Transmitter (USART). It is compatible with 8085, 8086, 8088 and 8748 system. The I.C. chip is fabricated using N-channel silicon gate technology. The 8251 can be used to transmit/receive serial data. It accepts data in parallel format from the microprocessor and converts them into serial data for transmission. It also receives serial data and converts them into parallel and sends the data in parallel format to the CPU. Figure 7.28 shows a schematic diagram of Intel 8251. The description of some important pins is as follows:

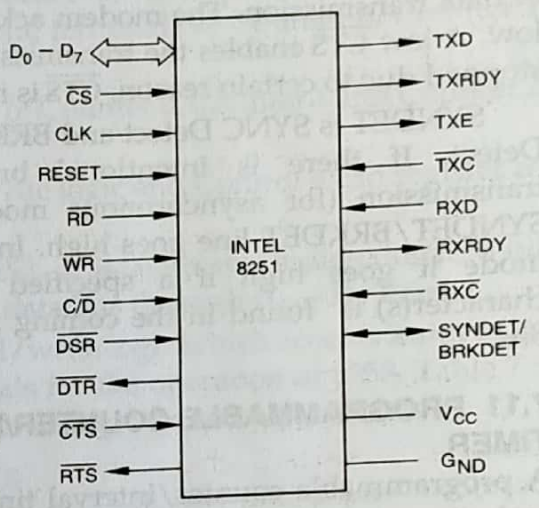

Fig. 7.28 Schematic Diagram of Intel 8251

$C/\overline{D}$ (Control/data). When it is low data is transmitted on data bus. When it is high control signals are transmitted on the data bus.

$\overline{RD}$ (Read). When it is low the CPU reads data from Intel 8251.

$\overline{WR}$ (Write). When it is low the CPU writes data into Intel 8251.

$\overline{DSR}$ Data set ready.

$\overline{DTR}$ Data terminal ready.

$\overline{RTS}$ Request to send.

$\overline{CTS}$ (Clear to send). A low on this pin enables 8251 to transmit serial data.

$\overline{TxC}$ (Transmitter clock). It governs the rate of data transmission.

TxE (Transmitter empty). TxE goes high when 8251 has no characters to transmit.

TxRDY Transmitter ready.

RxRDY Receiver ready.

RxD Line for receiving data.

TxD Line for serial data transmission.

RxC (Receiver clock). It governs the rate at which characters are received.

A high on TxRDY line informs CPU that the transmitter is ready to accept data. When data are received from CPU, this line becomes low. On receiving the data byte the 8251 sends serial data on TxD line, when $\overline{CTS}$ is low and the transmitter is enabled by CPU. The clock input at $\overline{TxC}$ governs the rate of serial data transmission. In synchronous transmission mode, the baud rate is equal to the clock frequency. In asynchronous transmission mode the baud rate is equal to the clock frequency or a fraction of the clock frequency. The fraction may be 1/16 or 1/64. Serial data are received on RxD line. RxRDY is normally low. When serial data are received by 8251, it

becomes high, and informs CPU that the 8251 contains data which are ready to be sent to CPU.

$\overline{DSR}$, $\overline{DTR}$, $\overline{RTS}$ and $\overline{CTS}$ are standard modem control signals. These control lines are connected to the corresponding lines of the modem being used. Modem is a modulating/demodulating device. It adds carrier frequency for data transmission over a high frequency channel. It removes carrier frequency when data are received from a high frequency channel.

The 8251 sends a low $\overline{DTR}$ signal to modem to indicate that it is ready to send data. By making $\overline{DSR}$ low the modem replies that it is also ready for onward transmission of data to the communication system. When modem and USART both are ready for data transmission, the 8251 sends a low $\overline{RTS}$ signal to modem for the initiation of data transmission. The modem acknowledges this request of 8251 by making $\overline{CTS}$ low. A low $\overline{CTS}$ enables the transmission logic of 8251. If data transmission has to be stopped due to certain reason, $\overline{CTS}$ is made high.

SYNDET is SYNC Detect and BRKDET is Break Detect. If there is intentional break in data transmission (for asynchronous mode only), the SYNDET/BRKDET line goes high. In synchronous mode it goes high if a specified synchronous character(s) is found in the coming string of data bits.

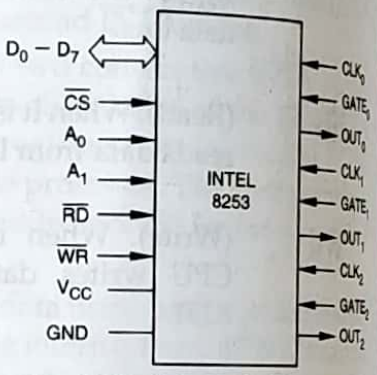

Fig. 7.29 Schematic Diagram of Intel 8253

7.11 PROGRAMMABLE COUNTER/INTERVAL TIMER

A programmable counter/interval timer is used in real time application for timing and counting function such as BCD/binary counting, generation of accurate time delay, generation of square wave of desired frequency, rate generation, hardware/software triggered strobe signal, one-shot signal of desired width, etc. Popular programmable interval timer chips are Intel 8253 and 8254. The 8253 operates in the frequency range of d.c. to 2.6 MHz while 8254 in the range of d.c. to 10 MHz. The 8253 uses NMOS technology whereas 8254 HMOS technology. Both are pin to pin compatible and operate in the following six modes:

Mode 0 : Interrupt on terminal count
Mode 1 : Programmable one-shot
Mode 2 : Rate generator
Mode 3 : Square wave generator
Mode 4 : Software triggered mode
Mode 5 : Hardware triggered mode

The 8254 is compatible to 8086, 8088, 8085 and most other microprocessors. The 8253 is compatible to 8085 microprocessor. The 8254 is a superset of the 8253. The 8253 will be discussed in detail in the next section as the work has been done with 8085 microprocessor-kit containing 8253 chip. The descriptions also hold good for 8254.

7.11.1 Intel 8253

The 8253 is 24-pin IC and operates at 5 $V_{d.c.}$. It contains three independent 16-bit counters. The programmer can program 8253 to operate in any one of the 6 operating modes. It operates under software control. Figure 7.29 shows its pin diagram. The description of its important pins is as follows:

$\overline{RD}$ (Read). When this pin is low the CPU reads data.

$\overline{WR}$ (Write). When this is low, the CPU outputs data in the form of mode information or loading of counters.

A_0, A_1 These pins are connected to the address bus. These are used to select one of the three counters. These are also used to address the control word registers for mode selection.

$\overline{CS}$ Chip select.

$D_0 - D_7$ Bidirectional data bus.

CLK_0 CLK_1 and CLK_2 are clock for Counter 0, Counter 1 and Counter 2 respectively. $GATE_0$, $GATE_1$ and $GATE_2$ are gate terminals of Counter 0, Counter 1 and Counter 2 respectively.

OUT_0, OUT_1 and OUT_2 are output terminals of Counter 0, Counter 1 and Counter 2 respectively.

The 8253 contains a data bus buffer, read/write logic and Control word register as described below:

Data Bus Buffer. This buffer is within 8253. It is a 3-state, bidirectional, 8-bit buffer. It is used to interface 8253 to the system data bus through $D_0 - D_7$ lines.

Read/Write Logic. The 8253 contains a read/write logic which accepts input from the system bus and then generates control signals for the operation of 8253. Table 7.6 shows the status of pins associated with read/write logic for various controls.

Table 7.6

$\overline{CS}$	A_1	A_0	$\overline{RD}$	$\overline{WR}$	
0	0	0	0	1	Read Counter No. 0
0	0	1	0	1	Read Counter No. 1
0	1	0	0	1	Read Counter No. 2
0	0	0	1	0	Load Counter No. 0
0	0	1	1	0	Load Counter No. 1
0	1	0	1	0	Load Counter No. 2
0	1	1	1	0	Write Mode Word
0	1	1	0	1	No-Operation 3-State
0	X	X	1	1	No-Operation 3-State
1	X	X	X	X	Disable 3-State

Note: X indicates undefined state. It means that it does not matter whether the state is 0 or 1.

Control Word Register. When the pins A_0, A_1 are 11, the control word register is selected. The control word format is shown below:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

The bits D_7 and D_6 of the control word are to select one of the 3 counters. D_5 and D_4 are for loading/reading the count. D_3, D_2 and D_1 are for the selection of operating

mode of the selected counter. There are six modes of operation for each counter of 8253. The selected counter can be programmed to operate in any desired mode. The six modes of operation are: MODE 0, MODE 1, MODE 2, MODE 3, MODE 4 and MODE 5. The bit D_0 is for the selection of binary or BCD counting. When D_0 is set to 0, the selected counter operates as a binary counter. When it is set to 1, the counter operates as a BCD counter.

The function of counters of 8253 are programmed by the system software. The control word decides the selection of counter, its mode of operation, loading, sequence of the count and selection of binary or BCD counting. As soon as the control word is written into the control word register the desired counter is selected; mode of operation is set, and the loading sequence of the count and the selection of binary or BCD counting are defined.

SC (Select Counter)

To select the counter SC0 and SC1 are set as follows:

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Illegal

RL (Read/Load)

To load/read counts RL0 and RL1 are set as follows:

RL1	RL0	
0	0	Counter latching operation.
0	1	Read/Load least significant byte only.
1	0	Read/Load most significant byte only.
1	1	Read/Load least significant byte first, then most significant byte.

Mode. Mode selecting bits M0, M1 and M2 are set as follows:

M2	M1	M0	
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

Note: X indicates undefined state. It does not matter whether it is 0 or 1.

BCD

0 Binary counter 16-bits

1 Binary coded decimal (BCD) counter (4 Decades).

7.11.2 Reading while Counting

There is a command for latching the content of a counter to read its count while count is still going on. The bit pattern for the control word for latching operation is as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	0	0	X	X	X	X

SC1 and SC0 — specify counter to be latched.
D_5 and D_4 – 00 makes counter latching operation.
X - indicates "don't care".

On receiving the latch command, the 8253 latches the content of the selected counter into a storage register. Now, the microcomputer reads the content of the storage register by issuing a normal read command. The read operation is done as specified by RL1 and RL0 bits of the original mode set command, *i.e.* reading of only LSBs of the count or only MSBs of the count or read LSBs of the count first, then MSBs of the count. The latching command is to be issued every time when the microcomputer wants to latch the counter contents first and then read it. The counter's mode or its operation is not affected by the latching command.

7.11.3 Operation

The 8253 contains three independent negative edge triggered presentable 16-bit down counters namely, counter 0, counter 1 and counter 2. As the counters are fully independent each counter can have separate mode configuration and counting operation. Generally, the 8253 is available on a microprocessor kit. One of the counters is used for single step operation. In Vinytics kit terminals of two counters namely counter 1 and counter 2 are available at connector space P1 and P2. The counter 0 has been used for single step operation. Though the connections of counter 1 are available to users, it is also used for generating programmable baud rate while using 8251. The clock for 8253 should be less than 2.6 MHz. The frequency of the clock output of 8085 is about 3 MHz. An edge triggered flip-flop 7474 has been used to divide this clock frequency by two. Thus a clock frequency of about 1.5 MHz is available on the kit. The three terminals available at connector space for each counter are CLK, GATE and OUT. In Vinytics microprocessor kit CLK terminal of one of the counters (whose terminals are available at connector space) is connected to 1.5 MHz clock. CLK terminal of the other counter has been left unconnected. The user can use the clock frequency of his choice for this counter. The user must check the CLK terminal of the counter which he is going to use, whether it has already been connected to the clock or he has to apply a suitable clock frequency to it.

The addresses for control word register and the counters of 8253 for Vinytics microprocessor kit are as follows:

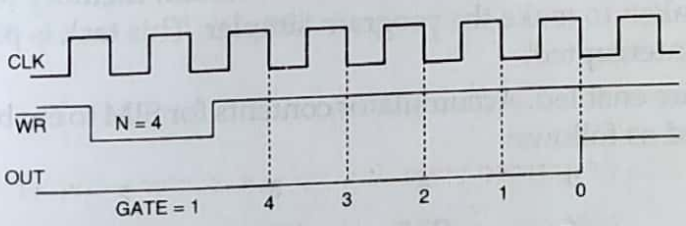

Fig. 7.30 MODE 0, Interrupt on Terminal Count

Counter 0 — 10
Counter 1 — 11
Counter 2 — 12
Control Word register — 13

A counter can be used for various applications such as BCD/binary counter, programmable rate generator, square wave generator, hardware/software triggered

strobe, programmable one-shot, to generate time delay etc. Descriptions of some applications are given below:

7.11.4 MODE 0: Interrupt on Terminal Count

Mode 0 is used for the generation of accurate time delay under software control. One of the counters of 8253 is initialised and loaded with suitable count for the desired time delay. When counting is over the counter interrupts CPU. On interruption the microprocessor performs the required task which is to be performed after the desired time delay.

The mode is set by loading the control word into the control word register. In MODE 0 operation the output of the counter becomes initially low after the mode is set. After mode set operation the counter is loaded by the desired count N. As soon as the count register is loaded counting starts if GATE is high. For MODE 0 operation GATE is kept high. While counting is going on the counter output OUT remains low. When the terminal count is reached *i.e.* count reaches 0, the output becomes high and remains high until the count is reloaded or a new count is loaded. Figure 7.30 shows timing diagram for MODE 0 operation. When the count is reloaded or a new count is loaded, OUT becomes low and the counter starts its counting operation again.

If GATE is made low during counting operation, counting stops. When GATE returns to 1, counting is resumed from the count at which the counting had discontinued. The count value can be changed at any time. A new count can also be loaded when counting is going on. After the loading of the new count, counting restarts from the new value of the count. When the new count is of one byte, counting starts immediately after loading the new value of the count. When the new count is of two bytes, the counter stops counting after the 1st byte of the new count is loaded. The counter starts counting only after the second byte of the new count is loaded.

The OUT terminal is used to interrupt the microprocessor. This mode of operation is used to generate precise time delay. It can be used when the microcomputer wants to execute one task exactly `*t*' second after performing another task.

Example 1. Use 8253 in MODE 0 to perform a particular task after certain delay. Take count for delay = FF50 hex. After counting is over, the counter is required to interrupt microprocessor to transfer contents of memory locations FD01 and FD02 to memory locations FC31 and FC32.

A very simple task to transfer the content of certain memory location to another location has been taken to make the program simpler. This task is performed after the microprocessor is interrupted.

All interrupts are enabled. Accumulator contents for SIM to enable RST 7.5, 6.5 and 5.5 are programmed as follows:

7	6	5	4	3	2	1	0
SOD	SOE	X	R7.5	MSE	M7.5	M6.5	M5.5
0	0	0	0	1	0	0	0 = 08

See explanation of this bit pattern in Section 7.5.3, Interrupts of Intel 8085. The control word for MODE 0 operation is formed as follows.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
0	1	1	1	0	0	0	0	= 70 hex

D_7 and D_6 have been set to 01 to initialize counter 1. D_5 and D_4 have been set 11 to load the least significant byte of the count first, then most significant byte.

D_3, D_2, D_1 are set to 000 for MODE 0.

D_0 is set to 0 as counting is to be done in binary because the given count value is in hexadecimal.

Connect OUT terminal of counter 1 to RST 7.5.

PROGRAM

Memory address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
	FB	EI		Enable all interrupts.
FC00	3E, 08	MVI	A, 08	Load bit pattern to enable RST
FC01				7.5, 6.5 and 5.5.
	30	SIM		Enable RST 7.5, 6.5 and 5.5.
FC03	3E, 70	MVI	A, 70	Control word for MODE 0 to
FC04				initialize counter 1
	D3, 13	OUT	13	Write into control word register.
FC06	3E, 50	MVI	A, 50 H	Least significant byte of the count.
FC08	D3, 11	OUT	11	Load counter 1 by LSBs of count.
FC0A	3E, FF	MVI	A, FF	MSB of count.
FC0C	D3, 11	OUT	11	Load counter by MSBs of count.
FC0E	C3, 10, FC	JMP	HERE	
FC10				

When RST 7.5 interrupts microprocessor, the program jumps to the memory location 003C. Then the monitor transfers the program from 003C location to FFBD. Now, the user has to transfer the program from FFBD to the strating address of the service subroutine. In this case service subroutine starts from memory location FC20.

FF3D	C3, 20, FC	JMP		Jump to service subroutine

SERVICE SUBROUTINE

FC20	2A, 01, FD	LHLD	FD01	Get contents of FD01 and FD02 in H-L pair.
FC23	22, 31, FC	SHLD	FC31	Store them in FC31 and FC32.
FC26	FB	EI		Enable all interrupts
FC27	C9	RET		

DATA

FD01 — 5B
FD02 — 38

Result

FC31 — 5B
FC32 — 38

Store the desired data in memory locations FD01 and FD02 and execute the program. See the result in memory locations FC31 and FC32. You will see that contents of FD01 and FD02 are transferred tc memory locations FC31 and FC32 respectively. Before the program returns from the service subroutine to the main program, all interrupts are enabled so that any further interrupts may be recognized.

Example 2. Use 8253 as a simple counter. Count N = FFFF. After certain delay read the count when counting is still going on.

The control word for MODE 0 to initialize counter 1 is determined as explained in Example 1. That is = 70 H. The counter acts as a binary counter. While counting is still going on the count value can be latched and read. For this purpose the control word is different than that for loading the count.

The control word for loading the count in counter 1 in MODE 0 is as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	1	1	1	0	0	0	0 = 70 H

This control word is for mode set command.

For latching only the bits D_5 and D_4 are 00, other bits remain same as those for loading. The control word for latching is as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	1	0	0	0	0	0	0 = 40 H

In the control word for latching only bits $D_4 - D_7$ are effective. $D_0 - D_3$ are ignored. But for simplicity $D_0 - D_3$ have been kept same as they were in the original mode set command.

PROGRAM

Memory address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
FC00	3E, 70	MVI	A, 70 H	Get control word for MODE 0 to initialize counter 1.
FC02	D3, 13	OUT	13	Write into control word register.
FC04	3E, FF	MVI	A, FF	Get LSBs of count.
FC06	D3, 11	OUT	11	Load LSBs of count into counter 1.
FC08	3E, FF	MVI	A, FF	Get MSBs of count.
FC0A	D3, 11	OUT	11	Load MSBs of count into counter 1.
FC0C	CD, 30, FC	CALL	DELAY	Produce desired delay.
FC0F	3E, 40	MVI	A, 40 H	Get control word for latching the count.
FC11	D3, 13	OUT	13	Write into control word register.
FC13	DB, 11	IN	11	Read LSBs of count.
FC15	32, 01, FD	STA	FD01	Store it in location FD01.
FC18	DB, 11	IN	11	Read MSBs of count
FC1A	32, 02, FD	STA	FD02	Store it in location FD02.
FC1D	76	HLT		Stop.

DELAY SUBROUTINE

FC30	06, FF		MVI	B, FF	Get count in register B for desired delay.
FC32	05	LOOP	DCR B		Decrement count of register B.

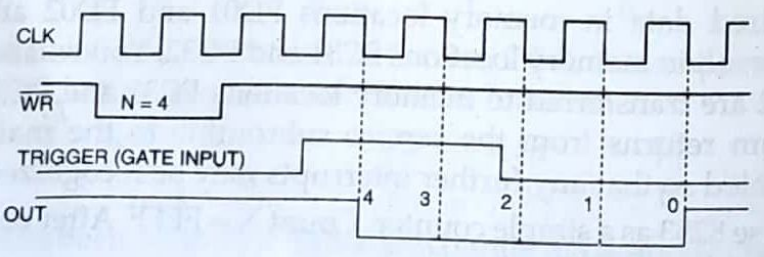

Fig. 7.31 MODE 1, Programmable one-shot

```
                C2, 32, FC          JNZ  LOOP
FC33            C9                  RET
FC36
```

Result
FD01 – DD LSBs of count
FD02 – F1 MSBs of count

The count value after the given delay is F1DD hex. The counter has decremented count by (FFFF – F1DD) = 0E22 in the given period of time.

Now, change the count value which has been loaded in counter 1 and see the result. For N = 60, 00 H:

Result
FD01 — DE LSBs of count.
FD02 — 51 MSBs of count.

Counter has counted (60, 00 – 51 DE) = 0E22. Since the delay time is same as that in the previous case, the counter has decremented the count by same value, *i.e.* 0E22.

Similarly, change delay period and count value of counter 1, and see results.

7.11.5 MODE 1 : Programmable One-Shot

In MODE 1 the counter acts as a retriggerable and programmable one-shot. The low to high transition of the signal applied to GATE acts as a trigger signal. In this mode of operation OUT becomes initially high after the mode is set. After mode set operation the counter is loaded by a count value of N. Then the counter decrements count, and the output (OUT) goes low for N clock cycles for every low to high transition of the GATE input. The output goes low at the first negative edge of the clock after the rising edge of the GATE input as shown in Fig. 7.31. The output remains low while counting is going on. It goes high on the terminal count (*i.e.* counter content = 0). The width of the output pulse can be varied by varying N. As the width of the output pluse is programmable this mode of operation is known as perogrammable one-shot.

If the GATE input is made low to high again, the counter is reloaded by the count value N. The counter starts decrementing the count once again, and the output goes low for N clock pulses again. Thus one-shot mode of operation is retriggerable.

If a new count is loaded while OUT is still low, the duration of the one-shot pulse is not affected until the occurrence of the succeeding trigger.

Consider situation when the occurrence of low to high transition of the GATE input has initialised a counter, counting is going on and OUT is still low. Now, suppose another low to high transition of the GATE signal occurs. This will force the output to remain low for N clock pluses after the occurrence of the second low to high transition of the GATE signal.

Example. Use 8253 in MODE 1 and see the wave shape of the output on CRO.

To test the working of 8253 in MODE 1 in the laboratory the program as described below was developed. The control word for counter 1 and MODE 1 was determined as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	1	0	1	0	0	1	0 = 52 H

D_7 and D_6 are set to 01 to select counter 1.

D_5 and D_4 are set to 01 for loading only LSB of the count.

D_3, D_2 and D_1 are set to 001 for MODE 1.

D_0 is set to 0 for binary counting.

PROGRAM

Memory address	*Machine Codes*	*Labels*	*Mnemonics*	*Operands*	*Comments*
FC00	3E, 52		MVI	A, 52H	Get control word to initialize counter 1 in MODE 1.
FC02	D3, 13		OUT	13	Write into control word register.
FC04	3E, 08		MVI	A, 08	Get count.
FC06	D3, 11		OUT	11	Load counter 1 with the count.
FC08	3E, 80		MVI	A, 80 H	Initialize PPI 8255.2
FC0A	D3, 0B		OUT	0B	

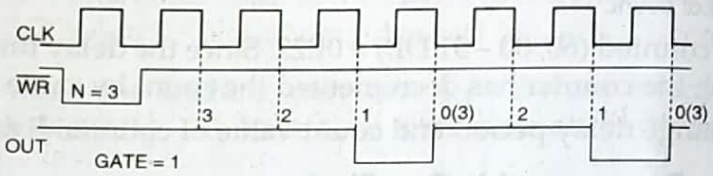

Fig. 7.32 MODE 2, Rate Generator

FC0C	3E, 00	LOOP	MVI	A, 00	Generate a pulse to be output at PC_0 which is connected to GATE of 8253.
FC0E	D3, 0A		OUT	0A	
FC10	3E, 01		MVI	A, 01	
FB12	D3, 0A		OUT	0A	
FC14	C3, 0C, FC		JMP	LOOP	

See the wave shape of the output and CLK on CRO. The pin PC_0 has been connected to GATE of 8253. A pulse is generated by the microprocessor and it is output at PC_0. The pulse is applied to GATE of 8253. In MODE 1 the counting starts following a low to high transition of the GATE input. Therefore, the triggering signal has to be applied to GATE. The count has been kept low because the counting must be over before another pluse is applied to GATE. If the count is kept sufficiently high, a delay period has to be included before the program jumps from FC14 to FC0C. The width of the pulse can be varied by varying the count value. Introducing a loop in the program the triggering signal is applied to GATE again. This results in the repetition of the pulse so that it can be seen on CRO.

7.11.6 MODE 2 : RATE Generator

In MODE 2 the counter acts as a simple divide by N counter. When this mode is set, the output of the counter becomes initially high. After mode set operation the counter is loaded by a count of value N. For MODE 2 operation GATE is kept high. In this mode the output remains high for (N – 1) clock pulses and then goes low for one clock pulse as shown in Fig. 7.32. After this, the output becomes high again and the count N is automatically reloaded into counter. Again the output remains high for (N – 1) clock pulses and goes low for one clock pulse. In this way the whole process is repeated as long as the GATE input is high. If the GATE input is made low the counter is disabled and the output stays high. When GATE returns to high, the counter resumes its normal operation and starts from initial count.

If a new count is loaded into the count register between output pulses, the present period is not affected, but the subsequent period reflects the new value of the count.

Example 1. Use 8253 in MODE 2 to act as a device-by-N binary counter. N = 16 decimal = 10 hexadecimal.

The counter 2 has been used for this purpose. The control word has been determined as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	1	0	0 = 94 hex

D_7 and D_6 are the bits of control word to select counter. They have been set to 10 to select counter 2. D_5 and D_4 are for loading count N. They have been set to 01 to load only the least significant byte of the count. In this example N = 16, so only LSBs of the count are required. D_3, D_2 and D_1 are for mode select. For MODE 2 they have been set to 010 = 2.

D_0 is for BCD or binary counting. It has been set to 0 for binary counting.

For counter 2, GATE 2, OUT 2 and CLK 2 terminals are available at connector space of Vinytics microprocessor kit. See the manual for the microprocessor kit which you are using to note down the correct connector for the counter 2 and correct pins for GATE 2, OUT 2 and CLK 2. In some Vinytics kit terminals of counter 1 are at connector space P_1 and in some kits at P_2. Similarly, the pin numbers are not in the same fashion for counter 1 and counter 2. Therefore, the user must ascertain the connector space and pins for the counter he is using.

Apply + 5 to GATE to make it high. Check terminal CLK whether clock has been connected to it by the manufacturer or not. If not, connect the clock 1.5 MHz which is available at the output terminal of 7474 or use your own suitable clock of other frequency as desired. Connect CLK and OUT to CRO to see the waves.

PROGRAM

Memory address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
FC00	3E, 94	MVI	A, 94 H	Get control word for counter 2, MODE 2, binary counting.
FC02	D3, 13	OUT	13	13 is address for writing control word into microprocessor kit used.
FC04	3E, 10	MVI	A, 10 H	Get count value N. Put it in hexadecimal. 10 hex = 16 decimal.
FC06	D3, 12	OUT	12	12 is the address for counter 2.
FC08	76	HLT		Stop

94 is the control word for MODE 2 operation. It has been determined for counter 2 for binary counting. 13 is the address for control word register, and 12 for counter 2 for the microprocessor kit which has been used in the laboratory. On CRO it can be seen that the output will become low for one clock pulse for every 16 clock pulses as shown in Fig. 7.32. For binary counting the count value N is kept in hexadecimal.

Example 2. Use 8253 in MODE 2 to act as a rate generator. The counter is to act as a BCD (decade) counter.

N = 10 decimal.

The counter 2 has been used. The control word for MODE 2 using counter 2 for BCD counting is as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

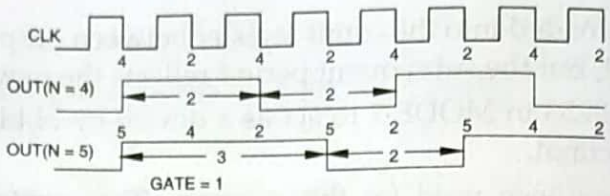

Fig. 7.33 MODE 3, Square Wave Generator

1	0	0	1	0	1	0	
							1 = 95 hex

D_7 and D_6 have been set to 10 to select counter 2.

D_5 and D_4 have been set to 01 for loading only the least significant byte of the count.

D_3, D_2 and D_1 have been set to 010 = 2 for MODE 2.

D_0 has been set to 1 for BCD counting

PROGRAM

Memory address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
FC00	3E, 95	MVI	A, 95 H	Get control word to initialize counter 2.
FC02	D3, 13	OUT	13	Write into the control word register.
FC04	3E, 10	MVI	A, 10	Get count in decimal, N = 10.
FC06	D3, 12	OUT	12	Load counter 2 with count.
FC08	76	HLT		Stop.

For BCD counter the control word is 95. The count N = 10 has been taken in decimal. In case of BCD counter the output will go low for one clock period for every 10 clock periods. As the control word is for BCD counter, the 8253 treats N as BCD number. See waves of CLK and OUT on CRO.

7.11.7 MODE 3 : Square Wave Generators

In MODE 3 the counter acts as a square wave generator. After mode set operation the counter is loaded by a count of value N. For MODE 3 operation GATE is kept high. For even values of N the output remains high for N/2 clock pulses and then goes low for next N/2 clock pulses. On the terminal count, the output state is changed and the counter is automatically reloaded with the full count and the whole process is repeated. Thus a continuous square wave is obtained. For odd values of N the output remains high for $\frac{N+1}{2}$ clock pluses and low for $\frac{N-1}{2}$ clock pulses. Wave shapes are shown in Fig. 7.33. When GATE is made low, the counter is disabled and the output stays high. When GATE returns to high, the counter resumes its normal operation.

Example 1. Use 8253 in MODE 3 as a square wave generator.

N = 16. The counter operates as a binary counter.

The counter 2 has been used for the purpose and the control word has been determined as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	1	1	0 = 96 hex

D_7 and D_6 have been set to 10 to select counter 2.

D_5 and D_4 have been set to 01 for loading only LSBs of the count.

D_3, D_2 and D_1 have been set to 011 for MODE 3.
D_0 has been set to 0 for binary counting.

PROGRAM

Memory address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
FC00	3E, 96	MVI	A, 96 H	Get control word for MODE 3 to initialize counter 2.
FC02	D3, 13	OUT	13	Write into control word register.
FC04	3E, 10	MVI	A, 10 H	Get count in hexadecimal for binary counting.
FC06	D3, 12	OUT	12	Load counter 2 with the count.
FC08	76	HLT		Stop.

See wave shape of the output on CRO. Waves will be as shown in Fig. 7.33. For N = 16 the output remains high for 8 clock cycles and then low for next 8 clock cycles. For N = 17 (odd number), the output remains high for 9 clock pulses and then low for next 8 clock pulses. A continuous wave is obtained as the process is repeated automatically by the timer.

Example 2. Use one counter of 8253 to generate square wave and another for rate generation. N = 8 for square wave. N = 6 for rate generation.

The counter 2 has been used for square wave generation in MODE 3. Counting is done in binary. The control word is 96 H as explained in Example 1.

The Counter 1 has been used for rate generation in MODE 2. The control word is determined as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	1	0	1	0	1	0	0 = 54 hex

D_7 and D_6 have been set to 01 to select counter 1.

D_5 and D_4 have been set to 01 for loading only LSBs of the count.

D_3, D_2, D_1 have been set to 010 for MODE 2.

D_0 has been set to 0 for binary counting.

PROGRAM

Memory address,	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
FC00	3E, 96	MVI	A, 96 H	Get control word to initialize counter 2 in MODE 3.
FC02	D3, 13	OUT	13	Write into control word register.
FC04	3E, 08	MVI	A, 08 H	Get count N = 08.
FC06	D3, 12	OUT	12	Load counter 2 with the count.
FC08	3E, 54	MVI	A, 54 H	Get control word to initialize counter 1 in MODE 2.
FC0A	D3, 13	OUT	13	Load control word register.
FC0C	3E, 06	MVI	A, 06 H	Get count N = 06.
FC0E	D3, 11	OUT	11	Load counter 1 with count.
FC10	76	HLT		Stop.

See the wave shapes of the output of counter 2 and counter 1. The output of counter 2 remains high for 4 clock pulses and then goes low for next 4 clock pluses. You will get a continuous square wave as the process is repeated again and again by the timer. The

output of the counter 1 remains high for 5 clock pulses and then goes low for one clock period. This process is repeated.

Example. 3 Use 8253 as a square wave generator. Use counts for BCD counting. N = 10 decimal.

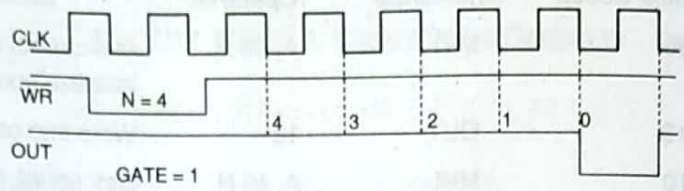

Fig. 7.34 MODE 4, Software Triggered Strobe

The control word will be determined as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	1	1	1 = 97 hex

D_7 and D_6 have been set to 10 to select counter 2.

D_5 and D_4 have been set to 01 for loading only LSBs of the count.

D_3, D_2 and D_1 have been set to 011 for MODE 3.

D_0 has been set to 1 for BCD counting.

PROGRAM

Memory address,	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
FC00	3E, 97	MVI	A, 97 H	Get control for MODE 3, BCD counting.
FC02	D3, 13	OUT	13	Load control word register.
FC04	3E, 10	MVI	A, 10	Get count 10, for BCD counting, it will be taken in decimal.
FC06	D3, 12	OUT	12	Load counter 2 with count. .
FC08	76	HLT		Stop.

See wave shape of the output on CRO. The output remains high for 5 clock pulses and then it goes low for next 5 clock pulses. Also check for N = 11 (odd value). The output will remain high for 6 clock pulses and then low for next 5 clock pulses. A continuous square wave is obtained as the process is repeated automatically by the timer.

7.11.8 MODE 4: Software Triggered Strobe

In MODE 4 operation the output of the counter becomes initially high after the mode is set. GATE is kept high for this mode of operation. The counter begins counting immediately after the count is loaded into the count register. When the counter reaches terminal count (*i.e.* counter content = 0), the output goes low for one clock period, then it returns to high as shown in Fig. 7.34. The output signal may be used as strobe. This mode of operation is referred to as a software triggered strobe because the generation of the strobe signal is triggered by loading the count into the count register.

Example. Use 8253 in MODE 4 and see the output wave on CRO.

To test the working of 8253 in the laboratory in MODE 4 the program as described below was developed. The control word for counter 1 for MODE 4 operation was determined as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

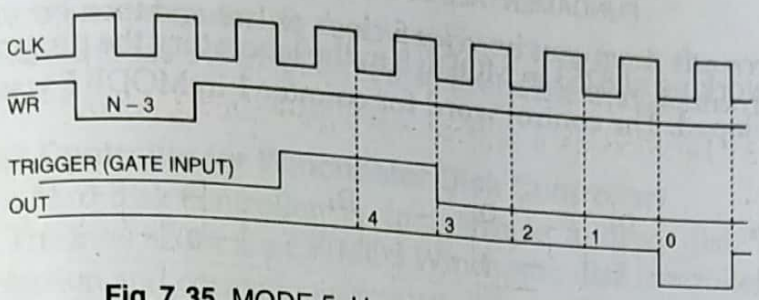

Fig. 7.35 MODE 5, Hardware Triggered Strobe

0	1	1	1	1	0	0	0 = 78 hex

D_7 and D_6 have been set to 01 to select counter 1.

D_5 and D_4 have been set to 11 for loading the LSBs of the count first, then MSBs of the count.

D_3, D_2 and D_1 have been set to 100 = 4 for MODE 4.

D_0 has been set to 0 for binary counting.

PROGRAM

Memory address	*Machine Codes*	*Labels*	*Mnemonics*	*Operands*	*Comments*
FC00	3E, 78		MVI	A, 78 H	Get control word to initialize counter 1 in Mode 4.
FC02	D3, 13		OUT	13	Write into control word register.
FC04	3E, 05	LOOP	MVI	A, 05	Get LSBs of the count.
FC06	D3, 11		OUT	11	Load counter 1 with LSBs of the count.
FC08	3E, 00		MVI	A, 00	Get MSBs of the count.
FC0A	D3, 11		OUT	11	Load counter 1 with MSBs of the count.
FC0C	C3, 04, FC		JMP LOOP		

See the wave shape of the CLK and output on CRO. Keep GATE high. It can be seen on CRO that the output goes low for one clock period. The count has been kept very low because the counting should be over before the program reloads the counter. If the count is sufficiently large, a delay has to be introduced before the program jumps from the memory location FC0C to FC04. Introducing a loop in the program the strobe signal has been generated again and again so that it can be seen on CRO.

7.11.9 MODE 5: Hardware Triggered Strobe

In this mode of operation GATE input acts as a trigger. After the mode is set, the output becomes initially high. A count value of N is loaded into the counter. Following a low to high transition of the GATE input, the counter starts decrementing the count. The counting begins at the first negative edge of the clock after the rising edge of the GATE input as shown in Fig. 7.35. On terminal count the output goes low for one clock period, then it goes high again.

As the low to high transition of the GATE input causes triggering, this mode is referred to as hardware triggered strobe.

This mode of operation is also retriggerable like MODE 1. If the GATE input is made low to high again, the counter is reloaded by the count value N. The counter starts decrementing the count once again and the output goes low for one clock period on terminal count.

Example. USE 8253 in MODE 5 and see the wave shape of the output on CRO.

To test the working of 8253 in MODE 5 in the laboratory the program as described below was developed. The control word for counter 1 in MODE 5 was determined as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	1	1	1	1	0	1	0 = 7A hex

D_7 and D_6 are set to 01 to select counter. 1.

D_5 and D_4 are set to 11 for loading the LSBs of the count first, then MSBs of the count.

D_3, D_2 and D_1 are set to 101 = 5 for MODE 5.

D_0 is set to 0 for binary counting.

PROGRAM

Memory address	*Machine Codes*	*Labels*	*Mnemonics*	*Operands*	*Comments*
FC00	3E, 7A		MVI	A, 7A	Get control word to initialize counter 1 in MODE 5
FC02	D3, 13		OUT	13	Write into control word register
FC04	3E, 08		MVI	A, 08	Get LSBs of count.
FC06	D3, 11		OUT	11	Load counter 1 with LSBs of the count.
FC08	3E, 00		MVI	A, 00	Get MSBs of the count.
FC0A	D3, 11		OUT	11	Load MSBs of count into counter 1.
FC0C	3E, 80	LOOP	MVI	A, 80 H	Initialize ports of 8255.2.
FC0E	D3, 0B		OUT	0B	
FC10	3E, 00		MVI	A, 00	Generate a pulse to be output at PC_0 which is connected to GATE of 8253.
FC12	D3, 00		OUT	0A	
FC14	3E, 01		MVI	A, 01	
FC16	D3, 0A		OUT	0A	
FC18	C3, 0C, FC		JMP	LOOP	

See the wave shape of the output and clock on CRO. The pin PC_0 of 8255.2 has been connected to the GATE. A pulse is generated by the microprocessor and it is available on PC_0. This pulse is applied to the Gate. In MODE 5 the counting starts following a low to high transition of the GATE input. Therefore, the triggering signal has to be applied to the GATE. The count has been kept at a low value because the counting must be over before another pulse is applied to the GATE. If the count is kept sufficiently large, a delay period has to be introduced before the program jumps from FC18 to FC0C. The loop introduced in the program makes the mode of operation retriggerable resulting in the repetition of the output signal so that it can be seen on CRO.

7.12 SPECIAL PURPOSE INTERFACING DEVICES

Special purpose interfacing devices have been developed to interface special I/O devices such as CRT, optical disk system, dot matrix printer, keyboard etc. Very brief descriptions of a few devices are given below. For details the reader should see the Intel microprocessor and Peripheral Handbook.

7.12.1 Programmable CRT Controller (Intel 8275 H)

The Intel 8275H is a programmable CRT controller. It is a single clip device. It is a 40 pin I.C. package and requires + 5 V supply. Its function is to interface CRT raster scan

display with the microcomputer. It receives characters from the system memory, displays them and refreshes the display. It also keeps the track of the display position on the screen.

7.12.2 Hard Disk Controller (or Winchester Disk Controller)

The function of a hard disk controller is to interface a hard disk system to the microprocessor. The Intel 82064 is a CHMOS Winchester disk controller. It contains an on-chip error detection and correction circuitry. It uses CHMOS III technology. It is a 40-pin IC and needs + 5 V supply. It is compatible with all Intel and most other microprocessors. It interfaces hard disk drives to the processor. It can control hard disk drives which employ Seagate Technology ST506/ST412 interface. Its data transfer rate is 5 MB/second. It has 128, 256, 512 or 1024 bytes/sector. It has multiple sector transfer capability.

7.12.3 Dot Matrix Printer Controller (Intel 8295)

The Intel 8295 is a dot martix printer controller. It is a 40 pin I.C. package. Its function is to interface LRC 7040 series dot martix printers to microprocessor. It can also be used to interface other similar printers. It may be used in serial or parallel mode of operation with the microcomputer. Data are transferred using DMA, interrupts or polling in case of parallel mode of operation. There is an internal buffering of up to 40 characters. The 8295 contains a 7×7 matrix character generator to accommodate 64 ASCII characters.

7.12.4 Programmable Keyboard/Display Interface (Intel 8279)

The Intel 8279 is a programmable keyboard interfacing device. Data input and display are the integral part of microprocessor kits and microprocessor-based systems. The designer requires a suitable interface for the same. The 8279 has been designed for the purpose of 8-bit Intel microprocessors. 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 microprocessor. It can also interface toggle or thumb switches. The purpose of the display section is to drive alphanumeric displays or indicator lights. It is directly connected to the microprocessor bus. The microprocessor is relieved from the burden of scanning the keyboard or refreshing the display. 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.

7.12.5 Memory Controllers

A dynamic RAM stores data on gate to source capacitor of a transistor. Due to leakage, the charge on these stray capacitors leaks away after a few milliseconds. Therefore, dynamic RAMs must be periodically refreshed, usually every 2 miliseconds or less. For this purpose a dynamic RAM requires refreshing circuitry. Intel has developed dynamic RAM controllers for refreshing dynamic RAMs. Some of them will be described in this section. An error detection and correction IC will also be described.

Intel 8203. The 8203 is a 64K dynamic RAM controller. It can directly address and drive up to 64 devices without external drivers. It generates all signals necessary to control 64K and 16K dynamic RAMs. It is compatible with 8085A, 8086 and 8088 family of microprocessors. It operates in two modes; one for 64K DRAM and other for 16K DRAM. It is a 40-pin IC and operates at + 5 V supply.

Intel 8207. The 8207 is dual-port DRAM controller. It is capable of providing all necessary signals to control 16K, 64K and 256K DRAMS. It can directly address up to 2 MB without external drivers. It can provide necessary signals to control an error detection and correction unit (Intel 8206), if it is in the system. It allows two different buses to access memory independently. It is compatible with 80286, 8086, 8088, 80186 and 80188 microprocessors. It is a 68-pin IC and requires + 5 V supply for its operation.

Intel 82C08. The 82C08 is a CHMOS dynamic RAM controller. It can support 64K and 256K DRAMs. It is compatible with 80286, 8086, 8088, 80186 and 80188 microprocessors. It can directly address up to 1MB without external drivers. It consumes less power and can operate with battery back up in case of power failure. It is available in 48 and 68-pin packages. It requires +5 V supply. it has zero wait state with Intel microprocessors.

Intel 8206. The 8206 is an error detection and correction unit. It is a 68-pin, 5V IC. It is capable of detecting and correcting errors for both SRAMs and DRAMs. It detects all single bit and double bit, and most multiple bit errors. It is a high speed device, being implemented in HMOS III technology.

7.12.6 Cache Controller

The instruction codes and data which are currently needed by the processor, are stored in a cache memory. These informations are copied from the main memory by the cache controller. When the processor sends an address of an instruction code or data, the cache controller examines whether the required contents of the specified address are present in the cache memory or not. If the required code or data is present in the cache memory, the cache controller enables the cache memory to send the required code/data to the processor. If the required code/data is not present in the cache memory, the cache controller enables the controller of the main memory to send the required code/data from the main memory. The main memory sends the required code/data on the data lines. From the data lines the required code/data is sent to the processor as well as cache memory.

Intel 82496, 82497 and 82498 Cache Controllers. These controllers have been developed to control second-level cache in the Pentium-based computers. They handle concurrently the CPU bus, memory bus and cache operation for maximum performance. The main features provided are: synchronous, asynchronous and strobed memory bus operation; selectable bus width, line sizes, transfers and burst orders. The 82496 and 82497 support 32, 64 and 128-bit wide memory bus; 16, 32 and 64-byte line sizes; and 256 KB to 512 KB cache memory with parity. The 82498 supports 64 and 128-bit wide memory bus; 32 and 64-byte line sizes; and 1M byte to 2M byte cache memory. The 82496, 82497 and 82498 cache controllers control 82491, 82492 and 82493 SRAM cache respectively.

The Pentium Pro microprocessor contains the second-level cache and its controller within the processor-chip itself.

7.12.7 Multifunction Microprocessor Support Controller, Intel 8256AH

It is a multifunction IC. It contains programmable universal asynchronous receiver-transmitter (UART), five 8-bit programmable timer/counters out of which four can be cascaded to two 16-bit timer/counters, two 8-bit programmable I/O ports (the Port 1 can be programmed for Port 2 handshake control and event counter inputs), 8-level priority interrupt controller and a programmable system clock unit, etc.

CHAPTER 10

MICROCONTROLLERS

INTRODUCTION

A microcomputer built on a single semiconductor chip is called single-chip microcomputer. It is used for dedicated applications such as automatic control of equipment, machines and process in industry, instrumentation, commercial and consumer appliances. Since the single-chip microcomputers are generally used in control applications, they are also called **microcontrollers**. Single-chip microcontrollers are embedded in the system which they control and, therefore, they are also called **embedded controllers**. The microcontrollers are particularly suitable for real-time control (event control or closed-loop control).

A single-chip microcomputer contains all essential components of a microcomputer such as CPU, RAM, ROM/EPROM/OTPROM, I/O lines, etc. Some single-chip microcomputers contain devices to perform specific functions such as DMA channels, A/D converter, serial port, pulse-width modulation, etc. All microcontrollers do not contain all sorts of specific functions. Rather, they possess some selected specific functions to meet the need of certain applications for which such microcontrollers are designed and manufactured. In case of a microcontroller, program is fixed. The user does not prepare program for a particular application. A program for a microcontroller is developed in a laboratory and tested. Then it is stored in ROM/EPROM/OTPROM of the microcontroller. Hence, ROM/EPROM/OTPROM of a microcontroller is known as **program memory**. As data and intermediate results are stored in RAM, it is called **data memory**.

In 1976, Intel introduced the 8048 series of single-chip microcontrollers. It is also known as MCS-48. It contains 8-bit CPU, 1/2/4K ROM/EPROM, 64/128/256 bytes of RAM, timer/counter, parallel I/O lines and A/D converter. The microcontrollers of this series are: 8048, 8748, 8041, 8042, 8049 and 8035. These microcontrollers were popular and widely used for automatic control applications. These microcontrollers do not have multiplication and division instructions.

In 1980, Intel introduced a more powerful 8051 series of 8-bit microcontrollers. These are the second generation microcontrollers. They are faster, contain a number of additional electronic circuits, have enhanced instruction set including instructions for multiplication and division, and more memory capacity.

Other companies also developed 4-bit and 8-bit microcontrollers. Examples of 4-bit microcontrollers are: Texas Instruments TMS1000 (It was the first single-chip microcontroller. It was developed in the year 1974), Hitachi's HMCS40, National's COP420, OKI's MSM6411, Toshiba's TLCS47, etc. Examples of 8-bit microcontrollers are: Motorola's 6801, 6805 and MC68HC11, Zilog's Z8 and Z86C83, NEC's 7800 series PD7810, PD7811 and PD7814, National's COP820, Rockwell's 6500/1, Philips' 87C552, Texas Instruments' TMS7500 and TMS370CO50, Microchip's PIC16C56, Atmel 89C51

and 89C2051, PIC16C6X, 16C7X and 16F8XX etc. 4-bit microcontrollers are used for appliances and toys control. 8-bit microcontrollers are used for simple industrial control applications, instrumentation, etc.

In 1983, Intel introduced 16-bit microcontrollers, the 8096 series. These are more powerful than 8-bit microcontrollers. Later on, Intel introduced 80C196 series of 16-bit microcontrollers for sophisticated industrial and commercial control applications, instrumentation, etc.

16-bit microcontrollers of some other companies are: Texas Instruments' TMS9940, Mostek's MK68200, Digital Equipment's LSI-11, Hitachi's H8/532, National's HPC16164, etc.

32-bit microcontrollers have been developed by IBM and Motorola. MPC505 is a 32-bit RISC microcontroller of Motorola. The 403GA is a 32-bit RISC embedded controller of IBM. 32-bit microcontrollers are used for sophisticated control applications.

10.1 INTEL 8051 SERIES OF MICROCONTROLLERS (MCS-51)

The 8051 series of microcontrollers were developed in the year 1980. They are the second generation of 8-bit microcontrollers. They are faster and more powerful than Intel 8048 series (1976) of microcontrollers. They have been provided with improved instruction set and contain a number of additional electronic circuitry for specific functions. The 8-bit microcontrollers are used for a variety of applications involving limited calculations and relatively simple control strategies. They are used for industrial and commercial control applications, appliances control, instrumentation, etc. The 8051 contains Boolean processor, full duplex serial port and power saving circuitry in addition to essential components such as 8-bit CPU, RAM, ROM/EPROM/OTPROM, timer/counter and parallel I/O lines. This series has a wide variety of versions with some of special functions such as DMA channels, A/D converter, pulse-width modulation, watch-dog timer, etc. Table 10.1 shows the details of its family members.

Table 10.1 Intel 8051 Series of Microcontrollers

Product	*On-chip Program Memory*	*On-Chip Data Memory in Bytes (RAM)*	*CLK MHz*	*Version Available*
80C51BH	4 KB ROM	128	24	
80C31BH	None	128	24	
87C51	4 KB EPROM	128	24	
8XC52/54/58	8/16/32 KB	256	12/24/33	80C32 ROMless, 80C52/54/58 with ROM, 87C52/54/58 with EPROM.
8XL52/54/58 (Low-Voltage Version)	8/16/32 KB	256	16/20	80L52/54/58 with ROM, 87L52/54/58 with OTP ROM.
8XC51FA/FB/FC	8/16/32 KB	256	12/24/33	80C51FA ROMless, 83C51FA/FB/FC with ROM, 87C51FA/FB/FC with EPROM.
8XL51FA/FB/FC (Low Voltage Version)	8/16/32 KB	256	20	80L51FA ROMless, 83L51FA/FB/FC with ROM, 87L51FA/FB/FC with OTPROM.
8XC51RA/RB/RC	8/16/32 KB	512	24	80C51RA ROMless, 83C51RA/RB/RC with ROM,

8XC51GB	8 KB	256	3.5 to 12/24	87C51 RA/RB/RC with EPROM 80C51GB ROMless, 83C51GB with ROM, 87C51GB with OTPROM
8XC51SL (low voltage version also available)	16 KB	256	2-16	80C51SL ROMless, 83C51SL with ROM, 87C51SL with OTPROM, 81C51SL with ROM programmed with system soft keyboard controlled and scanner firmware.
83C51KB	4 KB (ROM)	128	4-6	High performance keyboard microcontroller.
8XC152JA/JB/JC/JD	8 KB	256	3.5-16.5	80C152JA/JB/JC/JD ROMless, 83C152JA/JC with ROM.
8XC151SA/SB	8/16 KB	256	0-16	80C151SA ROMless, 83C151SA/SB with ROM, 87C151SA/SB with OTPROM.
8XC251SA/SB/SP/SQ	8/16 KB	512/1000	0-16	80C251SB/SQ ROMless, 83C251SA/SB/SP/SQ with ROM, 87C251SA/SB/SP/SQ with EPROM/OTPROM.

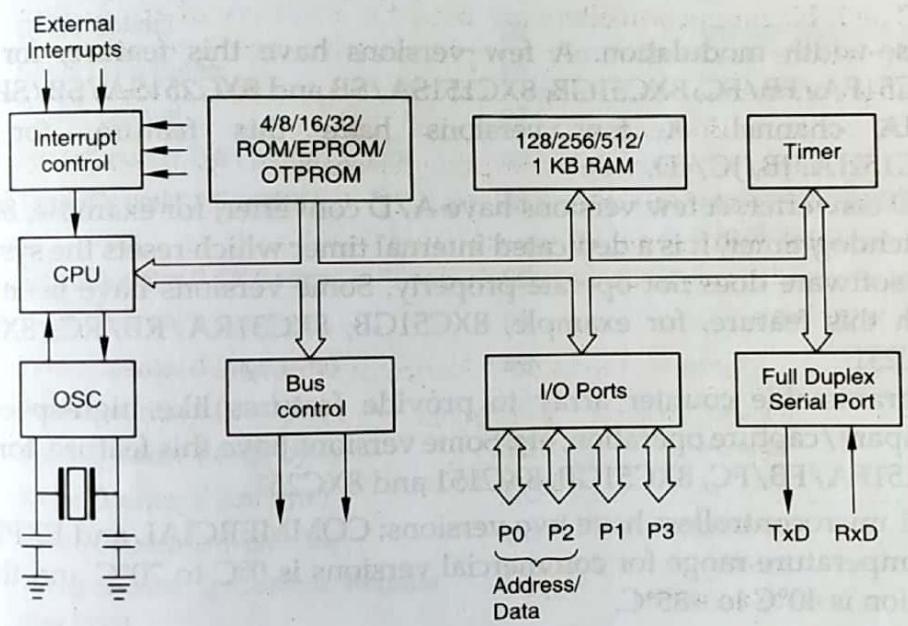

Fig. 10.1 Block Diagram of Intel 8051

Figure 10.1 shows the block diagram of Intel 8051. The common features of 8051 series of microcontrollers are:

(i) 8-bit CHMOS CPU

(ii) Most of the microcontrollers of 8051 family contain 256 bytes on-chip RAM which acts as data memory. The 8XC51RA/RB/RC and 8XC251SP/SQ contain 512 bytes, and the 8XC251SA/SB contains 1 KB RAM. A few entry level microcontrollers such as 80C51BH, 80C31BH and 87C51 contain 128 bytes RAM.

(iii) 4/8/16/32 KB ROM/EPROM/OTPROM. The capacity varies from version to version, see Table 10.1. ROMless versions are also available. In case of

ROMless versions external program memory is used. The capacity depends on the particular application.

(iv) Most of the microcontrollers contain 3 multimode 16-bit timer/counters. The 8XC152JA/JB/JC, 80C51BH, 80C31BH and 87C51 contain 2 timers/counters.

(v) Most of the microcontrollers contain 32 I/O lines i.e. four 8-bit ports. The 8XC152JA/JB/JC/JD, the universal communication controller contains 5 or 7 I/O ports. The 8XC51SL, keyboard controller contains 3 multifunction I/O ports.

(vi) Boolean processing (single-bit logic) capabilities. It is needed in industrial control applications.

(vii) Multimode, full duplex serial port. The 8XC152JA/JB/JC/JD contains UART.

(viii) The number of interrupt sources differs from version to version. It may be 5, 6, 7, 10 or 15. Four-level interrupt priority has been provided. The important interrupt sources are: one from the serial port when a transmission or reception is complete, two from timers, two from input pins INT0 and INT1, etc.

(ix) 64 KB external data memory space.

(x) 64 KB external program memory space.

(xi) Power saving modes.

Some versions are designed for specific applications, and therefore, they are provided with special features required by the typical applications. Special features are:

(i) Pulse-width modulation. A few versions have this feature, for example, 8XC51FA/FB/FC, 8XC51GB, 8XC151SA/SB and 8XC251SA/SB/SP/SQ.

(ii) DMA channel. A few versions have this feature, for example 8XC152JA/JB/JC/JD.

(iii) A/D converter. A few versions have A/D converter, for example, 8XC51GB.

(iv) Watchdog-timer. It is a dedicated internal timer which resets the system when the software does not operate properly. Some versions have been provided with this feature, for example, 8XC51GB, 8XC51RA/RB/RC, 8XC151 and 8XC251.

(v) Programmable counter array to provide features like high-speed output, compare/capture operation, etc. Some versions have this feature, for example, 8XC51FA/FB/FC, 8XC51GB, 8XC151 and 8XC251.

The 8051 microcontrollers have two versions: COMMERCIAL and EXPRESS. The operating temperature range for commercial versions is 0°C to 70°C and that for the express version is 40°C to +85°C.

10.1.1 Registers

The 8051 is an accumulator based microcontroller. Its registers are: register A (an accumulator), PSW, register B, 8-bit stack pointer, 16-bit data pointer, program counter, program address register, 16-bit timer registers for timer/counters, instruction register, control registers, RAM address register, serial data buffer, capture registers, special function registers, etc. Register B is used during multiply and divide operations. For other instructions it is used as another scratch pad register. The data pointer consists of a high byte and a low byte. It holds 16-bit address. It can be used as a 16-bit register or two independent 8-bit registers. The serial data buffer is actually two separate registers: a transmit buffer and a receive buffer register.

The 8051 has been provided with 4 banks of working registers. Each bank consists of 8 working registers, R0-R7. Physically these banks occupy the first 32 bytes of on-chip data RAM (address 0-1F hex). Only one bank is active at a time. Bits 3 and 4 of PSW decide which bank is to be made active. As the 8051 is a bit as well as byte microcontroller, some of its registers are both bit as well as byte addressable.

Besides working registers, there are a number of special function registers (SFRs). Some of SFRs are both bit- and byte-addressable. A list of SFRs is given below:

Symbol	*Name*	*Address*	*Remarks*
ACC	Accumulator	E0	Both bit- and byte-addressable
B	B register	F0	do
PSW	Program Status Word	D0	do
SP	Stack Pointer	81	—
DPTR	Data Pointer (DPH & DPL)	83 and 82	Consists of DPH and DPL
P0	Port 0	80	Both bit- and byte-addressable
P1	Port 1	90	do
P2	Port 2	A0	do
P3	Port 3	B0	do
IP	Interrupt Priority Control	B8	do
IE	Interrupt Enable Control	A8	do
SCON	Serial Control	98	Both bit- and byte-addressable
SBUF	Serial Data Buffer	99	—
PCON	Power Control	97	—
TMOD	Timer/Counter 0 &1 Mode Control	89	—
T2CON	Timer/Counter 2 Control	88	Both bit- and byte-addressable. It is in 8052 only.
TCON	Timer/Counter 0 &1 Control	C8	—
TL0	Timer/Counter 0 (low byte)	8A	—
TH0	Timer/Counter 0 (high byte)	8C	—
TL1	Timer/Counter 1 (low byte)	8B	—
TH1	Timer/Counter 1 (high byte)	8D	—
TL2	Timer/Counter 2 (low byte)	CC	—
TH2	Timer/Counter 2 (high byte)	CD	—
RCAP2L	Timer/Counter 2 Capture Register (low byte)	CA	—
RCAP2H	Timer/Counter 2 Capture Register (high byte)	CB	—

Description of some registers are given below:

Data Pointer. It consists of DPH (a high byte) and DPL (a low byte). It holds 16-bit address. It can be used as a 16-bit register or as two independent 8-bit registers.

P0, P1, P2 and P3. These are SFR latches for Port 0, 1, 2 and 3 respectively.

Serial Data Buffer. It consists of two separate registers, a transmit buffer register and a receive buffer register.

Timer Registers. (TL0, TH0), (TL1, TH1) and (TL2, TH2) are register pairs. These register pairs are 16-bit counting registers for Timer/Counter 0, 1 and 2 respectively.

Capture Registers. RCAP2L and RCAP2H is a register pair. These registers are capture registers for the Timer2 capture mode.

Control Registers. Special Function Registers IE, IP TMOD, TCON, T2CON and SCON hold control and status bits for the interrupt system, timer/counters, and the serial port. PCON is power control register. The 8051 is provided with power-saving modes of operation. For applications where power consumption is critical, both HMOS and CHMOS versions provide reduced power modes of operation. For CHMOS version of the 8051 microcontroller, the reduced power modes, Idle and Power Down modes are the standard features. In HMOS versions only reduced power mode is available.

PSW (Program Status Word). PSW register contains program status information as shown in Fig. 10.2. Its bits are indicated as PSW.0, PSW.1, PSW.2,.....,PSW.7.

7	6	5	4	3	2	1	0	← Bit No
CY	AC	F0	RS1	RS0	OV	–	P	← Status information

Fig. 10.2 PSW: Program Status Word Register

Bit No. 0, PSW.0. It is for parity status (parity flag, P).
Bit No. 1, PSW.1. Reserved.
Bit No. 2, PSW.2. Overflow flag (OV).
Bit No. 3, PSW.3. (RS0)
Bit No. 4, PSW.4. (RS1) — These bits are to select working register bank.
Bit No. 5, PSW.5. It is Flag 0 (F0) available to users for general purpose.
Bit No. 6, PSW.6. It is auxiliary carry flag (AC).
Bit No. 7, PSW.7. It is carry flag (CY).

Stack Pointer (SP). Intel 8051 microcontroller contains an 8-bit stack pointer register. It is incremented before data is stored during PUSH and CALL operations. It is decremented when POP or RET (Return) operation takes place. Any area of on-chip RAM can be used as stack.

Program Counter (PC). The Intel 8051 microcontroller contains a 16-bit Program Counter (PC) register. It points to the address of the next instruction of the program, which is to be fetched and executed. It is automatically incremented after fetching an instruction. It keeps the track of memory addresses of the instructions in the program being executed. It is affected by JUMP and CALL instructions.

10.1.2 Pins of Intel 8051

Figure 10.3 shows the pin diagram of Intel 8051 Microcontroller.

Pins 1 – 8 are for the Port 1, Pins 10 – 17 for Port 3, Pins 21 – 28 for Port 2 and Pins 32 – 39 for the Port 0. The alternate function of Pins 10 – 17 are given under the heading of I/O Lines (Next Subsection).

RST (Pin 9). It is the resetting pin for the device (8051).

XTAL2 (Pins 18). It is the output of inverting amplifier which is a part of the on-chip oscillator. When external clock is used, it is left unconnected.

XTAL1 (Pin 19). It is input to the inverting amplifier which is a part of the on-chip oscillator circuit. When external clock is used, it is connected to the external oscillator signal.

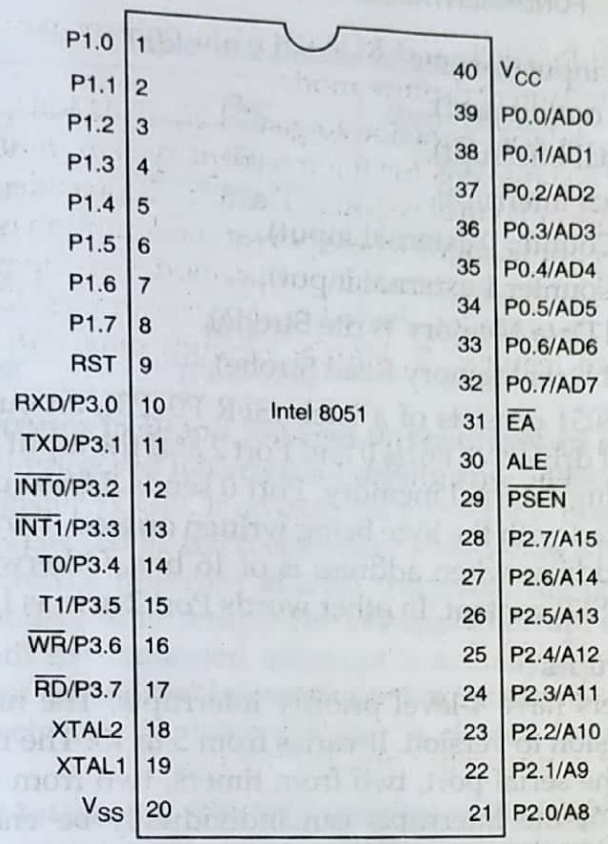

Fig. 10.3 Pin Diagram of Intel 8051 Microcontroller

V_{SS}(Pin 20). It is the circuit ground. All the voltage are specified with respect to it.

V_{CC}(Pin 40). It is for power supply, +5V.

$\overline{PSEN}$(Pin 29). It is program Store Enable. It is output control signal. It is a read strobe to external program memory.

ALE (Pin 30). It is Address Latch Enable output (control signal) for latching the low byte of the address during accesses to external memory.

$\overline{EA}$(Pin 31). It is External Access. It controls the access of program memory. See Table 10.5.

10.1.3 I/O Lines

Most of the 8051 microcontrollers contain four 8-bit parallel ports : P0, P1, P2 and P3. Altogether there are 32 I/O lines. All ports in 8051 are bidirectional. The I/O lines of 8051 are not simply input/output lines, rather they are multifunctional lines. If an application does not need any external memory besides on-chip memory, then all four ports can be used as input/output ports. If external memory is used, then Port 0 and Port 2 act as a multiplexed address/data bus. When external memory is employed, two lines of Port 3 are used to generate the $\overline{RD}$ and $\overline{WR}$ signals. Two pins of Port 3 act as RXD and TXD for serial data transmission. Two pins of Port 3 can be used as external input for timers, one for Timer 0 and one for Timer 1. Two pins of Port 3 can be used as external interrupts.

Alternative function of port pins are given below:

P1.0 T2 (Timer/Counter 2 external input). P1.0 provides alternative function only on the 8052 microcontroller.

P1.1 T2EX (Timer/Counter 2 capture/reload trigger). P1.1 provides alternative function only on the 8052 microcontroller.

P3.0 RXD (Serial input port).
P3.1 TXD (Serial output port).
P3.2 $\overline{\text{INT0}}$ (External interrupt).
P3.3 $\overline{\text{INT1}}$ (External interrupt).
P3.4 T0 (Timer/Counter 0 external input).
P3.5 T1 (Timer/Counter 1 external input).
P3.6 $\overline{\text{WR}}$ External Data Memory Write Strobe).
P3.7 $\overline{\text{RD}}$ (External Data Memory Read Strobe).

Each port of Intel 8051 consists of a latch (SFR P0–P3), an output driver and an input buffer. The output drivers of Ports 0 and Port 2 and the input buffer of Port 0, are employed while accessing external memory. Port 0 sends 8 LSBs of external memory address, time-multiplexed with the byte being written or read. Port 2 sends 8 MSBs of the external memory address when address is of 16 bits. Otherwise the Port 2 pins continue to emit the P2 SFR content. In other words Port 2 acts as I/O port.

10.1.4 The 8051 Interrupts

The 8051 microcontrollers have 4-level priority interrupts. The number of interrupt sources differs from version to version. It varies from 5 to 15. The important interrupt sources are: one from the serial port, two from timers, two from external interrupts INT0 and INT1. Each of the interrupts can individually be enabled/disabled by setting/clearing a bit in the special function register IE (Interrupt Enable). The IE register also contains a global disable bit, which disables all the interrupts. Each interrupt can also be programmed to one of the priority-level scheme by setting/clearing bits in the special function register IP (Interrupt Priority Register). A low-priority interrupt can be interrupted by a high-priority interrupt, but it can not be interrupted by another low-priority interrupt. A high-priority interrupt cannot be interrupted by a low-priority interrupt.

The 8051 microcontroller has five vectored interrupt sources namely, external interrupt 0 through the input pin INT0, external interrupt 1 through the input pin INT1, timer/counter 0 interrupt, timer/counter 1 interrupt and serial port interrupts. An interrupt for which microcontroller's hardware automatically transfers the program execution to a specific memory location is known as **vectored interrupt.** The specific memory location corresponding to each vector interrupt has a address as shown in Table 10.2. For each vectored interrupt there is an interrupt flag which is set when microcontroller's hardware detects the occurrence of an interrupt. Corresponding to above mentioned vectored interrupt sources, the 8051 has interrupt flags IE0, IE1, TF0, TF1, RI and TI respectively. Among these flags IE0, IE1 TF0 and TF1 are bits of the Timer/Counter Control Register TCON, as shown in Fig. 10.8. RI and TI are the bits of Serial Port Control Register SCON. RI is receive interrupt flag and TI transmit interrupt flag. These are connected through an OR gate and hence any one of RI or TI can send an interrupt signal to the microcontroller for the serial port. The other versions of 8051 series of microcontrollers have more number of interrupt sources for example 8052 contains 6 interrupt sources, 8XL51 FA/FB/FC has 7 interrupt sources and 8XC51 GB has 15. In 8052, Timer 2 interrupt is generated by the logical OR of TF2 and EXF2. The flags TF2 and EXF2 are not cleared by microcontroller's hardware when the service routine is vectored. The service routine determines whether TF2 or EXF2 has generated interrupt and the flag is cleared by software. Table 10.2 shows interrups, flags, vector address and priority of interrupts within the same level.

Table 10.2 Details of Interrupts

Interrupt	*Flag*	*Vector Address*	*Priority within level*
External Interrupt 0, $\overline{INT0}$	IE0	0003	Highest
Timer/Counter 0 Interrupt	TF0	000B	
External Interrupt1, $\overline{INT1}$	IE1	0013	
Timer/Counter1 Interrupt	TF1	001B	
Serial Port, RI or TI	RI or TI	0023	
Timer/Counter 2 Interrupt	TF2 or EXF2	002B	Lowest

The external interrupts $\overline{INT0}$ and $\overline{INT1}$ can be programmed to act as a falling edge triggered or low level triggered interrupt by setting and clearing the bits IT0 and IT1 respectively in the register TCON. If an external interrupt is made edge triggered, IE0 or IE1 interrupt flag is cleared by microcontroller's hardware when the service routine is vectored. If the external interrupt is level triggered, IE0 or IE1 is controlled by external requesting source of interrupt. The external interrupt source has to hold the interrupt request until the requested interrupt is actually generated. Then it has to deactivate the request before the interrupt service routine is completed. When the service routine is vectored, the interrupt flag is cleared by the external interrupt requesting source.

Enabling of 8051 Interrupts. The 8051 contains an interrupt enable register IE. It is a special function register (SFR). Each of interrupt sources can be individually enabled/disabled by setting/clearing a bit in the interrupt enable register IE. It also contains a global disable bit, EA which can disable all interrupts, if it is made equal to 0 *i.e.* EA = 0. It is a bit addressable register. To enable an interrupt, the bit EA and the bit corresponding to the desired interrupt is set. Fig. 10.4 shows the details of bits of register IE.

7	6	5	4	3	2	1	0	← Bit No.
EA	X	ET2	ES	ET1	EX1	ET0	EX0	← Symbol

Fig. 10.4 Interrupt Enable Register IE (Bit Addressable)

Bit No. 0, EX0. It is for External Interrupt 0 ($\overline{INT0}$). When EX0 = 1, $\overline{INT0}$ is enabled provided EA = 1. When EX0 = 0, $\overline{INT0}$ is disabled.

Bit No. 1, ET0. It is for Timer 0 Overflow interrupt. If ET0 = 1, the Timer 0 Overflow interrupt is enabled, provided EA = 1. If ET0 = 0, Timer 0 Overflow interrupt is disabled.

Bit No. 2, EX1. It is for External Interrupt 1 ($\overline{INT1}$). If EX1 = 1, $\overline{INT1}$ is enabled, provided EA = 1. If EX1 = 0, $\overline{INT1}$ is disabled.

Bit No. 3, ET1. It is for Timer 1 Overflow interrupt. If ET1 = 1, Timer 1 Overflow interrupt is enabled, provided EA = 1. If ET1 = 0, Timer 1 Overflow interrupt is disabled.

Bit No. 4, ES. It is for serial port interrupt. If ES = 1, the serial port interrupt is enabled, provided EA = 1. If ES = 0, the serial port interrupt is disabled.

Bit No. 5, ET2. It is for Timer 2 Overflow or capture interrupt. If ET2 = 1, the Timer 2 Overflow or capture interrupt is enabled. If ET2 = 0, the Timer 2 Overflow or capture interrupt is disabled.

Bit No. 6. Reserved.

Bit No. 7, EA. It is a global disable bit. If EA = 0, all interrupts are disabled. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

Interrupt Priority. By setting or clearing a bit in the special function register IP, the user can program each interrupt individually in either high-priority level or low-priority level. A low-priority interrupt can be interrupted by a high-priority interrupt, but it can not be interrupted by any other low-priority interrupt. A high-priority interrupt can not be interrupted by a low-priority interrupt. Table 10.2 shows priority level of various interrupts of Intel 8051 series of microcontrollers. If two interrupts of different priority level (i.e. one interrupt of high-priority level and another interrupt of low-priority level) occur simultaneously, the interrupt of the higher-priority level will be served first. If two interrupts of the same priority level occur simultaneously, an internal polling determines which interrupt is of higher-priority level on the basis of the priority level given in the Table 10.2.

Figure 10.5 shows the details of the bits of Interrupt Priority Register, IP.

7	6	5	4	3	2	1	0	← Bit No.
X	X	PT2	PS	PT1	PX1	PT0	PX0	← Symbol

Fig. 10.5 Interrupt Priority Register

Bit No. 0, PX0. It is for External Interrupt 0 ($\overline{INT0}$) priority level. PX0 = 1, higher-priority. PX0 = 0, lower-priority.

Bit No. 1, PT0. It is for Timer 0 Interrupt priority level. PT0 = 1, higher-priority. PT0 = 0, lower-priority.

Bit No. 2, PX1. It is for External Interrupt 1 ($\overline{INT1}$) priority level. PX1 = 1, higher-priority. PX1 = 0, lower-priority.

Bit No. 3, PT1. It is for Timer 1 Interrupt priority level. PT1 = 1, higher-priority. PT1 = 0, lower-priority.

Bit No. 4, PS. It is for Serial Port Interrupt priority level. PS = 1, higher-priority. PS = 0, lower-priority.

Bit No. 5, PT2. It is for Timer 2 Interrupt priority level. PT2 = 1, higher-priority level. PT2 = 0, lower-priority.

Bit No. 6 and Bit No. 7 are reserved.

10.1.5 Timer/Counters

The 8051 has two 16-bit on-chip programmable timer/counters namely, Timer 0 and Timer 1. Besides these two timer/counters the 8052 has one more timer/counter, Timer 2. All these three timer/counters can operate as timers or event counters. Real time applications can easily be implemented using these on-chip timer/counters, for example, pulse counting, baud rate generation, frequency measurement, pulse-width measurement/modulation and so on. The only difference between a timer and counter is the source of clock supply. When the timer/counter is used as a timer, the clock pulses are supplied from the oscillator after dividing the oscillator frequency by 12. When the timer/counter is used as a counter, the clock pulses are supplied from external source through the pin T0 (pin P3.4, i.e. the pin no. 4 of Port 3) to the Timer/Counter 0, and through the pin T1 (pin P3.5, *i.e.* pin no. 5 of Port 3) to the Timer/Counter 1. When a microcontroller operates as a timer, the timer/counter

register of the microcontroller is incremented every machine cycle. In other words the timer counts machine cycles and gives time delay etc. One machine cycle of the microcontroller consists of 12 oscillator periods (*i.e* 12 input clock periods of the microcontroller). The clock input of the microcontroller is at 12 MHz at which the time of one machine cycle will be 1 microsecond. When a microcontroller operates as a counter, its register is incremented in response to 1 to 0 transitions (the falling edge of the input signals) of the external input signals (pulses/clocks). The external signal is applied at T0 or T1 Pin of 8051 and at T2 pin of 8052. In the counting function, the hardware of the microcontroller takes 2 machine cycles (24 oscillator cycles) to detect 1 to 0 transition of the external pulse. TL0, TL1 and TL2 are low-order byte timer/counter registers and TH0, TH1 and TH2 are high-order byte of timer/counter registers.

Timer/Counter 0 and Timer/Counter 1

Both microcontroller 8051 and 8052 contain timer/counters, Timer/Counter 0 and Timer/Counter 1. TMOD register (Timer/Counter Mode Control Register) controls the modes of operation of Timer/Counter 0 and Timer/Counter 1. There are 4 modes of operation for these timer/counters. Mode 0, Mode 1 and Mode 2 are same for both timer/counters; Mode 3 is different. Figure 10.6 shows the various bits of TMOD Register.

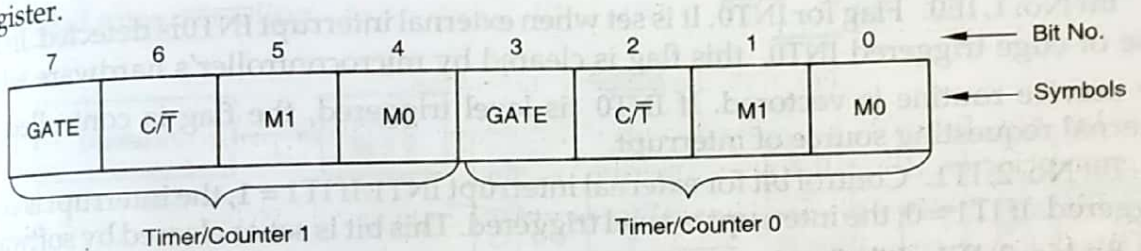

Fig. 10.6 TMOD Timer/Counter Mode Control Register

M1	M0		
0	0	Mode 0	13-bit timer/counter
0	1	Mode 1	16-bit timer/counter
1	0	Mode 2	8-bit timer/counter with automatic reload
1	1	Mode 3	Timer/Counter 0 operates as two separate timer/counters. Timer/Counter 1 does not work in Mode 3. However, it can work in Mode 0, 1 and 2 etc.

C/$\overline{T}$: Timer or Counter Selector. When C/$\overline{T}$ = 1, counter operation (input from pins T0 or T1). T0 and T1 are the pin Port 3.4 and P3.5 respectively used for alternative function. C/$\overline{T}$ = 0, timer operation (input from internal system clock).

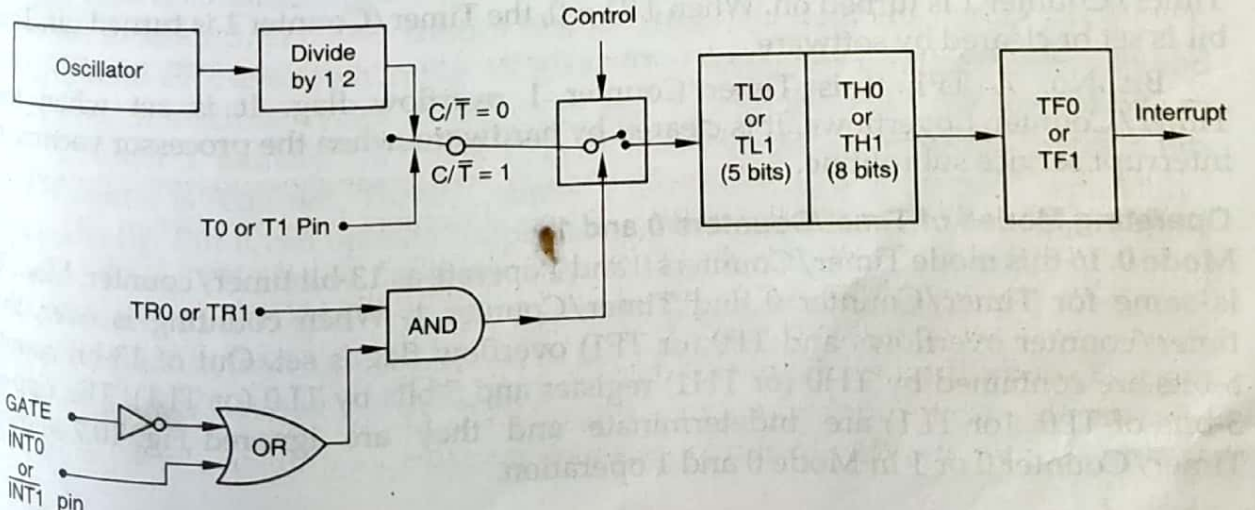

Fig. 10.7 Timber/Counter 0 and 1 in Mode 0 and 1 Operation

GATE = 1, Timer/Counter X is enabled, only if $\overline{INTX}$ pin is high and TRX is set. X = 0 or 1, see Fig. 10.7.

GATE = 0, Timer/Counter X is enabled whenever TRX is set. X = 0 or 1, see Fig. 10.7. Figure 10.7 shows Timer/Counter 0 and 1 in Mode 0 and 1 operation. Inverted GATE and $\overline{INT0}$ (or $\overline{INT1}$) are connected to an OR gate. If GATE is 0, the output of the OR gate is 1 and hence, it is immaterial whether $\overline{INT0}$ (or $\overline{INT1}$) is 0 or 1. If GATE = 1, $\overline{INT0}$ (or INT1) must be 1 to get 1 output from the OR gate. To enable Timer/Counter 0 or 1, the output of the OR gate as well as TR0 (or TR1) should be high. The SFR TCON controls the operation of Timer/Counter 0 and Timer/Counter 1. Its various bits are shown in Fig. 10.8.

7	6	5	4	3	2	1	0	← Bit No.
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	← Symbols

Fig. 10.8 TCON: Timer/Counter 0 and 1 Control Register

Bit No. 0, IT0. Control bit for external interrupt $\overline{INT0}$. If IT0 = 1, the interrupt is edge triggered. If IT0 = 0, the interrupt is level triggered. This bit is set or cleared by software.

Bit No. 1, IE0. Flag for $\overline{INT0}$. It is set when external interrupt $\overline{INT0}$ is detected. In the case of edge triggered $\overline{INT0}$, this flag is cleared by microcontroller's hardware when the service routine is vectored. If $\overline{INT0}$ is level triggered, the flag is controlled by external requesting source of interrupt.

Bit No. 2, IT1. Control bit for external interrupt $\overline{INT1}$. If IT1 = 1, the interrupt is edge triggered. If IT1 = 0, the interrupt is level triggered. This bit is set or cleared by software.

Bit No. 3, IE1. It is flag for $\overline{INT1}$. It is set when the external interrupt $\overline{INT1}$ is dected. In the case of edge triggered $\overline{INT1}$, this flag is cleared by microcontroller's hardware when the service routine is vectored. If $\overline{INT1}$ is level triggered, this flag is controlled by external requesting source of interrupt.

Bit No. 4, TR0. Run control for Timer/Counter 0. When TR0 = 1, the Timer/Counter 0 is turned on. When TR0 = 0, the Timer/Counter 0 is turned off. This bit is set or cleared by software.

Bit No. 5, TF0. It is Timer/Counter 0 overflow flag. It is set by hardware, when the Timer/ Counter 0 overflows. It is cleared by hardware, when the processor vectors to interrupt service subroutine.

Bit No. 6, TR1. Run control for Timer/Counter 1. When TR1 = 1, the Timer/Counter 1 is turned on. When TR1 = 0, the Timer/Counter 1 is turned off. This bit is set or cleared by software.

Bit No. 7, TF1. It is Timer/Counter 1 overflow flag. It is set when the Timer/Counter 1 overflows. It is cleared by hardware, when the processor vectors to interrupt service subroutine.

Operating Modes of Timer/Counters 0 and 1

Mode 0. In this mode Timer/Counters 0 and 1 operate as 13-bit timer/counter. Mode 0 is same for Timer/Counter 0 and Timer/Counter 1. When counting is over, the timer/counter overflows and TF0 (or TF1) overflow flag is set. Out of 13-bit count, 8-bits are contained by TH0 (or TH1) register and 5 bits by TL0 (or TL1). The upper 3-bits of TL0 (or TL1) are indeterminate and they are ignored Fig. 10.7 shows Timer/Counter 0 or 1 in Mode 0 and 1 operation.

Mode 1. This mode is same for Timer/Counter 0 as well as Timer/Counter 1. In Mode 1 Timer/Counter 0 and 1 operate as 16-bit timer/counter. When counting is over, the timer/counter overflows and sets TF0 or TF1 overflow flag. Figure 10.7 shows the block diagram for Model 1 (or Mode 0) for Timer/Counter 0 or 1. The register of the timer/counter is incremented every machine cycle. Suppose, initial value in the register is 0000. This value will be incremented every machine cycle and the value will finally reach FFFF. When this value is further incremented, the value will change from FFFF to 0000 and there will be overflow from the timer/counter. The initial value of the timer/counter's register can be programmed to any value by software. Suppose, a programmer keeps its initial value EF96. Now, counting will be done from EF97 to FFFF. Finally when FFFF will be incremented, there will be overflow.

Mode 2. This mode is same for Timer/Counter 0 as well as Timer/Counter 1. In this mode the timer/counter register TL0 or TL1 is made an 8-bit counter with automatic reload. The overflow from TL0 (or TL1) sets TF0 (or TF1) and it also reloads TL0 (or TL1) with the contents of TH0 (or TH1) which is preset by software. Figure 10.9 shows Timer/Counter 0 and 1 in Mode 2 operation. TH0 (or TH1) remains unchanged after reloading.

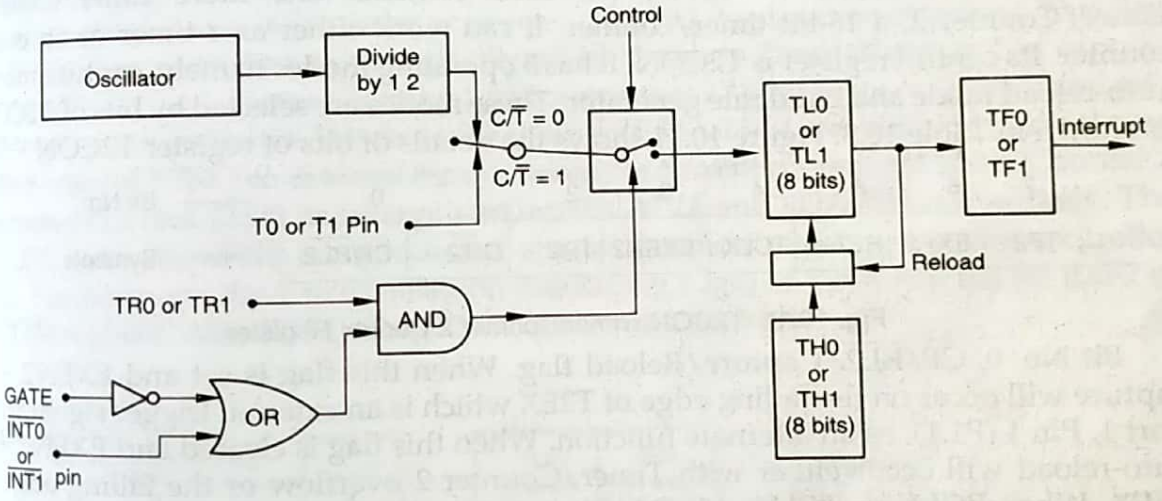

Fig. 10.9 Timber/Counter 0 and 1 in Mode 2 Operation: 8-Bit Auto Reload

Mode 3. In this mode Timer/Counter 0 makes TL0 and TH0 to operate as two separate Timer/counters. Figure 10.10 (a) and (b), show Timer/Counter 0 in Mode 3 operation. In Mode 3, TL0 uses control bits of Timer/Counter 0 such as GATE, C/T, TR0, INT0 and TF0, as shown in Fig. 10.10 (a). TH0 is locked into timer operation and simply counts machine cycles as shown in Fig. 10.10 (b). It uses the control bits of Timer/Counter 1 namely, TR1 and TF1. MODE 3 is used where one extra 8-bits timer/Counter is required. Timer/Counter 1 does not work in Mode 3 separately/independently. But it can operate simply as a timer since TH0 is locked into a timer function. Thus TH0 now controls the 'Timer 1' interrupt. However, while Timer/Counter 0 is in Mode 3, Timer/Counter 1 may be used in Mode 0, 1 or 2 which does not require an interrupt. It can also be used for baud rate generation for the serial port while Timer/Counter 0 is in Mode 3.

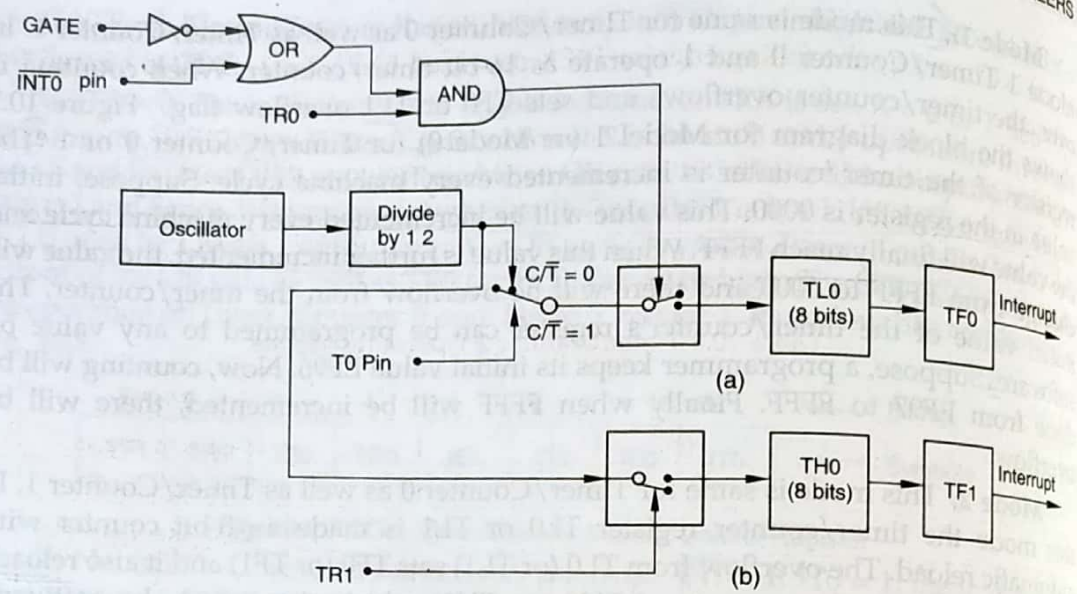

Fig. 10.10 Timber/Counter 0 in Mode 3 Operation

Timer/Counter 2

Besides Timer/Counter 0 and 1, the 8052 contains one more timer/counter, Timer/Counter 2, a 16-bit timer/counter. It can work either as a timer or an event counter. Its control register is T2CON. It has 3 operating modes namely, capture mode, auto-reload mode and baud rate generator. These modes are selected by bits of T2CON as shown in Table 10.3. Figure 10.11 shows the details of bits of register T2CON.

7	6	5	4	3	2	1	0	
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$	← Symbols
								← Bit No.

Fig. 10.11 T2CON: Timer/Counter 2 Control Register

Bit No. 0, CP/$\overline{RL2}$. Capture/Reload flag. When this flag is set and EXEN2 = 1, capture will occur on the falling edge of T2EX which is an external trigger signal from Port 1, Pin 1 (P1.1), as an alternate function. When this flag is cleared and EXEN2 = 1, auto-reload will occur either with Timer/Counter 2 overflow or the falling edge of T2EX. When RCLK or TCLK = 1, this bit is ignored and the timer/counter is auto-reloaded on its overflow.

Bit No. 1, C/$\overline{T2}$. When it is set, the timer/counter operates as external event counter (falling edge triggered). When it is cleared, the timer/counter operates as a timer on internal clock, OSC/12.

Bit No. 2, TR2. When it is set, the timer/counter is turned on. When it is cleared, the timer/counter is turned off.

Bit No. 3, EXEN2. Timer/Counter 2 external enable flag. When EXEN2 = 1, capture or reload will occur at the falling edge of T2EX, if the timer/counter is not being used to clock the serial port. When EXEN2 = 0, the timer/counter ignores T2EX.

Bit No. 4, TCLK. Transmit clock flag. When TCLK = 1, the serial port uses Timer/Counter 2 overflow pulses for its transmit clock in mode 1 and 3. When TCLK = 0, Timer/Counter 1 overflows are used for the transmit clock.

Bit No. 5, RCLK. Receive clock flag. When RCLK = 1, the serial port uses Timer/Counter 2 overflow pulses for its receive clock in mode 1 and 3. When RCLK = 0, Timer/Counter 1 overflow is used for the receive clock.

Bit No. 6, EXF2. Timer/Counter 2 external flag. It is set when either capture or reload is caused by the falling edge of T2EX provided EXEN2 = 1. EXF2 is cleared by software.

Bit No. 7, TF2. Timer/Counter 2 overflow flag. It is set by Timer/Counter 2 overflow. It is cleared by software. It is not set when TCLK or RCLK = 1.

Operating Modes of Timer/Counter 2. The operating modes of Timer/Counter 2 are: capture mode, auto-reload mode and baud generator mode. These modes are selected by setting T2CON bits as shown in Table 10.3.

Table 10.3 Timer/Counter 2 Operating Modes

RCLK + TCLK	CP/$\overline{RL2}$	TR2	*Modes*
0	0	1	16-bit Auto-reload
0	1	1	16-bit Capture
1	X	1	Baud Rate Generator
X	X	0	Off

X–may by either 0 or 1.

Capture Mode. In the capture mode there are two options which are selected by setting/clearing the bit EXEN2 of the T2CON register. If EXEN2 = 0, the Timer/Counter operates as a 16-bit timer or counter. The TF2 flag is set on overflow. This flag can be used to generate an interrupt. Figure 10.12 shows Timer/Counter 2 in capture mode. If EXEN 2 = 1, Timer/Counter 2 still operates as a 16-bit timer or counter and sets TF2 to generate interrupt. But there are some additional features also, for example the falling edge of T2EX (an external input) causes the current content of Timer/Counter 2 registers (TL2 and TH2), to be captured into RCAP2L and RCAP2H respectively. The RCAP2L and RCAP2H are additional special function registers in microcontroller 8052. Furthermore, the 1 to 0 transition (the falling edge) of T2EX sets the bit EXF2 of T2CON register. Also EXF2, like TF2 can generate interrupt.

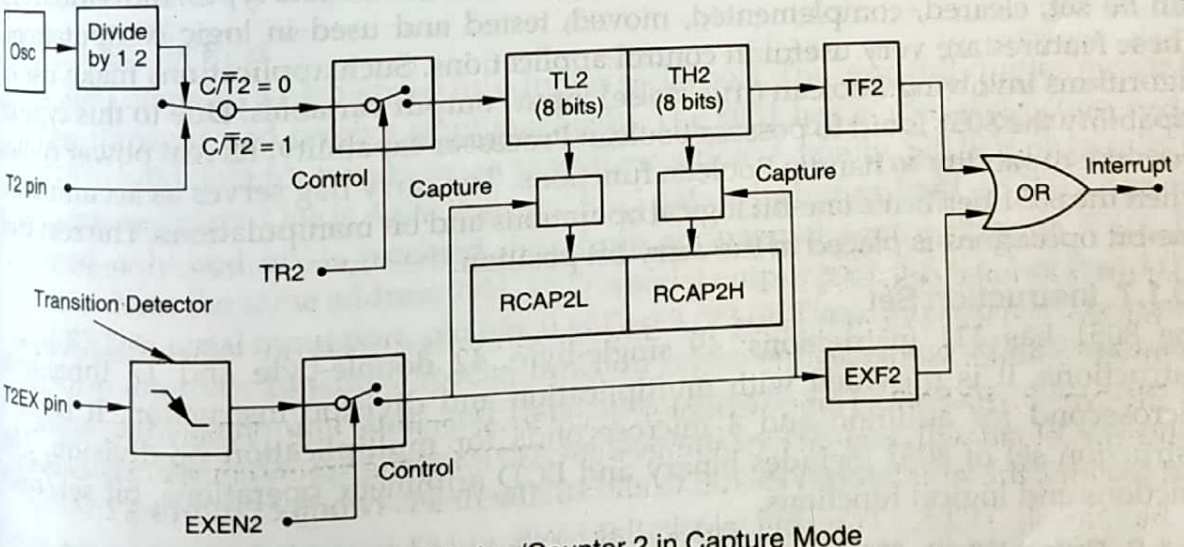

Fig. 10.12 Timber/Counter 2 in Capture Mode

Auto-Reload Mode. In the auto-reload there are two options which are selected by setting/clearing the bit EXEN2 of the T2CON register. If EXEN2 = 0, the Timer/Counter 2 operates as a 16-bit timer/counter and sets TF2 flag on overflow which can generate interrupt. Also, when TF2 is set, it causes reloading of TL2 and TH2 registers

with 16-bit values of registers RCAP2L and RCAP2H respectively. The values of RCAP2L and RCP2H are preset by software. If EXEN2 =1, the Timer/Counter 2 still performs all the operations described above, but there are some additional operations also. These are: (i) the falling edge of T2EX also triggers reloading of TL2 and TH2 registers with the 16-bit preset values of RCAP2L and RCAP2H and (ii) sets EXF2 which can also generate interrupt. Figure 10.13 shows Timer/Counter 2 in auto-reload mode.

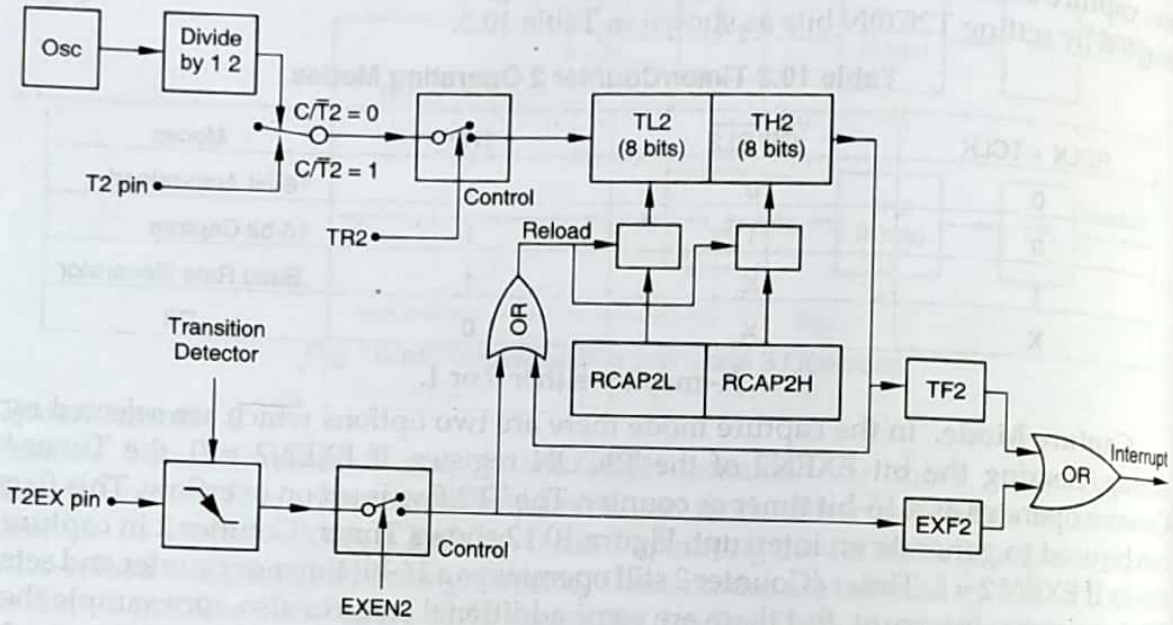

Fig. 10.13 Timber/Counter 2 in Auto-Reload

Baud Rate Generator Mode. This mode is selected by setting RCLK and/or TCLK. It will be discussed while describing serial port.

10.1.6 Boolean Processor

The ALU of 8051 can process one-bit data types besides 8-bit data types. Individual bits can be set, cleared, complemented, moved, tested and used in logic computations. These features are very useful in control applications. Such applications make use of algorithms involving Boolean (true/false) input/output variables. Due to this type of capability the 8051 is aid to possess Boolean Processor capability. Its real power comes from the capability to handle Boolean functions. The carry flag serves as accumulator when the 8051 performs one-bit logical operations and bit manipulations. The result of one-bit operations is placed in the carry-bit position.

10.1.7 Instruction Set

The 8051 has 111 instructions: 49 single-byte, 42 double-byte and 17 three-byte instructions. It is provided with multiplication and division instruction. It takes 1 microsecond for addition and 4 microseconds for multiplication or division. The instruction set of 8051 includes binary and BCD arithmetic operations, bit set/reset functions and logical functions.

10.1.8 Pulse-Width Modulation (PWM)

PWM signal can be generated by 8051. This feature of the 8051 can be used in a variety of control applications such as speed control of motors, duty cycle of heaters, etc. Several types of motors or switching power supply need pulse-width modulated signal for efficient operation. In such cases a true analog signal is not necessary. Where true

analog signal is required, a filter can be employed to give smooth analog signal output. The filter may be a simple RC network or active filter.

To fabricate a digital to analog converter on a microcontroller chip is a difficult task because it takes larger silicon area. Digital to analog conversion can be achieved employing PWM technique. Though the PWM output does not give smooth analog output (i.e. it does not act as a true D/A converter), it serves all practical purposes as required in industrial applications. It takes much less silicon area than a true D/A converter.

Figure 10.14 shows pulse-width modulated output. The wave-form is software programmable. The desired pulse-width of the PWM output signal can be obtained by loading the PWM control register by suitable 8-bit number.

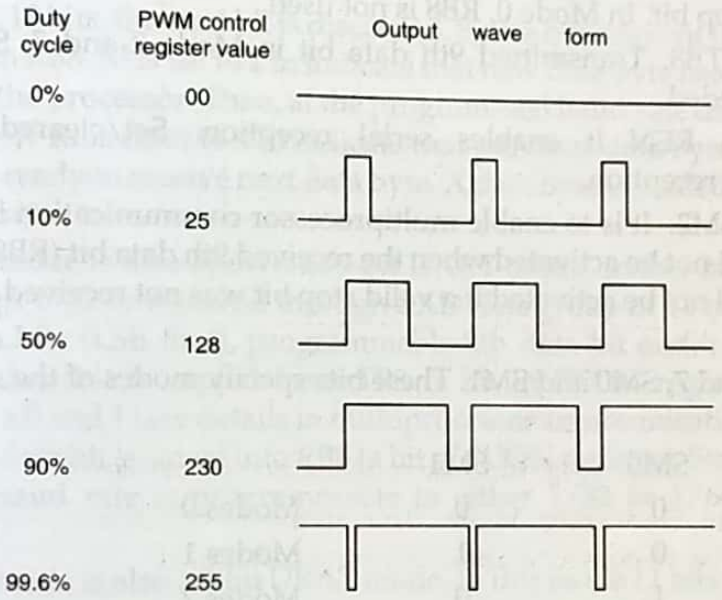

Fig. 10.14 Pulse-width Modulated Output

10.1.9 Serial Port

The 8051 contains a high-speed full-duplex serial port. The term full-duplex means that it can transmit and receive simultaneously. The 8051 has a special function register SBUF to hold data for transmission as well reception. Actually, SBUF is physically two separate registers. One is write only and hold data to be transmitted via TXD. The other is read only and holds received data from an external source through RXD. Both register have the same address 99H. TXD is serial output port, pin No. 1 of Port 3 (P3.1) and RXD is serial input port, pin No. 0 of Port 3 (P3.0). These pins of Port 3 are used to perform alternate function. Special function registers SCON and PCON are used to control data communication and data rates respectively. There are four operating modes for the serial port, which can be selected by 6th and 7th bits of the register SCON. Figure 10.15 shows the details of bits of SCON register. The four modes are:

Mode 0 – Shift Register Mode, Baud Rate – fosc/12.

Mode 1 – 8-bit UART Mode, Baud Rate – variable (set by Timer/Counter1).

Mode 2 – 9-bit UART Mode, Baud Rate – fosc/64 or fosc/32.

Mode 3 – 9-bit UART Mode, Baud Rate – variable set by (Timer/Counter1).

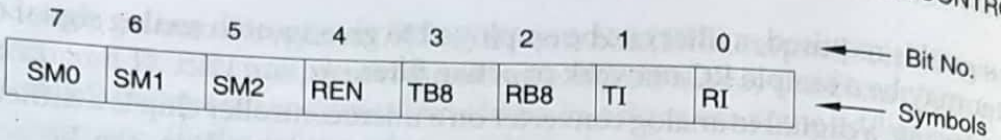

Fig. 10.15 SCON: Serial Port Control Register

Bit No. 0, RI. It is a receive interrupt flag. In the serial reception, it is set by hardware at the end of the 8th bit time in Mode 0 or half way through the stop bit time in other modes. See exception given in the description of SM2. It is cleared by software.

Bit No. 1, TI. It is a transmit interrupt flag. In the serial transmission, it is set by hardware at the end of the 8th bit time is mode 0 or at the beginning of the stop bit in other modes. It is cleared by software.

Bit No. 2, RB8. Received 9th data bit in Mode 2 and 3. In Mode 1, if SM2 = 0, RB8 is the received stop bit. In Mode 0, RB8 is not used.

Bit No. 3, TB8. Transmitted 9th data bit in Mode 2 and 3. Set and cleared by software as needed.

Bit No. 4 REN. It enables serial reception. Set/cleared by software to enable/disable reception.

Bit No. 5, SM2. It is to enable multiprocessor communication in Mode 2 and 3. If SM2 = 1, RI will not be activated when the received 9th data bit (RB8) is 0. In Mode 1, if SM2 = 1, RI will not be activated if a valid stop bit was not received. In Mode 0, SM2 is kept 0.

Bit No. 6 and 7, SM0 and SM1. These bits specify modes of the serial port as given below:

SM0	SM1	
0	0	Modes 0
0	1	Modes 1
1	0	Modes 2
1	1	Modes 3

SMOD is the 7th bit of the power control register PCON. It is the serial baud rate modify bit. When SMOD = 0, the serial port uses Timer/Counter 1 baud rate. When SMOD = 1, baud rate is doubled in Mode 1, 2 and 3, using Timer/Counter 1. SMOD is set/cleared by program.

When a data byte is written to SBUF, transmission of data byte begins. When the written data byte to SBUF has been transmitted, TI flag is set to indicate that SBUF has become empty. Now, next byte of data, if any, can be sent to SBUF for transmission. When a data byte is received in SBUF, RI is set as explained earlier. RI interrupts processor. The processor then executes program to read the serial byte from SBUF. When a serial byte has been read, RI is reset. RI reset to 0 indicates that the program has read the previous data byte and is read to receive the next byte of data.

Operating Modes of Serial Port. The four operating modes of the serial port are as described below:

Mode 0. This mode is also known as shift register mode. In this mode of operation RXD is used for both functions serial transmission as well as reception. TXD outputs shift clock. 8 data bits (LSB first) are transmitted/received. The baud rate is fixed. It is equal to 1/12 the oscillator frequency.

Mode 1. This mode is also known as 8-bit UART mode. In this mode 10 bits are transmitted through TXD or received through RXD. The group of the 10 bits consists of

a start bit (0), 8 data bits (LSB first) and stop bit (1). On receive, the stop bit goes into RB8 of the special function resister SCON. The baud rate is variable. The baud rate is generated using Timer/Counter 1 as given below:

$$\text{Baud rate} = \frac{\text{Timer/Counter 1 overflow rate}}{32}, \text{ if SMOD} = 0, \text{ Timber/Counter 1 in Mode 1}$$

$$= \frac{\text{Timer/Counter 1 overflow rate}}{16}, \text{ if SMOD} = 1, \text{ Timer/Counter 1 in Mode 1}$$

$$= \frac{2^{\text{SMOD}}}{32} \times \frac{\text{Oscillator frequency}}{12 \times [256 - (\text{TH1})]}, \text{ SMOD} = 0 \text{ or } 1, \text{ Timer/Counter 1 in Auto-reload Mode}$$

Out of original, 10 bits, the start bit is discarded, 8 data bits are sent to SBUF and the stop bit is saved into RB8. RI is set to 1 to indicate that new data byte has been received. Now RI interrupts the processor. Then, at the programmed baud rate data bits are read by the program. Now RI is reset to 0 to indicate that previous data byte has been read and the program is ready to receive next data byte. A data byte is loaded to SBUF when RI = 0 and stop bit = 1.

Mode 2. This mode is also known as 9-bit UART mode. In this mode 11 bits are transmitted through TXD or received through RXD. The group of 11 bits consists of a start bit (0), 8 data bits (LSB first), programmable 9th data bit and a stop bit (1). In transmission, 9th data bit is copied from TB8 (a bit in SCON register). It may be programmed to be a 0 and 1 (see details in multiprocessor communication). When data is received, the 9th data bit is saved into RB8 (a bit of SCON register). Start and stop bits are ignored. The baud rate is programmable to either 1/32 or 1/64 the oscillator frequency.

Mode 3. This mode is also a 9-bit URAT mode. In this mode 11 bits are transmitted through TXD or received through RXD. The group of 11 bits consists of a start bit (0), 8 data bits (LSB first), a programmable 9th bit and a stop bit (1). The Mode 3 is same as Mode 2 in all respects except the baud rate. The baud rate in Mode 3 is variable. The baud rate is determined exactly as in Mode 1 using Timer/Counter 1.

In the 8052, the baud rate can be generated by Timer 1 or Timer 2 or by both (one for transmit and the other for receive). Timer 2 is used to generate baud rate by setting TCLK and/or RCLK in T2CON register.

10.1.10 Multiprocessor Communication

When a system contains several 8051 microcontrollers and communication among these microcontrollers is required, the provision of multiprocessor communication is needed. Modes 2 and 3 of the serial port of the microcontroller 8051, have provision of multiprocessor communication. In these modes, 9th data bit which is programmable, is used. The 9th data bit goes into RB8 when data is received. When SM2 = 1, multiprocessor feature is enabled. With SM2 = 1, the serial port interrupt, RI is activated only if RB8 = 1. If RB8 = 0 and SM2 = 1, RI will not be activated. In a multiprocessor system, the device which transmits data is called master and the devices which receive data are called slaves. When a master wants to transmit a block of data, it first sends out an address to identify the target slave. The status of the 9th bit differentiates an address byte from a data byte. The 9th bit of an address byte is 1, whereas the 9th bit of the data byte is 0. SM2 of all the slaves is kept 1 when the system is

initialized for multiprocessor communication. When SM2 = 1, no slave will be interrupted by data byte because its 9th bit is 0 resulting in RB8 = 0. When RB8 has become 0, RI will not be activated. But with SM2 = 1, all slaves will be interrupted by an address byte as it has its 9th bit 1. Now each slave examines the received address and checks whether it has been addressed. The addressed slave clears its SM2 bit and then receives the data bytes which are coming. The slaves which were not addressed keep their SM2 set and continue what they were doing. They ignore data bytes.

10.1.11 Power Saving Modes

To reduce power consumption the 8051 has been provided with two power saving modes which can be invoked by user's software. The two power saving modes are : idle mode and power down mode. In the idle mode the CPU is turned off while the RAM and other on-chip peripherals continue their operation. In this mode current drawn is reduced to 15 percent of the current drawn when the device is fully active. The Idle mode can be terminated by any enabled interrupt or by a hardware reset. Backup power is used in the power down mode. When power failure is detected, processor is interrupted. After receiving such an interrupt, the processor enables backup power supply to transfer relevant data to on-chip RAM. In the power down mode all on-chip activities are suspended. The on-chip RAM continues to hold its data. In this mode the device draws less than 10 microamp. The on-chip RAM and Special Function Registers retain their values until the Power Down Mode is terminated.

10.1.12 Brief Description of Some 8051 Microcontroller Versions

The **80C51BH, 80C31BH** and **87C51** are 8-bit CHMOS control-oriented microcontrollers. The 80C31BH is ROMless. The 80C51BH contains 4 KB ROM and 87C51 contains 4 KB EPROM. All the three contain 128 bytes RAM. They contain two 16-bit timer/counters and 5 interrupt sources. Commercial/Express versions are available.

The **8XC52/54/58** are enhanced versions of 80C51/80C31/87C51. X = 0 and 7, see Table 10.1. The versions ending with 52 contain 8 KB program memory, ending with 54 contain 16 KB program memory and ending with 58 contain 32 KB program memory. The additional features include : 256 bytes RAM, more program memory, three 16-bit timers, programmable clock output, up/down timer/counter and hence, up/down counting capability, 6 interrupt sources, etc. The clock output and up/down counting capability make it suitable for motor control. They operate at 5 V d.c. The low-voltage versions 8XL52/54/58 which operate at 3 V are also available.

The **8XC51FA/FB/FC** are enhanced versions of 8XC52/54/58. Their added features are: pulse-width modulation, high-speed I/O, watch-dog timer capability, 7 interrupt sources, etc. These are more powerful microcontrollers for applications which require pulse-width modulation, high-speed I/O and up/down counting capabilities such as motor control. Their low-voltage versions 8XL51FA/FB/FC are also available.

The **8XC51RA/RB/RC** are also the enhanced versions of the 8XC52/54/58. The added features are: 512 bytes RAM, dedicated hardware watch-dog timer.

The **8XC51GB** is the enhanced version of 8XC51FA. Its added features are : on-chip A/D converter, dedicated watch-dog timer, more versatile serial channel which facilitates multiprocessor communications, memory protection features, etc. It has 15 interrupt sources: 7 external and 8 internal. The added features make it a very powerful control-oriented microcontroller.

The **8XC51SL** is a keyboard controller suitable for laptop and notebook PCs. They include an 8042-style UPI host interface with expanded memory, keyboard scan and

powerful management, 16 KB ROM/OTP, 256 bytes RAM; 4 channel, 8-bit A/D; 10 interrupt sources, three multifunction I/O ports, 2MHz-16MHz clock frequency, 5 LED drivers, etc. Their low-voltage versions are also available.

The **83C51KB** is high performance keyboard microcontroller. It is highly integrated keyboard microcontroller for standard and advanced desktop computers. It has 4 KB ROM and 128 bytes RAM. It has 8 dedicated key scan input pins, 16 dedicated key scan output pins, configurable timer (16-bit or two 8- bit), etc.

The **8XC152JA/JB/JC/JD** microcontrollers are universal communication controller. These microcontrollers contain superset of 80C51 architecture, multiprotocol serial communications, UART (universal asynchronous receiver/transmitter), dual on-chip DMA channel, HOLD/HLDA, two general purpose timer/counters, 5 or 7 I/O ports, 56 special function registers, 11 interrupt sources, etc. The serial communication is provided with SDLC (synchronous data link control)/HDLC (high-level data link control), CSMA/CD and user definable protocols. The JA and JC versions contain 8 KB ROM. The JB and JD versions are ROMless.

The **8XC152** series of microcontrollers are well suited for implementing integrated services digital network (ISDN) and emerging local area networks (LAN). Besides multiprotocol communication capability, the 80C152 offers traditional microcontroller features for peripheral I/O interface and control.

The **8XC151SA/SB** are high-performance microcontrollers of 80C51 family. They contain fast instruction pipeline, 16-bit internal code fetch; 8-bit, minimum 2 clock external code fetch in page mode; user selectable configurations : external wait states (0-3 wait states) and page mode; 256 bytes RAM, ROM/OTPROM 8 KB for SA version and 16KB for SB version, ROMless versions are also available, seven maskable interrupt sources with four programmable priority levels, hardware watchdog timer, programmable counter array with high-speed output, compare/capture operation, pulse-width modulation and watchdog timer; programmable serial I/O port with framing error detection and automatic address recognition, 16 MHz operation, etc. The new features like programmable wait states, page mode, etc. can be selected using the new user-programmable configuration.

The **8XC251SA/SB/SP/SQ** microcontrollers are the next generation of Intel MCS51 architecture that increases system performance by a factor 5 using the MCS251 microcontroller codes. By rewriting codes using the MCS251 architecture instructions, designers can increase performance upto 15 times. The MCS251 contains a 3-stage pipeline CPU architecture. The pipeline stages are: instruction fetch/decode, address generation/data fetch, and execution/writeback. It has 24-bit linear addressing which gives 16 MB unified memory space for code and data. It is register-based machine and can perform register-to-register operations. It is provided with 40 bytes general purpose register file, which is accessible as bytes, words and double words (sixteen 8-bit, sixteen 16-bit or ten 32-bit registers). The 40 bytes register file is designed with accumulator functionality and data indexing capability. It has an 8-bit ALU with high-speed 8-bit source buses SRC1(8) and SRC2(8), and 16-bit destination bus, DST(16). It has 16-bit internal code bus with 2 bytes/state code fetch. It has 8-bit, 2 clock external code fetch in page mode. Its other important features are: new instructions and addressing modes, 64 KB stack space, 128KB external code/data memory, supports 64 interrupt sources - 1 TRAP, 1 NMI and 62 maskable with four-level priority level; 1 KB on-chip data RAM for SA/SB versions and 512 bytes for SP/SQ versions, ROM/OTPROM 16 KB for SB/SQ versions and 8KB for SA/SP versions, 32 programmable I/O lines, 3 flexible timer/counters; programmable counter array with

high-speed output, compare/capture operation, PWM and watchdog timer; programmable serial I/O port with framing error detection and automatic address recognition; user selectable configurations with external wait states (0-3 wait states), address range and memory mapping, and page mode; hardware watchdog timer, 16 MHz clock, efficient C-language support, etc.

10.1.13 Memory Organization of Intel 8051

The memory address space for Intel 8051 is as given below:

(i) 2/4/8/16/32 KB on-chip program memory in the form of ROM/EPROM/ flash Memory. The capacity differs from version to version.

(ii) Up to 64 KB of program memory external to the microcontroller chip can be addressed.

(iii) 128, 256, 512, 1024 etc bytes of on-chip data memory *i.e.* (RAM) are available. The capacity differs from version to version.

(iv) 64 KB of data memory external to the microcontroller chip can be addressed.

The Data Memory address space consists of an internal and an external memory space. External Data Memory is accessed using MOVX instruction. The special function registers have their addresses in the upper part of internal data memory space. In some versions of the 8051 microcontrollers, the upper part of the internal data memory are used only for SFRs. For example, in 8051, 8751, 8031 etc, the upper internal data memory locations, from 80H to FF, are used for SFRs. The lower memory locations of internal data memory from 00 to 7F are used for internal data RAM. But in some versions the same addresses of the upper part of the internal data memory are used for both special function registers as well as upper bytes of RAM. Here, it is to be noted that physically there are two sets of memory locations having same memory addresses. The memory locations of two sets are addressed by different addressing modes. For example, in 8052, 8032 etc, the upper internal data memory addresses, from 80H to FF, are used for SFRs employing direct addressing. The same upper internal data memory addresses, from 80H to FF, are also used for the upper 128 bytes of internal RAM, employing register-indirect addressing. The lower 128 bytes of internal data memory can be addressed using direct addressing or register-indirect addressing. The SFRs can not be accessed using register-indirect addressing. Table 10.4 shows how to access data memory. Also see Fig. 10.16 and 10.17.

Table 10.4 Data Memory Accessing

Microcontrollers	*Internal Data Memory*		*External Data Memory*
	Address	*Addressing Modes*	
8051, 8751, 8031 etc.	00 to 7F for Internal RAM 80 to FF for SFRs	Direct or Register-Indirect Addressing Direct Addressing	External Data Memory is accessed using MOVX instructions
8052, 8032 etc.	00 to 7F for Internal RAM 80H to FF for SFRs 80 H to FF for Internal RAM	Direct or Register-Indirect Addressing Direct Addressing Register-indirect Addressing	External Data Memory is accessed using MOVX instructions

Also, in program memory some memory addresses are used for both internal program memory as well as external program memory. In 8051, 8751, 8351 etc when $\overline{EA}$ =1, internal program memory locations from 0000 to 0FFF and external program

memory from 1000H to FFFF are accessed. When $\overline{EA} = 0$, only external program memory locations from 0000 to FFFF are accessed. Similar is the case for 8052, 8032 etc, but ranges are different. Table 10.5 shows how to access program memory. Also, see Figs. 10.16 and 10.17.

Table 10.5 Program Memory Accessing

Microcontrollers	*Status of $\overline{EA}$ signal*	*Internal Program Memory (ROM etc.)*	*External Program Memory*
8051, 8751, 8351, etc.	$\overline{EA} = 1$	0000 to 0FFF	1000H to FFFF
DO	$\overline{EA} = 0$	—	0000 to FFFF
8052, 8032 etc.	$\overline{EA} = 1$	0000 to 2FFF	3000H to FFFF
DO	$\overline{EA} = 0$	—	0000 to FFFF

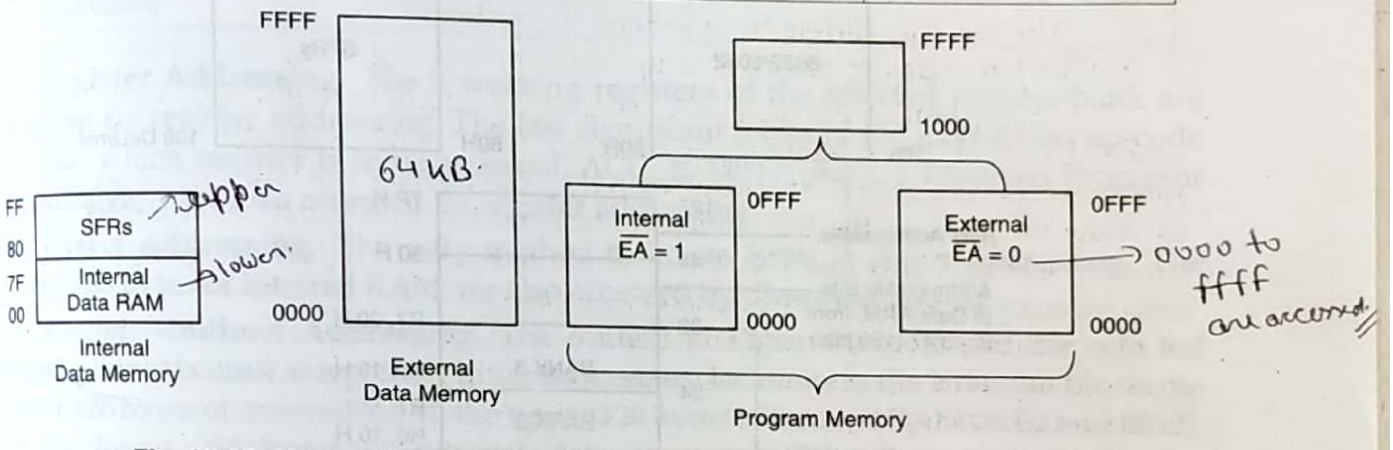

Fig. 10.16 Memory Organization of Intel 8051 (Except 8032 and 8052)

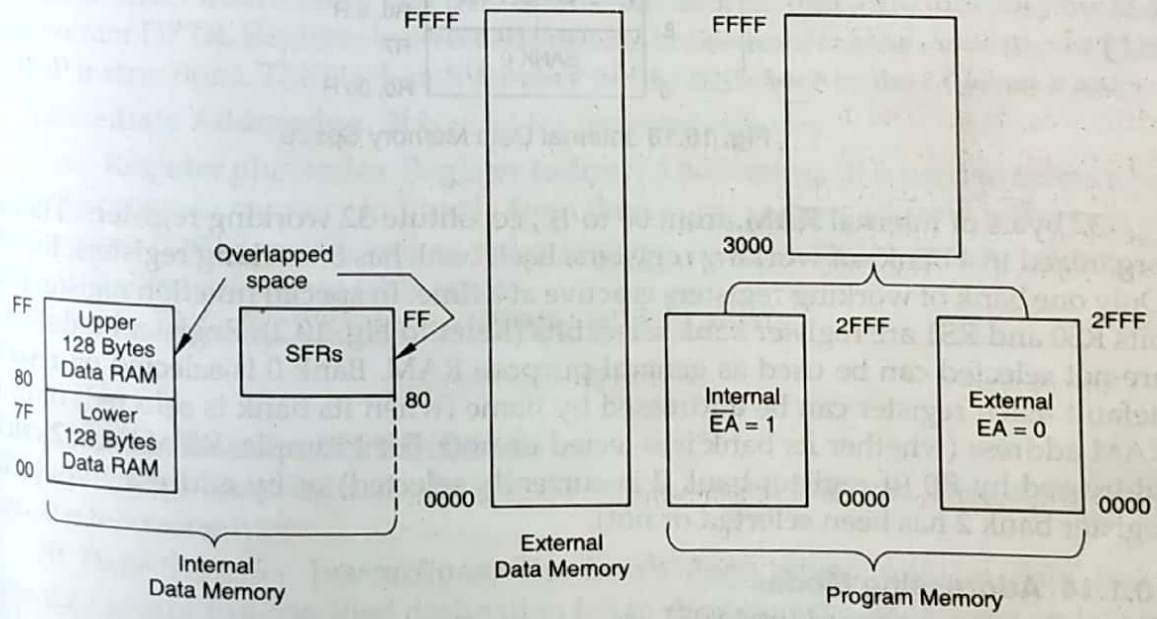

Fig. 10.17 Memory Organization of Intel 8032/8052

Internal Data Memory Space (Internal RAM)

The addresses for the lower 128 bytes are:

Address

00 – 07H BANK 0: – R0 – R7 (working registers).

08 – 0FH BANK 1: – R0 – R7 (working registers).
10H – 17H BANK 2: – R0 – R7 (working registers).
18H – 1FH BANK 3: – R0 – R7 (working registers).
20H – 2FH Bit addressable RAM locations (128 memory locations for 128 bits).
30H – 7FH General byte addressable.

From 80H to FFH are for SFRs for 8051, 8751, 8031, 8052, 8032 etc, 80H to FFH are also for upper 128 bytes of internal RAM for 8052 and 8032 microcontroller. The memory addresses from 80H to FFH are used by both SFRs as well as upper 128 bytes of internal RAM employing different addressing modes in 8052 and 8032 microcontrollers. See Fig. 10.18.

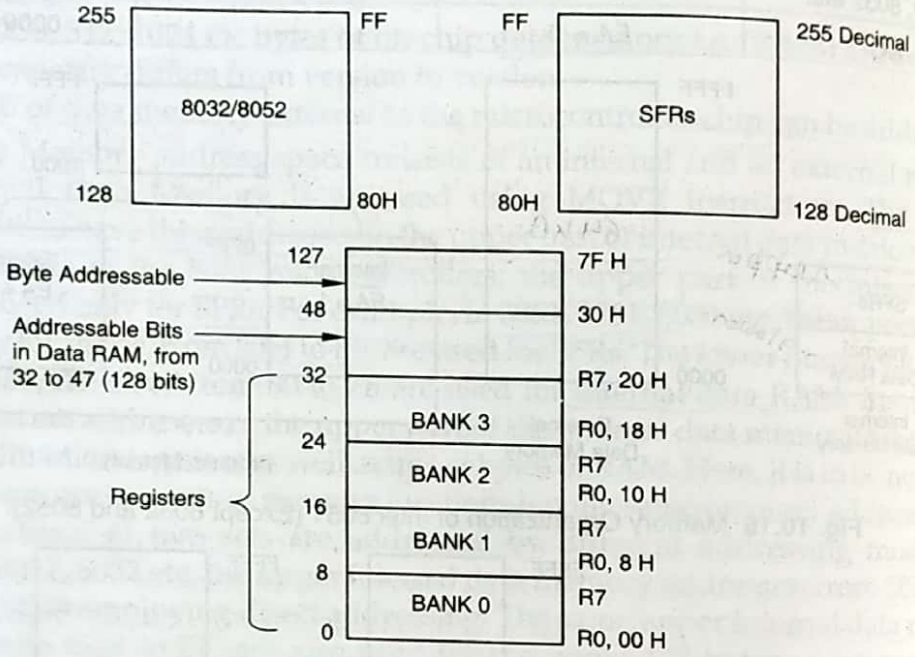

Fig. 10.18 Internal Data Memory Space

32 bytes of internal RAM, from 00 to 1F, constitute 32 working registers. They are organized in 4 banks of working registers. Each bank has 8 working registers, R0 – R7. Only one bank of working registers is active at a time. In special function register PSW, bits RS0 and RS1 are register bank select bits (Refer to Fig. 10.2). Register banks which are not selected can be used as general-purpose RAM. Bank 0 is selected on reset by default. Each register can be addressed by name (when its bank is selected) or by its RAM address (whether its bank is selected or not). For example, R0 of bank 2 can be addressed by R0 (if register bank 2 is currently selected) or by address 10 (whether register bank 2 has been selected or not).

10.1.14 Addressing Modes

The addressing modes of Intel 8051 are:

(i) Register Addressing
(ii) Direct Addressing
(iii) Register Indirect Addressing
(iv) Immediate Addressing
(v) Base-register plus Index-register Indirect Addressing

Table 10.6 gives the summary of memory of memory spaces which are accessed by each of the addressing modes.

Table 10.6 Addressing Modes of Intel 8051 and Associated Memory Spaces

Addressing Mode	*Memory Space*
Register Addressing	R0 – R7, ACC, B, CY(bit), DPTR
Direct Addressing	Lower 128 bytes of Internal RAM, Special Function Registers
Register-Indirect Addressing	Internal RAM (@R1, @R0, SP), External Data Memory (@R1, R0, @DPTR)
Immediate Addressing	Program Memory
Base-register plus Index-Register Indirect Addressing	Program Memory (@DPTR + A, @PC + A)

Register Addressing. The 8 working registers of the selected register-bank are accessed by register addressing. The last significant 3 bits of the instruction op-code indicate which register is to be accessed. ACC, B, DPTR and CY (Boolean Processor Accumulator) are also accessed by register addressing.

Direct Addressing. The only method to access SFRs is direct addressing. The lower 128 bytes of internal RAM are also accessed by direct addressing.

Register-Indirect Addressing. The content of either R0 or R1 (in the selected register bank) is used as a pointer to access memory locations in the 256 bytes block: the lower 128 bytes of internal RAM, the upper 128 bytes of internal RAM (8052 and 8032), or the lower 256 bytes of external data memory. SFRs are not accessable by register-indirect addressing. Full 64K of external data memory are accessed by 16-bit data pointer DPTR. Register-indirect addressing is also used for the execution of PUSH or POP instructions. The stack-pointer may reside anywhere in the internal RAM.

Immediate Addressing. It is used for program memory.

Base-Register plus Index-Register Indirect Addressing. It is used to access a byte from the program memory indirectly from the location whose address is the sum of a base register (DPTR or PC) and an index register. ACC is used as index register. This mode is facilitates look-up table access.

Program memory Address @ DPTR + A

or @ PC + A

10.1.15 Classification of Instructions

Instructions are classified according to the function they perform. They are classified into the following types:

(i) **Data Transfer Instructions.** The instructions which transfer data from a specified source to a specified destination fall in this category. The source of data may be a register, memory locations(s), stack, port etc. The destination may be a register, memory location, stack, port etc. The examples of data transfer instructions are: MOV, XCH (exchange), PUSH, POP, etc.

(ii) **Arithmetic Instructions.** These instructions perform addition, subtraction, multiplication, division, increment, decrement, comparison and decimal adjust. Examples are: ADD, ADDC, SUBB, INC, DEC, MUL, DIV, and DA.

(iii) Logical Instructions. The instructions of this group perform logical AND, OR, Exclusive-OR, clear, rotate, complement etc operations. Examples are: ANL (AND), ORL (OR), XRL (EX-OR), CLR (clear), CPL (complement), RL, RLC, RR, RRC etc.

(iv) Control Transfer Instructions. The instructions of this group include call, return and jump instructions. Examples are: ACALL, LCALL, RET, JZ, JNZ, JC, JNC, NOP etc.

10.1.16 Description of Intel 8051 Instructions

Some terms and symbols which are used while describing the instructions of Intel 8051 microcontrollers are:

Rn: – working registers R0, R1,....., and R7 of the currently selected Register Bank.

@Ri: – @ R0 or @ R1. 8-bit data RAM address specified indirectly by the register R0 or R1.

data: – 8-bit immediate data.

data: 16 – 16-bit immediate data.

direct: – 8-bit internal RAM address (00 – 7F) or SFR address (80 – FF).

bit: – bit address of the directly addressed bit in the internal data RAM or SFR.

addr 16: – 16-bit destination address used by LCALL and LJMP. A branch may be anywhere within the 64K-byte program memory.

addr 11: – 11-bit destination address used by ACALL and AJMP. The branch will be within the same 2K-byte page of the program as the first byte of the following instruction.

rel: – 8-bit signed (2's complement) offset byte used by SJMP and conditional jumps. Range is – 128 to + 127 bytes relative to the first byte of the following instruction.

1 machine cycle of the microcontroller = 12 input clock cycles (or clock periods) of microcontroller = 12 clock cycles (or clock periods) of the oscillator. While describing the instruction 1 machine cycle of the microcontroller has been written as 1 M/C cycle.

10.1.16.1 Data Transfer Instructions

MOV A, Rn. [A] ← [Rn]. Move data from the specified working register Rn to the accumulator. n = 0, 1, 2,....., or 7. Byte: 1. No flag affected. Register Addressing. M/C cycle: 1, Oscillator periods: 12. Encoding: 1110 1rrr, where rrr = address of register Rn.

MOV Rn, A. [Rn] ← [A]. Move data from the accumulator to the specified working register Rn, where n = 0, 1, 2,...., or 7. Byte: 1. No flag affected. Register Addressing. M/C cycle: 1, Oscillator periods: 12. Encoding: 1111 1rrr, where rrr = address of register Rn.

MOV A, direct. [A] ←[direct address]. The word 'direct' given in the mnemonic is the address which is directly given in the instruction. It is an 8-bit internal data RAM address (00 to 7FH) or an SFR address. This instruction moves data from the address given in the instruction to the accumulator. Bytes: 2. Direct Addressing. No flag affected. M/C cycle:1; oscillator periods: 12. Encoding: 1110 0101 direct address.

MOV direct, A. [direct address] ←[A]. Move the content of the accumulator to the address [8-bit internal RAM (00-7F) or SFR address] directly given in the instruction. Bytes: 2, M/C cycle: 1, oscillator periods: 12. Direct addressing. No flag affected. Encoding 1111 0101 direct address.

MOV A,# data. [A]←# data, where symbol # stands for immediate. This instruction moves immediate data to the accumulator. Bytes: 2. Immediate Addressing. M/C cycle: 1, oscillator periods: 12. Encoding: 0111 0100 immediate data.

MOV Rn, # data. [Rn]←# data. Move immediate data to the specified working register Rn, where n = 0, 1, 2,...., or 7, Bytes: 2, immediate addressing. No flag affected. M/C cycle: 1, oscillator periods: 12. Encoding: 0111 1rrr immediate data.

MOV direct, # data. [direct address]←# data. Move immediate data to the address [8-bit internal RAM (00 – 7F) or SFR address] given in the instruction itslef. Bytes: 3. Immediate addressing. Flags affected none. M/C cycles: 2, oscillator priods: 24. Encoding: 0111 0101 Direct address Immediate address.

MOV Rn, direct. [Rn] ←[direct address]. Move data from the address [8-bit internal RAM (00 – 7F) or SFR address] given in the instruction to the specified working register Rn. Bytes: 2. Direct Addressing. Flags affected none. M/C cycles: 2, oscillator periods: 24. Encoding: 1010 1rrr direct address, where rrr = the address of the working register Rn.

MOV direct, Rn. [direct address]←[Rn]. Move the content of the specified working register Rn to the address [8-bit internal RAM (00 – 7F) or SFR address] directly given in the instruction. Bytes. 2. Direct Addressing. No flag affected. M/C cycles: 2, oscillator periods: 24. Encoding: 1000 1rrr direct address.

MOV direct, direct. [direct address]←[direct address]. Move the content from the specified direct address to the specified direct address. A direct address is an 8-bit internal RAM address (00 – 7FH) or SFR address. Bytes: 3. Direct addressing. Flags affected none. M/C cycle: 2, oscillator periods: 24. Encoding: 1000 0101 direct source address direct destination address.

MOV A,@Ri. [A]←[[Ri]]. Move the content of the internal RAM memory location whose address is in Ri (where i = 0 or 1) to the accumulator. Byte: 1. Register-Indirect Addressing for source. Flags affected none. M/C cycle: 1, oscillator periods: 12. Encoding: 1110 011i.

MOV @Ri, A. [[Ri]]←[A]. Move the content of the accumulator to the internal data RAM memory location whose address is in Ri, where i = 0 or 1. Byte: 1. Register Indirect Addressing for destination. Flags affected none. M/C cycle: 1, oscillator periods: 12. Encoding 1111 011i.

MOV @Ri, # data. [[Ri]]← # data. Move immediate data to the internal RAM memory location whose address is in Ri, where i = 0 or 1. Bytes: 2. Register Indirect Addressing. Flags affected none. M/C cycle: 1, oscillator periods: 12. Encoding: 0111 011i immediate data.

MOV @Ri, direct. [[Ri]]←[direct address]. Move data from the direct address [8-bit internal RAM (00–7F) address or SFR address] to the internal RAM memory location whose address is in Ri, where i = 0 or 1. Bytes: 2. Register Indirect Addressing for destination. Flags affected non. M/C cycles: 2, oscillator periods: 24. Encoding: 1010 011i direct address.

MOV direct, @Ri. [direct address]←[[Ri]]. Move the content of the internal RAM memory location whose address is in Ri (i = 0 or 1) to the direct address [8-bit internal (00–7F) RAM or SFR address] given in the instruction. Bytes: 2. Register Indirect Addressing for source. Flags affected none. M/C cycles: 2, oscillator period: 24. Encoding: 1000 011i direct address.

MOV DPTR, # data 16. [DPTR]←# data 16. Move 16-bit immediate data to the 16-bit data pointer DPTR. Bytes: 3. Immediate Addressing. Flags affected: none. M/C cycles: 2, oscillator periods: 24. Encoding: 1001 0000 8 MSBs of immediate data 8 LSBs of immediate data. 8 MSBs of the immediate data will be moved to DPH and 8 LSBs of the immediate data will be moved to DPL.

MOV C, bit. [C]←[bit address]. It is an instruction to move a bit whose address is directly given in the instruction. 'bit' given in the instruction is an 8-bit address of internal RAM or SFR. 'C' is carry flag. It is the 7th bit (the MSB) of the PSW (Program Status Word). It is written as PSW. 7. This instruction moves a bit of data from the bit address specified in the instruction to the carry. Bytes: 2. Direct Bit Addressing. Flags affected: carry flag. M/C cycle: 1, oscillator periods: 12. Encoding: 1010 0010 bit address.

MOV bit, C. [bit address]←[C]. This instruction moves carry status to the bit address (internal RAM or SFR) given in the instruction. Bytes: 2. Direct Bit Addressing. Flags affected: Carry flag. M/C cycle: 2, oscillator periods: 24. Encoding: 1001 0010 bit address.

MOVC A, @A + DPTR. [A]←[[A]+[DPTR]]. MOVC instruction loads accumulator with a code byte or constant from the program memory (internal or external). This instruction moves a byte of data from the program memory (whose address is the sum of the original 8-bit unsigned content of the accumulator and 16-bit content of the data pointer DPTR) to the accumulator. The word 'original' appears here as this instruction is used for look-up table accessing. Byte: 1. Base Register plus Index Register Indirect Addressing. In this case DPTR is the base register and the accumulator is an index register. M/C cycles: 2, oscillator periods: 24. Encoding: 1001 0011.

MOVC A, @A + PC. [A]←[[A] + [PC]]. MOVC instruction loads accumulator with a code byte or constant from the program memory (internal or external). This instruction moves a byte of data from the program memory, (whose address is the sum of the original 8-bit unsigned content of the accumulator and 16-bit content of the program counter PC) to the accumulator. The word 'original' appears here as this instruction is used for look-up table accessing. Byte: 1: Base Register plus Index Register Indirect Addressing. In this case PC is the base register and the accumulator is an index register. M/C cycles: 2, oscillator periods: 24. Encoding: 1000 0011.

MOVX A, @Ri. [A]←[[Ri]]. The instruction MOVX moves data from external data memory to the accumulator or vice versa. 'X' is for external. The instruction MOVX A, @ Ri moves data from the external memory, whose address is specified by Ri, i = 0 or 1, to the accumulator. Since Ri can hold an 8-bit address, using this instruction only upto 2K bytes of external data memory can be addressed. So, it is suitable for small external RAM. Byte: 1. Register-Indirect Addressing. Flags affected: none. M/C cycles: 2, oscillator periods: 24. Encoding: 1110 001i.

MOVX @Ri, A. [[Ri]]← [A]. This instruction moves data from the accumulator to the external data memory whose address is specified by Ri, i = 0 or 1. Since Ri can hold an 8-bit address, using this instruction only upto 2K external data memory can be addressed. So, it is suitable small external RAM. Byte: 1. Register-Indirect Addressing. Flags affected: none. M/C cycels: 2, oscillator periods: 24. Encoding: 1111 001i.

MOVX A, @DPTR. [A]← [[DPTR]]. This instruction moves data from the external data memory, whose address is specified by the 16-bit data pointer DPTR, to the accumulator. As the DPTR can hold 16-bit memory address, upto 64K bytes of external data memory can be addressed using this instruction. So it is suitable for a large external data RAM. Byte: 1. Register-Indirect Addressing. Flags affected: none. M/C cycle: 2, oscillator periods: 24. Encoding: 1110 0000.

MOVX @DPTR, A. [[DPTR]]← [A]. This instruction moves data from the accumulator to the external data memory whose address is specified by the 16-bit data pointer DPTR. Since the DPTR holds 16-bit memory address, using this instruction upto 64K bytes of external data memory can be addressed. So, it is suitable for a large external data RAM. Bytes: 1, Register-Indirect Addressing. Flags affected: none. M/C cycles: 2, oscillator periods: 24. Encoding: 1111 0000.

PUSH direct. [SP]← [SP] + 1. [[SP]]←[direct address]. When data is to the placed on the stack, the content of the stack pointer is incremented by one before storing data on the stack so that the stack grows up as data is stored. The content of the specified direct address is then copied into the internal RAM location addressed by the stack pointer. Bytes: 2, M/C cycles: 2, oscillator periods: 24. Encoding: 1100 0000 direct address.

POP direct. [direct address]← [[SP]]. [SP]□[SP] – 1. The content of the internal RAM location addressed by the stack pointer is read, and then the stack pointer is decremented by one. The value read is stored in the destination address which is directly given in the instruction. Bytes: 2, M/C cycles: 2, oscillator periods: 24. Encoding 1101 0000 direct address.

XCH A, Rn. [A]$\leftrightarrows$[Rn]. Byte:1, M/C cycle: 1 oscillator periods: 12, Addressing: register. This instruction exchanges accumulator with the specified working register. Encoding: 1100 1rrr, where rrr = address of specified register, Rn.

XCH A, direct. [A]$\leftrightarrows$[direct address]. Bytes: 2, M/C cycle: 1, direct addressing. Oscillator periods: 12. This instruction exchanges accumulator with the internal data RAM location which is directly specified in the instruction. Encoding: 1100 0101 direct address.

XCH A, @Ri. [A]$\leftrightarrows$[[Ri]]. Byte: 1, M/C cycle: 1., register-indirect addressing. Oscillator periods: 12. This instruction exchanges accumulator with the data RAM location which is specified indirectly in the instruction. Encoding: 1100 011i.

XCHD A, @Ri. [$A_{3\text{-}0}$]$\leftrightarrows$[[$R_{i\,3\text{-}0}$]]. Byte: 1, M/C cycle: 1., register-indirect addressing. Oscillator periods: 12. This instruction exchange low-order nibble of the accumulator with the low-order nibble of the data RAM location which is indirectly addressed in the instruction. Encoding: 1101 11i.

10.1.16.2 Arithmetic instructions

Table 10.7 Summary of Arithmetic Instructions

Mnemonic	*Description*
ADD A Rn	Adds the content of the specified working register to the content of the accumulator and keeps the result in accumulator. Flags affected: all, Byte: 1, M/C cycle: 1, oscillator periods: 12. Register addressing. Encoding: 0010 1rrr, where rrr = address of Rn.
ADD A, direct	Adds the content of the specified address to the content of the accumulator, and places the result in accumulator. Flags affected: all, Bytes: 2, M/C cycle: 1, oscillator periods: 12, direct addressing. Encoding: 0010 0101 direct address.

ADD A, @ Ri	Adds the content of the data RAM location (addressed by Ri) to the content of the accumulator, and places the result in the accumulator. Flags affected: all, Byte: 1, M/C cycle: 1, oscillator periods: 12, register-indirect addressing. Encoding: 0010 011i.
ADD A, # data	Adds immediate data to the content of the accumulator, and places the result in the accumulator. Flags affected: all, immediate addressing: Bytes: 2, M/C cycle: 1, oscillator periods: 12. Encoding: 0010 0100 immediate data.
ADDC A, Rn	[A] ←[A] + [Rn] + [C], C is carry flag. Adds the content of the specified working register to the content of the accumulator with carry. Places the result in the accumulator. Flags affected: all, Bytes: 1, M/C cycle: 1, oscillator periods: 12. Register addressing. Encoding: 0011 1rrr, where rrr = address of Rn.
ADDC A, direct	Adds the content of the specified address to the content of the accumulator with carry. Places the result in the accumulator. Flags affected: all, Bytes: 2, M/C cycle: 1, oscillator periods: 12. Direct addressing. Encoding: 0011 0101 direct address.
ADDC A, @Ri	Adds the content of the data RAM location (specified by Ri) to the content of accumulator with carry. Places the result in the accumulator. Flags affected: all, Byte: 1, M/C cycle: 1, oscillator periods: 12, register-indirect addressing. Encoding 0011 011i.
ADDC A, # data	Adds immediate data to the content of the accumulator with carry. Places the result in the accumulator. Flags affected: all, Bytes: 2, M/C cycle: 1, oscillator periods: 12, immediate addressing. Encoding: 0011 0100 immediate data.
SUBB A, Rn	[A] – [Rn] – [C] Subtracts the content of the specified working register from the content of the accumulator with borrow. Result in accumulator. Flags affected: all, Byte: 1, M/C cycle: 1, oscillator periods: 12, register addressing. Encoding: 1001 1rrr, where rrr = address of Rn.
SUBB A, direct	Subtracts the content of the specified address from the content of the accumulator with borrow. Result in accumulator. Flags affected: all, Byte: 1, M/C cycle: 1, oscillator periods: 12, direct addressing. Encoding: 1001 0101 direct address.
SUBB A, @Ri	Subtracts the content of data RAM location (addressed by Ri) from the content of the accumulator with borrow. Result in accumulator; Flags affected: all, Byte: 1, M/C cycle: 1, oscillator periods: 12, register-indirect addressing. Encoding 1001 011i.
SUBBA, # data	Subtracts immediate data from the content of the accumulator with borrow. Result in accumulator; Flags affected: all, Bytes: 2, M/C cycle: 1, oscillator periods: 12, immediate addressing. Encoding: 1001 0100 immediate data.
INC A	Increments the content of the accumulator by 1 and places the result in the accumulator. No flag is affected. Byte: 1, M/C cycle: 1, oscillator periods: 12, register addressing. Encoding: 0000 0100

Note: 1. In case of ADD or ADDC instruction, OV flag is set if there is carry out of 7th bit but not out of 6th bit, or there is carry out of 6th bit but not out of 7th bit. This feature is utilized when signed arithmetic operation is performed.

2. In case of SUBB instruction, OV flag is set if a borrow is needed into 6th bit, but not into 7th bit, or into 7th bit but not into 6th bit. This feature is utilized when signed arithmetic operation is performed.

Mnemonic	Description
INC Rn	Increments the content of the specified working register, Rn, by 1. Places the result in Rn. No flag is affected. Byte: 1, M/C cycle: 1, oscillator periods: 12, register addressing. Encoding: 0000 1rrr, where rrr = address of Rn.
INC direct	Increments the content of the address given in the instruction, by 1. Places the result in the specified address. Bytes: 2, M/C cycle: 1, oscillator periods: 12. Direct addressing, flags affected: none. Encoding: 0000 0101 direct address.
INC @Ri	Increments the content of data RAM location (addressed by Ri) by 1. Places the result in specified RAM location. Byte: 1, M/C cycle: 1, oscillator periods: 12, register-indirect addressing, flags affected: none. Encoding: 0000 011i.
INC DPTR	Increments the content of the 16-bit data pointer DPTR by 1. No flag is affected. Result is placed in DPTR. Byte: 1, M/C cycles: 2, oscillator periods: 24, register addressing. Encoding: 1010 0011.
DEC A	Decrements the content of the accumulator by 1. Places the result in the accumulator. Flags affected: none, Byte: 1, M/C cycle: 1, oscillator periods: 12, register addressing. Encoding: 0001 0100.
DEC Rn	Decrements the content of the specified working register, Rn, by 1. Places the result in Rn. No flag is affected. Byte: 1, M/C cycle: 1, oscillator periods: 12, register addressing. Encoding: 0001 1rrr, where rrr = address of Rn.
DEC direct	Decrements the content of the address given in the instruction, by 1. Places the result in the specified address. No flag is affected. Bytes: 2, M/C cycle: 1, oscillator periods: 12, direct addressing. Encoding: 0001 0101 direct address.
DEC @Ri	Decrements the content of the data RAM location (addressed by Ri) by 1. Places the result in the specified RAM location. No flag is affected. Byte: 1, M/C cycle: 1, oscillator periods: 12, register-indirect addressing. Encoding: 0001 011i.
MUL AB	This instruction is used to multiply unsigned 8-bit content (integer) of the accumulator by the unsigned 8-bit content (integer) of the register B. The low-order byte of the 16-bit product is placed in the accumulator and the high-order byte in register B. When the product is greater than FFH, the overflow flag OV is set to 1, otherwise it is cleared. The carry flag is always cleared i.e. it is 0. Byte: 1, M/C cycles: 4, oscillator periods: 48, register addressing. Encoding: 1010 0100.
DIV AB	This instruction divides the unsigned 8-bit content (integer) of the accumulator by unsigned 8-bit content (integer) of the register B. The quotient is placed in the accumulator and remainder in register B. The carry and overflow flag OV are cleared. Byte: 1, M/C cycles: 4, oscillator periods: 48, register addressing. Encoding: 1000 0100.
DA A	It is 'decimal adjust accumulator'. After the execution of ADD or ADDC instruction, the result is in hexadecimal and placed in the accumulator. This instruction operates on the result and gives the final result in decimal system. Byte: 1, M/C cycle: 1, oscillator periods: 12, Encoding: 1101 0100.

10.1.16.3 Logical Instructions

Table 10.8 Summary of Logical Instructions

Mnemonic	Description
ANL A, Rn	Bitwise logical-AND operation between the content of the specified working register and the content of the accumulator. Result in the accumulator No flag is affected. Byte: 1, M/C cycle: 1, oscillator periods: 12, register addressing. Encoding: 0101 1rrr, where rrr = address of Rn.

ANL A, direct	Bitwise logical-AND operation between the content of the specified address and the content of the accumulator. Result in the accumulator. No flag is affected. Bytes: 2, M/C cycle: 1, oscillator periods: 12, direct addressing. Encoding 0101 0101 direct address.
ANL A, @Ri	Bitwise logical-AND operation between the content of the data RAM location (addressed by Ri) and the content of the accumulator. Result in the accumulator. No flag is affected. Byte: 1, M/C cycle: 1, oscillator periods: 12, register-indirect addressing. Encoding: 0101 011i.
ANL A, # data	Bitwise logical-AND operation between immediate data and the content of the accumulator. Result in the accumulator. No flag is affected. Bytes: 2, M/C cycle: 1, oscillator periods: 12, immediate addressing. Encoding: 0101 0100 immediate data.
ANL direct, A	Bitwise logical-AND operation between the content of the specified address and the content of the accumulator. Result in the specified address. No flag is affected. Bytes: 2, M/C cycle: 1, oscillator periods: 12, direct addressing. Encoding 0101 0010 direct address.
ANL direct, # data	Bitwise logical-AND operation between the immediate data and the content of the specified address. Result in specified address; No flag affected; Bytes: 3, M/C cycles: 2, oscillator periods: 24, immediate addressing. Encoding: 0101 0011 direct address immediate address.
ANL C, bit	Logical-AND operation between the content of the given bit address and the carry bit. Result in carry bit; only carry flag affected; bytes: 2, M/C cycle: 2, oscillator periods: 24, direct bit addressing. Encoding: 1000 0010 bit address.
ANL C, / bit	Logical-AND operation between the complement of the content of the given bit address and the carry bit. Result in carry bit; only carry flag affected, Bytes: 2, M/C cycles: 2, oscillator periods: 24, direct bit addressing. Encoding: 1011 0000 bit address.
ORL A, Rn	Bitwise logical-OR operation between the content of the specified working register and the content of the accumulator. Result in accumulator; No flag affected; Byte: 1, M/C cycle: 1, oscillator periods: 12, register addressing. Encoding: 0100 1rrr, where rrr = address of Rn.
ORL A, direct	Bitwise logical-OR operation between the content of the specified address and the accumulator. Result in a accumulator; No flag affected; Bytes: 2, M/C cycle: 1, oscillator periods: 12, direct addressing. Encoding: 0100 0101 direct address.
ORL A, @Ri	Bitwise logical-OR operation between the content of data RAM location (addressed by Ri) and the content of the accumulator. Result in the accumulator; No flag affected; Byte: 1, M/C cycle: 1, oscillator periods: 12, register-indirect addressing. Encoding: 0101 011i.
ORL A, # data	Bitwise logical-OR operation between immediate data and the content of the accumulator. Result in accumualtor; No flag affected; Bytes: 2, M/C cycle: 1, oscillator periods: 12, immediate addressing. Encoding: 0100 0100 immediate data.
ORL direct, A	Bitwise logical-OR operation between the content of the specified address and the content of the accumulator. Result in specified address, No flag affected; Bytes: 2, M/C cycle: 1, oscillator periods: 12, direct addressing. Encoding: 0100 0010 direct address.
ORL direct, # data	Bitwise logical-OR operation between immediate data and the content of the specified address. Result in specified address; No flag affected; Bytes: 3, M/C cycles: 2, oscillator periods: 24, immediate addressing. Encoding: 0100 0011 direct address immediate address.

ORL C, bit	Logical-OR operation between the content of the given bit address and the carry bit. Result in carry bit; only carry flag affected; Bytes: 2, M/C cycles: 2, oscillator periods: 24, direct bit addressing. Encoding: 0111 0010 bit address.
ORL C, / bit	Logical-OR operation between the complement of the content of the given bit address and the carry bit. Result in carry bit, only carry flag affected, Bytes: 2, M/C cycles: 2, oscillator periods: 24, direct bit addressing. Encoding: 1010 0000 bit address.
XRL A, Rn	Logical Exclusive-OR operation between the content of the specified working register and the content of the accumulator. Result in accumulator; No flag affected, Byte: 1, M/C cycle: 1, oscillator periods: 12 register addressing. Encoding: 0110 1rrr, where rrr = address of Rn.
XRL A, direct	Logical Exclusive-OR operation between the content of the specified address and the content of the accumulator. Result in accumulator, no flag affected, Bytes: 2, M/C cycle: 1, oscillator periods: 12, direct addressing. Encoding: 0110 0101 direct address.
XRL A, @Ri	Logical Exclusive-OR operation between the content of data RAM location (addressed by Ri) and the content of the accumulator. Result in accumulator, no flag affected; Byte: 1, M/C cycle: 1, oscillator periods: 12, register-indirect addressing. Encoding: 0110 011i.
XRL A, # data	Logical Exclusive-OR operation between immediate data and the content of the accumulator. Result in accumulator, no flag affected, Bytes: 2, M/C cycle: 1, oscillator periods: 12, immediate addressing. Encoding: 0110 0100 immediate data.
XRL direct, A	Logical Exclusive-OR operation between the content of the specified address and the content of the accumulator. Result in specified address, no flag affected, Bytes: 2, M/C cycle: 1, oscillator periods: 12, direct addressing. Encoding: 0110 0010 direct address.
XRL direct, # data	Logical Exclusive-OR operation between the content of the specified address and immediate data. Result in the specified address, no flag affected, Bytes: 3, M/C cycles: 2, oscillator periods: 24, immediate addressing. Encoding: 0110 0011 direct address immediate address.
CLR A	Clears (resets to zero) accumulator. No flag affected. Byte: 1, M/C cycle: 1, oscillator periods: 12. Encoding: 1110 0100.
CLR C	Clears (resets to zero) carry flag. Only carry flag affected. Byte: 1, M/C cycle: 1, oscillator periods: 12, Encoding: 1100 0011.
CLR bit	Clears (reset to zero) the content of the specified bit address. No flag affected; Bytes: 2, M/C cycle: 1, oscillator periods: 12. Encoding: 1100 0010 bit address.
CPL A	Complements accumulator. No flag affected. Byte: 1, M/C cycle: 1, oscillator periods: 12. Encoding: 1111 0110.
CPL C	Complements carry flag. Only carry flag affected. Byte: 1, M/C cycle: 1, oscillator periods: 12, Encoding: 1011 0011.
CPL bit	Complements the content of the specified bit address. No flag affected; Bytes: 2, M/C cycle: 1, oscillator periods: 12, Encoding: 1011 0010 bit address.
RL A	Rotate accumulator left. All the 8 bits in the accumulator are rotated one bit to the left. Bit 7 moves into the bit 0 position. No flag affected. Byte: 1, M/C cycle: 1, oscillator periods: 12, Encoding: 0010 0011.
RLC A	Rotate accumulator left through carry flag. All the 8 bits in the accumulator and the carry flag are rotated together one bit to the left. Bit 7 moves into the carry flag. The original value of the carry flag moves into the bit 0 position. Only carry flag affected. Byte: 1, M/C cycle: 1, oscillator periods: 12. Encoding: 0011 0011.

Mnemonic	*Description*
RR A	Rotate accumulator right. All the 8 bits in the accumulator are rotated one bit to the right. Bit 0 moves into bit 7 position. No flag affected. Byte: 1, M/C cycle: 1, oscillator periods: 12, Encoding: 0000 0011.
RRC A	Rotate accumulator right through carry flag. All the 8 bits in the accumulator and the carry flag are rotated together one bit to the right. Bit 0 moves into the carry flag. The original value of the carry flag moves into the bit 7 position. No flag affected, Byte: 1, M/C cycle: 1, oscillator periods: 12, Encoding: 0001 0011.
SWAP A	Swap nibbles within the accumulator. It interchanges the low-order and high-order nibbles of the accumulator. No flag affected, byte: 1, M/C cycle: 1, oscillator periods: 12, Encoding: 1100 0100.

10.1.16.4 Program Branching Instructions

Table 10.9 Summary of Program Branching Instructions

Mnemonic	*Description*
ACALL addr 11	Absolute Call. It unconditionally calls a subroutine located at the specified address. Before the control is transferred to the subroutine, the address of the next instruction of the main program is saved in the stack. No flag affected, Bytes: 2, M/C cycles: 2, oscillator periods: 24, immediate addressing. Encoding: $a_{10}\ a_9\ a_8\ 1\ 0001\ a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0$, where $a_0 - a_{10}$ is 11-bit address.
LCALL addr 16	Long subroutine Call. It calls subroutine located at the specified address. Before the control is transferred to the subroutine, the address of the next instruction of the main program is saved in the stack. No flag affected, Bytes: 3, M/C cycles: 2, oscillator periods: 24, immediate addressing, Encoding: 0001 0010 16-bit address.
RET	Return from subroutine. No flag affected. RET is used at the end of subroutine. Before the execution of a subroutine the address of the next instruction of the main program is saved in the stack. The execution of RET instruction brings back the saved address from the stack and keeps in the program counter for further execution. Byte: 1, M/C cycles: 2, oscillator periods: 24, register-indirect addressing, Encoding: 0010 0010.
RETI	Returns from interrupt service subroutine to the main program. RETI is used by the subroutine called by an interrupt. It pops the return address from the stack and keeps in PC for further execution. It also restores the interrupt logic to accept additional interrupt at the same priority level as the one just processed. PSW is not automatically restored to its pre-interrupt status. No other registers are affected. Byte: 1, M/C cycles: 2, oscillator periods: 24, register-indirect addressing. Encoding: 0011 0010.
AJMP addr 11	Absolute jumpt. It transfers program execution at the specified address. No flag affected, Bytes: 2, M/C cycles: 2, oscillator periods: 24, immediate addressing. Encoding: $a_{10}\ a_9\ a_8\ 0\ 0001\ a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0$, where $a_0 - a_{10}$ is 11-bit address.
LJMP addr 16	Unconditional long jump. It transfers program at the specified address. No flag affected, immediate addressing, Bytes: 3, M/C cycles: 2, oscillator periods: 24, Encoding: 0000 0010 16-bit address.

Mnemonic	Description
SJMP rel (rel is signed-displacement)	Unconditional short jump. 'rel' is relative. The branch destination address is computed by adding the specified signed displacement to the PC. No flag affected, Bytes: 2, M/C cycles: 2, Encoding: 1000 0000 relative address (signed-displacement).
JMP @A + DPTR	Jump indirect relative to DPTR. The branch destination address is computed by adding 8-bit unsigned content of the accumulator to the 16-bit pointer. No flag affected, Byte: 1, M/C cycles: 2, oscillator periods: 24, register-indirect addressing, Encoding: 0111 0011.
JZ rel (rel is signed-displacement)	Jump if accumulator is zero. No flag affected, Bytes: 2, M/C cycles: 2, oscillator periods: 24, Encoding: 0110 0000 relative address (singed-displacement). The branch destination address is computed by adding the specified signed-displacement to PC.
JNZ rel (rel is signed-displacement)	Jump if accumulator is not zero. No flag affected, Bytes: 2, M/C cycles: 2, oscillator periods: 24, Encoding: 0111 0000 relative address (signed-displacement). The destination address is computed by adding the signed displacement to PC.
CJNE A, direct, rel (rel is signed-displacement)	Compare direct byte to ACC. and jump if not equal, Bytes: 3, M/C cycles: 2, oscillator periods: 24, Encoding: 1011 0101 direct address relative address (singed-address). The branch destination address is computed by adding the signed displacement to PC).
CJNE A, # data, (rel is signed-displacement)	Compare immediate data to ACC. and jump if not equal. No flag affected, Bytes: 3, M/C cycles: 2, oscillator periods: 24, Encoding: 1011 0100 immediate data rel. address (signed- displacement). The branch destination address is computed by adding the signed displacement to PC.
CJNE Rn, # data, rel (rel is signed-displacement)	Compare immediate data to working register and jump if not equal. No flag affected, Bytes: 3, M/C cycles: 2, oscillator periods: 24, Encoding: 1011 1rrr immediate address rel. address (signed-displacement). Branch destination address is computed by adding the signed displacement to PC.
CJNE @ Ri, # data, rel (rel is signed-displacement)	Compare immediate data to content of data RAM location (addressed by Ri) and jump if not equal. No flag affected, Bytes: 3, M/C cycles: 2, oscillator periods: 24, Encoding: 1011 011i immediate data rel. address (signed- relative displacement). The branch destination address is computed by adding the signed displacement to PC.
DJNZ Rn, rel (rel is signed-displacement)	Decrement the specified working register and jump if not zero. The branch destination address is computed by adding the signed displacement to PC. No flag affected, Bytes: 2, M/C cycles: 2, oscillator periods: 24, Encoding: 1101 1rrr rel. address (signed-displacement).
DJNZ direct, rel (rel is signed-displacement)	Decrement the content of the specified direct address and jump if not zero. The branch destination address is computed by adding the signed-displacement to PC. Bytes: 3, M/C cycles: 2, oscillator periods: 24, Encoding: 1101 0101 direct address rel. address (signed-displacement).
NOP	No operation. Byte: 1, M/C cycle: 1, oscillator periods: 12. Encoding: 0000 0000.

Result

A-09H

B-14H

Note: In normal case the overflow flag is zero. When register B contains 00 (i.e. divisor is zero), the values of the quotient and remainder are undefined, and the overflow flag is set to 1. In this example, it was zero as it was an example of a normal case.

10.1.18 Atmel Microcontrollers

Atmel 89C51 and 89C2051 are CMOS 8-bit microcontrollers. The 89C51 is a 40 pin microcontroller whereas the 89C2051 is 20 pin microcontroller. Both 89CXX and 89C20XX have flash programmable and erasable read only memory (PEROM). Operating frequency is in the range of 0 – 24 MHz. Operating voltage is 2.7 – 6V. The 89CXX and 89C20XX are compatible with Intel 8051 family of microcontrollers. These Atmel microcontrollers have same instruction set as that of the 8051 microcontroller. The assembler and simulator for 8051 can also be used for 89C51 and 89C2051.

10.1.19 PIC Microcontrollers

PIC stands for Peripheral Interface Controller. PIC 16C6X and 16C7X are 8-bit PIC microcontrollers. They use RISC architecture and employ Harward bus system. The Harward bus architecture uses separate memory for program and data resulting in faster processing. The PIC microcontrollers have only 35 instructions which can easily be remembered. The instruction execution time for most of the instructions is 0.2 microsecond. The machine cycle is of 4 clock cycles whereas that of Intel 8051 is 12 clock cycles. The PIC 16F8XX series of microcontrollers use flash memory for storing program and EEPROM for data.

10.2 INTEL 8096 SERIES OF MICROCONTROLLERS (MCS-96)

Intel introduced 16-bit high-performance 8096 series of microcontrollers in the year 1983. These microcontrollers can perform complex calculations and can handle fast input-output operations. They are designed to have register-to-register architecture. The ALU operates directly on the register file. The CPU contains a 256-byte RAM which is called a register file. Each location of the register file acts as a register. The content of any location of the register file can be output to a port. This type of architecture is known as register-to-register architecture. In such an architecture accumulator bottleneck is eliminated and direct control of I/O operations is done. Fig. 10.19 shows the block diagram of MCS-96.

The 16-bit microcontrollers are used in applications requiring complex calculations and sophisticated control. They are used for real-time control applications involving closed-loop control, instrumentations, robotics, digital signal processing, the control of 3-phase motor, hard disk drive, printer, modems, missile, video recorder, oscilloscopes, intelligent transducers, tape drives, plotters, aerospace guidance system, torpedo guidance, etc.

Table 10.10 shows the latest family members of MCS-96.

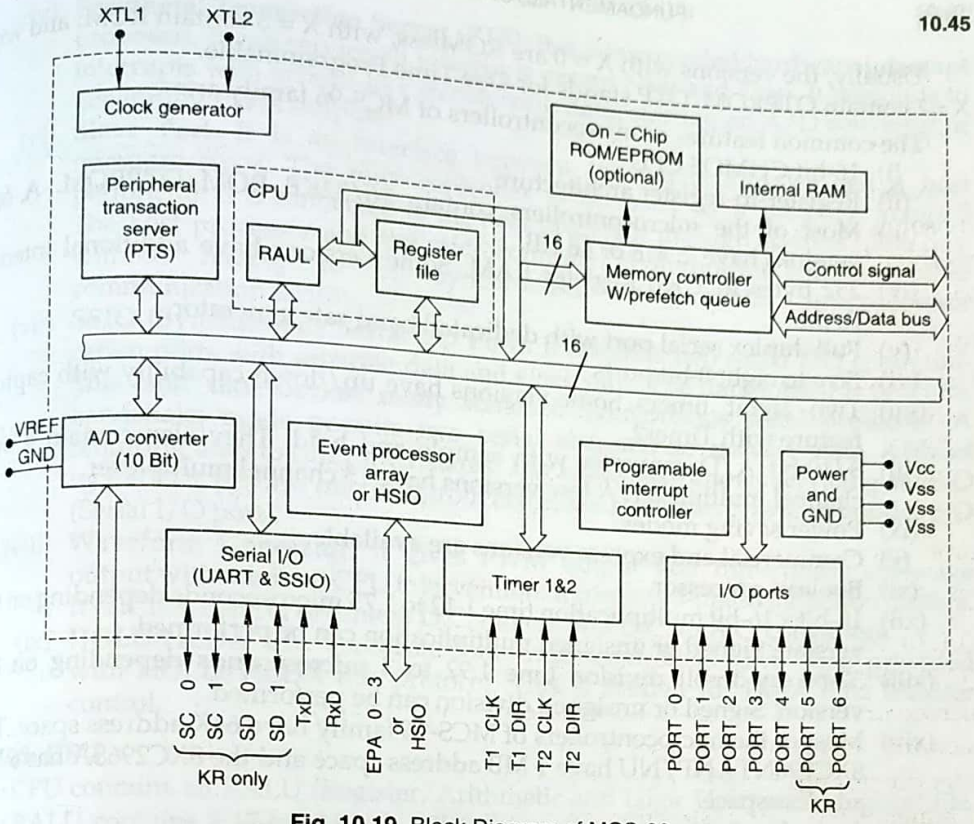

Fig. 10.19 Block Diagram of MCS-96

Table 10.10 MCS-96 Family Members

Product	*On-chip Program Memory*	*On-chip Data Memory in Bytes*	*CLK MHz*	*Versions Available*
8XC196KB	8KB ROM/OTP	232	16	X = 0, 3, 7
8XC198	8KB ROM/OTP	232	16	X = 0, 3, 7
8XC196KC	16KB ROM/OTP	488	16/20	X = 0, 3, 7
8XC196KD	32KB ROM/OTPROM	1000	16/20	X = 3, 7
8XC196KR/JR	16KB OTPROM	512 Reg.RAM 256 Internal RAM	16	X = 0, 7
8XC196KQ/JQ	12KB OTPROM	384 Reg. RAM, 128 internal RAM	16	X = 0, 7
8XC196NT	32KB OTPROM	1KB Reg. RAM, 512 internal RAM	20	X = 0, X = 7
8XC196NU	48KB ROM	1000	50	X = 0, X = 3
8X296SA	2KB ROM	512 Reg. RAM, 2KB Code/data RAM	40/50	X = 0, X = 3
8XC196MC	16KB OTPROM	488	8-16	X = 0, 3, 7
8XC196MD	16KB OTPROM	488	8-16	X = 0, 3, 7
8XC196MH	32KB OTPROM	744	16	X = 0, 3, 7

Usually, the versions with X = 0 are ROMless, with X = 3 contain ROM; and with X =7 contain OTPROM. OTP stands for One Time Programmable.

The common features of microcontrollers of MCS-96 family are:

(i) 16-bit CHMOS CPU
(ii) Register-to-register architecture
(iii) Most of the microcontrollers contain 16/32KB ROM/OTPROM. A few versions have 2, 4, 8 or 48 KB.
(iv) 232 bytes to 1 KB Register RAM. Some versions have additional internal RAM.
(v) Full duplex serial port with dedicated baud rate generator.
(vi) Five to eight 8-bit ports
(vii) Two 16-bit timers. Some versions have up/down capability with capture feature with Timer2.
(viii) 8/10-bit A/D converter with sample and hold. They also contain 4 or 8 channel multiplexer. A few versions have 13 channel multiplexer.
(ix) Power saving modes.
(x) Commercial and express versions are available.
(xi) Boolean processor.
(xii) 16-bit x 16-bit multiplication time 1.12 to 1.75 microseconds depending on the version. Signed or unsigned multiplication can be performed.
(xiii) 32-bit by 16-bit division time 1.92 to 3 microseconds depending on the version. Signed or unsigned division can be performed.
(xiv) Most of the microcontrollers of MCS-96 family have 64K address space. The 8XC196NT/ NP/NU have 1 MB address space and the 8XC296SA has 6 MB address space.

The special features possessed by some of the versions are:

(i) **Pulse-Width Modulated Output (PWM).** The PWM output is used as a rough D/A converter, for speed control of motor, duty cycle of heater, etc.
(ii) **Watchdog timer** is dedicated internal timer that resets the system if the software fails to operate properly.
(iii) **High-Speed Input/Output (HSIO) Units.** Besides standard I/O ports some MCS-96 microcontrollers contain high-speed input/output units. The high-speed input (HSI) unit records times of external events. It can monitor four independently configurable HSI lines. It captures the value of Timer1 when an event occurs. The HSI unit can store upto 8 entries.
The high-speed output (HSO) unit can trigger events at specified times based on Timer1 and Timer2 such as starting of A/D conversion, generation of one or more of four software timers, setting and clearing of one or more of the 6 HSO output lines, etc.
(iv) **Event Processor Array (EPA).** Some microcontrollers contain EPA instead of HSIO units. The EPA performs input event capture and output event generation functions employing Timer1 and Timber2. It contains 10 capture/compare modules, 2 compare only modules and the 2 timers. In input event capture mode it stores the value of the timer, generates and interrupt or both, when an external event occurs. In compare mode i.e. output mode), when the timer matches the value in compare register, the EPA changes the state of an output pin, generates an interrupt or both. The EPA sets, resets or toggles the output pin, when comparison occurs. The EPA also allows two channels to control a single output pin which is useful for high-speed PWM generation.

(v) **Peripheral Transaction Server (PTS).** It is a microcoded hardware interrupt processor. It can service all interrupts except NMI and Trap. It responds to interrupts with actions like transferring data, starting an A/D converter or generating PWM outputs.

(vi) **Slave Port.** It is an interface between the microcontroller and a host microprocessor. The microcontroller is accessed as a memory mapped peripheral. It is connected to the host processor through address/data bus. The host processor and the microcontroller communicate with each other without having to be synchronised. It provides interprocessor communication facility.

(vii) **SSIO (Synchronous Serial I/O Port).** It contains two serial I/O communication ports with separate data and clock pins. The data format is of 8 bits. This unit can support many standard synchronous serial protocols. A handshake mode permits two serial channels to transfer data without requiring lines to carry their status. The handshake allows servicing of SSIO by the PTS. In some microcontrollers the SSIO has been provided besides SIO (Serial I/O port).

(viii) **Waveform Generator.** It gives PWM output and three-phase sinewave output with minimal CPU intervention. It controls 3-phase induction motors. It can also control brushless D.C. motors and D.C. to A.C. converters.

(ix) **HOLD/HLDA Bus Protocol.** Some microcontrollers have been provided with HOLD/HLDA bus protocol. This is needed for DMA data transfer control.

10.2.1 CPU

The CPU contains an RALU (Register/Arithmetic and Logic Unit) and a register file. The RALU contains a 17-bit ALU, a PSW (program status word), a loop counter, an instruction register, a program counter (PC) and three temporary registers. The 17-bit ALU uses 16 bits for data and one bit for sign extension. The RALU does not use an accumulator. It operates directly on each location of the register file which is a 256-byte RAM. Every location of the register file acts as a register. This type of architecture is known as **register-to-register** architecture. The contents of any location of the register file can be output to a port. So, in register-to-register architecture accumulator bottleneck is eliminated and direct control of I/O operations is performed. Any location of the register file can act as a source or destination for most of the instructions.

The register file is a 256-bytes (or more) RAM. The RAM locations from 00H to 17H are used to serve as I/O control registers or **special function registers (SFRs)**. Locations 18H and 19H are used as stack pointer. When stack operation is not needed, these locations can be used as general purpose RAM. The remaining bytes of the register file are used as general purpose RAM which is known as **Register RAM**. The register RAM can be accessed as bytes, words or double words. Some microcontrollers contain a register file of capacity more than 256 bytes, see Table 10.10. Besides register RAM, some microcontrollers contain additional RAM called **internal RAM**.

10.2.2 Serial Port

The serial port is provided with one synchronous mode and three asynchronous modes. The mode 0 is synchronous mode. Mode 1, Mode 2 and Mode 3 are asynchronous modes. In synchronous mode 8 bits of data, without start or stop bits, are transmitted or received. All three asynchronous modes can support full or half duplex operations. In full-duplex operation each system can transmit and receive information at the same time. It requires two serial links. The transfer of information takes place in

both directions simultaneously and independently. In half-duplex operation communication can take place in either direction between two systems using a single link. But data are transmitted only in one direction at a time.

The Mode 1 is the standard asynchronous mode which is used for normal serial communication. The data frame for Mode 1 consists of 10 bits : 8 data bits, a start bit and a stop bit. If parity bit is used, one bit is used for the parity and only 7 bits are used for data.

The Mode 2 and Mode 3 are used for multiprocessor communication. The data frame consists of 11 bits : a start bit, 9 data bits and a stop bit.

The major components of microcontrollers of MCS-96 family have already been described. Table 10.11 shows major components of MCS-96 family members. All members contain two 16-bit timers. Many microcontrollers contain 16-bit watchdog timer, four 16-bit software interrupts, etc.

Table 10.11 Major Components of MCS-96 Family Members

Product	*HSIO or EPA*	*ADC*	*Serial Port*	*SSIO*	*PWM*	*PTS*	*$\overline{HOLD}$ / $\overline{HLDA}$*	*Slave Port*	*3-Phase Wave-generator*
8XC196KB	HSIO	Yes	Yes	—	1	—	Yes	—	—
8XC198	HSIO	Yes	Yes	—	1	—	—	—	—
8XC196KC	HSIO	Yes	Yes	—	3	Yes	Yes	—	—
8XC196KD	HSIO	Yes	Yes	—	3	Yes	Yes	—	—
8XC196KR/ KQ/JR/HQ	EPA	Yes	Yes	Yes	—	Yes	Yes	Yes	—
8XC196NT	EPA	Yes	Yes	Yes	—	Yes	Yes	Yes	—
8XC196NP	EPA	—	Yes	—	3	Yes	—	—	—
8XC196NU	EPA	—	Yes	—	3	Yes	—	—	—
8X196MC	EPA	Yes	—	—	2	Yes	—	—	Yes
8X196MD	EPA	Yes	—	—	2	Yes	—	—	Yes
8X196MH	EPA	Yes	Yes	—	2	Yes	—	—	Yes
8XC296SA	EPA	—	Yes	—	3	—	—	—	—

10.2.3 Brief Description of 8096 Microcontrollers

8XC198. It is a low-cost entry-level CHMOS 16-bit microcontroller. It contains 8 KB ROM/ OTPROM, 232 bytes register RAM, 28 interrupt sources/16 vectors, high-speed I/O subsystem, PWM, two 16-bit timers, 16-bit watchdog timer, 10-bit A/D converter, full duplex serial port, four 16-bit software timers, etc. It has 8-bit external bus. It is an 8-bit bus version of 8XC196KB. It is provided with four high-speed inputs to record times when events occur, and six high-speed outputs for pulse or wave generation. Both commercial and express versions are available. It has register-to-register organisation and operates at 16 MHz. It has general features of MCS-96 as already explained. ROM, OTP and ROMless versions are available.

8XC196KB. It is a high-performance CHMOS 16-bit microcontroller. It has all general features of MCS-96. Besides general features, it has 10-bit A/D converter, $\overline{HOLD}$/$\overline{HLDA}$ bus protocol and dynamically configurable 8-bit or 16-bit buswidth. It has 8 KB on-chip ROM/OTP, 232 bytes Reg. RAM, 16-bit timer, 16-bit up/down counter with capture, etc. It has 4 high-speed inputs to record times when events occur,

and six high-speed outputs for pulse or waveform generation. It has 1.75 microseconds 16 × 16 multiplication and 3 microseconds 32/16 division time, five 8-bit I/O ports, 28 interrupt sources/16 vectors, four 16-bit software timers, etc. Both commercial and express versions are available. ROM, OTP and ROMless versions are available.

8XC196KC. It is an enhanced version of 8XC196KB. It has 488 bytes of Reg. RAM and 16 KB ROM/OTP, 1.4 micro-second 16 × 16 multiplication and 2.4 microseconds 32/16 division time, 16/20 MHz clock, 3 PWMs, PTS, etc. All other features are same as those of 8XC196KB. ROM, OTP and ROMless versions are available.

8XC196KD. It is an enhanced version of 8XC196KC. It has 32 KB ROM/OTP and 1000 bytes Reg. RAM. All other features are same as those of 8XC196KC. ROM and OTP versions are available.

8XC196KR/KQ/JR/JQ. These are the 4th generation of MCS-96 microcontrollers implemented on Intel's 1 micron process technology. They have all general features of MCS-96 microcontrollers. Besides the general features they have been provided with EPA (Event Processor Array with 10 high-speed capture/compare), SSIO (Synchronous serial I/O), 128/256 bytes internal RAM, 512/384 bytes Reg. RAM, 16K/12K OTP, 37 prioritized interrupt sources, 41/56 I/O lines, interprocessor communication slave port, PTS, ADC, two dedicated high-speed compare registers, 1.75 microsecond 16 × 16 multiplication and 3 microseconds 32/16 division time, $\overline{\text{HOLD}}/\overline{\text{HLDA}}$ bus protocol, etc. ROMless and OTP versions are available.

8XC196NT. It is an enhanced version of 8XC196KR with 1 Mbyte address space, 1000 bytes Reg. RAM, 512 bytes internal RAM and 20 MHz clock. Its OTP and ROMless versions are available. Its 16 × 16 multiplication time is 1.4 microsecond and 32/16 division time is 2.4 microseconds. Other features are same as those of 8XC196KR.

8XC196NP. Besides general features of MCS-96 microcontrollers it has 1 Mbytes linear address space, 1KB Reg. RAM, a demultiplexed bus and a chip select unit. The external bus can dynamically switch between multiplexed and demultiplexed operation. It has 3 PWMs, 4 external interrupt pins and NMI, 25 MHz clock, 16 prioritized interrupt sources, 4 KB optional ROM, PTS, full duplex serial port, 32 I/O lines, EPA, etc. It can perform unsigned multiplication and division. Its ROM and ROMless versions are available.

8XC196NU. Its features are similar to 8XC196NP with optional 48 KB ROM and 50 MHz clock. Other features are same as those of 8XC196NP. Unsigned division time is 960 ns and multiplication time 640 ns.

8X296SA. It is footprint and functionality compatible upgrade for 8XC196NP and 8XC196NU with 512 Reg. RAM, 6MB linear address space, 2KB code/data RAM, 19 interrupt sources (14 with programmable priorities), multiplication time 80 ns, and division time 880 ns, etc. Other features are same as those of 8XC196NU. Its external bus can dynamically switch between multiplexed and demultiplexed operation. The device has hardware and instructions to support digital signal processing algorithms.

8XC196MC. It is 16-bit industrial motor control microcontroller. It is mainly designed to control 3 phase A.C. induction and D.C. brushless motors. It contains 16KB program memory, 488 bytes Reg. RAM, 53 I/O lines, PTS, EPA, 3-phase waveform generator, 13 channel 8/10-bit ADC with sample and hold, 14 prioritized interrupt sources, flexible 8/16-bit external bus, watchdog timer, 2 PWMs, 8-16 MHz clock, etc.16 × 16 multiplication time is 1.75 microsecond and 32/16 division time is 3 microseconds. The 87C196MC contains 16KB on-chip user programmable OTPROM.

CHAPTER 11

INTEL 8086 AND INTEL'S OTHER 16-BIT MICROPROCESSORS

11.1 INTRODUCTION

Intel's 16-bit microprocessors are 8086 , 80186 and 80286. These microprocessors have 16-bit internal architecture and contain 16 data lines. The 8088 and 80188 have their internal architecture of 16-bit but their data lines are only 8. All these microprocessors come under Intel's 8086 family. The Intel 8086, 80186, 80286, 8088 and 80188 microprocessors have the same basic set of registers, instructions and addressing modes. The 8088 microprocessor was very popular CPU for cheaper desktop computers, PC/XT (Personal Computer Extended Technology). PC/XT computers were widely used in 1980s. As the 8088 microprocessor had only 8 data lines, it used cheaper 8-bit peripherals. The 80286 microprocessor was used for costlier computers, called PC/AT (Personal Computer Advanced Technology). These computers used 16-bit peripherals. The 80286 microprocessor was also very popular in 1980s.

The 80186 and 80188 were integrated microprocessors. They contain 8086 and 8088 CPU respectively. Besides CPU, they also contain on-chip clock generator, programmable memory and chip select logic, programmable interrupt controller, high-speed DMA channels, programmable 16-bit timers, local bus controllers etc. These CPUs were not popular for general-purpose computers, but the computers based on these CPUs were very chip and used for industrial control, instrument and equipment control etc.

The computers based on 8086 family CPUs are no longer manufactured. They are described here as their knowledge provides the base to understand the latest CPUs which are used today.

11.2 INTEL 8086

The 8086 is a 16-bit, N-channel, HMOS microprocessor. The term HMOS is used for "high-speed MOS". Its CMOS (Complementary MOS) version, the 80C86 is also available. It consumes less power. The 8086 draws 360 mA on 5 V whereas the 80C86 draws only 10 mA. The 8086 is manufactured for standard temperature range 32° F to 180°F as well as extended temperature range (40°F to +225°F). Its clock frequencies for its different versions are: 5, 8 and 10 MHz. It was introduced in 1978. It contains an electronic circuitry of 29000 transistors. It is built on a single semiconductor chip and packaged in a 40-pin IC package. The type of the package is DIP (Dual In-Line Package).

The 8086 uses 20 address lines and 16 data lines. It can directly address up to $2^{20} = 1$ Mbytes of memory. The 16-bit data word is divided into a low-order byte and a high-order byte. The 20 address lines are time multiplexed lines. The 16 low-order address lines are time multiplexed with data, and the 4 high-order address lines are time multiplexed with status signals. Figure 11.1 shows the pin diagram of Intel 8086.

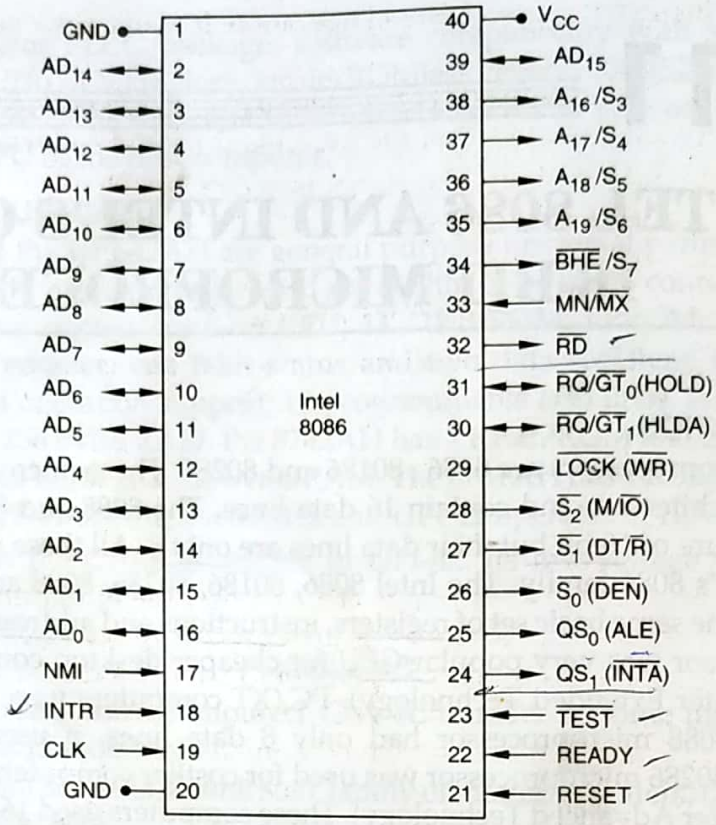

Fig. 11.1 Pin Diagram of Intel 8086 Microprocessor

AD0-AD15 are 16 low-order address lines. 8 LSBs of data are transmitted on AD0-AD7 and 8 MSBs of data on AD8-AD15. $\overline{BHE}$ at pin number 34 is multiplexed with status signals S7. The Intel 8284 clock generator/driver is used to generate the clock signal required for 8086 microprocessor. In a microprocessor-based system when bus controller is required the Intel 8288 is used with 8086 CPU. In a single bus configuration the 8288 may not be used.

11.2.1 Pin Description of 8086

The description of the pins of 8086 is as follows:

AD0-AD15. (Bidirectional) Address/Data lines. These are low-order address bus. They are multiplexed with data.

When AD lines are used to transmit memory address the symbol A is used instead of AD, for example A0-A15. When data are transmitted over AD lines the symbol D is used in place of AD, for example D0-D7, D8-D15 or D0-D15.

A16-A19 (Output). High-order address lines. These are multiplexed with status signals.

A16/S3, A17/S4. A16 and A17 are multiplexed with segment identifier signals S3 and S4.

A18/S5. A18 is multiplexed with interrupt status S5.

A19/S6. A19 is multiplexed with status signal S6.

$\overline{BHE}$/S7 (Output). Bus High Enable/Status. During T1 it is low. It is used to enable data onto the most significant half of data bus, D8-D15. 8-bit device connected to upper half of the data bus use BHE signal. It is multiplexed with status signal S7. S7 signal is available during T3 and T4.

$\overline{RD}$ (Read). This signal is used for read operation. It is an output signal. It is active when LOW.

READY(Input). The addressed I/O or memory sends acknowledgment through this pin. When HIGH it indicates that the peripheral is ready to transfer data.

RESET (Input). System reset. The signal is active HIGH.

CLK (Input). Clock. 5, 8 or 10 MHz.

INTR (Interrupt request)

NMI (Input). Non-maskable interrupt request.

$\overline{TEST}$ (Input). Wait for test control. When it is low the microprocessor continues execution otherwise waits.

VCC. Power supply, + 5 V d.c.

GND. Ground.

11.2.2 Operating Modes of 8086

There are two modes of operation for Intel 8086, namely the minimum mode and the maximum mode. When only one 8086 CPU is to be used in a microcomputer system the 8086 is used in the minimum mode of operation. In this mode the CPU issues the control signals required by memory and I/O devices. In a multiprocessor system it operates in the maximum mode. In case of maximum mode of operation control signals are issued by Intel 8288 bus controller which is used with 8086 for this very purpose. The level of the pin MN/$\overline{MX}$ decides the operating mode of 8086. When MN/$\overline{MX}$ is high the CPU operates in the minimum mode. When it is low the CPU operates in the maximum mode. From pin 24 to 31 issue two different sets of signals. One set of signals is issued when the CPU operates in the minimum mode. The other set of signals is issued when the CPU operates in the maximum mode. Thus, the pins from 24 to 31 have alternate functions.

11.2.3 Pin Description for Minimum Mode

For the minimum mode of operation the pin MN/$\overline{MX}$ is connected to 5 V d.c. supply, i.e. MN/$\overline{MX}$ = VCC. The description of the pins from 24 to 31 for the minimum mode is as follows:

$\overline{INTA}$ (Output). Pin No. 24. Interrupt acknowledge. On receiving interrupt signal the processor issues an interrupt acknowledge signal. It is active LOW.

ALE (Output). Pin No. 25. Address latch enable. It goes HIGH during T1. The microprocessor sends this signal to latch the address into the Intel 8282/8283 latch.

$\overline{DEN}$ (Output). Pin No. 26. Data enable. When Intel 8286/8287 octal bus transceiver is used this signal acts as an output enable signal. It is active LOW.

DT/$\overline{R}$ (Output). Pin No. 27. Data Transmit/Receive. When Intel 8286/8287 octal bus transceiver is used this signal controls the direction of data flow through the transceiver. When it is HIGH data are sent out. When it is LOW data are received.

M/$\overline{IO}$ (Output). Pin No. 28. Memory or I/O access. When it is HIGH the CPU wants to access memory. When it is LOW the CPU wants to access I/O device.

$\overline{WR}$ (Output). Pin No. 29. Write. When it is LOW the CPU performs memory or I/O write operation.

HLDA (Output). Pin No. 30. HOLD acknowledge. It is issued by the processor when it receives HOLD signal. It is active HIGH signal. When HOLD request is removed HLDA goes LOW.

HOLD(Input). Pin No. 31. Hold. When another device in microcomputer system wants to use the address and data bus, it sends a HOLD request to CPU through this pin. It is an active HIGH signal.

11.2.4 Pin Description For Maximum Mode

For the maximum mode of operation the pin MN/$\overline{MX}$ is made LOW. It is grounded. The description of the pins from 24 to 31 is as follows:

QS1, QS0 (Output). Pin Nos. 24, 25. Instruction Queue Status. Logics are given below:

QS1	QS0	
0	0	No operation
0	1	1st byte of opecode from queue
1	0	Empty the queue
1	1	Subsequent byte from queue

$\overline{S0}$, $\overline{S1}$, $\overline{S2}$ (Output). Pin Nos. 26, 27, 28. Status signals. These signals are connected to the bus controller Intel 8288. The bus controller generates memory and I/O access control signals. Table 11.1 shows the logic for status signals.

Table 11.1 Logic for Status Signals

S2	S1	S0	
0	0	0	Interrupt acknowledge
0	0	1	Read data from I/O port
0	1	0	Write data into I/O port
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Passive state.

$\overline{LOCK}$ (Output). Pin No. 29. It is an active LOW signal. When it is LOW all interrupts are masked and no HOLD request is granted. In a multiprocessor system all other processors are informed by this signal that they should not ask the CPU for relinquishing the bus control.

$\overline{RQ}/\overline{GT_1}$, $\overline{RQ}$ / $\overline{GT_0}$ (Bidirectional). Pin Nos. 30, 31. Local Bus Priority Control. Other processors ask the CPU through these lines to release the local bus. After receving the request ($\overline{RQ}$), the CPU sends a request grant signal ($\overline{GT}$) through the same line indicating that the request has been granted and buses will be released after the end the processor's current bus cycle. $\overline{RQ}/\overline{GT_0}$ has higher priority than $\overline{RQ}/\overline{GT_1}$.

In the maximum mode of operation signals $\overline{WR}$, ALE, $\overline{DEN}$, DT/$\overline{R}$ etc. are not available directly from the processor. These signals are available from the controller 8288.

11.2.5 Functional Units of Intel 8086

The 8086 contains two independent functional units: a bus interface unit (BIU) and an execution unit (EU). Figure 11.2 shows the block diagram of 8086 microprocessor. The general purpose registers, stack pointer, base pointer and index registers, ALU, flag register (FLAGS), instruction decoder and timing and control unit constitute execution unit (EU). The segment registers, instruction pointer and 6-byte instruction queue are associated with the bus interface unit (BIU).

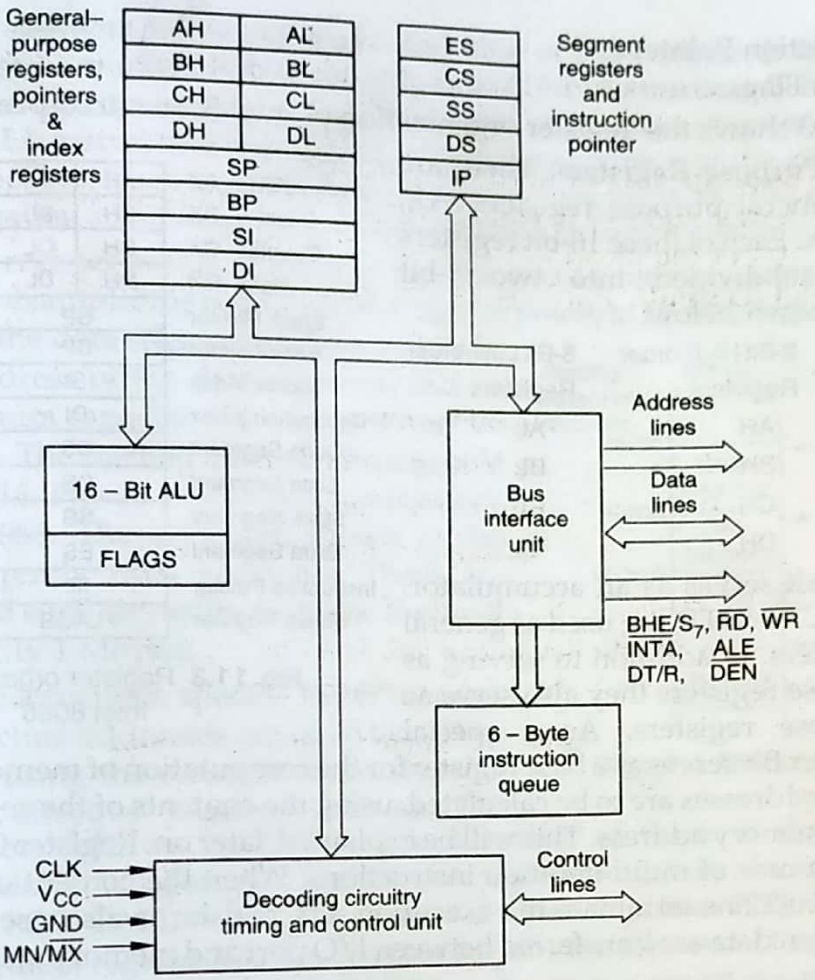

Fig. 11.2 Block Diagram of Intel 8086 Microprocessor

The BIU handles transfer of data and addresses between the processor and memory/IO devices. It computes and sends out addresses, fetches instruction codes, stores fetched instruction codes in a first-in-first-out register set called a **queue**, reads data from memory and I/O devices, writes data to memory and I/O devices. It relocates addresses of operands since it gets unrelocated operand addresses from EU. The execution unit tells BIU from where to fetch instructions or to read data.

The EU receives opecode of an instruction from the queue, decodes it and then executes it. While EU is decoding an instruction or executing an instruction, the BIU fetches instruction codes from the memory and stores them in the queue. The BIU and EU operate in parallel independently. This makes processing faster except in the cases of jump and CALL instructions, where the queue must be dumped and then reloaded from a new address.

While EU executes instructions, the BIU fetches instructions. This type of overlapped operation of the functional units of a microprocessor is called **pipelining**. This technique has been extended to more than two functional units in case of 32-bit and 64-bit microprocessors.

11.2.6 Registers of Intel 8086

The Intel 8086 contains the following registers:

(i) General Purpose Registers
(ii) Pointer and Index Registers
(iii) Segment Registers

(iv) Instruction Pointer
(v) Status Flags

Figure 11.3 shows the register organization of Intel 8086 micro-processor.

General Purpose Registers. There are four 16-bit general purpose registers: AX, BX, CX and DX. Each of these 16-bit registers are further subdivided into two 8-bit registers as shown below:

16-Bit Registers	8-Bit High-order Registers	8-Bit Low-order Registers
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

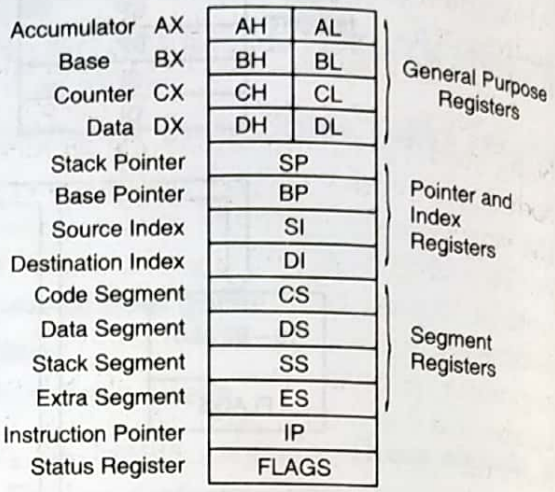

Fig. 11.3 Register organization of Intel 8086

Register AX serves as an accumulator. Registers BX, CX and DX are used as general purpose registers. In addition to serving as general purpose registers they also serve as special purpose registers. As a special purpose register BX serves as a base register for the computation of memory address. In 8086 memory addresses are to be calculated using the contents of the segment register and effective memory address. This will be explained later on. Register CX is also used as a counter in case of multi-iteration instructions. When the content of CX becomes zero such instructions terminate the execution. DX register is also used for memory addressing when data are transferred between I/O port and memory using certain I/O instructions.

Pointer and Index Registers. The following four registers are in the group of Pointer

and Index Registers:

1. Stack Pointer, SP
2. Base Pointer, BP
3. Source Index, SI
4. Destination Index, DI

The function of SP is same as the function of stack pointer in Intel 8085. BP, SI and DI are used in memory address computation.

Segment Register. There are four segment registers in 8086 as given below:

1. Code Segment Register, CS
2. Data Segment Register, DS
3. Stack Segment Register, SS
4. Extra Segment Register, ES

In an 8086 microprocessor-based system memory is divided into the following four segments:

1. Code Segment
2. Data Segment
3. Stack Segment
4. Extra Segment

The code segment of the memory holds instruction codes of a program. The data, variables and constants given in the program are held in the data segment of the

memory. Stack segment holds addresses and data of subroutines. It also holds the contents of registers or memory locations given in PUSH instruction. Before attending an interrupt, the microprocessor saves the content of program counter on the stack. Also, when CALL instruction is executed, before the execution of the subroutine, the address of the next instruction of the program is saved on the stack. The extra segment holds the destination addresses of some data of certain string instructions, and so on.

A segment register points to the starting address of a memory segment currently being used. For example, the code segment register points to the starting address of the code segment, the data segment register points to the starting address of the data segment, and so on. The maximum capacity of a segment may be upto 64 Kbytes. The starting address of a segment is divisible by 16. Figure 11.4 shows the memory segments of 8086. The segments shown in the figure are currently used segments. There are more number of such segments to make the total memory capacity 1 Mbytes.

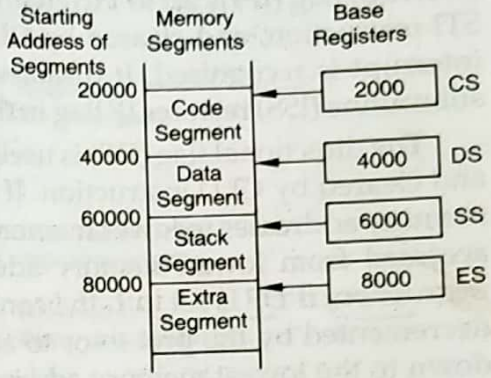

Fig. 11.4 Memory Segments

The 8086 instructions specify 16-bit memory address. The actual addresses are of 20 bits. They are calculated using the contents of the segment registers and effective memory address. The effective memory address is computed in a variety of ways. It depends on the addressing modes. It will become clear later on. For the execution of stack instructions such as PUSH, POP, CALL or RET, the content of the stack pointer (SP) and the content of the stack segment register (SS) are used to compute the address of the stack location to be accessed. The index registers SI and DI together with segment registers DS and ES are used to perform string operations. The source addresses for string operations are computed using the contents of SI and DS. The destination addresses for string operations are computed using the contents of DI and ES.

Instruction Pointer (IP). The instruction pointer in the 8086 microprocessor acts as a program counter. It points to the address of the next instruction to be executed. Its content is automatically incremented when the execution of a program proceeds further. The content of the instruction pointer (IP) and the content of the code segment register (CS) are used to compute the memory address of the instruction code to be fetched. This is done during instruction fetch operation.

Status Register. The 8086 contains a 16-bit status register. It is also called flag register or program status word (PSW). There are 9 status flags as shown in Fig. 11.5. Seven bit positions of the status register remain unused. Out of nine flags 6 are condition flags, and three are control flags. The six condition flags are: carry, auxiliary

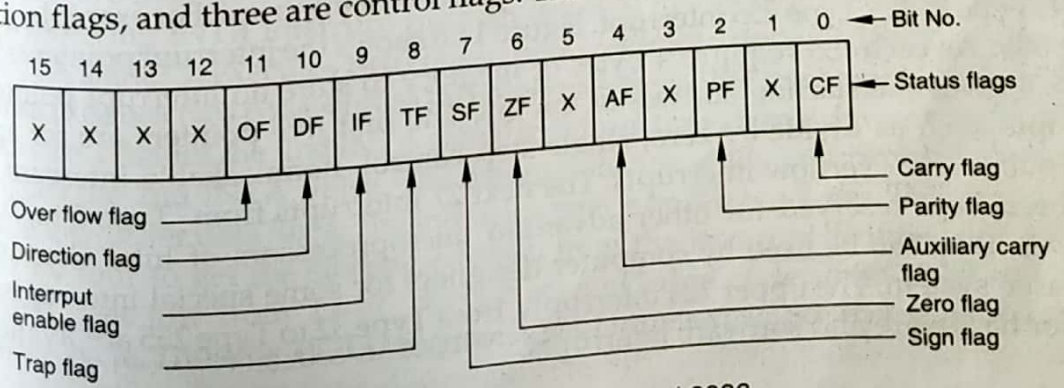

Fig. 11.5 Status Flags of Intel 8086

carry, zero, sign, parity and overflow flags. These flags are set/reset by the processor after the execution of an arithmetic or logical instruction. The three control flags are: trap (or trace), interrupt and directional flag. These flags are set/reset by the programmer as required by certain instructions in the program. The overflow, trap, interrupt and directional flags are new. Other flags are same as those available in Intel 8085. The overflow flag is set to 1, if the result of a signed operation becomes out of range, otherwise it is reset *i.e.* it is made 0.

When the trap flag (TF) is set to 1, a program can be run in single-step mode. The interrupt flag (IF) is set to 1 to enable INTR of 8086. If it is 0, INTR is disabled. It is set by STI instruction, and cleared by CLI instruction. IF is automatically cleared when an interrupt is recognised. It disables INTR. IRET used at the end of interrupt service subroutine (ISS) restores IF flag in the state in which it was before interrupt occurred.

The directional flag (DF) is used in string operation. It can be set by STD instruction and cleared by CLD instruction. If it is set to 1, string bytes are accessed from higher memory addresses to lower memory address. When it is set to zero, the string bytes are accessed from lower memory addresses to higher memory addresses. For MOVS instruction, if DF is set to 1, the contents of index registers SI and DI are automatically decremented by the processor to access the string from the highest memory address down to the lowest memory address. If DF is made zero, SI and DI are automatically incremented to access the string starting with the lowest address.

11.2.7 Interrupts

External devices or internal abnormal conditions can interrupt the normal program execution of a microprocessor. An interrupt caused by an external device is called **hardware interrupt**. A microprocessor can also be interrupted by internal abnormal conditions such as overflow, division by zero, etc. A programmer can also interrupt microprocessor by inserting INT instruction at the desired point in the program while debugging a program. Such an interrupt is called a **software interrupt**. The interrupts caused by internal abnormal conditions also come under the heading of software interrupt.

The 8086 can handle up to 256, hardware and software interrupts. To store the starting addresses of ISS for 256 interrupts, 1KB memory from 00000 to 003FF (hex) is set aside. To store the starting address of each ISS 4 bytes of memory space is needed : 2 bytes to store the value for CS and 2 bytes for IP values. The starting address of an ISS stored in the 1 KB memory space is called interrupt vector or interrupt pointer.

1 KB memory space acts as a table to contain interrupt vectors (or interrupt pointers), and hence it is called **interrupt vector table (or interrupt pointer table).** The 256 interrupt pointers have been numbered from 0 to 255 (FF hex). The number assigned to an interrupt pointer is known as type of that interrupt. For example, Type 0, Type 1, Type 2, ... , Type 255 interrupt. Figure 11.6 shows the interrupt pointer table for Intel 8086. As each ISS requires 4 bytes of memory to store an interrupt pointer, the Type × 4 gives the starting address of the ISS. The first five pointers are for specific interrupts such as divide-by-zero, single-step control, nonmaskable interrupt NMI, break point, and overflow interrupts. The next 27 interrupts from Type 5 to Type 31 have been kept reserved for other advanced microprocessors of Intel Corporation. However, they can be used by computer designers for some special interrupts in an 8086 based system. The upper 224 interrupts, from Type 32 to Type 255 are available to users for hardware and software interrupts.

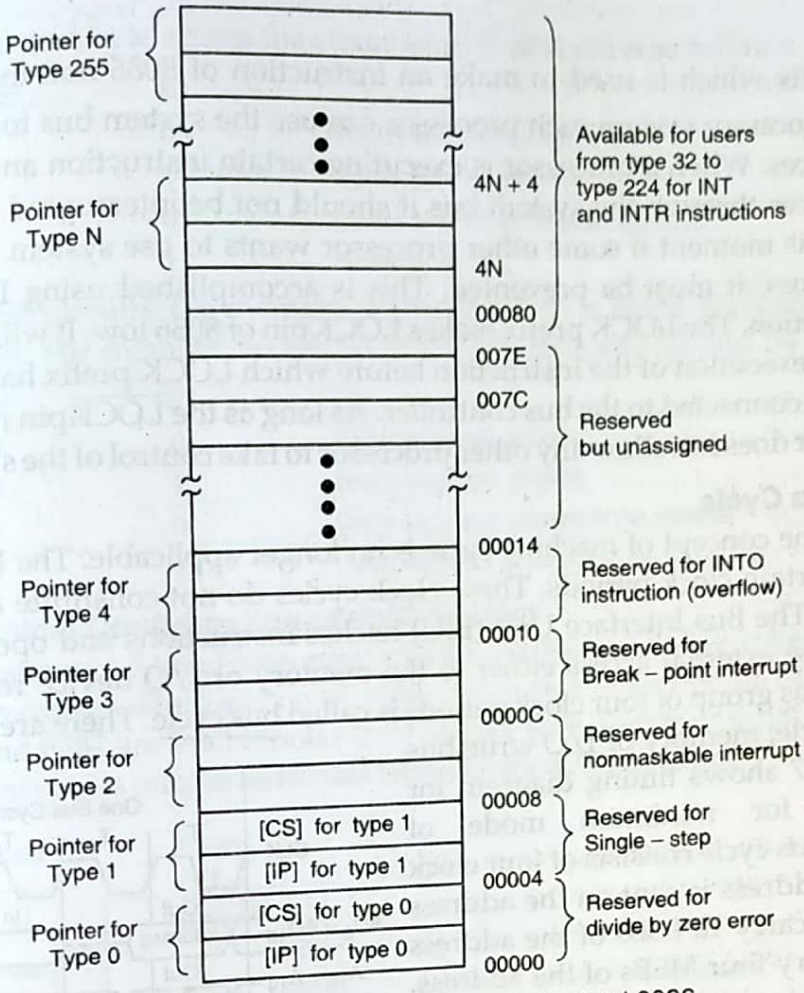

Fig. 11.6 Interrupt Pointer Table for Intel 8086

The 8086 has two hardware interrupts: NMI and INTR. The NMI is a onmaskable interrupt. It can not be disabled (masked) by user by using software. It is used by the processor to handle emergency conditions. For example, it can be used to save program and data in case of power failure. An external electronic circuitry is used to detect power failure, and to send an interrupt signal to 8086 through NMI line. In case of a.c. power failure, the D.C. supply to the computer can be maintained for a short period, say 50 milliseconds using suitable capacitors in the filter circuit. This time is sufficient to save program and data. The ISS for Type 2 interrupt saves program after power failure in some RAM provided with battery backup. The program alongwith data is restored, and it can be executed again from the point at which it was interrupted.

The INTR is a maskable interrupt. It can be enabled/disabled using interrupt flag (IF). After receiving INTR interrupt from an external device, the 8086 acknowledges through INTA signal. It executes two consecutive interrupt acknowledge bus cycles. In other words it issues two consecutive INTA signals to get the interrupt type from the external device. It issues the first INTA signal to inform the external device to get ready. Then it issues the 2nd INTA pulse. The external device sends interrupt type to 8086 through the low-order data lines. The interrupt type is an 8-bit number. This number is multiplied by four to get the starting address from the interrupt pointer table. The $\overline{\text{LOCK}}$ signal goes low from T2 of the 1st acknowledge cycle until T2 of the 2nd acknowledge cycle. Thus the HOLD request is not honoured till the end of the 2nd bus cycle.

11.2.8 LOCK

LOCK is a prefix which is used to make an instruction of 8086 non-interruptable.

In a multiprocessor system each processor can use the system bus to make use of common resources. When a processor is executing certain instruction and is utilising common resources through the system bus it should not be interrupted by the other processor. At this moment if some other processor wants to use system bus and use common resources, it must be prevented. This is accomplished using LOCK prefix before an instruction. The LOCK prefix makes $\overline{\text{LOCK}}$ pin of 8086 low. It will remain low throughout the execution of the instruction before which LOCK prefix has been used. The $\overline{\text{LOCK}}$ pin is connected to the bus controller. As long as the $\overline{\text{LOCK}}$ pin remains low, the bus controller does not allow any other processor to take control of the system bus.

11.2.9 8086 Bus Cycle

In case of 8086 the concept of machine cycle is no longer applicable. The EU executes instruction in certain clock periods. These clock cycles do not constitute any form of machine cycles. The Bus Interface Unit (BIU) fetches instructions and operands from the memory. Any external access either to the memory or I/O device requires four clock periods. This group of four clock periods is called bus cycle. There are memory or I/O read bus cycle; memory or I/O write bus cycle. Figure 11.7 shows timing diagram for memory read for minimum mode of operation. One bus cycle consists of four clock periods. 20-bit address is sent on the address bus. AD0-AD15 carry 16 LSBs of the address and A16-A19 carry four MSBs of the address. The address remains on the address lines only during T1 because these lines operate in multiplexed mode. AD0-AD15 lines remain in high impedance state during T2. During T3 and T4 data are transmitted on AD0-AD15. During T2, T3 and T4 status signals S3, S4, S5 and S6 are carried by A16-A19 lines. The ALE signal goes high during T1 to latch the address into external latch. MN/$\overline{\text{MX}}$ is high indicating that the CPU is operating in the minimum mode. M/$\overline{\text{IO}}$ is high. It indicates that the address sent by the CPU is for memory. $\overline{\text{RD}}$ goes low during T2. When it goes low data can be read by the CPU. It goes again high in T4. DT/$\overline{\text{R}}$ goes low during T1. This signal is used by transceiver 8286 or 8287 if it is used in the system. $\overline{\text{DEN}}$ goes low during T2 and again it becomes high during T4. It is provided for data enable for 8286/8287 if it is used in the system. DT/$\overline{\text{R}}$ and $\overline{\text{DEN}}$ signals are used to control bidirectional latched buffers. These are specially designed to work with 8286/8287. $\overline{\text{BHE}}$ in conjunction with A0 decides whether a byte or a word is to be transferred from/to memory locations. $\overline{\text{BHE}}$ identifies high order byte of memory word whereas A0 identifies low order byte. On $\overline{\text{BHE}}$ line the status signal S7 is transmitted during T2, T3 and T4.

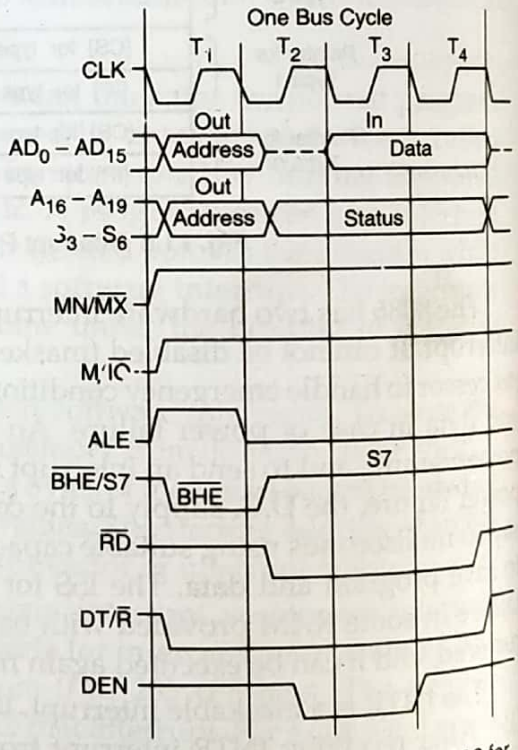

Fig. 11.7 Timing Diagram of Intel 8086 for Memory Read in Minimum Mode

For word and byte access the status of $\overline{BHE}$ and A0 is as follows:

$\overline{BHE}$	A0	
0	0	Whole word is transferred
0	1	Upper byte from/to odd address
1	0	Lower byte from/ to even address
1	1	None

S3 and S4 identify the memory segment which is being accessed. S5 is interrupt enable status. For memory access S6 is always zero and S3 and S4 are as follows:

S4	S3	
0	0	Extra segment access
0	1	Stack segment access
1	0	Code segment access or no access
1	1	Data segment access

11.2.10 Typical Configuration of 8086 System

Figure 11.8 shows a typical 8086 system in the minimum mode configuration. The 8286 is a bus transceiver. The function of a bus transceiver is to provide buffer for data bus. The 8286 and 8287 are 8-bit bipolar transceivers with 3-state outputs. The 8287 inverts the input data at its output terminals whereas the 8286 does not.

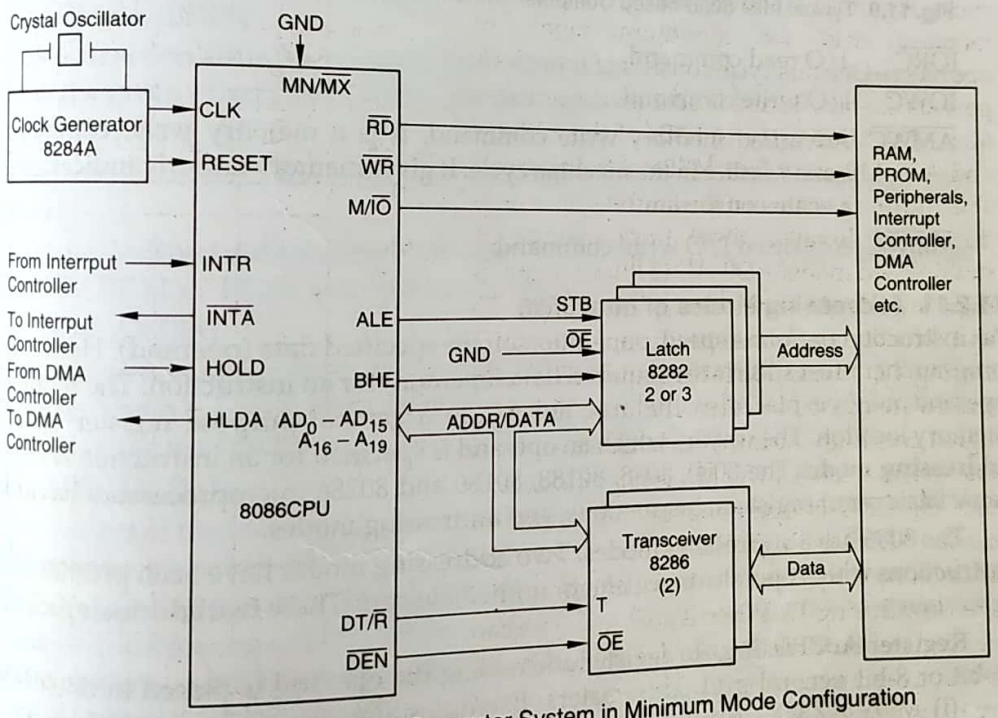

Fig. 11.8 Typical 8086-based Computer System in Minimum Mode Configuration

Figure 11.9 shows a typical 8086 system in the maximum mode configuration. In addition to latches and bus transceivers, a bus controller is also employed for this configuration. The bus controller provides control signals as shown in the figure. The important signals are:

$\overline{MRDC}$ Memory read command.

$\overline{MWTC}$ Memory write command.

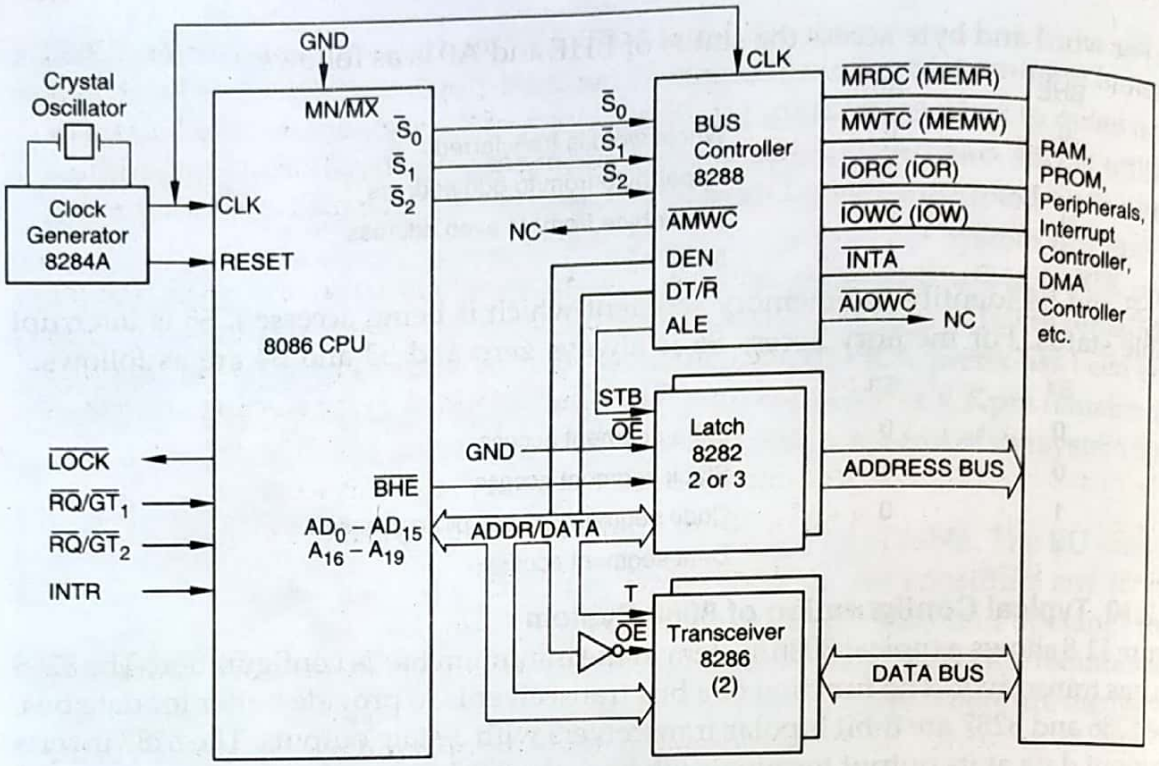

Fig. 11.9 Typical Intel 8086-based Computer System in Maximum Mode Configuration

$\overline{IORC}$ I/O read command.

$\overline{IOWC}$ I/O write command.

$\overline{AMWC}$ Advanced memory write command. It is a memory write command issued earlier in the machine cycle. It gives memory an early indication of a write instruction.

$\overline{AIOWC}$ Advanced I/O write command.

11.2.11 Addressing Modes of Intel 8086

An instruction performs specific operation on the specified data (operand). Hence, the programmer must specify the required data (operand) for an instruction. The required operand may be placed in the accumulator, in a general purpose register or in a memory location. The way by which an operand is specified for an instruction is called **addressing mode**. The 8086, 8088, 80188, 80186 and 80286 microprocessors have the same basic set of registers, instructions, and addressing modes.

The 8086 has 8 addressing modes. Two addressing modes have been provided for instructions which operate on register or immediate data. These two addressing modes are:

Register Addressing. In register addressing the operand is placed in one of the 16-bit or 8-bit general purpose registers. Examples are:

(i) MOV AX,CX (ii) ADD AL,BL (iii) ADD CX,DX

Immediate Addressing. In immediate addressing the operand is specified in the instruction itself. Examples are:

(i) MOV AL,35H (ii) MOV BX,0301H
(iii) MOV [0401],3598H (iv) ADD AX,4836H

The remaining 6 addressing modes specify the location of an operand which is placed in a memory. The memory address of an operand consists of two components: the starting address of the memory segment and an offset. The 16 MSBs of the starting

address of the memory segment resides in the corresponding segment register. When an operand is stored in a memory location, how for the operand's memory location is within a memory segment from the starting address of the segment, is called **offset** or **effective address**. An offset is determined by adding any combination of three address elements: displacement, base and index. The combination depends on the addressing mode of the instruction.

The 8086 uses 20-bit memory address. The segment register gives 16 MSBs of the starting address of the memory segment. The BIU generates 20-bit starting address of the memory segment by shifting the content of segment register left by 4 bits. In other words it puts 4 zeros in 4 LSBs positions. The offset is added to the starting address of the memory segment to get the memory address.

The memory address = Starting address of memory segment + Offset.

Displacement. It is an 8-bit or 16-bit immediate value given in the instruction.

Base. It is the content of the base register, BX or BP.

Index. It is the content of the index register, SI or DI.

The combinations of these three address elements give six memory addressing modes as described below.

Direct Addressing. In direct addressing mode the operand's offset is given in the instruction as an 8-bit or 16-bit displacement element. Examples are:

(i) ADD AL, [0301]

This instruction adds the content of the offset address 0301 to AL. The operand is placed at the given offset (0301) within the data segment DS.

(ii) ADD [0301], AX

This instruction adds the content of AX to the contents of memory locations 0301 and 0302.

Register Indirect Addressing. The operand's offset is placed in any one of the registers BX,BP,SI or DI as specified in the instruction. Examples are:

(i) MOV AX,[BX]

This instruction moves the contents of memory locations addressed by the register BX to the register AX. For example, BX contains 0301, and the content of 0301 is 53H and the content of the next memory location is 95H. The 9553H will move to AX.

(ii) ADD AL, [SI]

The content of the memory location addressed by SI is added to the content of AL, and the result is placed in AL.

Based Addressing. The operand's offset is the sum of an 8-bit or 16-bit displacement and the contents of the base register BX or BP. BX is used as a base register for data segment, and BP is used as a base register for stack segment.

Offset (Effective address) = [BX + 8-bit or 16-bit displacement]

Examples are:

(i) MOV AL, [BX + 05]; an example of an 8-bit displacement.

Suppose, the register BX contains 0301. The offset will be 0301 + 05 = 0306. The content of the memory location 0306 will move to AL.

(ii) MOV AL, [BX + 1346H]; an example of 16-bit displacement

If [BX] = 0301. The offset = 0301 + 1346 = 1647H.

The content of 1647H will move to AL.

Indexed Addressing. The operand's offset is the sum of the content of an index register SI or DI and an 8-bit or 16-bit displacement.

Offset = [SI or DI + 8-bit or 16-bit displacement]

Examples are:

MOV AX, [SI + 05]; an example of 8-bit displacement

MOV AX, [SI + 1528H]; an example of 16-bit displacement

Based Indexed Addressing. The operand's offset is the sum of the content of a base register BX or BP and an index register SI or DI.

Offset = [BX or BP] + [SI or DI]

BX is used as a base register for data segment, and BP is used as a base register for stack segment.

Examples are:

(i) ADD AX,[BX + SI] (ii) MOV CX,[BX + SI]

Based Indexed with Displacement. In this mode of addressing the operand's offset is given by

Offset = [BX or BP] + [SI or DI] + 8-bit or 16-bit displacement

Examples are:

(i) MOV AX, [BX + SI + 05]; an example of 8-bit displacement

(ii) MOV AX, [BX + SI + 1235H]; an example of 16-bit displacement

11.2.12 8086 Instructions with Program Examples

The instructions of 8086 are much more powerful than those of 8085. The 8085 is an accumulator-based processor. For arithmetic and logic operations the address of only one operand is given in the instruction. The other operand is supposed to be present in the accumulator. The result is placed in the accumulator. For example, ADD B (an 8085 instruction) specifies only one operand i.e. the content of register B. The other operand is in the accumulator. The ADD B instruction adds the content of register B to the content of register A, and keeps the sum in register A. This type of processor is also called **one address processor**.

The 8086 is a general purpose registers type processor. For arithmetic and logic operations all general purpose registers : AX, BX, CX and DX, act as an accumulator. Both operands are specified in the instruction. The following are some typical examples:

ADD AX, CX; The content of CX is added to the content of AX and result is placed in AX.

ADD CX, AX; The content of AX is added to the content of CX, and the result is placed in CX.

ADD DX, [Mem]; Add the content of the specified memory to DX, and place the result in DX.

The 8086 processor falls under the category of two address processor. The machine codes of the 8086 instructions can be obtained by hand assembly, assembler or DEBUG program. These topics are discussed in Sections 11.3-11.8. Assembler and DEBUG programs run on computers. One can write assembly language program and can get machine codes on the computer. Those machine codes can be used as required. Hand assembly is difficult and a tedious task. One who wants to learn it can refer to the book: "The 8086 Book" by Russell Rector and George Alexy, Pub: Osborne/McGraw Hill,

1980 and "Advanced Microprocessors and Interfacing", by B. Ram, Tata McGraw Hill, 2001.

Some important instructions are described below:

11.2.12.1 MOV register, register

Data can be transferred from a 16-bit register to a 16-bit register or from an 8-bit register to an 8-bit register. Examples are:

(i) MOV AX, DX ; 16-bit data transfer
(ii) MOV BX, AX ; 16-bit data transfer
(iii) MOV AL, CH ; 8-bit data transfer
(iv) MOV CL,BL ; 8-bit data transfer.

Example 1. Test **MOV CX, BX** on microprocessor kit.

Memory Address	*Machine Codes*	*Mnemonics*	*Comments*
0201	BB,38,85	MOV BX,8538H	; Move 16-bit immediate data 8538H to BX
0204	8B,CB or (89,D9)	MOV CX,BX	; Move contents of BX to CX
0206	CC	INT3	; Interrupt

Result: Examine BX and CX registers
BX : 8538H
CX : 8538H

Note: BB is opcode to move 16-bit immediate data to BX.
8B,CB or (89,D9) are opcodes to move content of BX to CX.
CC is code for interrupt INT3.

Example 2. Test **MOV BL,CH** on microprocessor kit.

Memory Address	*Machine Codes*	*Mnemonics*	*Comments*
0201	B5, 53	MOV CH, 53H	; Move 8-bit immediate data 53H to CH
0204	88, EB or (8A, DD)	MOV BL, CH	; Move contents of CH to BL
0205	CC	INT3	; Interrupt

Result: Examine CH and BL registers.
CH : 53H
BL : 53H

11.2.12.2 MOV Register, Memory

16-bit or 8-bit data can be moved to a 16-bit or 8-bit register. Examples are:

(i) MOV CX,[BX] ; 16-bit data transfer. Memory locations are specified by BX. Since the register CX is a 16-bit register, data from two consecutive memory locations specified by BX will move to CX. The register BX will contain the address of the 1st memory location. If BX contains the memory address 0301. The data from 0301 and the next memory location 0302 will move to CX.
(ii) MOV AL,[BX] ; 8-bit data transfer. If BX contains 0301, data will move from 0301 to AL.
(iii) MOV AX,[SI] ; 16-bit data transfer.
(iv) MOV AX,[BX + DISP] ; DISP is an 8-bit or 16-bit number called displacement. Suppose, BX contains 0301 and DISP is 05, then data will move from 0306 (0301 + 5) and 0307 to AX.

(v) MOV CX,[Memory address]
or MOV CX,[0301] ; An example of direct addressing. The contents of 0301 a
0302 will move to CX.

Example 1. Test **MOV CX,[BX]** on microprocessor kit.

This is an example to move content of a memory location addressed by BX regis
to CX.

Memory Address	*Machine Codes*	*Mnemonics*	*Comments*
0201	BB,01,03	MOV BX,0301H	; Move memory address to BX
0204	8B,0F	MOV CX, [BX]	; Move data from memory addressed by B to register CX
0206	CC	INT3	; Interrupt

Data: load data into specified memory locations.
0301 - 25H
0302 - 98H
Result: Examine register CX.
CX - 9825H

Example 2. MOV CX,[0301] ; Move 16-bit data from memory locations to CX.

Memory Address	*Machine Codes*	*Mnemonics*	*Comments*
0201	8B,0E,01,03	MOV CX,[0301]	; Move memory address to BX
0205	CC	INT3	

Data:
0301 - 53H
0302 - 85H

Result: CX - 8553H

Note: 8B, 0E are opcodes to move data from memory to CX using direct addressing. 01 is LSB memory address and 03 MSB of the memory address.

11.2.12.3 MOV Memory, Register

16-bit or 8-bit data is moved from 16-bit or 8-bit register to memory location(s
Examples are:

(i) MOV [0301],AX ; 16-bit data transfer
The contents of AX will move to the memory locations 0301 and 0302.
(ii) MOV [0301],CX ; 16-bit data transfer
(iii) MOV [0301], AL ; 8-bit data transfer.
The content of AL will move to the memory location 0301
(iv) MOV [BX], CX ; 16-bit data transfer.
The contents of CX will move to memory location addressed by BX.
(v) MOV [BX + DISP], CX

Example 1. MOV [BX],CX ; move content of register CX to memory locatio
addressed by BX

Memory Address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
0201	B9, 83, 45	MOV	CX, 4583h	; 16-bit data in CX

```
0204    BB,01,04    MOV    BX, 401H    ; Memory address in BX
0207    89, 0F      MOV    [BX], CX    ; Move data from CX to memory
                                         location addressed by BX
0209    CC          INT3                ; Interrupt
```

Result :
0401 - 83H
0402 - 45H

Example 2. **MOV [0301], CX** ; Move CX to memory location specified in the instruction.

Memory Address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
0201	B9, 35, 14	MOV	CX, 1435H	; 16-bit data in CX
0204	89, 0E, 01, 03	MOV	[0301], CX	; Move CX to [0301]
0208	CC	INT3		

Result:
0301 - 35H
0302 - 14H

Example 3. **MOV [0301], AL**

Memory Address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
0201	B0, 38	MOV	AL, 38H	; Move 38H to AL
0203	A2, 01, 03	MOV	[0301], AL	; Move AL to [0301]
0206	CC	INT3		; Interrupt

Result:
0301 - 38H

11.2.12.4 MOV register, 8-bit or 16-bit immediate data

8 or 6-bit data can be moved to a register using immediate addressing.

Examples are:

(i) MOV AL, 8-bit data
(ii) MOV BX, 16-bit data
(iii) MOV SI, 16-bit data
(iv) MOV AX, 16-bit data

Example 1. **MOV AX, 1593H**
MOV BX, 5836H
MOV SI, 1346H

Memory Address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
0201	B8,93,15	MOV	AX, 1593H	; Move 1593H to AX
0204	BB,36,58	MOV	BX, 5836H	; Move 5836H to BX
0207	BE,46,13	MOV	SI, 1346H	; Move 1346 to SI
020A	CC	INT3		; Interrupt

Result: Examine AX,BX and SI registers
AX - 1593H
BX - 5836H
SI - 1346H

Example 2. MOV AL, 95H

Memory Address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
0201	B0,95H	MOV	AL, 95H	; Move 95H to AL
0203	CC	INT3		; Interrupt

Result: Examine AL
AL - 35H

11.2.12.5 MOV memory, 8-bit or 16-bit immediate data

Examples are:

(i) MOV [0301], 67H ; 67 will move to memory location 0301.
(ii) MOV [0301], 1584H ; 1584 will move to memory locations 0301 and 0302.
(iii) MOV [BX], 94H ; 94H will move to the memory location addressed by BX.
(iv) MOV [BX], 1439H ; 1439 will move to two consecutive memory locations.
(v) MOV [BX + DISP], 58H ; If BX contains 0301 and DISP is 08, 58 will move to the memory location 0309 (0301 + 8).
(vi) MOV [BX + DISP], 1583 ; if [BX] = 0301, and DISP = 05, 1583 will move to 0306 and 0307.
(vii) MOV [BX + SI], 53H
(viii) MOV [BX + SI], 1593H
(ix) MOV [BX + SI + DISP], 48H
(x) MOV [BX + SI + DISP], 1439H
If [BX] = 0301, [SI] = 1548, DISP = 04,
[BX + SI + DISP] = 184D

The 16-bit data 1439 will move to memory locations 184D and 184E.

Example 1. MOV [0301], 8435H

Memory Address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
0201	C7,06,01,03,35,84	MOV	[0301], 8435H	; Move 8435 to memory locations 0301 and 0302.
0207	CC	INT3		; Interrupt

Result:
0301 - 35H
0302 - 84H

Note: C7,06 are opcodes to move immediate 16-bit data to memory using direct addressing. 01 and 03 are LSB and MSB of memory address respectively. 35 and 84 are LSB and MSB of 16-bit immediate data respectively.

Example 2. MOV [BX], 8354H

Memory Address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
0201	Bb,01,03	MOV	BX,0301H	; Memory address in BX
0204	C7,07,54,83	MOV	[BX],8354H	; Move 8354H to memory locations addressed By BX
0208	CC	INT3		; Interrupt

Result:
0301 - 54H
0302 - 83H

Result: 54H + 35H = 89H
AL - 89H
Note: 04 is the opcode to add 8-bit immediate data to AL.

Example 3. ADD CL, 8-bit data

Opcode : 80,C1, 8-bit immediate data.

80,C1 are opcodes for above instruction. Note that opcodes are of 2 bytes to move 8-bit or 16-bit immediate data to registers other than accumulator. Write assembly language program. Opcode for MOV CL, 8-bit data is B1, 8-bit data.

Example 4. ADD CX, 16-bit data

Opcodes for the above instruction are : 81,C1.

Write assembly language program. Opcodes for MOV CX, 16-bit data are: B9, 16-bit data.

11.2.12.7 ADD [memory], 8-bit or 16-bit data

Example 1. ADD [BX], 54H

PROGRAM

```
0201    BB,01,03    MOV    BX,0301H    ; memory address in BX
0204    80,07,54    ADD    [BX],54H    ; Add 54H to [BX]
0208    CC          INT3                ; Interrupt
```

DATA:
Result: 0301 - 36H
0301 - 8AH

Note: 80,07 are opcodes to add 8-bit immediate data to memory location addressed by BX.

Example 2. ADD [BX], 16-bit data

PROGRAM

```
0201    BB,01,03    MOV    BX,0301H    ; Memory address in BX
0204    81,07,86,45 ADD    [BX], 4586H ; Add 4586H to [BX]
0208    CC          INT3                ; Interrupt
```

DATA: **Result:** 3597H + 4586H = 7BIDH
0301 - 97H 0301 - 1D
0302 - 35H 0302 - 7B

Note: 81,07 are opcodes to add 16-bit data to memory addressed by BX.

Example 3. ADD [BX + SI], 16-bit or 8-bit data

Machine Codes for this instruction are 81,00

Take ADD [BX + SI], 2345H

PROGRAM

```
0201    BB,01,03    MOV    BX, 0301H    ; Get memory address in BX
0204    BE,42,13    MOV    SI, 1342H    ; Index number in SI
0207    81,00,45,23 ADD    [BX+SI],2345H ; Add 2345H to [BX +SI]
020B    CC          INT3                ; Interrupt
```

Memory locations 0301 + 1342 = 1643 and the next location

Data: **Result:** 5187 + 2345 = 74CCH
1643 - 87H 1643 - CCH
1644 - 51H 1644 - 74H

Example 4. ADD [BX + SI + 16-bit DISP], 16-bit data

Machine Codes for this instruction are 81,80.

PROGRAM

```
0201    BB,01,03    MOV BX, 0301        ; Memory address in BX
0204    BE,42,13    MOV SI,1342         ; Index in SI
0207    81,80,52,14,84,35 ADD [BX + SI + 1452],3584H
020D    CC          INT3
```

Memory address = 0301 + 1342 + 1452 = 2A95 and 2A96.

DATA: **Result:**
2A95 - 15H 2A95 - 99H
2A96 - 37H 2A96 - 6CH

11.2.12.8 ADD register, [memory]

Example 1. Add AX, [BX + DISP]

Machine codes for the above instruction with 8-bit DISP are 03,47.

The value of 8-bit displacement is 08H.

PROGRAM

```
0201    BB,01,03    MOV    BX,0301H    ; Get memory address in BX
0204    B8,95,28    MOV    AX,2895H    ; 6-bit data in AX
0207    03,47,08    ADD    AX,[BX + 08] ; Add AX to [BX + 08]
020A    CC          INT3                ; Interrupt
```

DATA: **Result :** 2895 + 4354H in AX
0309 - 54H AX - 6BE9H
030A - 43H

Example 2. ADD CX, [0301]

Machine Codes = 03,0E, LSB of memory address, MSB of memory address.

PROGRAM

```
0201    B9,34,96    MOV    CX,9634H    ; 16-bit data in CX
0204    03,0E,01,03 ADD    CX,[0301]   ; Add CX to memory
0208    CC          INT3                ; Interrupt
```

DATA: **Result:** (9634 + 5738 = ED6C hex.)
0301 - 38H CX - ED6CH
0302 - 57H

11.2.12.9 ADD [Mem.], register

Example 1. ADD [BX], CL

Machine Codes = 00,0F

PROGRAM

```
0201    BB,01,03    MOV    BX,0301H    ; Memory address in BX
0204    B1,85       MOV    CL,85H      ; 8-bit data in CL
0206    00,0F       ADD    [BX],C      ; Add [BX] and CL
0208    CC          INT3                ; Interrupt
```

DATA: **Result:**
0301 - 49H 0301 - CE

Example 2. ADD [BX], CX

Machine Codes = 01,0F

PROGRAM

```
0201    BB,01,03    MOV    BX,0301H    ; Memory address in BX
0204    B9,78,84    MOV    CX,8478H    ; 16-bit data in CX
```

```
0207    01,0F    ADD     [BX],CX    ; Add [BX] and CX
0209    CC       INT3               ; Interrupt
```

DATA: **Result:** 6743H + 8478H = EBBB hex.
0301 - 43H 0301 - BBH
0302 - 67H 0302 - EBH

11.2.12.10 ADD Register, Register

Example 1. ADD CL, DL

Codes = 00,D1

PROGRAM

```
0201    B1,90    MOV     CL,90H     ; 8-bit data in CL
0202    B2,45    MOV     DL,45H     ; 8-bit data in DL
0205    00,D1    ADD     CL,DL      ; Add CL and DL
0207    CC       INT3               ; Interrupt
```

Result : 90 + 45 = D5H
CL - D5

Example 2. ADD CX,DX

Codes = 01,D1 or (03, CA)

PROGRAM

```
0201    B9,57,93          MOV     CX,9357H    ; Get 16-bit data in CX
0204    BA,63,24          MOV     DX,2463H    ; 16-bit data in DX
0207    01,D1 or (03, CA) ADD     CX,DX       ; Add CX and DX
0209    CC                INT3                ; Interrupt
```

Result : 9357H + 2463H = B7BA
CX - B7BAH

11.2.12.11 MUL mem/reg

Multiply AL or AX register by the specified register or specified memory location(s).

This instruction is used for the multiplication of two unsigned numbers. For 8-bit multiplication one operand is placed in AL and the other operand is placed in any other 8-bit register or a memory location. The result is placed in AX register. For 16-bit multiplication one operand is placed in AX register and the other operand is placed in any other 16-bit register or two consecutive memory locations. The result is placed in DX and AX. The low-order 16-bits are placed in AX and the high-order 16-bits in DX. In either case (8-bit multiplication or 16-bit multiplication), when the high-order half of the result is zero, the overflow flag (OF) and the carry flag (CF) are set to 0, otherwise they are set to 1. The remaining condition flags are undefined.

Example 1. MUL CL

Codes for this instruction = F6,E1
Take [AL] = 84H and [CL] = 56H

PROGRAM

```
0201    B0,84    MOV     AL,84H     ; 8-bit data in AL
0203    B1,56    MOL     CL,56H     ; 8-bit data in CL
0205    F6,E1    MUL     CL         ; Multiply AL by CL
0207    CC       INT3               ; Interrupt
```

Result:
AX - 2C58H

Example 2. MUL CX

Codes for this instruction = F7, E1

Take [AX] = 9273H, CX = 4835H

PROGRAM

0201	B8,73,92	MOV	AX,9273H
0204	B9,35,48	MOV	CX,4835H
0207	F7,E1	MUL	CX
0209	CC	INT3	

Result:

[AX] - A9CFH

[DX] - 294EH

Example 3. MUL [memory]

Take MUL [BX + 8-bit DISP]

Take DISP = 04, [BX] = 0301

Codes = F7, 67

Multiply 5348H by 8654H

PROGRAM

0201	B8,48,53	MOV	AX,5348H	; 16-bit data in AX
0204	BB,01,03	MOV	BX,0301H	; Memory address in BX
0207	F7,67,04	MUL	[BX+04]	; Multiply AX by [BX + 04]
020A	CC	INT3		; Interrupt

DATA: **Result :**

0305 - 54H AX - 03A0H

0306 - 86H DX - 2BB3H

11.2.12.12 IMUL mem/reg

Multiply AL or AX register by the specified register or specified memory location(s).

This instruction is used for the multiplication of two signed numbers. For 8-bit multiplication one operand is placed in AL and the other operand is placed in any other 8-bit register or a memory location. The result is placed in AX. For 16-bit multiplication one operand is placed in AX and the other operand is placed in any other 16-bit register or two consecutive memory locations. The result is placed in AX and DX. In either case (8-bit or 16-bit multiplication), when the high-order half of the result is the sign extension of the low-order half of the result, the overflow flag (OF) and the carry flag (CF) are set to zero, otherwise they are set to 1. The remaining condition flags are undefined.

Example. IMUL CX

Opcodes are = F7, E9

Take multiplication of -7593H and 6845H

In case of 16-bit processor 15 bits represent number and one bit (MSB) represents its sign.

2's complement representation of -7593H = 8A6D

Put [AX] = 8A6DH and [CX] = 6845H.

PROGRAM

0201	B8,6D,8A	MOV	AX,8A6DH	; 16-bit signed number in AX
0204	B9,45,68	MOV	CX,6845H	; 16-bit signed number in CX
0207	F7,E9	IMUL	CX	; Multiply AX by CX
0209	CC	INT3		; Interrupt.

Result:
[AX] = 9761H
[DX] = D01CH

Product = D01C9761 hex. As the result is negative, it is in 2's complement. To get correct answer take 2's complement of D01C9761 and put a negative sign before it. It will be equal to -2FE3689F.

11.2.12.13 DIV mem/reg

Divide AX by an 8-bit number or DX : AX by 16-bit number.

This is used to divide a 32-bit unsigned number by a 16-bit unsigned number or to divide a 16-bit unsigned number by an 8-bit unsigned number. The 32-bit dividend is placed in DX and AX. The 16-bit divisor is placed in a 16-bit register or memory locations. The 16-bit quotient is placed in the AX register, and the 16-bit remainder is placed in DX register. It is called 16- bit operation. If the quotient is greater than FFFF, a divide-by-zero (Type 0) interrupt is generated. All conditional flags remain undefined.

If the dividend is of 16-bit, it is placed in AX, and 8-bit divisor is placed in an 8-bit register or a memory location. The 8-bit quotient is placed in AL, and the 8-bit remainder is placed in AH. If the quotient is greater than FF, a divide-by-zero (Type 0) interrupt is generated. This operation is called an 8-bit operation.

Example 1. DIV CL

This instruction will divide AX by CL

Machine Codes = F6,F1

Take division of 2C5B by 56H.

PROGRAM

```
0201    B8,5B,2C    MOV     AX, 2C5B    ; Dividend in AX
0204    B1,56       MOV     CL, 56H     ; Divisor in CL
0206    F6,F1       DIV     CL          ; Divide AX by CL.
0208    CC          INT3                ; Interrupt
```

Result:
[AL] = 84H (Quotient)
[AH] = 03H (Remainder)

Example 2. DIV CX

This instruction will divide DX:AX by CX.

Machine Codes = F7,F1

Take division of 294EA9DF by 4835H.

PROGRAM

```
0201    B8,DF,A9    MOV     AX,A9DF     ; Low-order 16-bits of dividend in AX
0204    BA,4E,29    MOV     DX,294E     ; High-order 16-bits of dividend in DX
0207    B9,35,48    MOV     CX,4835     ; Divisor in CX
020A    F7, F1      DIV     CX          ; Divide DX : AX by CX.
020C    CC          INT3                ; Interrupt
```

Result:
[AX] = 9273H (Quotient)
[DX] = 0010H (Remainder)

Example 3. DIV [BX]

This instruction will divide a 32-bit or 16-bit number by a 16-bit or 8-bit number. The divisor is in memory location(s).

(i) Take the division of 15F6H by 68H

Machine Codes for 8-bit operation, when divisor is in a memory location = F6,37.

PROGRAM

```
0201   B8,F6,15    MOV   AX,15F6H   ; 16-bit dividend in AX
0204   BB,01,03    MOV   BX,0301H   ; Memory address in BX
0207   F6,37       DIV   [BX]       ; Divide AX by [BX]
0209   CC          INT3             ; Interrupt
```

DATA: **Result:**

0301 - 68H [AL] = 36H (Quotient)
AH] = 06H (Remainder)

(ii) Divide 294EA9D8H by 4835H

Machine Codes for 16-bit operation = F7,37, when divisor is in memory.

PROGRAM

```
0201   B8,D8,A9    MOV   AX, A9D8H  ; Low-order 16 bits of dividend in AX
0204   BA,4E,29    MOV   DX, 294EH  ; High-order 16 bits of dividend in DX
0207   BB,01,03    MOV   BX ,0301H  ; Memory address in BX
020A   F7,37       DIV   [BX]       ; Divide DX : AX by [BX]
020C   CC          INT3             ; Interrupt
```

DATA: **Result:**

0301 - 35H (LSB of divisor) [AX] = 9273H (Quotient)
0302 - 48H (MSB of divisor) [DX] = 0009H (Remainder)

11.2.12.14 IDIV mem/reg

Divide AX by 8-bit number and DX:AX by 16-bit number.

This instruction is used for signed numbers.

Example IDIV CL

This instruction will divide AX by CL.

Opcodes = F6,F9

Take division of 26FA by – 56H.

2's complement of 56 is AA.

PROGRAM

```
0201   B8,FA,26    MOV   AX, 26FA   ; Dividend in AX
0204   B1,AA       MOV   CL, AAH    ; Divisor in CL
0206   F6,F9       IDIV  CL         ; Divide AX by CL
0208   CC          INT3             ; Interrupt
```

Result:

[AL] = 8C (Quotient), –ve
[AH] = 02 (Remainder)
2's complement of 8C is 74. So the result will be –74.

11.2.12.15 Load Instructions

(i) LODSB Instruction

The LODSB Instruction loads the content of the memory location addressed by SI register into AL rgister. The machine code of LODSB instruction is AC. After the execution of this instruction, the SI register is incremented or decremented by one depending on the value of the DF flag. If the value of the DF flag is zero, the SI register is incremented by one. If the DF flag is one, the SI register is decremented by one.

Example. Load AL register with the content of memory location 0201H.

Example. Store the content of AX register into memory locations 0201 H and 0202 H.

PROGRAM

Memory Address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
0101	BF,01,02	MOV	DI, 0201H	; Memory address in DI
0104	B8,84,95	MOV	AX, 9584H	; data in AX
0107	AB	STOSW		; store [AX] into [DI]
0108	CC	INT3		; Interrupt program

Result:

0201 - 84H

0202 - 95H

11.3 CLASSIFICATION OF 8086 INSTRUCTIONS

Instructions are classified on the basis of function they perform. They are categorized into the following main types:

(i) Data Transfer Instructions

All the instructions which perform data movement come under this category. They are move, load, store, exchange, input, output, push and pop instructions. The source of data may be a register, memory location(s), port etc. The destination may be a register, memory location or port. Memory to memory or Port to port data are not transferred by single instruction. Examples of data transfer instructions are : MOV, XCHG, PUSH, POP, LDS, LEA, IN, OUT etc

(ii) Arithmetic Instructions

Instructions of this group perform addition, subtraction, multiplication, division, increment, decrement, comparison, ASCII and decimal adjustment etc. Examples are: ADD, SUB, INC, MUL, DIV, CMP, DAA etc.

(iii) Logical Instructions

Instructions of this group perform logical AND, OR, XOR, NOT and TEST operations. Rotate and shift instructions may be included in this group. Examples of rotate and shift instructions are : RCL, RCR, ROL, ROR, SAL, SAR, SHL, SHR etc.

(iv) Program Execution Transfer (or Branch) Instructions

Instructions of this group transfer program execution from the normal sequence of instructions to the specified destination or target. After the execution of such instructions, the processors starts fetching instructions from some new address, rather than continuing in sequence. Examples of instructions of this group are: JMP, JC, JB, JNB, JZ, JNZ, CALL, RET etc.

Iteration control instructions such as LOOP, LOOPE, LOOPZ, LOOPNE, LOOPNZ etc can also be included in this category of instructions

Interrupt instructions such as INT, INTO and IRET can also be included in this category of instructions

Processor Control instructions

Instructions of this group are related to flag manipulation and machine control . Examples are: CLC, CLD, STC,STD, HLT, NOP, LOCK, etc

String instructions

String is series of bytes or series of words stored in the sequential memory locations. The 8086 provides some instructions which handle string operations such as string movement, comparison, scan, load and store. Examples are: MOVS / MOVSB / MOVSW , CMPS / CMPSB / CMPSW, SCAS/ SCAB / SCASW, LODS / LODSB / LOSDW, STOS / STOSB / STOSW etc.

11.4 BINARY ADDRESS OF 8086 REGISTERS

8-Bit Registers	*16-Bit Registers*	*3-Bit Binary Address*
AL	AX	000
CL	CX	001
DL	DX	010
BL	BX	011
AH	SP	100
CH	BP	101
BH	SI	110
DH	DI	111

11.5 DESCRIPTION OF 8086 INSTRUCTIONS

At some places machine codes are given and at some places they are not given because in those cases machine codes are to be calculated for different conditions. The calculation of machine code is beyond the scope of this book. One can refer to author's book "Advanced Microprocessors", TMH. If assembler is used, it gives machine codes of all the instructions of the program which is being run on a computer having assembler.

11.5.1 Data Transfer Instructions

The above category of Instructions has been introduced in the Section 11.3 (i). The following instructions come under this category:

11.5.1.1 MOV Instructions

Examples of MOV instructions are given in the Section 11.2.12.1 to 11.2.12.5. Following are the types of MOV instruction:

(i) MOV reg, data

(Move 8-Bit or 16-Bit Immediate Data to an 8-Bit or 16-Bit Specified Register)

Object code = 1011 0 r r r for 8-bit operation

= 1011 1 r r r for 16-bit operation

r r r = 3-bit address of 8-bit or 16-bit register. Flags affected: none.

(ii) MOV mem/reg, data. Flags affected : none

Move 8-bit or 16-bit immediate data to the 8-bit or 16-bit specified register or memory location(s).

(iii) MOV ac, mem. Flags affected : none

Move 8-bit or 16-bit data from the specified memory location(s) to accumulator *i.e.* AL or AX register.

Object code = A0 for 8-bit operation

= A1 for 16-bit operation

(iv) **MOV mem, ac.** Flags affected : none

Move the contents of accumulator (AL or AX register) to the specified memory location(s).

Object code = A2, [Effective address] for 8-bit operation
= A3, [Effective address] for 16-bit operation

(v) **MOV reg/mem, mem/reg.** Flags affected : none

Following operations are allowed:

(a) MOV reg, reg
(b) MOV mem , reg
(c) MOV reg , mem

Memory to memory move is not allowed.

(vi) **MOV segreg, mem/reg.** Flags affected: none

Move the content of the specified 16-bit register or memory locations to the specified segment register.

(vii) **MOV mem / reg, segreg.** Flags affected: none

Move the content of the specified segment register to the 16-bit specified register or memory locations.

11.5.1.2 Load / Store / Exchange etc Instructions

(i) LDS **reg, mem.** Flags affected: none

This instruction loads a word (two bytes) from the specified memory locations into the specified register. Furthermore, it also loads a word from the next two memory locations into DS register.

Example: LDS SI, [0200]

(ii) **LES reg, mem.** Flags affected : none

This instruction loads a word (two bytes) from the specified memory locations into the specified register. Furthermore, it also loads a word from the next two memory locations into ES register.

Example: LES DI, [BX]

(iii) **LEA reg, mem.** Flags affected : none

This instruction loads offset address into the specified register. It can also determine the offset address of a variable memory location specified in the instruction to load it into the specified register.

Examples are:

(a) LEA CX , [BX + SI + 0532H]
(b) LEA BX , TEMPERATURE

(iv) **LAHF.** Machine code: 9F. Flags affected : none

This instruction loads low-order 8-bits of the flag register into AH register. Flags loaded into AH register are: SF , ZF , AF , PF and CF.

(v) **SAHF.** Code : 9E. Flags affected : none

This instruction stores the content of AH register into low-order bits of the Flags register.

(vi) XLAT/XLATB. Code: D7. Flags affected : none

This instruction reads a byte from the lookup table. The BX register is loaded with the starting address of the lookup table. An 8-bit data element is placed in AL. When XLAT is executed , the [AL] and [BX] are added to give the desired offset address of the look table. The content of the memory location having the offset address [AL] + [BX] is moved to AL register. This instruction is used for code conversion. An element of one code is placed in AL, corresponding other code is in the lookup table.

(vii) XCHG reg. Flag affected : None

This instruction exchanges the contents of the 16-bit specified register with the contents of AX register.

Object code = 10010 r r r

r r r = 3-bit address for a 16-bit register

(viii) XCHG reg, mem/reg. Flag affected : None

This instruction exchanges the contents of an 8-bit or 16-bit specified register with the content of an 8-bit or 16-bit specified register or memory location(s).

11.5.1.3 PUSH Instructions

(i) PUSH reg. Flag affected : None

This instruction pushes (sends) the content of a specified 16-bit register onto the top of the stack.

Machine code = 0101 0 r r r

r r r = 3-bit address for a 16-bit register

(ii) PUSH mem/reg. Flag affected : None

This instruction pushes (writes) the contents of a specified 16-bit register or memory location(s) onto the top of the stack.

(iii) PUSH segreg. Flag affected : None

This instruction pushes (moves) the content of a specified segment register onto the top of the stack.

(iv) PUSHF. Code : 9C. Flag affected : None

This instruction pushes the content of the Flags register onto the top of the stack.

11.5.1.4 POP Instructions

(i) POP reg. No flags are affected

This instruction pops (reads) two bytes from the top of the stack and keeps them in a 16-bit specified register. This instruction does not allow to pop data into segment registers. POP segreg instruction is used for segment registers.

Machine code = 0101 1r r r

r r r = 3-bit address of a 16-bit register

(ii) POP mem/reg. Flags affected: None

This instruction is used to pop (read) two bytes from the top of the stack and to place them in a 16-bit specified register or memory locations.

(iii) POP seg reg. Flags affected: None

This instruction pops (reads) two bytes from the top of the stack and keeps them in a specified segment register. POP CS is not allowed.

Machine code = 000 r r 111

11.12 INTEL 8088

The 8088 is an 8/16-bit microprocessor. Its internal architecture is of 16 bit but it has only 8 data lines. It was widely used in PC/XT personal computer. As it has only 8 data lines it employs 8-bit I/O devices which are cheaper as compared to 16-bit I/O devices. So personal computers using 8088 CPU were cheaper as compared to personal computers employing 80286 CPU.

Figure 11.10 shows the pin diagram of 8088. It has 8-bit data bus AD0-AD7 multiplexed with address lines. It has 20 address lines and it can directly address up to 1 MB of memory. It contains 16-bit registers and 16-bit wide internal data path. It is a 40 pin IC and operates at 5 Vdc. Its clock is 5-8 MHz. The register set, instructions and addressing modes of 8088 are same as those of 8086. Its CHMOS version is 80C88 which operates at 8 MHz. 8087 math coprocessor is used with 8088 CPU. Figure 11.11 shows a typical 8088 based computer system. The queue length in 8088 is of 4 bytes only whereas that in 8086 is of 6 bytes.

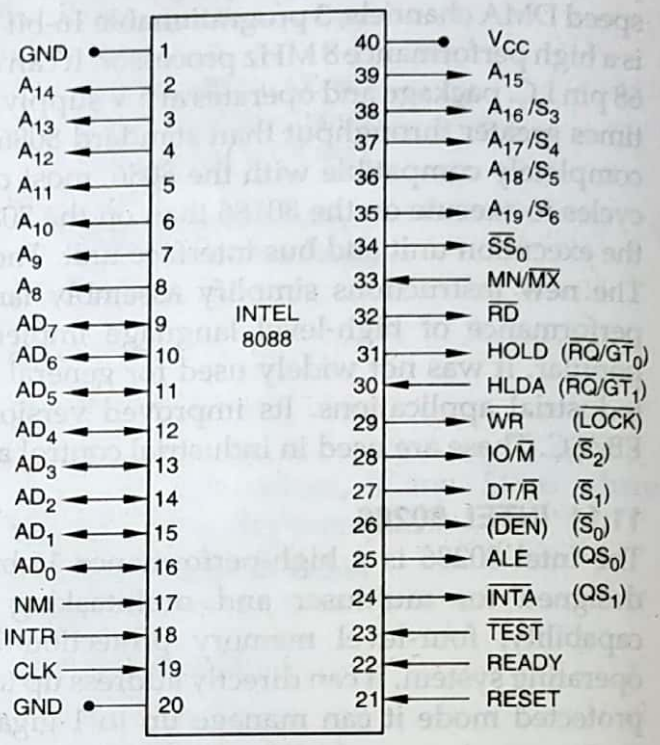

Fig. 11.10 Pin Configuration of Intel 8088

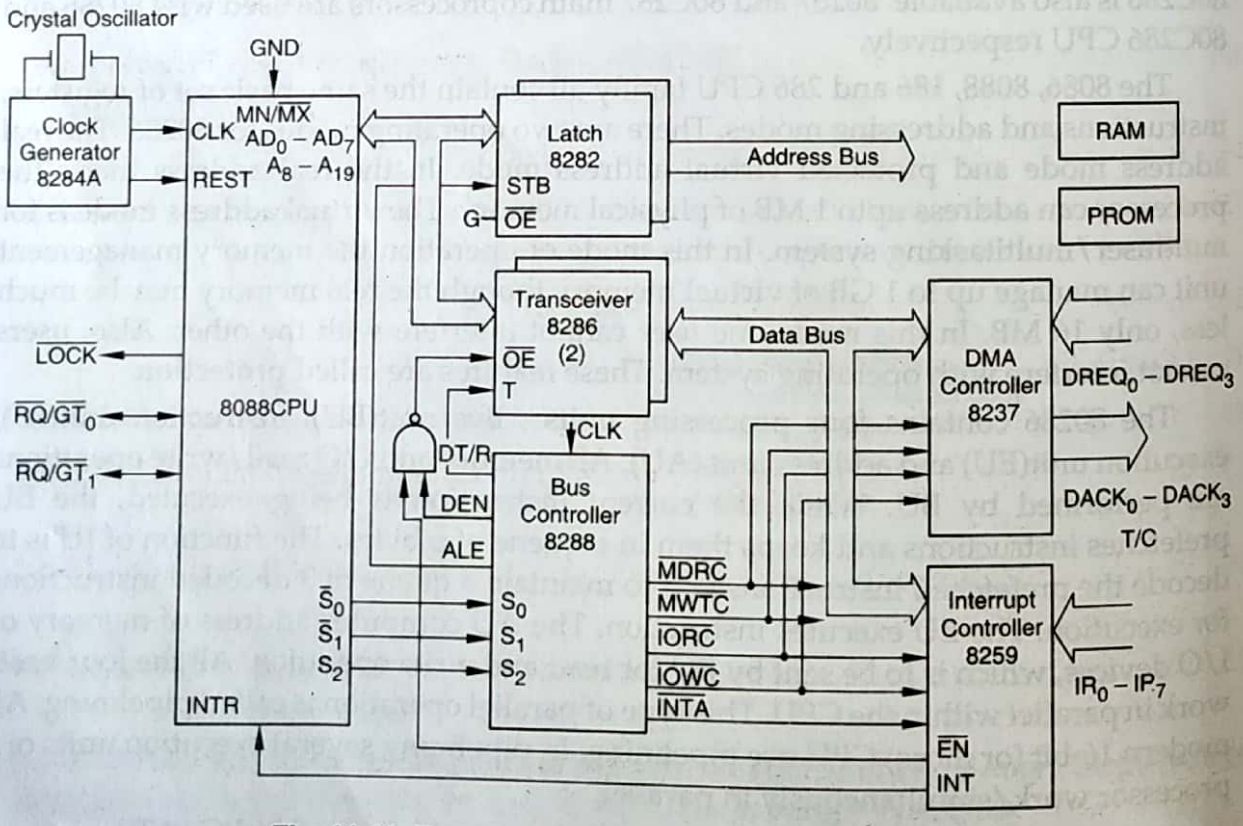

Fig. 11.11 Typical 8088 CPU-based computer system

11.12.1 Intel 80188

The Intel 80188 is a highly integrated 8-bit microprocessor. It contains 8088-2 CPU, programmable interrupt controller, clock generator, 3 programmable 16-bit timer, programmable memory and peripheral chip-select logic, local bus generator, 2 high speed DMA channels, etc. It can address directly upto 1 M bytes of memory. It has 68 pins and operates at 5V supply. It was not popular for general purpose microcomputer. Its improved versions are: 80C188, 80C188XL, 80C188EA/ EB/EC. These are used for industrial control applications.

PROBLEMS

1. What are important signals of Intel 8086? Discuss them in brief.
2. How many operating modes does 8086 have? Discuss them in brief.
3. How many functional units does 8086 contain? Discuss them in brief.
4. What is pipelining ? How is it achieved in 8086? What are its advantages?
5. Discuss the register organization of 8086. Explain the function of each register.
6. In addition to the function of a general purpose register what other functions are performed by the register BX, BP and CX?
7. What is the function of a segment register in 8086?
8. Discuss the function of instruction pointer and stack pointer in 8086?
9. How many status flags does 8086 have? Discuss the role of each flag.
10. What are conditional and control flags in 8086?
11. How many interrupt lines does 8086 have?
12. What do you understand by hardware interrupt and software interrupt?
13. Discuss the interrupt system of Intel 8086. What is interrupt pointer? What is 'type' of an interrupt?
14. What are LOCK and $\overline{\text{LOCK}}$? Discuss their roles.
15. Discuss the various addressing modes of 8086. What are displacement, base and index? What is an effective address or offset?
16. In what way does Intel 8088 differ from 8086?
17. Discuss the important features of 80286.
18. What are Intel 80186 and 80188? For what types of applications are they suitable?
19. Write an 8086 assembly language program to find the smallest 8-bit number in an 8-bit data array.
20. Write an 8086 assembly language program to find sum of a series of 16-bit numbers; sum: 16-bit.
21. Write an 8086 assembly language program to find sum of a series of 16-bit numbers; sum : 32-bit.
22. Write an 8086 assembly language program to move (or relocate) block of (i) 16-bit (*i.e.* word), (ii) 8-bit (*i.e.* byte) data.
23. What are assember directives? Explain.