UNIT-I

The 8086 Microprocessor

Introduction to 8086 – Microprocessor architecture – Addressing modes - Instruction set and assembler directives – Assembly language programming – Modular Programming - Linking and Relocation - Stacks - Procedures – Macros – Interrupts and interrupt service routines – Byte and String Manipulation.

1. Introduction to 8086

8086-HARDWARE ARCHITECTURE

AUQ: Explain the features of 8086 microprocessor. (May 2011, 8 Marks)

The features of 8086 are:

- The 8086 is a 16 bit processor
- The 8086 has a 16 bit data bus
- The 8086 has a 20 bit address bus
- Direct addressing capability 1M byte of memory(2^{20})
- It provides fourteen 16 bit register
- 24 operand addressing modes
- Bit, byte, word and block operations.
- 8 and 16 bit signed and unsigned arithmetic operations including multiply and divide
- Four general purpose 16 bit registers: AX, BX, CX, DX
- Two pointer group registers: stack pointer (SP), Base pointer(BP)
- Two index group registers: source index (SI), destination index (DI)
- Four segment registers: code segment (CS), Data segment (DS), Stack segment (SS), Extra segment(ES)
- 6 Status flag and 3 control flags.
- Memory is byte addressable- each address stores an 8 bit value.
- Address can be up to 32 bit long, resulting in 4GBof memory.
- Range of clock rates: 5MHZ for8086, 8MHZ for8086-2, 10MHZ for8086-1
- Multibus system compatible interface
- Available in 40 pin plastic package and lead cerdip.

2. 8086 Microprocessor Architecture:

AUQ: Explain the internal architecture of 8086 microprocessor.(Dec-2003,04,06,08,11,12,13, May-2003,05,07,08,10,11,15, May 2016, Dec 2016, May 2017)

The internal functions of 8086 processor are partitioned logically into two processing units.

The 8086 CPU is divided into two independent functional parts,

1. **Bus Interface Unit (BIU)**
2. **Execution Unit (EU)**

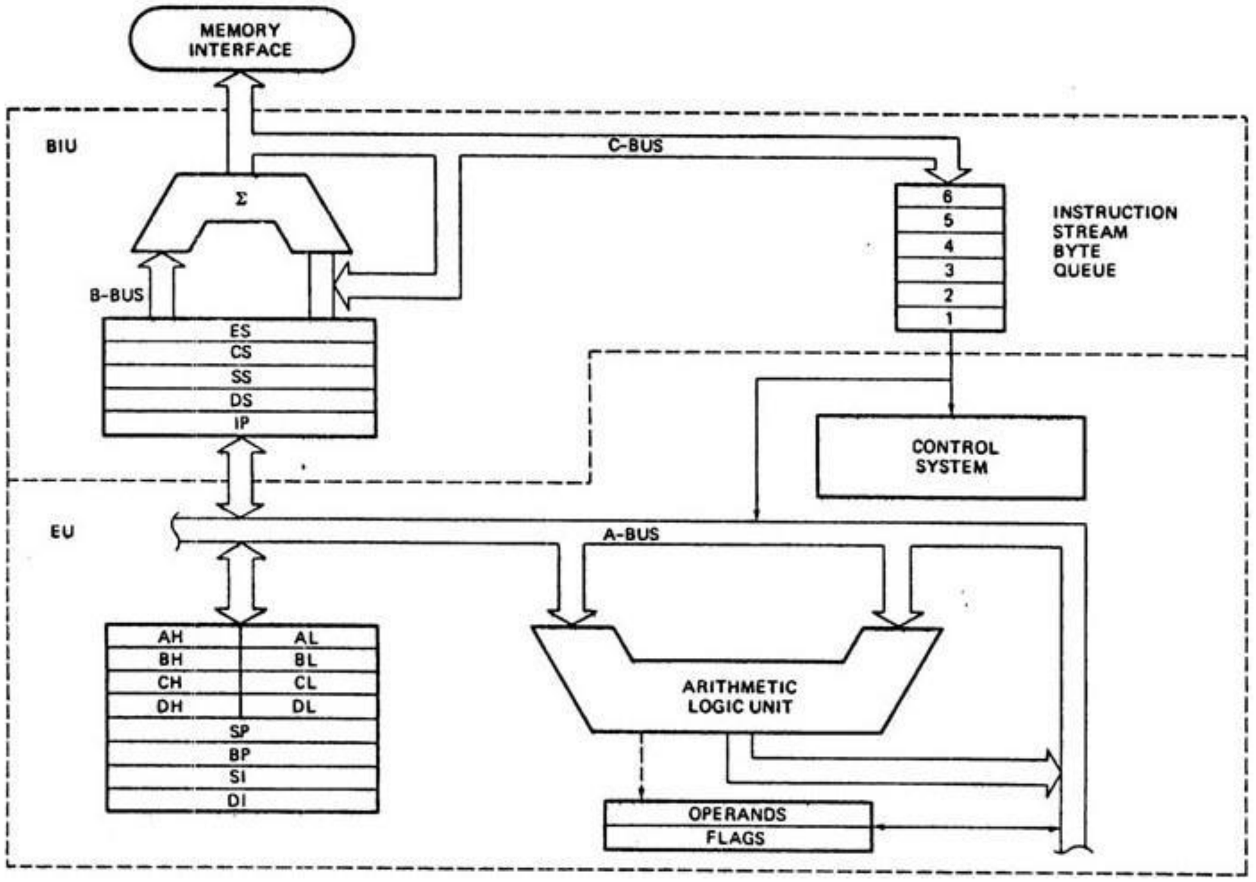
- ✓ The BIU and EU function independently.
- ✓ The BIU interface the 8086 to the outside world. The BIU fetches, reads data from memory and ports, and writes data to memory and I/O ports.
- ✓ EU receives program instruction codes and data from the BIU, executes these instructions and stores the results either in general registers or output them puts all its data through the BIU.

The BIU contains

1. Segment Registers, 2. Instruction Pointer (IP), 3.Instruction Queue

The EU contains

1. ALU
2. General purpose registers
3. Index registers
4. Pointers
5. Flag register



General Purpose Registers

All general registers of the 8086 microprocessor can be used for arithmetic and logic operations. The 16 bit general purpose registers are

1. Accumulator register (AX)
2. Base register (BX)
3. Count register (CX)
4. Data register (DX)

(i) Accumulator register (AX)

- ✓ Accumulator (AX) is a 16 bit register; consists of two 8-bit registers AL and AH.
- ✓ AL contains the low-order byte of the word, and AH contains the higher order byte.
- ✓ Accumulator can be used for Input/ Output (I/O) operations and string manipulation.

(ii) Base register (BX)

- ✓ Base register (BX) is a 16 bit register; consist of two 8-bit registers BL and BH.
- ✓ BL consist the lower order byte of the word, and BH contains the higher order byte.
- ✓ BX register contains a data pointer used for based, based indexed or register indirect addressing.

(iii) Count register (CX)

- ✓ Counter register (CX) is a 16 bit register; consists of two 8-bit registers CL and CH.
- ✓ CL register contain the low order byte of the word, and CH contains the high order byte.
- ✓ Count register can be used as a counter in string manipulation and shift/ rotate instructions.

(iv)Data register (DX)

- ✓ Data register (CX) is a 16 bit register; consists of two 8-bit registers DL and DH.
- ✓ DL register contain the low order byte of the word, and DH contains the high order byte.
- ✓ Data register can be used as a port number in I/O operations.
- ✓ In integer 32-bit multiply and divide instruction the DX register contains higher order word of the initial or resulting number.

Segment Registers

There are four different 64 KB segments for instructions, stack, data and extra data.

The segment registers are:

1. Code segment (CS)
2. Stack segment (SS)
3. Data segment (DS)
4. Extra segment (ES)

(i) Code segment (CS)

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

(ii) Stack segment (SS)

- ✓ Stack segment is a 16-bit register containing address of 64KB segment with program stack.
- ✓ By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers are located in the stack segment.
- ✓ SS register can be changed directly using POP instruction.

(iii) Data segment (DS)

- ✓ Data segment is a 16-bit register containing address of 64KB segment with program data.
- ✓ By default, the processor assumes that all data referenced by general registers (AX, BX, CX, and DX) and index register (SI, DI) is located in the data segment.
- ✓ DS register can be changed directly using POP and LDS instructions.

(iv) Extra segment (ES)

- ✓ Extra segment is a 16-bit register containing address of 64KB segment, usually with program data.
- ✓ By default, the processor assumes that the DI register references the ES segment in string 'manipulation instructions.
- ✓ ES register can be changed directly using POP and LES instructions.
- ✓ It is possible to change default segments used by general and index registers by prefixing instructions with a CS, SS, DS or ES prefix.

Pointer Registers

(i) Stack Pointer (SP)

Stack pointer is a 16-bit register pointing to program stack.

(ii) Base Pointer (BP)

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

Index Registers: (i) Source Index (SI)

- ✓ Source index is a 16-bit register.
- ✓ SI is used for indexed, based indexed and register indirect addressing, as well as a source data address in string manipulation instructions.

(ii) Destination Index (DI)

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

Instruction Pointer (IP)

- ✓ Instruction pointer is a 16-bit register. The operation is same as the program counter.
- ✓ The IP register is updated by the BIU to point to the address of the next instruction.
- ✓ Programs do not have direct access to the IP, but during execution of a program the IP can be modified or saved and restored from the stack.

Flag register

Flag register is a 16-bit register containing nine 1-bit flags:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Six status or condition flags (OF, SF, ZF, AF, PF, CF)

Three control flags (TF, DF, IF)

- **Overflow Flag (OF)** – It is set if an overflow occurs, i.e., a result is out of range.
- **Sign Flag (SF)** – It is set if the most significant bit of the result is set.
- **Zero Flag (ZF)** – It is set if the result is zero.
- **Auxiliary carry Flag (AF)** – It is set if there is a carry out of bit 3 during addition or borrow by bit 3 during subtraction. This flag is used exclusively for BCD arithmetic.
- **Parity Flag (PF)** – It is set to 1 if the low-order 8-bits of the result contain an even number of 1s.
- **Carry Flag (CF)** – It is set if carry from or borrow to the most significant bit during last result calculation.
- **Trap Flag (TF)** – if set, a trap is executed after each instruction.
- **Direction Flag (DF)** – Used by string manipulation instructions. If set then string manipulation instructions will auto- decrement index registers. If cleared then the index registers will be auto-incremented.
- **Interrupt-enable Flag (IF)** - Setting this bit enables maskable interrupts.

Instruction Queue:

- ✓ . The instruction queue is a First-In-First-out (FIFO) group of registers where 6 bytes of instruction code is pre-fetched from memory.
- ✓ ***It is being done to speedup program execution by overlapping instruction fetch and execution. This mechanism is known as PIPELINING.***
- ✓ If the queue is full, the BIU does not perform any bus cycle. If the BIU is not full and can store atleast 2 bytes and EU does not request it to access memory, the BIU may pre-fetch instructions.
- ✓ If the BIU is interrupted by the EU for memory access while pre-fetching, the BIU first completes fetching and then services the EU. In case of JMP instruction, the BIU will reset the queue and .begin refilling after passing the new instruction to the EU.

ALU: Arithmetic and Logic Unit

ALU is a 16 bit register. It can add, subtract, increment, decrement, complement, shift numbers and performs AND, OR, XOR operations.

Control unit: "

Generates timing and control signals to perform the internal operations of the microprocessor.

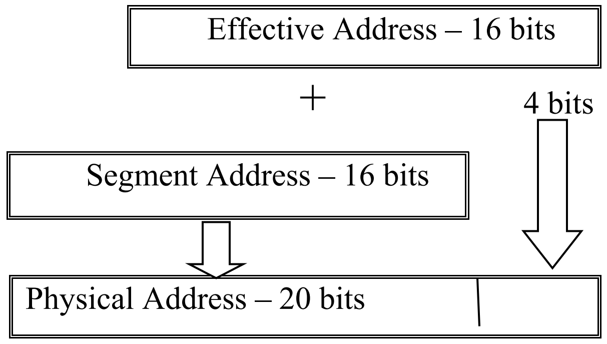
3. The 8086 Addressing Modes

AUQ: What are the addressing modes in 8086? Explain with example.(Dec-2006,07,08,10,11, May2006,07,08,09,11,15, May 2016, Dec 2016)

Addressing modes in 8086:

The 8086 memory addresses are calculated by adding the segment register contents to an offset address. The offset address calculation depends on the addressing mode being used. The total number of address lines in the 8086 is 20 whereas the segment registers are 16 bits. The actual address in memory (effective address) is calculated as per the following steps.

- The segment register contents are multiplied by 1OH, thus, shifting the contents left by 4 bits. This results in the starting address of the segment in memory.
- The offset address is calculated. The offset address is basically the offset of the actual memory location from the starting location of the segment. The calculation of this offset value depends on the addressing mode being used.
- The offset address is added to the starting address of the segment to get the effective address, i.e. the actual memory address.


Effective Address – 16 bits
+
4 bits
Segment Address – 16 bits
Physical Address – 20 bits

Suppose a segment register contents are xyzwH, and the offset value calculated is abcdH, then:

- Starting address of the segment
- Offset address
- Effective address

The addressing modes specify the location of the operand and also how its location may be determined. The following addressing modes are supported in the 8086.

- Register Addressing Mode
- Immediate Addressing Mode
- Direct Memory Addressing Mode
- Register Indirect Addressing Mode
- Base plus Index Register Addressing Mode
- Register Relative Addressing Mode
- Base plus Index Register Relative Addressing Mode
- String Addressing Mode

Register Addressing Mode

When both destination and source operands reside in registers, the addressing mode is known as register addressing mode. Following are the examples:

➢ MOVAX,BX

Move the contents of BX register to AX register. The contents of BX register remain unchanged.

➢ AND AL, BL

AND the contents of AL register with the contents of BL register and place the resultant contents in AL register.

Immediate Addressing Mode

When one of the operands is part of the instruction, the addressing mode is known as immediate addressing mode. Examples are given below

- MOV CX, 2346H

Copy into CX the 16-bit data 2346H.

➢ SUB AL, 24H

Subtract 24H from the contents of AL register and put the result in AL register.

Direct Memory Addressing Mode

In this mode, the 16-bit offset address is part of the instruction as displacement field. It is stored as 16-bit unsigned or 8-bit sign-extended number.

➢ MOV (4625H), *Dl*

Copy the contents of DL register into memory locations calculated from Data Segment register and offset 4625H.

➢ OR AL, (3030H)

OR the contents of AL register with the contents of memory location calculated from DS register and offset 3030H.

Register Indirect Addressing Mode

In this addressing mode, the offset address is specified through pointer register or index register.

For index register, the SI (Source Index) register or DI (Destination Index) register may be used, whereas for pointer register, BX (Base Register) register or BP (Base Pointer) register may be used. Following are some examples of the application of the register indirect addressing mode.

➢ MOV AL, (BP)

Copy into AL register the contents of memory location, whose address is calculated using offset as contents of BP register and the contents of DS register.

Base plus Index Register Addressing Mode

In this mode, both base register (BP or BX) and index register (SI or DI) are used to indirectly address the memory location. An example is given below.

➢ MOV (BX + DI), AL

Copy the contents of AL register into memory location whose address is calculated using the contents of DS (Data Segment), BX (Base Register) and DI (Destination Index) registers.

Register Relative Addressing Mode

This mode is similar to base plus index addressing mode. In this mode, the offset is calculated using either a base register (BP, BX) or an index register (SI, DI) and the displacement specified as an 8-bit or a 16-bit number, as part of the instruction.

- MOV AX, (DI + 06)

Copy to AL the contents of memory location whose address is calculated using DS (Data Segment), DI (Destination Index) register with displacement of 06, and copy to AH the contents of the next higher memory location.

Base plus Index Register Relative Addressing Mode

This addressing mode is basically the combination of base plus index register addressing mode and register relative addressing mode. To find the address of the operand in memory, a base register (BP or BX), an index register (DI or SI) and the displacement which is specified in instruction is used along with the data segment register. For example:

- MOV (BX + DI + 2), CL

Copy the contents of the CL register to the memory location whose address is calculated using DS (Data Segment), BX (Base Register) and DI (Destination Index) registers and 02 as displacement.

String Addressing Mode

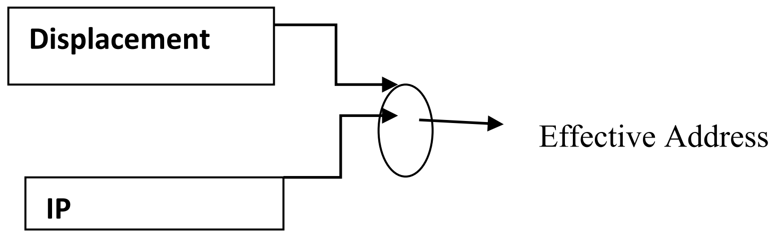
In this addressing mode, the string instruction uses index registers implicitly to access memory.

Example: MOVSB

Copy the byte from the source string location determined by DS and SI to the destination string location determined by ES and DI.

- The addressing modes for **branch related instructions** are
  - **Intrasegment direct (within the same segment)**
  - **Intrasegment Indirect**
  - **Intersegment Direct (Control transfer to different segment)**
  - **Intersegment Indirect**

Intrasegment direct (within the same segment)



If the displacement is 8 bit long, it is called short jump SJMP
If the displacement is 16 bit long, it is called Long jump LJMP.
For example **CALL NEAR**
A NEAR JMP is a jump where destination location is in the same code segment. In this case only IP

is changed.

Intrasegment Indirect

The content of register or memory is accessed using any of the above data related addressing mode except immediate mode**.**

Intersegment Direct (control transfer is in different segment)

The purpose of the addressing mode is to provide a means of branching from one code segment to another .Replaces the content of IP with the part of the instruction and the contents of the CS with another part of instruction**.**

Example: FAR CALL

A FAR JMP is a jump where destination location is from a different segment. In this case both IP and CS are changed as specified in the destination.

Intersegment Indirect

The content of memory block containing 4 bytes.ie IP (LSB),IP(MSB),CS (LSB),and CS(MSB) sequentially .The starting address of the memory block may be referred using any of the addressing mode except immediate mode.

4. The instruction set of 8086.

Explain the instruction set of 8086 microprocessor.

Give three examples for the following 8086 microprocessor instructions: String Instructions, Process Control Instruction, Program Execution Transfer Instructions and Bit manipulation Instructions. (May 2010)(June 2016)

Explain the data transfer, arithmetic and branch instructions with examples. (June 2016)

Intel 8086 has approximately 117 instructions. These instructions are used to transfer data between registers, register to memory, memory to register or register to I/O ports and other instructions are used for data manipulation.

But in Intel 8086 operations between memory to memory is not permitted. These instructions are classified in to six-groups as follows.

1. Data Transfer Instructions
2. Arithmetic Instructions
3. Bit Manipulation Instructions
4. String Instructions
5. Program Execution Transfer Instructions
6. Processor Control Instructions

Data Transfer Instructions

General-Purpose	
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push registers onto stack
POPA	Pop registers off stack
XCHG	Exchange byte or word
XLAT	Translate byte

Input/Output	
IN	Input byte or word
OUT	Output byte or word
Address Object and Stack Frame	
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
ENTER	Build stack frame
LEAVE	Tear down stack frame
Flag Transfer	
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags from stack
POPF	Pop flags off stack

1. MOV

MOV destination, source

This (Move) instruction transfers a byte or a word from the source operand to the destination operand.

(DEST) ← (SRC), DEST = Destination, SRC = Source

Example:

MOV AX, BX

MOVAX, 2150H

MOV AL, [1135]

MOV [4186], AL

MOV SS, DX

MOV [BX], DS

2. PUSH

PUSH Source

This instruction decrements SP (stack pointer) by 2 and then transfers a word from the source operand to the top of the stack now pointed to by stack pointer.

(SP) ← (SP)-2

((SP) + 1: (SP)) ← (SRC)

Example:

PUSH SI

PUSH BX

3. POP

POP destination

This instruction transfers the word at the current top of stack (pointed to by SP) **to** the destination operand and then increments SP by 2, pointing to the new top of the stack.

(DEST) ← ((SP) + 1:(SP))

(SP) ← (SP) + 2

Example:

POP DX

POP DS

4. LAHF

Load Register AH from Flags

This instruction copies Sign flag(S), Zero flag (Z), Auxiliary flag (AC). Parity flag (P) and Carry flag (C) of 8085 into bits 7, 6, 4, 2 and 0 respectively, of register AH. The content of bits 5, 3 and 1 is undefined.

AH←

S	Z	X	A	X	P	X	C

5. SAHF

Store Register AH into Flags

This instruction transfers bits 7, 6, 4, 2 and 0 from register AH into S, Z, AC, P and C flags respectively, thereby replacing the previous values.

S	Z	X	A	X	P	X	C

←AH

6. XCHG

XCHG destination, source

This (Exchange) instruction switches the contents of the source and destination operands.

(Temp) ← (DEST)

(DEST) ← (SRC)

(SRC) ← (Temp)

Example:

XCHG AX, BX

XCHG BL, AL

7. XLAT

XLAT table

- This (Translate) instruction replaces a byte in the AL register with a byte from a 256-byte, user-coded translation table.
- XLAT is useful for translating characters from one code to another like ASCII to EBCDIC. Register BX is the starting point of the table. The byte in AL is used as an index into the table and is replaced by the byte at the offset in the table corresponding to AL's binary value.

AL ← ((BX) + (AL))

Example :

XLAT ASCII_TAB

XLAT Table_3

8. LEA

LEA destination, source

This (Load Effective Address) instruction transfers the offset of the source operand (memory) to the destination operand (16-bit general register).

(REG) ← EA

Example :

LEA BX , [BP] [DI] LEA SI, [BX + 02AF H]

9. LDS

LDS destination, source

This (Load pointer using DS) instruction transfers a 32-bit pointer variable from the source operand (memory operand) to the destination operand and register DS.

(REG) ← (EA)

(DS← (EA+2)

Example:

LDS SI, [6AC1H]

10. LES

LES destination, source

This (Load pointer using ES) instruction transfers a 32-bit pointer variable from the source operand (memory operand) to the destination operand and register ES.

(REG) ← (EA)

(ES) ← (EA+2)

Example:

LES DI, [BX]

11. IN

IN accumulator, port

This (Input) instruction transfers a byte or a word from an input port to the accumulator (AL or AX).

(DEST) ← (SRC)

Example:

IN AX, DX

IN AL, 062H

12. OUT

OUT port, accumulator

This **(Output)** instruction transfers a byte or a word from the accumulator (AL or AX) to an output port.

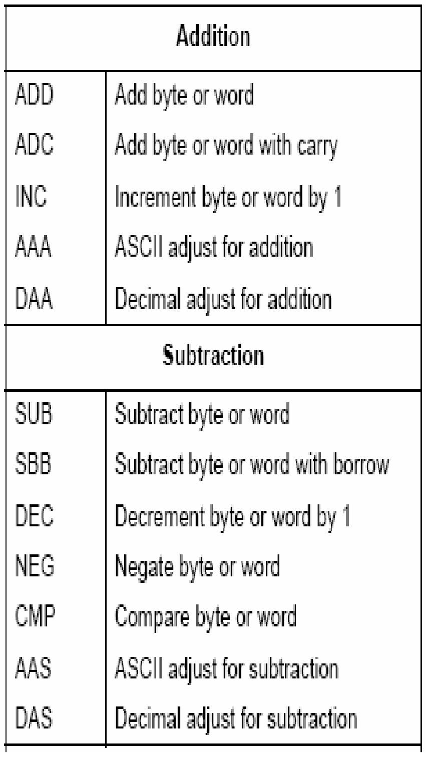
(DEST) ← (SRC)

Example:

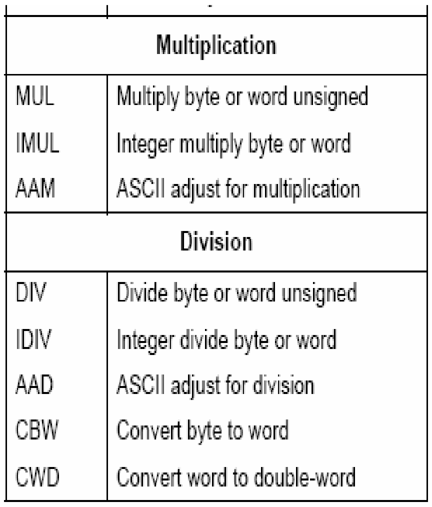
OUT DX, AL

OUT 31, AX

Arithmetic Instructions



Addition	
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
Subtraction	
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction



Multiplication	
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiplication
Division	
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to double-word

1. **ADD**

   ADD destination, source

   This **(Add)** instruction adds the two operands (byte or word) and stores the result in destination operand.

   (DEST) ← (DEST) + (SRC)

   Example:

   ADD CX, DX

   ADD AX, 1257 H

   ADDBX, [CX]

2. **ADC**

   ADC destination, source

   This **(Add with carry)** instruction adds the two operands and adds one if carry flag (CF) is set and stores the result in destination operand.

   (DEST) ← (DEST) + (SRC) + 1

Example:

   ADC AX, BX

   ADC AL, 8

   ADC CX, [BX]

3. **SUB**

   SUB destination, source

   This (Subtract) instruction subtracts the source operand from the destination operand and the result is stored in destination operand.

   (DEST) ← (DEST) - (SRC)

Example:

   SUB AX, 6541 H

   SUB BX, AX

   SUB SI, 5780 H

4. **SBB**

   SBB destination, source

   This (**Subtract with Borrow**) instruction subtracts the source from the destination and subtracts 1 if carry flag (CF) is set. The result is stored in destination operand.

   (DEST) ← (DEST) - (SRC) -1

Example:

SBB BX, CX

SBB AX, 2

5. **CMP**

CMP destination, source

This (Compare) instruction subtracts the source from the destination, but does not store the result.

(DEST) - (SRC)

Example:

CMP AX, 18

CMP BX, CX

6. **INC**

INC destination

This **(Increment)** instruction adds 1 to the destination operand (byte or word).

(DEST) ← (DEST) + 1

Example:

INC BL

INC CX

7. **DEC**

DEC destination

This **(Decrement)** instruction subtracts 1 from the destination operand.

(DEST) ← (DEST)-1

Example:

DEC BL

DEC AX

8. **NEG**

NEG destination

This (Negate) instruction subtracts the destination operand from 0 and stores the result in destination. This forms the 2's complement of the number.

(DEST) ← 0 - (DEST)

Example: NEG AX

NEG CL

9. DAA

This (Decimal Adjust for Addition) instruction converts the binary result of an ADD or ADC instruction in AL to packed BCD format.

If the auxiliary carry flag is set or the low 4 bits of AL are greater than 9, then 06 H is added to AL. If the carry flag is set or the high 4 bits of AL are greater than 9, then 60 H is added to the AL.

10. DAS

This **(Decimal Adjust for Subtraction)** instruction converts the binary result of a SUB or SBB instruction in AL to packed BCD format.

11.AAA

This **(ASCII Adjust for Addition)** instruction adjusts the binary result of ADD or ADC instruction.

If bits 0-3 of AL contain a value greater than 9, or if the auxiliary carry flag (AF) is set, the CPU adds 06 to AL and adds 1 to AH. The bits 4-7 of AL are set to zero.

(AL) ← (AL) + 6

(AH) ← (AH) +1

(AF) ←1

Example:

AAA

Before execution

AH	AL
00	0B

After execution

AH	AL
01	01

12. AAS

This **(ASCII Adjust for Subtraction)** instruction adjusts the binary result of a SUB or SBB instruction.

If D_3 - D_0 of AL>9,

(AL) ← (AL) – 6

(AH) ← (AH) – 1

(AF) ←1

13.MUL

MUL source

- This **(Multiply)** instruction multiply AL or AX register by register or memory location contents.
- Both operands are unsigned numbers.
- If the source is a byte (8 bit), then it is multiplied by register AL and the result is stored in AH and AL.
- If the source operand is a word (16 bit), then it is multiplied by register AX and the result is stored in AX and DX registers.

If 8 bit data, (AX) ← (AL) × (SRC)

If 16 bit data, (AX), (DX) ← (AX) × (SRC)

Example:

MUL25

MUL CX .

MULBL

14.IMUL

IMUL Source

This **(Integer Multiply)** instruction performs a signed multiplication of the source operand and the accumulator.

If 8 bit data, (AX) ← (AL) × (SRC)

If 16 bit data, (AX), (DX) ← (AX) × (SRC)

Example:

IMUL 250

IMUL BL

15. AAM

This (ASCII Adjust for Multiplication) instruction adjusts the binary result of a MUL instruction. AL is divided by 10(0AH) and quotient is stored in AH. The remainder is stored in AL.

(AH) ← (AL/OAH)

(AL) ← Remainder

16. DIV

DIV Source

- This (Division) instruction performs an unsigned division of the accumulator by the source operand.
- It allows a 16 bit unsigned number to be divided by an 8 bit unsigned number, or a 32 bit unsigned number to be divided by a 16 bit unsigned number.
- If byte (8-bit) operation is performed, the 8 bit quotient is stored to AL and 8 bit remainder is stored to AH register.

- If the source operand is a word (16 bit), the 16 bit quotient is stored in AX and the remainder is stored in DX register.

For 8 bit data, AX / source

(AL) ← Quotient

(AH) ← Remainder

For 16 bit data, AX, DX / Source

(AX) ← Quotient

(DX) ← Remainder

Example:

DIV CX

DIV 321

17.IDIV

IDIV source

This (Integer Division) instruction performs a signed division of the accumulator by the source operand.

For 8 bit data, AX / Source

(AL) ← Quotient

(AH) ← Remainder

For 16 bit data, AX, DX/Source

(AX) ← Quotient

(DX) ← Remainder

Example:

IDIV CL

IDIVAX

18. AAD

This **(ASCII Adjust for Division)** instruction adjusts the unpacked BCD dividend in AX before a division operation. AH is multiplied by 10(0AH) and added to AL. AH is set to zero.

(AL) ← (AH × 0AH) + (AL)

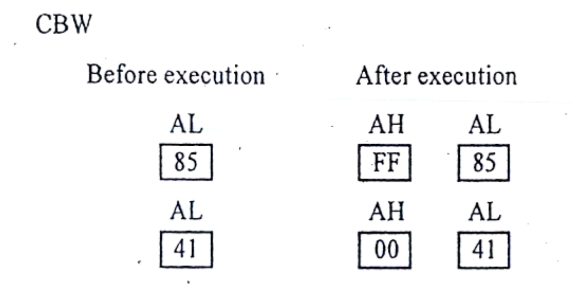
(AH) ←0

19. CBW

This **(Convert Byte to Word)** instruction converts a byte to a word. It extends the sign of the byte in register AL through register AH.

This instruction can be used for 16 bit IMUL or IDIV instruction.

Example:



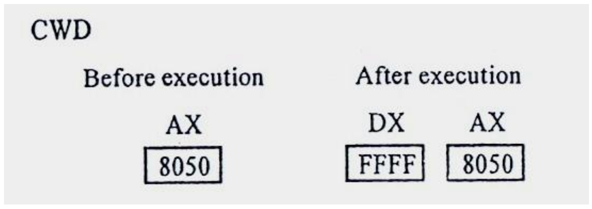
20. CWD

This (Convert Word to Double word) instruction converts a word to a double word. It extends the sign of the word in register AX through register DX.

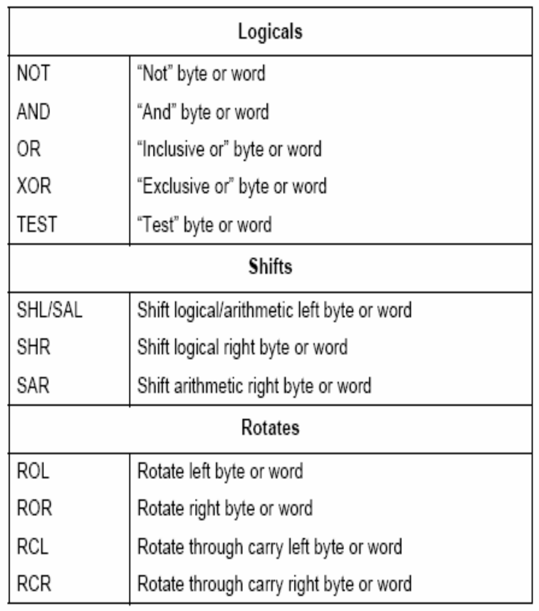
If AX < 8000 H, then DX = 0000 H

If AX > 8000 H, then DX = FFFFH

Example:



Bit Manipulation Instructions



Logicals	
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
Shifts	
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
Rotates	
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word

(i)	Logical Instructions	: AND, OR, XOR, NOT, TEST
(ii)	Shift Instructions	: SHL, SAL, SHR, SAR
(iii)	Rotate Instructions	: ROL, ROR, RCL, RCR

1. AND

AND destination, source

This **(AND)** instruction performs the logical "AND" of the source operand with the destination operand and the result is stored in destination.

(DEST) ← (DEST) "AND" (SRC)

Example:

AND BL, CL

AND AL, 0011 1100 B

2. OR

OR destination, source

This **(OR)** instruction performs the logical "OR" of the source operand with the destination operand and the result is stored in destination.

(DEST) ← (DEST) "OR" (SRC)

Example:

OR AX, BX

OR AL, 0FH

3. **XOR**

XOR destination, source

This **(Exclusive OR)** instruction performs the logical "XOR" of the two operands and the result is stored in destination operand.

(DEST) ← (DEST) "XOR"(SRC)

4. NOT

NOT destination

This (NOT) instruction inverts the bits (forms the l's complement) of the byte or word.

(DEST) ← 1 's complement of (DEST)

Example:

NOT AX

5. TEST

TEST destination, source

This (TEST) instruction performs the logical "AND" of the two operands and updates the flags but does not store the result.

(DEST) "AND" (SRC)

Example:

TEST AL, 15 H

TEST SI, DI

6. SHL

SHL destination, count

This (Shift Logical Left) instruction performs the shift operation. The number of bits to be shifted is represented by a variable count, either 1 or the number contained in the CL register.

Example

SHL AL, 1

Before execution:

CF	AL							
0	1	1	0	0	1	1	0	0

After execution:

CF	AL							
1	1	0	0	1	1	0	0	0

7. SAL

SAL destination, count

SAL **(Shift Arithmetic Left)** and SHL **(Shift Logical Left)** instructions perform the same operation and are physically the same instruction.

Example

SAL AL, CL

SAL AL, 1

8. SHR

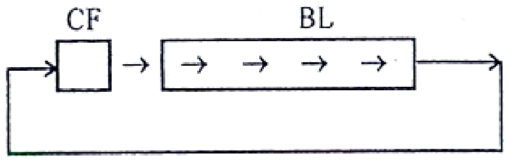
SHR destination, count

This **(Shift Logical Right)** instruction shifts the bits in the destination operand to the right by the number of bits specified by the count operand, either 1 or the number contained in the CL register.

Example

SHR BL, 1

SHR BL, CL



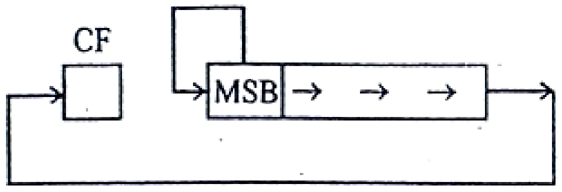
The SHR instruction may be used to divide a number by 2. For example, we can divide 32 by 2,

```
MOV  BL, 32  ;  0010  0000  (32)
SHR  BL, 1   ;  0001  0000  (16)
SHR  BL, 1   ;  0000  1000  (8)
SHR  BL, 1   ;  0000  0100  (4)
SHR  BL, I   ;  0000  0010  (2)
```

9. SAR

SAR destination, count

This **(Shift Arithmetic Right)** instruction shifts the bits in the destination operand to the right by the number of bits specified in the count operand. Bits equal to the original high-order (sign) bits are shifted in on the left, thereby preserving the sign of the original value.



Example :

SAR BL, 1

Before execution:

CF BL

CF		BL							
0		1	1	0	0	1	1	0	0

After execution:

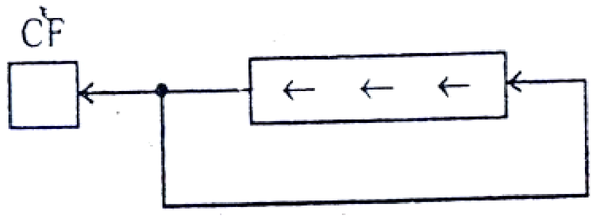
CF BL

0		1	1	1	0	0	1	1	0

10. ROL

ROL destination, count

This **(Rotate Left)** instruction rotates the bits in the byte/word destination operand to the left by the number of bits specified in the count operand.



Example:

ROL AL, 1

Before execution:

CF		AL							
0		1	1	0	0	1	1	0	0

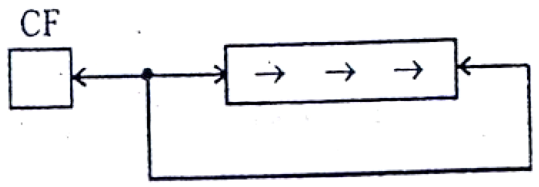
After execution:

CF		AL							
1		1	0	0	1	1	0	0	1

11. ROR

ROR destination, count

This **(Rotate Right)** instruction rotates the bits in the byte/word destination operand to the right by the number of bits specified in the count operand.



Example:

ROR AL, 1

Before execution:

CF		AL							
0		1	1	0	0	1	1	0	0

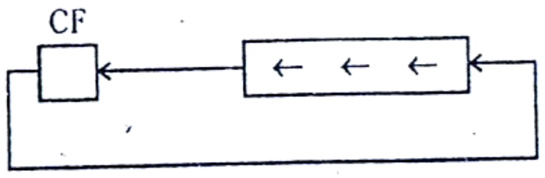
After execution:

CF		AL							
0		0	1	1	0	0	1	1	0

12. RCL

RCL destination, count

This **(Rotate through Carry Left)** instruction rotates the contents left through carry by the specified number of bits in count operand.



Example:

RCL AL, 1

Before execution:

CF		AL							
1		0	0	0	0	1	1	1	1

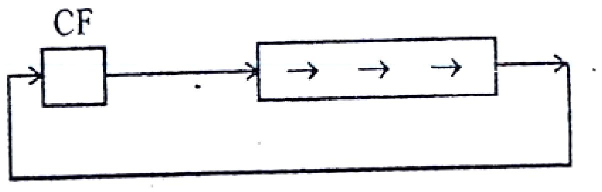
After execution:

CF		AL							
0		0	0	0	1	1	1	1	1

13.RCR

RCR destination, count

This **(Rotate through Carry Right)** instruction rotates the contents right through carry by the specified number of bits in the count operand.



Example:

RCR AL, 1

Before execution:

CF
1

AL							
1	1	0	0	0	0	1	0

After execution:

CF
0

AL							
1	1	1	0	0	0	0	1

String Instructions

REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero
MOVSB/MOVSW	Move byte string/word string
MOVS	Move byte or word string
INS	Input byte or word string
OUTS	Output byte or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string

1. REP

REP MOVS destination, Source

This **(Repeat)** instruction converts any string primitive instruction into a re-executing loop. It specifies a termination condition which causes the string primitive instruction to continue executing until the termination condition is met. REP is used in conjunction with the MOVS and STOS instructions.

Example:

REP MOVS CL, AL

The other Repeat instructions are:

REPE	-	Repeat while Equal
REPZ	-	Repeat while zero
REPNE	-	Repeat while Not Equal
REPNZ	-	Repeat while Not Zero

The above instructions are used with the CMPS and SCAS instructions.

Example:

REPE CMPS destination, source

REPNE SCAS destination

2.MOVS

MOVS destination - string, source-string

This **(Move String)** instruction transfers a byte/word from the source string (addressed by SI) to the destination string (addressed by DI) and updates SI and DI to point to the next string element.

(DEST) ← (SRC)

Example:

MOVS Buffer 1, Buffer 2

3. CMPS

CMPS destination-string, source-string

This **(Compare String)** instruction subtracts the destination byte/word (addressed by DI) from the source byte/word (addressed by SI). It affects the flags but does not affect the operands. -.

Example:

CMPS Buffer 1, Buffer 2

4. SCAS

SCAS destination-string

- This **(Scan String)** instruction subtracts the destination string element (addressed by DI) from the contents of AL or AX and updates the flags.

- ‘The contents of destination string or accumulator are not altered.
- After each operation, DI is updated to point to the next string element

Example:

SCAS Buffer

5. LODS

LODS source-string

This **(Load String)** instruction transfers the byte/word string element addressed by SI to register AL or AX and updates SI to point to the next element in the string.

(DEST)←(SRC)

Example:

LODSB name

LODSW name

6. STOS

STOS destination - string

This **(Store String)** instruction transfers a byte/word from register AL or AX to the string element addressed by DI and updates DI to point to the next location in the string.

(DEST) ← (SRC)

Example: STOS display

Program Transfer Instructions

Unconditional instructions	: CALL, RET, JMP
Conditional instructions	: JC, JZ, JA
Iteration control instructions	: LOOP, JCXZ
Interrupt instructions	: INT, INTO, IRET

1. CALL

- **CALL procedure - name**
- This (CALL) instruction is used to transfer execution to a subprogram or procedure.
- RET (return) instruction is used to go back to the main program.
- There are two basic types of CALL : NEAR and FAR

Intra-Segment CALL:

- A NEAR-CALL is a call to a procedure which is in the same code segment as the CALL instruction.
- When 8086 executes a NEAR-CALL instruction, it decrements the stack pointer (SP) by 2 and copies

the offset of the next instruction after the CALL on the stack.

- It loads IP with the offset of the first instruction of the procedure in same segment.
- This NEAR-CALL is known as Intra-segment CALL.

Inter-Segment CALL:

- A FAR-CALL is a call to a procedure which is in a different segment from that which contains the CALL instruction.
- When 8086 executes a FAR-CALL, it decrements the SP by 2 and copies the contents of the CS register to the stack.
- It then decrements SP by 2 again and copies the offset of the instruction after the CALL to the stack.
- Finally it loads CS with the segment base of the segment which contains the procedure and IP with the offset of the first instruction of the procedure in that segment.
- This FAR-CALL is known as Inter-segment CALL.

Example:

CALL NEAR

CALL AX

2.RET

This **(Return)** instruction will return execution from a procedure to the next instruction after the CALL instruction in the main program.

If intra-segment, IP is popped off the stack; SP =SP+2

If inter-segment, CS is popped off the stack; SP = SP+2

IP is popped off the stack; SP=SP+2

If optional POP value is used, then SP= SP + value.

Example:

RET

RET 6

3. JMP

JMP target

This **(Jump)** instruction unconditionally transfers control to the target location. The target operand may be obtained from the instruction itself (direct JMP) or from memory or a register referenced by the instruction (indirect JMP).

A NEAR-JMP (Intra-segment) is a jump where destination location is in the same code segment. In this case only IP is changed.

IP = IP + signed displacement

A FAR-JMP (Inter-segment) is a jump where destination location is from a different segment. In this case both IP and CS are changes as specified in the destination.

Example:

JMPBX

Conditional JMP

Instruction	**Operation**
JC	Jump if carry
JNC	Jump if no carry
JZ	Jump if Zero
JNZ	Jump if not zero
JS	Jump if sign or negative
JNS	Jump if positive
JP/JPE	Jump if parity/parity even
JNP/JPO	Jump if not parity/odd parity
JO	Jump if overflow
JNO	Jump if no overflow
JA/JNBE	Jump if above/not below or equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above or equal
JBE/JNA	Jump if below or equal / not above
JG/JNLE	Jump if greater/not less than nor equal
JGE/JNL	Jump if greater or equal/not less than
JL/JNGE	Jump if less/neither greater nor equal
JLE/JNG	Jump if less than or equal / not greater

5.LOOP

LOOP label

This (Loop if CX not zero) instruction decrements CX by 1 and transfers control to the target operand if CX is not zero. Otherwise the instruction following LOOP is executed.

If CX≠0, CX = CX - 1

IP = IP + displacement

If CX=0, then the next sequential instruction is executed.

Example:

LOOP again

6. LOOPE/LOOPZ

LOOPE/LOOPZ label

These **(LOOP while Equal/Loop while Zero)** are different mnemonics for the same instruction.

If CX≠0, CX=CX-1 and control is transferred '.o the target operand

If CX = 0, then next sequential instruction is executed.

Example:

LOOPE again

7. LOOPNE/LOOPNZ

LOOPNE Label

These **(LOOP while Not Equal/LOOP while Not Zero)** are different mnemonics for the same instruction. CX is decremented by 1 and control is transferred to target operand if CX is not zero and if ZF=0; otherwise the next sequential instruction is executed.

Example:

LOOPNE again

8. JCXZ

JCXZ Label

This **(Jump if CX register Zero)** instruction transfers control to the target operand if CX=0. It is useful at the beginning of a loop to bypass the loop if CX=0.

Example:

JCXZ again

9. INT

INT interrupt type (0-255)

This **(Interrupt)** instruction activates the interrupt procedure specified by the interrupt-type number (0-255). The address of the interrupt pointer is calculated by multiplying the interrupt-type number by 4.

Example :

INT 7, INT 180

10.INTO

This **(Interrupt on Overflow)** instruction generates a software interrupt if the overflow flag is set. Otherwise, control proceeds to the following instruction without activating an interrupt procedure.

11.IRET

This **(Interrupt on Return)** instruction transfers control back to the point of interruption by popping IP, CS and the flags from the stack.

IRET is used to exit any interrupt procedure, whether activated by hardware or software.

Processor Control Instructions

1. HLT

This (Halt) instruction will cause the 8086 to stop fetching and executing instructions. The 8086 will enter a halt state. The ways to get the processor out of halt state are with (i) an interrupt signal on the INTR pin. (ii) An interrupt signal on the NMI pin, (iii) a reset signal on the RESET pin.

2. WAIT

This (Wait) instruction causes the 8086 to enter the wait state while its test line is not active.

3. LOCK

- The LOCK prefix allow 8086 to make sure that another processor does not take control of the system bus while it is in the middle of a critical instruction which uses the system bus.
- The LOCK prefix is put in front of the critical instruction.
- When an instruction with a LOCK prefix executes, the 8086 will assert its bus lock signal output. This signal is connected to an external bus controller device which then prevents any other processor from taking over the system bus.

4. ESC

- ✓ This **(Escape)** instruction provides a mechanism by which other coprocessors may receive their instructions from the 8086 instruction stream and make use of the 8086 addressing modes.
- ✓ The 8086 does a no operation (NOP) for the ESC instruction other than to access a memory operand and place it on the bus.

5. NOP

This **(No operation)** instruction causes the CPU to do nothing. NOP does not affect any flags.

6. Flag operations

Instruction	Operation
CLC	Clear the carry flag (CF)
CMC '	Complement the carry flag (CF)
STC	Set the carry flag (CF)
CLD	Clear the direction flag (DF)
STD	Set the direction flag (DF)
CLI	Clear the interrupt flag (IF)
STI	Set the interrupt flag (IF)

5. <u>Assembler directives</u>

AUQ: Explain the assembler directives in 8086 Microprocessor. (Dec-2006,11,12, May2007,08,10,11,13, Dec 2016)

- An assembler is a program used to convert an assembly language program into the equivalent machine code modules which may further converted into executable codes.

Some directives such as ORG, EQU, DB, DW, etc which are common in different assemblers were also described. Though a representative set of directives for the 8086 assembler is presented, it is possible that some assemblers have a few additional directives. On the other hand, some of the directives presented here may not be present or may be present in different forms.

Directives for Constant and Variable Definition

An Intel 8086 assembly program uses different types of constants like binary, decimal, octal and hexadecimal. These can be represented in the program using different suffixes like B (for binary), D (for decimal), O (for octal) and H (for hexadecimal) to the constant. For example:

- 10H is a hexadecimal number (equivalent to decimal 16).
- 27O is an octal number (equivalent to decimal 23).
- 10100B is a binary number.

A number of directives are used to define and store different kinds of constants.

The DB (Define Byte), DW (Define Word), DDW (Define Double Word) are described in Chapter The 8086 assembler uses DD as directive for double word. Some of the 8086 assemblers also provide the following additional directives.

- DQ : Define Quadword
- DT : Define Ten Bytes

In addition, these directives can be used to store strings and arrays. For example:

- NAME DB "NEHA SHIKHA". The ASCII codes of alphanumeric characters specified within double quotes are stored.

- NUM DB 5, 6, 10, 12, 7. The array of five numbers NUM is declared and the numbers 5, 6, 10, 12 and 7 are stored.

The directives DUP and EQU are used to store strings and arrays.

DUP

Using the DUP directive, several locations may be initialized and the values may be put in those locations. The format is as follows. Name Type Num DUP (value)

For example: TEMP DB 20 DUP (5)

The above directive defines an array of 20 bytes in memory and each location is initialized to 5. The array is named TEMP.

EQU

The EQU directive may be used to define a data name with immediate value or another data name. It can also be used to equate a name to a string. For example:

NUMB EQU 20H .

NAME EQU “RASHMI”

Program Location Control Directives

The directives used for program location control in the 8086 assembler are (ORG, EVEN, ALIGN and LABEL.

ORG

The ORG directive is used to set the location counter to a particular value. For example:

- ORG 2375H

  The location counter is set to 2375H. If it occurs in data segment, the next data storage will start at 2375H. If it is in code segment, the next instruction will start at 2375H.

EVEN

Using EVEN directive, the next data item or label is made to start at the even address boundary. The assembler, on encountering EVEN directive, will advance the location counter to even address boundary. For example:

- EVEN TABLE DW 20 DUP (0)

This statement declares an array named TABLE of 20 words starting from the even address. Each word is initialized to zero.

ALIGN number

This directive will force the assembler to align the next segment to an address that is divisible by 2, 4, 8 or 16. The unused bytes are filled with 0 for data and NOP instruction for code. For example:

- ALIGN 2

It will force the next segment to the next even address.

LENGTH

It is an operator which is used to tell the assembler to determine the number of elements in a data item, such as string or array. For example:

- MOV CX, LENGTH ARRAY

This statement will move the length of the array to the CX register.

OFFSET

This operator is used to determine through the assembler, the offset of a data item from the start of the segment containing it. For example:

- MOV AX, OFFSET FACT

This statement will place in the AX register the offset of the variable FACT from the start of the segment.

LABEL

The LABEL directive is used to assign a name to the current value in the location counter. The location counter is used by the assembler to keep track of the current location. The value in the location counter denotes the distance of the current location from the start of the segment.

Segment Declaration Directives

These directives help in declaring various segments with specific names. The start and the end of segments may also be specified using this directive. The directives for segment declaration include SEGMENT, ENDS, ASSUME, GROUP, CODE, DATA, STACK etc.

SEGMENT and ENDS

The SEGMENT and ENDS directives signify the start and end of a segment.

```
INST  SEGMENT
ASSUME CS: FNST, DS: DATAW
__________
__________
```

INST ENDS **ASSUME**

This directive is used to assign logical segment to physical segment at any time. It tells the assembler as to what addresses will be in segment registers at the time of execution. For example:

- ASSUME CS: CODE, DS: DATA, SS: STACK

This directive tells the assembler that the CS register will store the address of the segment whose name is CODE, and so on.

.CODE (Name)

This code directive is the shortcut used in the definition of code segments. The name is optional and is specified if there is more than one code segment in the program.

.DATA and .STACK

Similar to .CODE, the .DATA and .STACK directives are shortcuts in the definition of data segment and stack segment, respectively.

GROUP

This directive is used to tell the assembler to group all the segments in one logical group segment. This allows the contents of all the segments to be accessed from the same segment base. For example:

- PROG GROUP CODE, DATA

The above statement will group the two segments CODE and DATA into one segment named PROG. Each segment must be declared using ASSUME statement as in the following statement.

- ASSUME CS: PROG DS: PROG

Procedure and Macro-related Directives

The directives in this group relate to the declaration of procedures and macros along with the variables contained in them. The directives include PROC, ENDP, PUBLIC, MACRO,ENDM and EXTRN.

PROC and ENDP

The PROC directive is used to define the procedures. The procedure name is a *must* and it must follow the naming convention of the assembler. Along with the name of the procedure, the field NEAR or FAR also needs to be specified.

The ENDP directive is used to mark the end of the procedure. Some examples are given below.

```
FUNCT PROC FAR
__________
__________
ENDP
FACT PROC NEAR
__________
__________
ENDP
```

The first procedure FUNCT is in a segment which is different from where it is called. The second procedure FACT is in the same segment where it is called. All the statements of the procedure are between PROC and ENDP directives.

PUBLIC

It is very much possible that a variable is defined in one module, but is used in other modules. In order to facilitate the linking, such variables are declared *public,* using the PUBLIC directive in the module where they are defined. For example:

- PUBLIC PX, PY, PZ

MACRO and ENDM

The MACRO directive is used to define macros in the program. The ENDM directive defines the end of MACRO

Other Directives

These directives are of general nature, or they relate to more than one group described earlier. The directives described include PTR, PAGE, TITLE, NAME and END.

PTR

The PTR is an operator used in instructions to assign a specific type to a variable or a label. The PTR operator can also be used to override the declared type of variable. Following is an example of the use of PTR directive.

- ARRAY DW 0125H, 1630H, 9275H ...

In the above array of words, suppose we wish to move a byte from the array, we may then simply insert the PTR operator as follows

- MOV AL, BYTE PTR ARRAY

PAGE

This directive is used for listing of the assembled program. At the start of the program, the directive is placed to specify the maximum number of lines on a page and the maximum number of characters in a line to be placed for listing. An example is given below.

- PAGE 60, 120

The above example specifies that 60 lines are to be listed on a page with a maximum of 120 characters in each line.

TITLE

This directive is also used for the listing of the program. The title declared in this directive defines the title of the program and is listed in line 2 of each page of program listing. The maximum number of characters allowed is 60. For example:

- TITLE PROGRAM TO FIND SQUARE ROOT

NAME

The NAME directive is used to assign a specific name to each module, when the program consists of a number of modules. This helps in understanding the program logic.

END

This is the last statement of the program and it specifies the end of the program to the assembler. It must be noted that not all of the above directives are used in all programs. User may deploy them depending on the need of the program logic.

6. <u>ASSEMBLY LANGUAGE PROGRAMS:</u>

1. 16-BIT ADDITION USING 8086

```
     MOV  DX,0000
     MOV AX,[2000]
     MOV BX,[2002]
     ADD AX,BX
     JNC L1
     INC DX
L1   MOV[2004],AX
     MOV[2006],DX
     HLT
```

2. 16-BIT SUBTRACTION USING 8086

```
     MOV  DX,0000
     MOV AX,[2000]
     MOV BX,[2002]
     SUB AX,BX
     JNC L1
     INC DX
L1   MOV[2004],AX
     MOV[2006],DX
     HLT
```

3. 16 BIT MULTIPLICATION USING 8086

```
MOV  DX,0000
MOV  SI,2000
MOV AX,SI
MOV  CX,[2002]
MUL   CX
MOV   SI,2100
MOV   [SI],AX
MOV  [2102],DX
HLT
```

4. 16 BIT DIVISION USING 8086

```
MOV  DX,0000
MOV  SI,2000
MOV AX,SI
MOV  CX,[2002]
DIV   CX
MOV   SI,2100
MOV   [SI],AX
MOV  [2102],DX
HLT
```

5. 16 BIT ASCENDING ORDER USING 8086

```
        MOV AX,0000H
START   MOV CX,0005H
        MOV DX,0005H
        MOV SI,2000H
LABEL   MOV AX,[SI]
        CMP AX,[SI+2]
        JC LOOP
        XCHG AX,[SI+2]
        XCHG AX,[SI]
LOOP    ADD SI,0002
        LOOP LABEL
        DEC DX
        JNZ START
        HLT
```

6. 16-BIT DESCENDING ORDER USING 8086

```
        MOV AX,0000
START   MOV CX,0005
        MOV DX,0005
        MOV SI,2000
LABEL   MOV AX,[SI]
        CMP AX,[SI+2]
        JNC LOOP
        XCHG AX, [SI+2]
        XCHG AX, [SI]
LOOP    ADD SI, 0002
        LOOP LABEL
        DEC DX
        JNZ START
        HLT
```

7. LARGEST NUMBER IN AN ARRAY USING 8086

```
      MOV CX,0004
      DEC CX
      MOV SI,2000
      MOV AX,[SI]
LABEL  CMP AX,[SI+2]
      JNC LOOP1
      MOV AX,[SI+2]
LOOP 1   ADD SI,0002
      LOOP LABEL
      MOV[2500],AX
      HLT
```

9. Write a program based on 8086 instruction set to compute the average of 'n' numberof bytes stored in the memory.(Nov/ Dec 2012)

```
MOV     SI, 2000H
MOV     DI, 3000H
MOV     CL, [SI]
INC     SI
MOV     AX, 0000H
ADD     AL, [SI]
JNC     STEP2
INC     AH
INC     SI
LOOP    STEP1
MOV     [DI]AX
HLT
```

8. SMALLEST NUMBER IN AN ARRAY USING 8086

```
      MOV CX,0004
      DEC CX
      MOV SI,2000
      MOV AX,[SI]
LABEL CMP AX,[SI+2]
      JC LOOP1
      MOV AX,[SI+2]
LOOP 1  ADD SI,0002
      LOOP LABEL
      MOV[2500],AX
      HLT
```

10. Write an 8086 ALP to sort the array of elements in ascending order. (Apr/ May 2011, May / June 2013)

```
        MOV SI, 2000H
        MOV CL,[SI]
        DEC CL
STEP1  MOV SI,2000H
        MOV CH, [SI]
        DEC CH
        INC SI
STEP2  MOV AL, [SI]
        INC SI
        CMP AL, [SI]
        JC STEP3
        XCHG AL, [SI]
        XCHG AL,[SI-1]
        DEC CH
        JNZ STEP2
STEP3  DEC CL
        JNZ STEP1
        HLT
```

11. Write an 8086 ALP to find the largest element in array elements. (Apr/ May 2011)

```
        MOV   SI.2000H
        MOV   DI, 3000H
        MOV   CL, [SI]
        INC   SI
        MOV   AL, [SI]
        DEC   CL
STEP1:  INC   SI
        MOV   BL, [SI]
        CMP   AL, BL
        JNC   STEP2:
        MOV   AL, BL
STEP2:  DEC   CL
        JNZ   STEP1:
        MOV   [DI],AL
        HLT
```

12. Write an 8086 program to convert BCD data to binary data. (Nov/ Dec 2010)

```
MOV BX,2000H
MOV AL,[BX]
MOV DL,AL
AND DL,0FH
AND AL,F0H
MOV CL,4
ROA AL,CL
MOV DH,OAH
MUL DH
ADD AL,DL
MOV [BX+1],AL
HLT
```

7. Linking And Relocation

Explain linking and relocation concepts in 8086 Processor.

The DOS linking program links the different object modules of a source program and function library routines to generate an integrated executable code of the source program.

The main input to the linker is the .OBJ file that contains the object modules of the source programs.

The linker program is invoked using the following options.

```
C> LINK
or
C>LINK MS.OBJ
```

The output of the link program is an executable file with the entered filename and .EXE extension. This executable filename can further be entered at the DOS prompt to execute the file.

The linked file in binary for ***run*** on a computer is commonly known as executable file or simply '.exe.' file. After linking, there has to be re-allocation of the sequences of placing the codes before actually placement of the codes in the memory.

The loader program performs the task of reallocating the codes after finding the physical RAM addresses available at a given instant. The DOS linking program links the different object modules of a source program and function library routines to generate an integrated executable code of the source program.

The loader program performs the task of reallocating the codes after finding the physical RAM addresses available at a given instant. The ***loader*** is a part of the operating system and places codes into the memory after reading the '.exe' file.

A program called *locator* reallocates the linked file and creates a file for permanent location of codes in a standard format.

Segment combination

In addition to the linker commands, the assembler provides a means of regulating the way segments in different object modules are organized by the linker.

Segments with same name are joined together by using the modifiers attached to the SEGMENT directives. SEGMENT directive may have the form Segment name SEGMENT Combination-type where the combine-type indicates how the segment is to be located within the load module.

Segments that have different names cannot be combined and segments with the same name but no combine-type will cause a linker error. The possible combine-types are:

PUBLIC – If the segments in different modules have the same name and combine type PUBLIC, then they are concatenated into a single element in the load module. The ordering in the concatenation is specified by the linker command.

COMMON – If the segments in different object modules have the same name and the combine-type is COMMON, then they are overlaid so that they have the same starting address. The length of the common segment is that of the longest segment being overlaid.

STACK – If segments in different object modules have the same name and the combine type STACK, then they become one segment whose length is the sum of the lengths of the individually specified segments. In effect, they are combined to form one large stack.

AT – The AT combine-type is followed by an expression that evaluates to a constant which is to be the segment address. It allows the user to specify the exact location of the segment in memory.

MEMORY – This combine-type causes the segment to be placed at the last of the load module. If more than one segment with the MEMORY combine-type is being linked, only the first one will be treated as having the MEMORY combine type; the others will be overlaid as if they had COMMON combine-type.

```
Source module 1
 DATA      SEGMENT   COMMON
 DATA            ENDS
 CODE            SEGMENT   PUBLIC
 CODE            ENDS

Source module 2
DATA       SEGMENT   COMMON
           .
           .
DATA       ENDS
CODE       SEGMENT    PUBLIC
           .
           .
CODE       ENDS
```

Access to External Identifiers

If an identifier is defined in an object module, then it is said to be a *local* (or *internal) identifier* relative to the module. If it is not defined in the module but is defined in one of the other modules being linked, then it is referred to as an *external* (or *global*) *identifier* relative to the module.

In order to permit other object modules to reference some of the identifiers in a given module, the given module must include a list of the identifiers to which it will allow access. Therefore, each module in multi-module programs may contain two lists, one containing the external identifiers that can be referred to by other modules.

Two lists are implemented by the EXTRN and PUBLIC directives, which have the forms:

```
EXTRN Identifier: Type..., Identifier: Type
  and
PUBLIC        Identifier,..., Identifier
```

where the identifiers are the variables and labels being declared or as being available to other modules.

8. <u>Stacks</u>

How stacks are accessed in 8086 processor? Explain briefly.(Dec-2007)

The stack is a block of memory that may be used for temporarily storing the contents of the registers inside the CPU. It is a top-down data structure whose elements are accessed using the stack pointer (SP) which gets decremented by two as we store a data word into the stack and gets incremented by two as we retrieve a data word from the stack back to the CPU register.

The stack is essentially *Last-In-First-Out* (LIFO) data segment. This means that the data which is pushed into the stack last will be on top of stack and will be popped off the stack first.

The stack pointer is a 16-bit register that contains the offset address of the memory location in the stack segment. T Stack Segment register (SS) contains the base address of the stack segment in the memory.

The Stack Segment register (SS) and Stack pointer register (SP) together address the stacktop as explained below:

SS $\Longrightarrow$ 5000H

SP $\Longrightarrow$ 2050H

If the stack top points to a memory location 52050H, it means that the location 52050H is already occupied with the previously pushed data. The next 16 bit push operation will decrement the stack pointer by two, so that it will point to the new stack-top 5204EH and the decremented contents of SP will be 204EH. This location will now be occupied by the recently pushed data.

Thus for a selected value of SS, the maximum value of SP=FFFFH and the segment can have maximum of 64K locations. If the SP starts with an initial value of FFFFH, it will be decremented by two whenever a 16-bit data is pushed onto the stack. After successive push operations, when the stack pointer contains 0000H, any attempt to further push the data to the stack will result in stack overflow.

After a procedure is called using the CALL instruction, the IP is incremented to the next instruction. Then the contents of IP, CS and flag register are pushed automatically to the stack. The control is then transferred to the specified address in the CALL instruction i.e. starting address of the procedure. Then the procedure is executed.

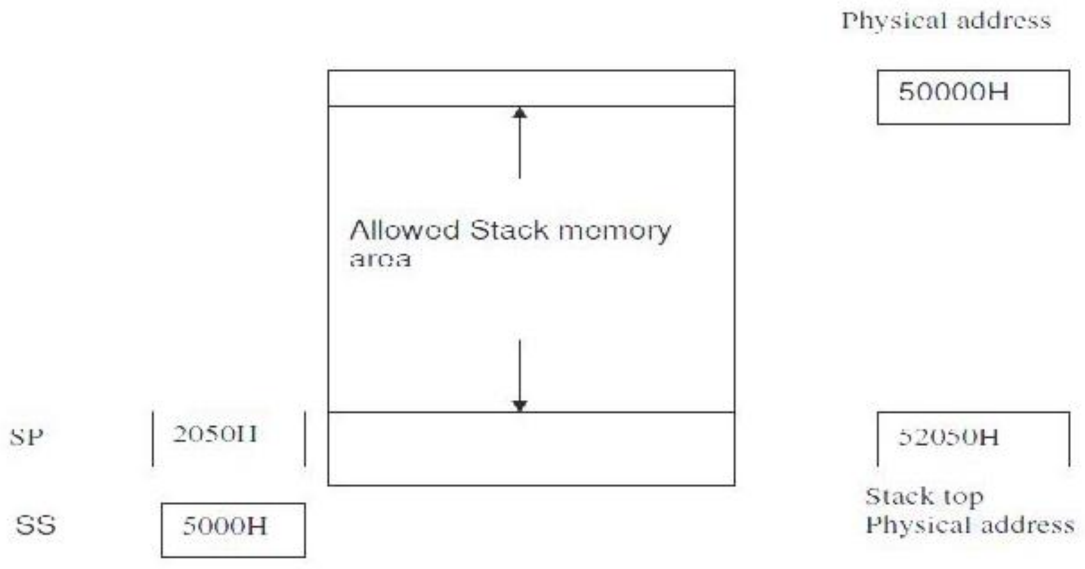


Fig. Stack-top address calculation

9. PROCEDURES & MACROS

Macros: AUQ: Define macro. Explain how macros are constructed in ASM-86 with example. (Dec-2010, May2006,10,11)

Macros look like procedures, but they exist only until our code is compiled, after compilation all macros are replaced with real instructions. If we declared a macro and never used it in out code, complier will simply ignore it.

```
Macro definition :

name MACRO [parameters,..]

        <statements>

ENDM
```

Example:

```
My Macro    MACRO P1,P2,P3
MOV AX, P1
MOV BX, P2
MOV CX, P3
ENDM
```

Advantages of macros

- Repeated small groups of instructions replaced by one macro
- Errors in macros are fixed only once, in the definition.
- Duplication of effort is reduced.

- In effect, new higher level instructions can be created
- Programming is made easier, less error prone
- Generally quicker in execute than subroutines.

Disadvantages of macros

In large programs, produce greater code size than procedures.

When to use Macros?

- To replaces small groups of instruction not worthy of subroutines.
- To create a higher instruction set for specific applications.
- To create compatibility with other computers,
- To replace code portions which are repeated often throughout the program.

Procedure (PROC)

This directive marks the start and end of a procedure block called label, the statements in the block can be called with they CALL instruction.

The directive PROC indicated the states of a procedure. The type of the procedure FAR of NEAR is to be specified after the directive, the type NEAR is used to call a procedure with is within the programmed module. The type FAT is used to call a procedure from some other program module. The PROC directive is used with ENDP directive to enclose a procedure.

PROC definition :

```
label  PROC [ [near / far ] ]

<Procedure instructions>

Label ENDP
```

Example:

```
WEST PROC  FAR
-
-
-
WEST ENDP
```

A procedure is a set of instructions that compute some value or take some action (such as printing or reading a character value). The definition of a procedure is very similar to the definition of an algorithm.

A procedure is a set of rules to follow which, if they conclude, produce some results. An algorithm is also such a sequence, but an algorithm is guaranteed to terminate whereas a procedure offers no such guarantee.

Nested Procedures

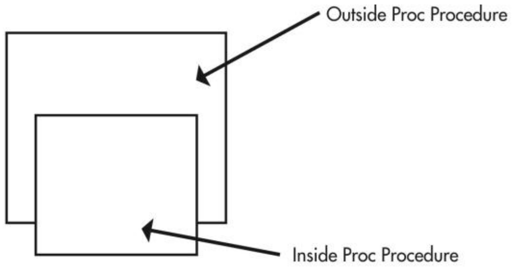
The nest procedure is one procedure definition may be totally enclosed inside another. The following is an example of such a pair of procedures:

```
OUTSIDEPROC       PROC  NEAR
JMP    ENDOFOUTSIDE
INSIDEPROC PROC  NEAR
MOC    AX, 0
RET
INSIDEPROC ENDP
ENDOFOUTSIDE:     CALL  INSIDEPROC
MOV    BX, 0
RET
OUTSIDEPROC       ENDP
```

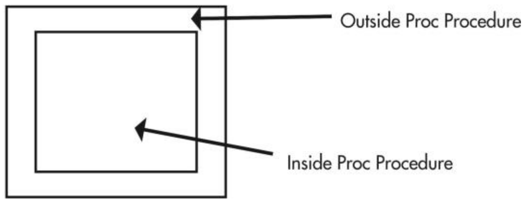
Whenever we nest one procedure within another, it must be totally contained within the nesting procedure. That is the PROC and ENDP statements for the nested procedure must lie between the PROC and ENDP directives of the outside, nesting procedure. The following is not legal.

```
OUTSIDEPROC       PROC  NEAR
*
*
*
INSIDEPROC PROC  NEAR
*
*
*
OUTSIDEPROC       ENDF
*
*
*INSIDEPROC       ENDP
```

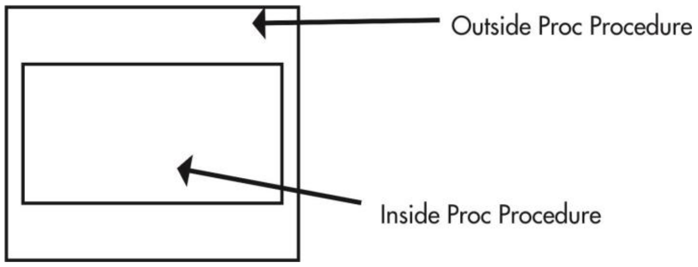
THE OUTISDE proc AND Inside PROC procedures overlap, they are not nested. If we attempt to create a set of procedures like this MASM would report a “block nesting error.



The only form acceptable to MASM



Besides fitting inside an enclosing procedure, PROC/ENDP groups must fit entirely within a segment. The ENDP directive must appear before the SEG ends statement since MyProc begins inside SEG. Therefore, procedures within segments must always take the form.



Not only can we nest procedures inside other procedures and segments, but we can nest segments inside other procedures and segments as well.

Difference between Macros and Procedures

1. To use a procedure use CALL instruction is needed. For example: CALL MyProc.

To use a macro, just type its name. For example, my Macro.

2. Procedure is located at some address in memory, and if use the same procedure 100 times, the CPU will transfer control to this part of the memory. The control will be return back to the program by RET Instruction. The stack is used to keep the return address. The CALL instruction takes about 3 bytes, so the size of the output executable file grows very insignificantly, no matter how many time the procedure is used.

Macro is expanded directly in program's code. So if use the same macro 100 times, the complier expands the macro 100 times, making the output executable file larger and larger, each time all instruction of a macro are inserted.

3. Use stack or any general purpose registers to pass parameters to procedure.

To pass parameters to macro, just type them after the macro name. For example :My Macro ,1.2.3

4. To mark the end of the macro ENDM directive is enough.

To mark the end of the procedure, type the name of the procedure before the ENDP directive.

Differentiate procedures and macros.

The difference between procedures and macros are given in the Table.

S.No.	PROCEDURES	MACROS
1	To use procedure use CALL and RET instructions are needed	To use a macro, just type its name
2	It occupies less memory	It occupies more memory
3	Stack is used	Stack is not used
4	To mark the end of the procedure, type the name of the procedure before the ENDP directive.	To mark the end of the macro ENDM directive is enough
5	Overhead time is required to call the procedure and return to the calling program	No overhead time during the execution.

10. <u>INTERRUPTS AND INTERRUPT SERVICE ROUTINES</u>

AUQ: What are the interrupts in 8086? Explain interrupt related service routines.(Dec-2007,08,12, May-2007,08,11,12,13,15, May 2016, May 2017)

Interrupts:

A signal to the processor to halt its current operation and immediately transfer control to an interrupt service routine is called as interrupt. Interrupts are triggered either by hardware as when the keyboard detects a key press, or by software, as when a program executes the INT instruction.

- Interrupts are triggered by different hardware, these are called hardware interrupts.
- To make a software interrupt there is an INT instruction, it has very simple syntax : INT

Value.

Where value can be a number between 0 to 255 (or 00 to FF h).

Interrupt Service Routings (ISRs)

IST is a routing that receives processor control when a specific interrupt occurs.

The 8086 will directly call the service routing for 256 vectored interrupts without any software processing. This is in contrast to non vectored interrupts that transfer control directly to a single interrupt service routine, regardless of the interrupt source.

The 8086 provides a 256 entry interrupt vector table beginning at address 0:0 in memory. This is a 1K table containing 256 4-byte entries. Each entry in the table contains a segmented address that points at the interrupt service routing in memory. Generally, interrupts referred by their index into this table, so interrupt zero's address (vector) is at memory location 0:0 interrupt one's vector is at address 0:4, interrupt two's vector is at address 0:8, etc.

Interrupt vector table:

It is a table maintained by the operating system. It contains addresses (vectors) of current interrupt service routine. When an interrupt occurs, the CPU branches to the address in the table that corresponds to the interrupt's number.

When an interrupt occurs, regardless of source, the 8086 does the following :

1. The CPU pushes the flags register onto the stack.
2. The CPU pushes a far return address (segment: offset) onto the stack, segment value first.
3. The CPU determines the cause of the interrupt (i.e, the interrupt number) and fetches the

four byte interrupt vector from address 0 : vector x 4 (0:0, 0;4, 0:8 etc)

4. The CPU transfers control to the routine specific by the interrupt vector table entry.

After the completion of these steps, the interrupt service routine takes control. When the interrupt service routine wants to return control, it must execute an IRET (interrupt return) instruction. The interrupt return pops the far return address and the flags of the stack.

	Available Interrupt pointers (224)	3FFH	Type 255 pointer
			(Available)
			:
			:
		080H	Type 32 pointer
			(Available)
	Reserved Interrupts (27)	07FH	Type 31 pointer
			(Reserved)
			;
			Type 5 pointer
			(Reserved)
	Dedicated Interrupt Pointers(6)		Type 4 pointer
		014H	overflow

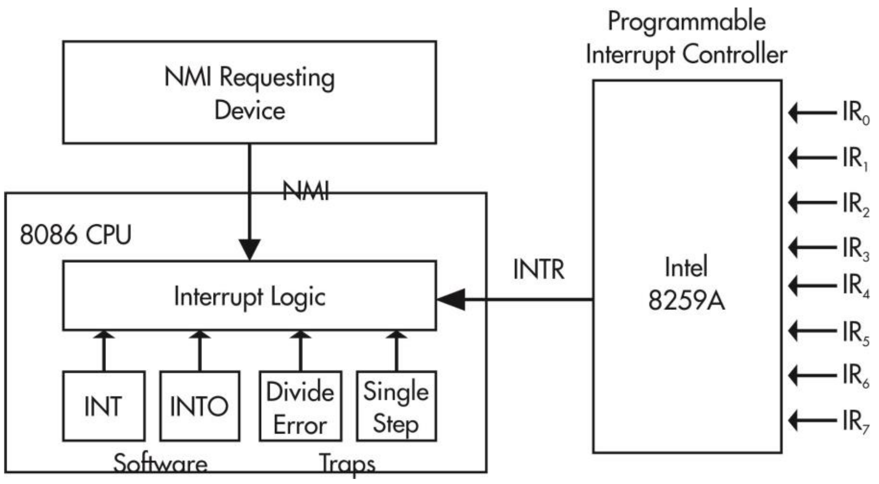
		010h	Type 3 pointer 1Byte INT instruction
		00CH	Type 2 Pointer NonMaskable
		008H	TYPE 1 Pointer Single step
CS Base address		004H	TYPE 0 Pointer Divide by zero
IP offset		000H	

Types of Interrupts :

1. Hardware Interrupt – External used INTR and NMI
2. Software Interrupt – Internal – from INT or INTO
3. Processor Interrupt – Traps and 10 Software Interrupts

External – generated outside the CPU by other hardware, (INTR, NMI)

Internal – generated within CPU as a result of an instruction of operation (INT, INTO, Divide error and single step)



Dedicated Interrupts:

(i) <u>Divide Error Interrupt (Type 0)</u>

This interrupt occurs automatically following the execution of DIV or IDIV instruction when the quotient exceeds the maximum value that the division instruction allows.

(ii) <u>Single Step Interrupt (Type 1)</u>

This interrupt occurs automatically after execution of each instruction when the Trap Flag (TF) is set of 1. It is used to execute programs one instruction at a time, after which an interrupt is requested.

Following the ISR, the next instruction is executed and another single stepping interrupt request occurs.

(iii)Non Maskable Interrupt (Type 2)

It is the highest priority hardware interrupt that triggers on the positive edge. This interrupt occurs automatically when it received a low-to-high transition on its NMI input pin. This interrupt cannot be disabled or masked. It is used to save program data or processor status in case of system power failure.

(iv)Breakpoint Interrupt (Type 3)

This interrupt is used to set break point is software debugging programs.

(v) Overflow Interrupt (Type 4)

This interrupt is initiated by INTO (Interrupt on Overflow) instruction. It is used to check overflow condition after any signed arithmetic operation in the system. The overflow flag (OF) will be set if the signed arithmetic operation generates a result whose size is larger than the size of destination register or memory location. At this time overflow interrupt is used to indicate an error condition.

Software Interrupts (INT n)

The software interrupts are non maskable interrupts. They are higher priority than hardware interrupts.

The software interrupts are called within the program using the instruction INT n. Here 'n' means value and is in the range of 0 to 255. These interrupts are useful for debugging, testing ISRs and calling procedures.

Hardware Interrupts

INTR and NMI are called hardware interrupts. INTR is maskable and NMI is non maskable interrupts.

INTR interrupts (type 0-255) can be used to interrupt a program execution. This interrupt is implemented by using two pins: INTR and INTA. This interrupts can be enabled or disabled by STI (IF=1) or (IF=0) respectively.

Interrupt Priority

The priority of interrupts of 8086 is shown in Table. The software interrupts except single step interrupt have the highest priority; followed by NMI, followed by INTR. Single step interrupt has the least priority. The 8086 checks for internal interrupts before for any hardware interrupt. Therefore software interrupts have higher priority than hardware interrupts.

Interrupt	**Priority**
INT n, INT 0, Divide Error	Highest
NMI	↓
INTR	↓
Single Step	Lowest

UNIT-II
8086 SYSTEM BUS STRUCTURE

8086 signals – Basic configurations – System bus timing –System design using 8086 – IO programming – Introduction to Multiprogramming – System Bus Structure - Multiprocessor configurations – Coprocessor, Closely coupled and loosely Coupled configurations – Introduction to advanced processors.

- ✓ The 8086 Microprocessor is a 16-bit CPU.
- ✓ Available clock rates: 5, 8 and 10MHz
- ✓ Packages: 40 pin CERDIP or plastic package
- ✓ Operates in single processor or multiprocessor configurations
- ✓ Modes of operation: Minimum mode (single processor mode) and Maximum mode (multiprocessor mode) configuration.

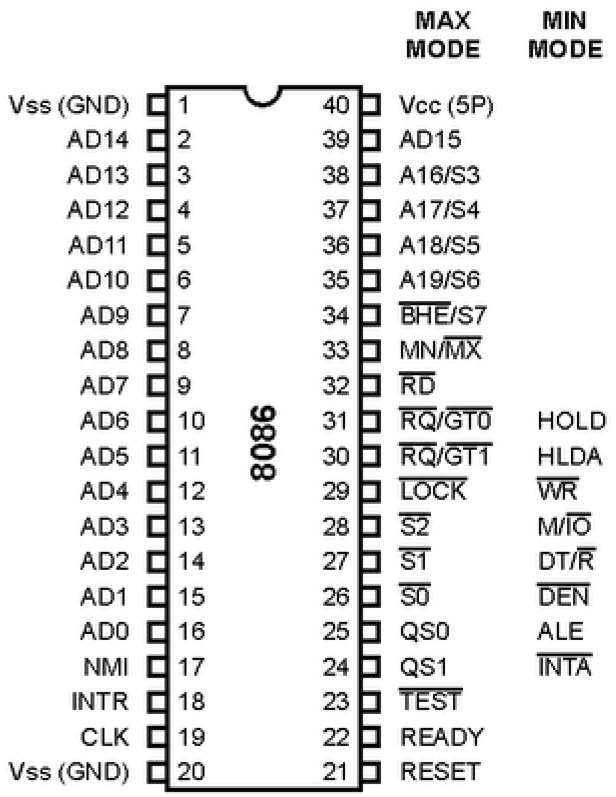
SIGNAL DESCRIPTION OF 8086

AUQ: Explain the signal used in 8086 processor. (Dec 2003,06,07,09,10,13, May 2006,07,08,09,11)

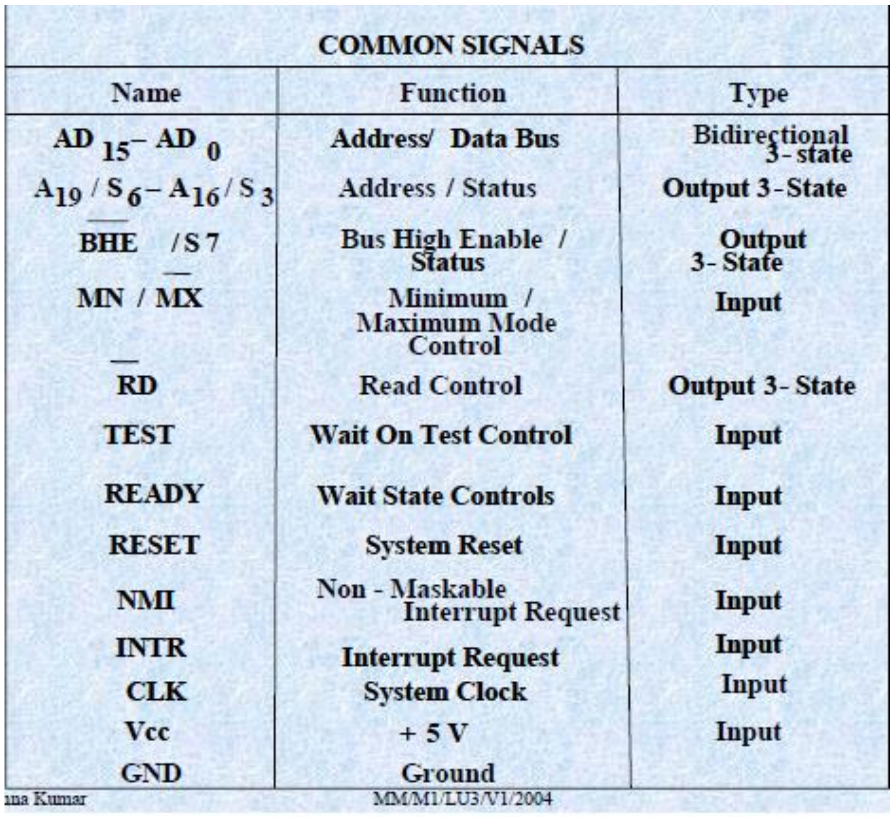
The 8086 signals can be categorized in three groups.

- ➢ Signals having common function in minimum and maximum mode.
- ➢ Signals having special functions in minimum mode
- ➢ Signals having special functions in maximum mode

PIN DIAGRAM



Signals having common function in minimum and maximum mode



COMMON SIGNALS		
Name	**Function**	**Type**
$AD_{15} - AD_0$	Address/ Data Bus	Bidirectional 3-state
$A_{19}/S_6 - A_{16}/S_3$	Address / Status	Output 3-State
$\overline{BHE}$ /S 7	Bus High Enable / Status	Output 3-State
MN / $\overline{MX}$	Minimum / Maximum Mode Control	Input
$\overline{RD}$	Read Control	Output 3-State
TEST	Wait On Test Control	Input
READY	Wait State Controls	Input
RESET	System Reset	Input
NMI	Non - Maskable Interrupt Request	Input
INTR	Interrupt Request	Input
CLK	System Clock	Input
Vcc	+ 5 V	Input
GND	Ground	

ına Kumar MM/M1/LU3/V1/2004

$AD_{15} – AD_0$

- These are time multiplexed address and data lines. They act as address lines during first part of machine cycle and data lines in later part.

$A_{19}/S_6 – A_{16}/S_3$

- These are time multiplexed address and status lines. They act as address lines during first part of machine cycle and status lines in later part.
- These are most significant address lines for memory operations. During I/O operations these lines are low.
- The status signals S_4 and S_3 indicate which segment registers is being used for memory access.
- The status of interrupt enable flag bit will be displayed on S_5.
- The status line S_6 is always low.

S_4	S_3	Indications
0	0	ES
0	1	SS
1	0	CS
1	1	DS

$\overline{BHE}/S_7$- Bus high enable/ status

- Low signal on $\overline{BHE}$ indicates access to higher order memory banks, otherwise access is to only lower order memory banks.
- $\overline{BHE}$ and A_0 decide the memory bank and type of access.
- S_7 has no function.

$\overline{BHE}$	A_0	**Indications**
0	0	Both higher and lower order banks for word read/ write
0	1	Higher order bank for byte read/ write
1	0	Lower order bank for byte read/ write
1	1	None

$\overline{RD}$

- Read control signal
- $\overline{RD}$ is low when 8086 is receiving the data from memory or I/O.

READY

- Wait state request signal.
- A HIGH on READY input causes the 8086 to extend the machine cycle by inserting wait states.

$\overline{TEST}$

- This input is examined by WAIT instruction.
- If the $\overline{TEST}$ input goes low, execution will continue, else, the processor remains in idle state.

INTR

- This is level triggered input.
- INTR is sampled during the last clock cycle of each instruction to determine the availability of request.
- These interrupts can be masked internally by resetting the interrupt enable flag.

NMI

- NMI (Non-Maskable interrupt) is positive edge triggered non-maskable interrupt request.

CLK

- CLK is clock signal from external crystal oscillator.
- 8086 requires clock signal with 33% duty cycle.

RESET

- System reset signal must be high for atleast 4 clock periods to cause reset.
- Reset operation takes about 10 clock periods.

VCC

- +5V supply with ±5% tolerance.

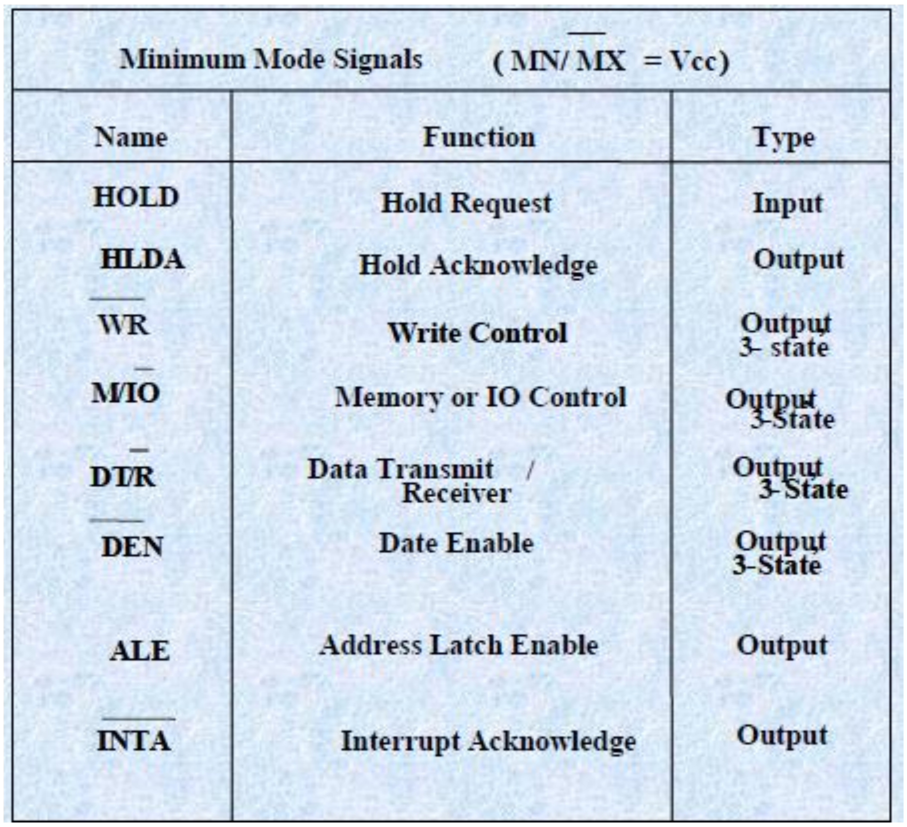
GND

- Ground for internal circuits.

$\mathrm{MN}/\overline{MX}$

- High on this pin selects minimum mode and low signal selects maximum mode.

<u>Signals having special functions in minimum mode</u>



Minimum Mode Signals ($\mathrm{MN}/\overline{\mathrm{MX}}$ = Vcc)

Name	Function	Type
HOLD	Hold Request	Input
HLDA	Hold Acknowledge	Output
$\overline{\mathrm{WR}}$	Write Control	Output 3- state
$\mathrm{M}/\overline{\mathrm{IO}}$	Memory or IO Control	Output 3-State
$\mathrm{DT}/\overline{\mathrm{R}}$	Data Transmit / Receiver	Output 3-State
$\overline{\mathrm{DEN}}$	Date Enable	Output 3-State
ALE	Address Latch Enable	Output
$\overline{\mathrm{INTA}}$	Interrupt Acknowledge	Output

ALE

- Address latch enable.
- High on this pin indicates valid address on address/data bus.

$\overline{\mathrm{WR}}$

- Write control signal.
- $\overline{WR}$is low when 8086 sends the data to memory or I/O.

$\mathrm{M}/\overline{\mathrm{IO}}$

- $\mathrm{M}/\overline{\mathrm{IO}}$ = High, indicates memory access.

- $M/\overline{IO}$ = low, indicates I/O access.

$\overline{INTA}$

- $\overline{INTA}$ is the acknowledgement for the interrupt request on INTR pin.
- It is pulsed low in two consecutive bus cycles.
- First pulse indicates interrupt acknowledgement.
- During second pulse, external logic puts the interrupt type on data bus.

$DT/\overline{R}$

- Data transmit/receive.
- This signal, when high indicates data is being transmitted by 8086.
- The low signal indicates that 8086 is receiving the data.

$\overline{DEN}$

- Data bus enable.
- This signal, when low indicates that the 8086 processors address/data bus is used as data bus.
- It is used to enable data buffers.

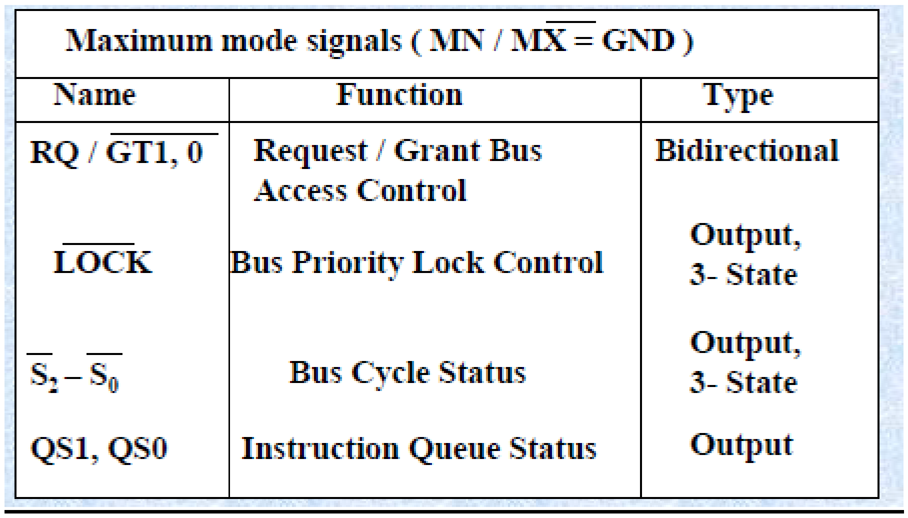
HOLD

- HOLD signal when high indicates another master has requested for direct memory access.
- When HOLD becomes low, it indicates that direct memory access is no more required.

HLDA

- The microprocessor sends high signal on HLDA to indicate acknowledgement of DMA request. It then tristates the buses and control.
- When HOLD becomes low, the microprocessor makes HLDA low and regains the control of buses.

<u>Signals having special functions in maximum mode</u>



Maximum mode signals ($MN / \overline{MX}$ = GND)		
Name	**Function**	**Type**
$RQ / \overline{GT1, 0}$	Request / Grant Bus Access Control	Bidirectional
$\overline{LOCK}$	Bus Priority Lock Control	Output, 3- State
$\overline{S_2} - \overline{S_0}$	Bus Cycle Status	Output, 3- State
QS1, QS0	Instruction Queue Status	Output

$\overline{\text{LOCK}}$

- This signal indicates that an instruction with lock prefix is being executed and the bus is not to be used by any other processor.

$\overline{RQ}/\overline{GT}_1$, $\overline{RQ}/\overline{GT}_0$

- In maximum mode DMA request is received and acknowledged using these signals.
  $\overline{RQ}/\overline{GT}_0$has highest priority compared to $\overline{RQ}/\overline{GT}_1$

$\bar{S}_2, \bar{S}_1, \bar{S}_0$

- These are the status lines which indicate the type of operation being carried out by the processor.

$\bar{S}_2$	$\bar{S}_1$	$\bar{S}_0$	**Control functions**
0	0	0	Interrupt acknowledge
0	0	1	I/O read
0	1	0	I/O write
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	No operation

$\overline{QS}_1, \overline{QS}_0$

- These two signals are decoded to provide instruction queue status.

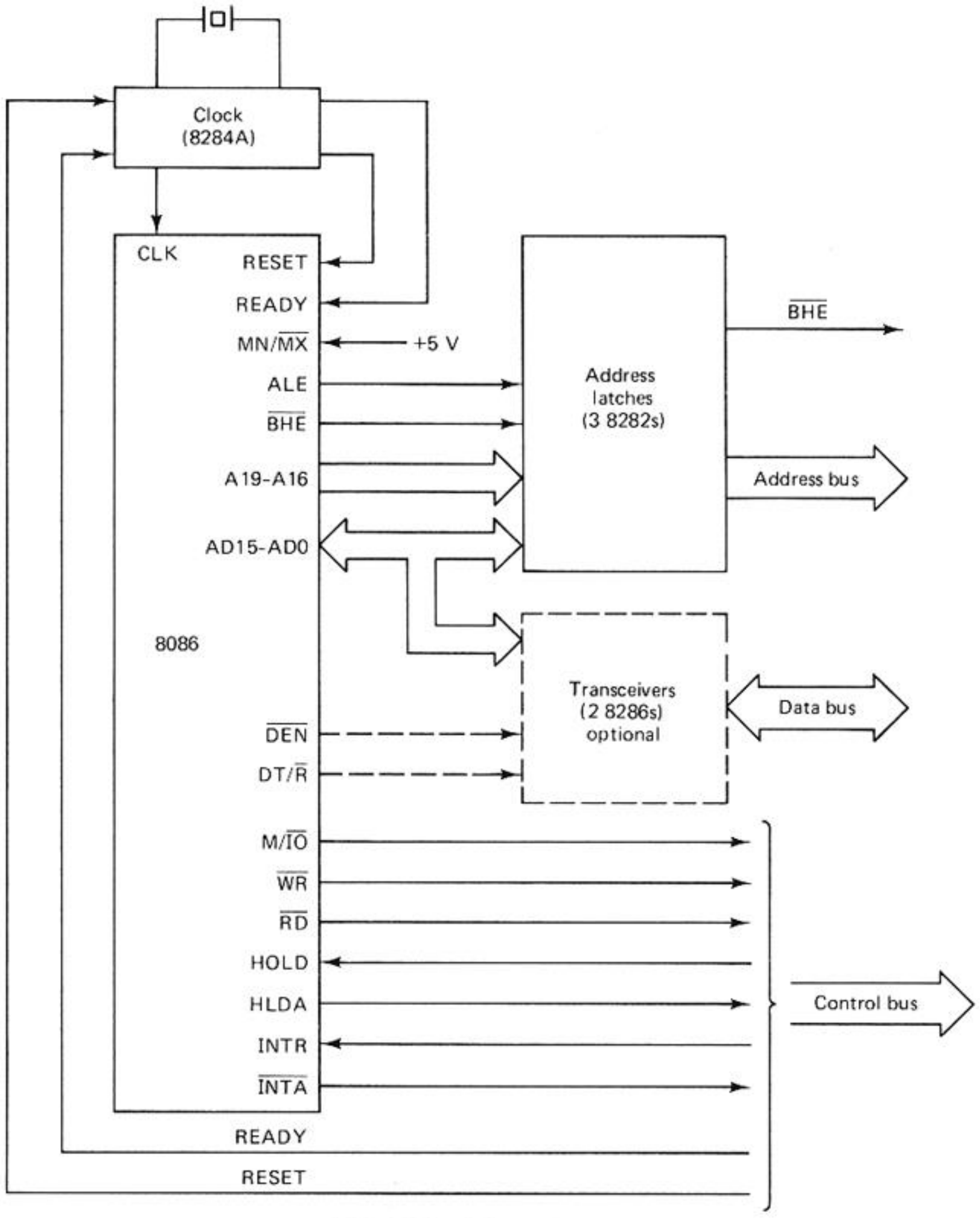
$\overline{QS}_1$	$\overline{QS}_0$	**Indications**
0	0	Queue is in idle state
0	1	First byte of opcode has entered queue
1	0	Queue empty
1	1	Subsequent byte of opcode has entered queue

<u>Basic configurations :</u>

1. Minimum Mode configuration:

AUQ: Explain with neat diagram minimum mode configuration of 8086 system. (Dec 2006,08, May 2006,07)

- ✓ A processor is in minimum mode when its MN / /MX pin is strapped to +5V. In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX pin to logic 1.
- ✓ In this mode, all the control signals are given out by the microprocessor chip itself.
- ✓ There is a single microprocessor in the minimum mode system.
- ✓ The remaining components in the system are latches, transreceivers, clock generator, memory and I/O devices. Some type of chip selection logic may be required for selecting memory or I/O devices, depending upon the address map of the system.
- ✓ Latches are generally buffered output D-type flip-flops like 74LS373 or 8282.
- ✓ They are used for separating the valid address from the multiplexed address/data signals and are controlled by the ALE signal generated by 8086.
- ✓ Transreceivers are the bidirectional buffers and sometimes they are called as data amplifiers. They are required to separate the valid data from the time multiplexed address/data signals.
- ✓ They are controlled by two signals namely, DEN and DT/R.
- ✓ The DEN signal indicates the direction of data, i.e. from or to the processor. The system contains memory for the monitor and users program storage.
- ✓ The clock generator generates the clock from the crystal oscillator and then shapes it and divides to make it more precise so that it can be used as an accurate timing reference for the system.
- ✓ The clock generator also synchronizes some external signal with the system clock. The general system organisation is as shown in below fig.



Note: In an 8088 system $\overline{BHE}$ is $\overline{SSO}$, M/$\overline{IO}$ is IO/$\overline{M}$, and only one 8286 is needed.

Figure 8-4 Minimum mode system.

2. **Maximum Mode configuration:**

AUQ: Explain with neat diagram maximum mode configuration of 8086 system. (Dec 2007)

A processor is in maximum mode when its MN / /MX pin is grounded. The maximum mode definitions of pins 24 through 31 are given in table and a typical maximum mode configuration is shown in Fig.

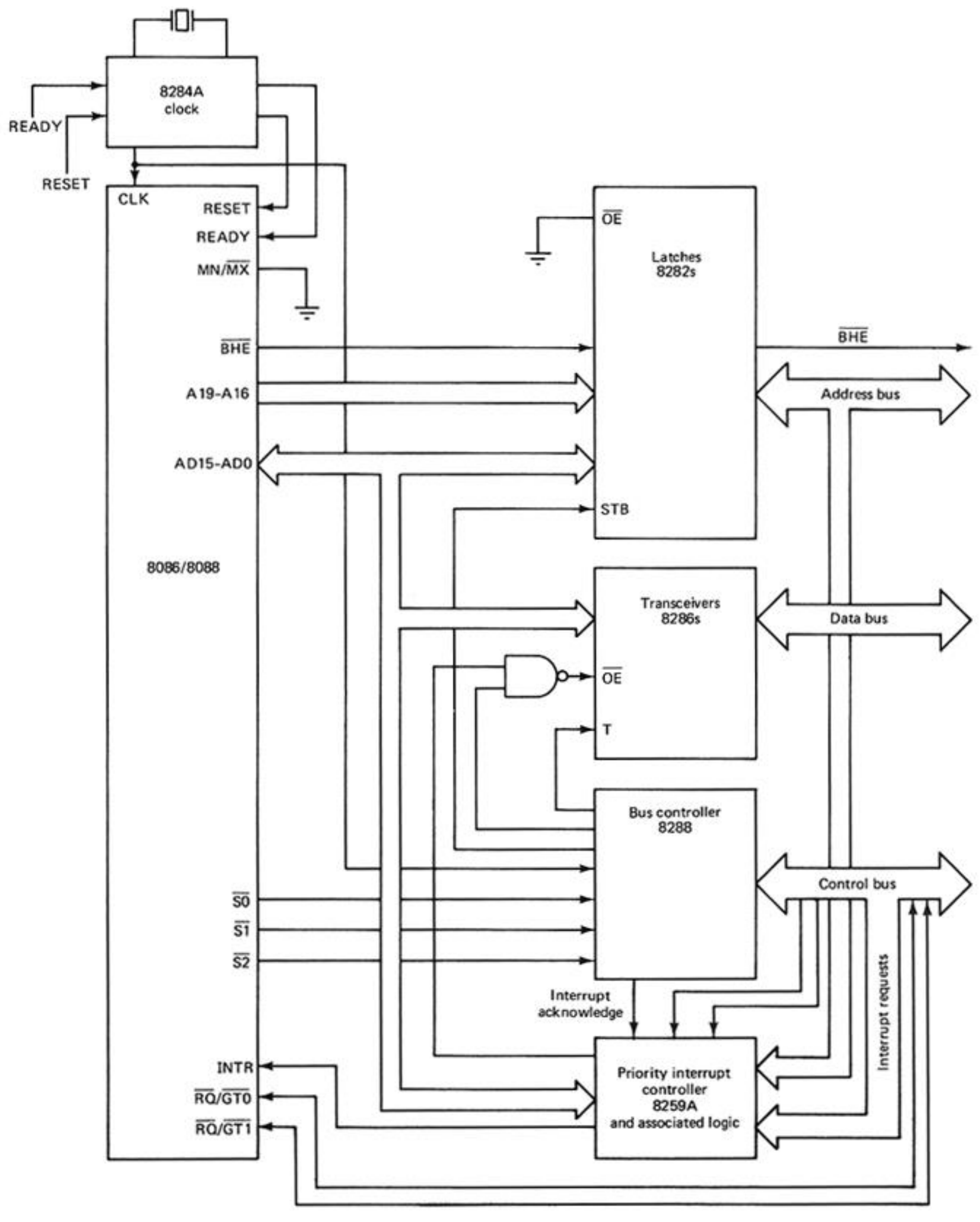
The circuitry is for converting the status bits /S0, /S1 and /S2 into the I/O and memory transfer signals needed to direct data transfers and for controlling the 8282 latches and 8286 transceivers.

It is normally implemented with an Intel 8288 bus controller. Also included in the system is an interrupt priority management device: however, its presence is optional.

- ✓ In the maximum mode, there may be more than one microprocessor in the system configuration. The components in the system are same as in the minimum mode system.
- ✓ The basic function of the bus controller chip IC8288, is to derive control signals like RD and WR (for memory and I/O devices), DEN, DT/R, ALE etc. using the information by the processor on the status lines.
- ✓ The bus controller chip has input lines S2, S1, S0 and CLK. These inputs to 8288 are driven by CPU. The process to be activated for this combination is listed below.
- ✓ It derives the outputs ALE, DEN, DT/R, MRDC, MWTC,AMWC, IORC, IOWC and AIOWC. The AEN, IOB and CEN pins are specially useful for multiprocessor systems.
- ✓ AEN and IOB are generally grounded. CEN pin is usually tied to +5V. The significance of the MCE/PDEN output depends upon the status of the IOB pin.
- ✓ If IOB is grounded, it acts as master cascade enable to control cascade 8259A, else it acts as peripheral data enable used in the multiple bus configurations.

The HOLD and HLDA pins become the /RQ / /GT0 and /RQ / /GT1 pins. Both bus requests and bus grants can be given through each of these pins. They are exactly the same except that if requests are seen on both pins at the same time, then one on /RQ / /GT0 is given higher priority. A request consists of a negative puls arriving before the start of the current bus cycle. The grant is negative puls that is issued at the beginning of the current bus cycle provided that:

1. The previous bus transfer was not the low byte of a word to or from an odd address if the CPU is an 8086. For 8088, regardless of the address alignment the grant signal will not be sent until second byte of a word reference is accessed.
2. The first pulse of an interrupt acknowledgement did not occure during the previous bus cycle.
3. An instruction with a LOCK prefix is not being executed.
4. If condition 1 or 2 is not met, then the grant will not be given until the next bus cycle and if condition 3 is not met, the grant will wait until the locked instruction is completed. In response to the grant the three-state pins are put in their high-impedance state and the next bus cycle will be given to the requesting master.



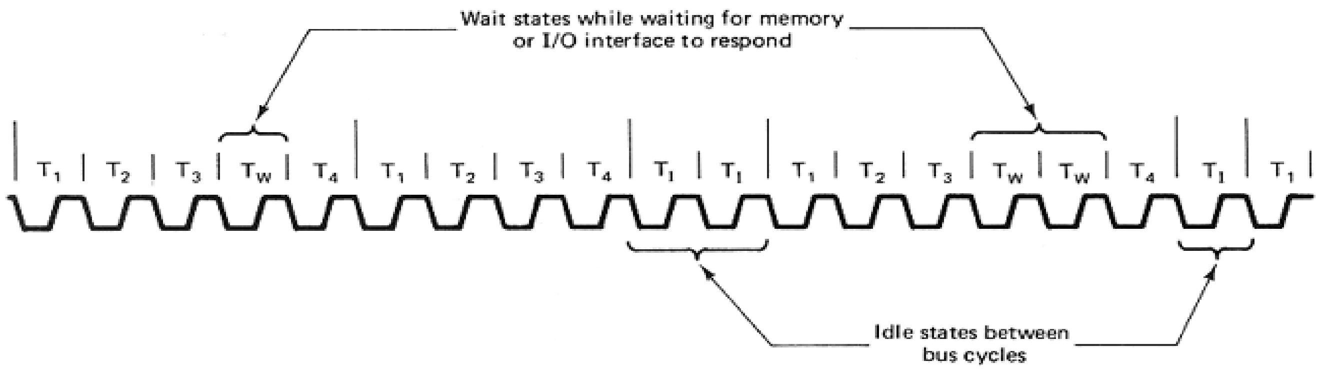
Note: $\overline{BHE}$ is not present in an 8088 system

Figure 8-9 Typical maximum mode configuration.

Read Write Timing Diagram

AUQ: Draw and explain the timing diagram of different cycle in 8086 processor. (Dec 2007, May 2009,13, Dec 2016, May 2017)

The typical sequence of bus cycles is shown below;


Wait states while waiting for memory or I/O interface to respond
T1 T2 T3 TW T4 T1 T2 T3 T4 T1 T1 T1 T2 T3 TW TW T4 T1 T1
Idle states between bus cycles

Bus timing for Minimum Mode:

- ✓ The opcode fetch and read cycles are similar. Hence the timing diagram can be categorized in two parts, the first is the timing diagram for read cycle and the second is the timing diagram for write cycle.
- ✓ The read cycle begins in T1 with the assertion of address latch enable (ALE) signal and also M / IO signal. During the negative going edge of this signal, the valid address is latched on the local bus.
- ✓ The BHE and A0 signals address low, high or both bytes. From T1 to T4 , the M/IO signal indicates a memory or I/O operation.• At T2, the address is removed from the local bus and is sent to the output. The bus is then tristated. The read (RD) control signal is also activated in T2.
- ✓ The read (RD) signal causes the address device to enable its data bus drivers. After RD goes low, the valid data is available on the data bus.
- ✓ The addressed device will drive the READY line high. When the processor returns the read signal to high level,the addressed device will again tristate its bus drivers.

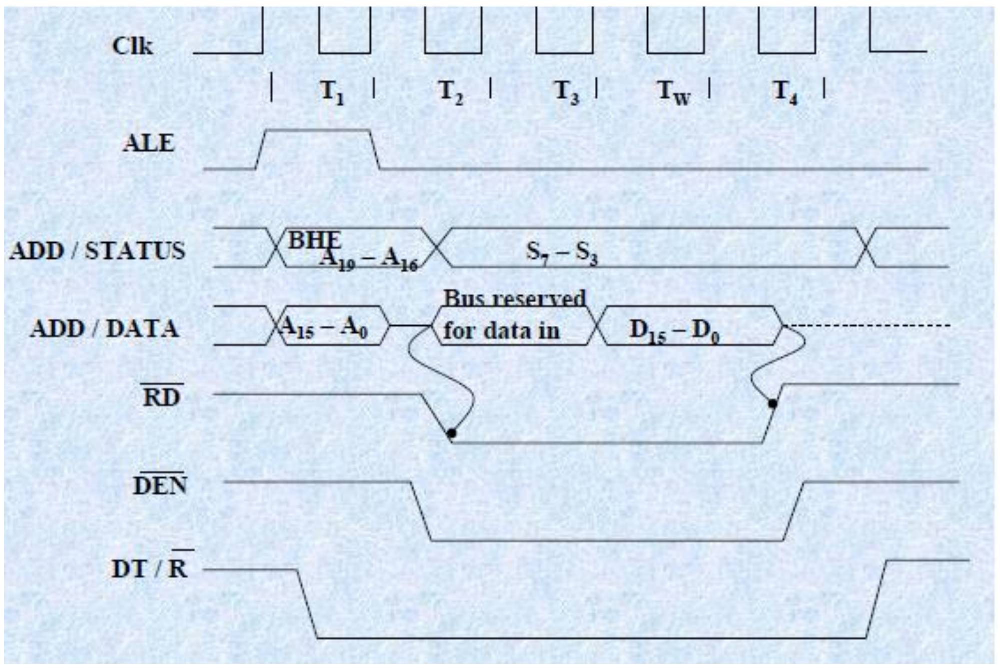


Figure:Readcycle timing diagarm of minimum mode8086

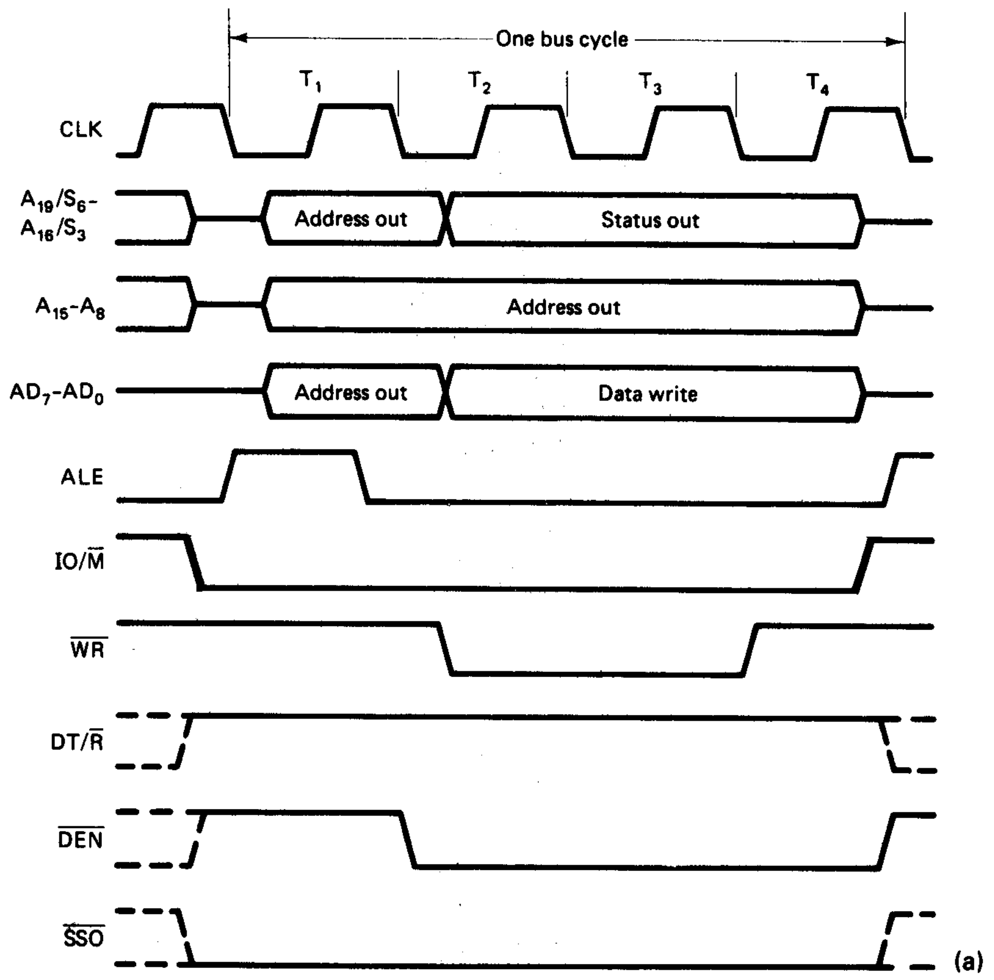
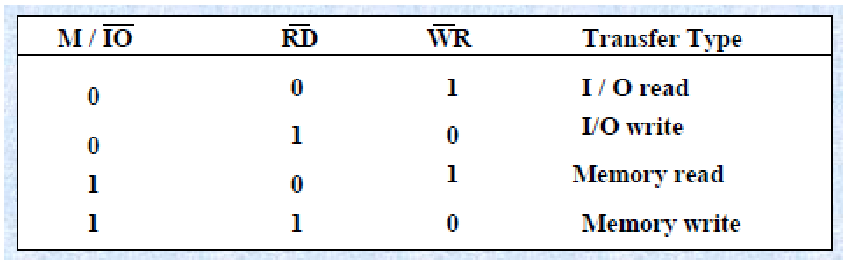


Figure: Write cycle timing diagarm of minimum mode8086

- A write cycle also begins with the assertion of ALE and the emission of the address. The M/IO signal is agai asserted to indicate a memory or I/O operation. In T2, after sending the address in T1, the processor sends th data to be written to the addressed location.
- The data remains on the bus until middle of T4 state. The WR becomes active at the beginning of T2 (unlike RD is somewhat delayed in T2 to provide time for floating).
- The BHE and A0 signals are used to select the proper byte or bytes of memory or I/O word to be read or write.
- The M/IO, RD and WR signals indicate the type of data transfer as specified in table below.



M / $\overline{IO}$	$\overline{RD}$	$\overline{WR}$	Transfer Type
0	0	1	I / O read
0	1	0	I/O write
1	0	1	Memory read
1	1	0	Memory write

Bus Timing for Maximum Mode:

- ✓ The maximum mode system timing diagrams are divided in two portions as read (input) and write (output) timing diagrams.
- ✓ The address/data and address/status timings are similar to the minimum mode.
- ✓ ALE is asserted in T1, just like minimum mode. The only difference lies in the status signal used and the available control and advanced command signals.
- ✓ Here the only difference between in timing diagram between minimum mode and maximum mode is the status signals used and the available control and advanced command signals.
- ✓ R0, S1, S2 are set at the beginning of bus cycle.8288 bus controller will output a pulse as on the ALE and apply a required signal to its DT / R pin during T1.
- ✓ In T2, 8288 will set DEN=1 thus enabling transceivers, and for an input it will activate MRDC or IORC. These signals are activated until T4. For an output, the AMWC or AIOWC is activated from T2 to T4 and MWTC or IOWC is activated from T3 to T4.
- ✓ The status bit S0 to S2 remains active until T3 and become passive during T3 and T4**.**
- ✓ If reader input is not activated before T3, wait state will be inserted between T3 and T4.

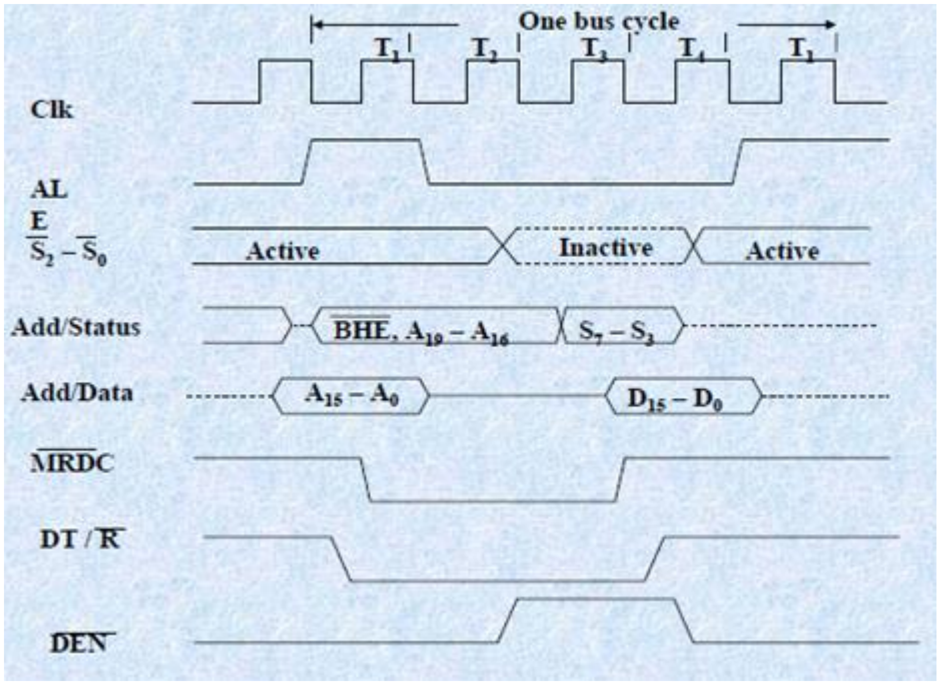


Figure:Memory read timing diagram in maximum mode

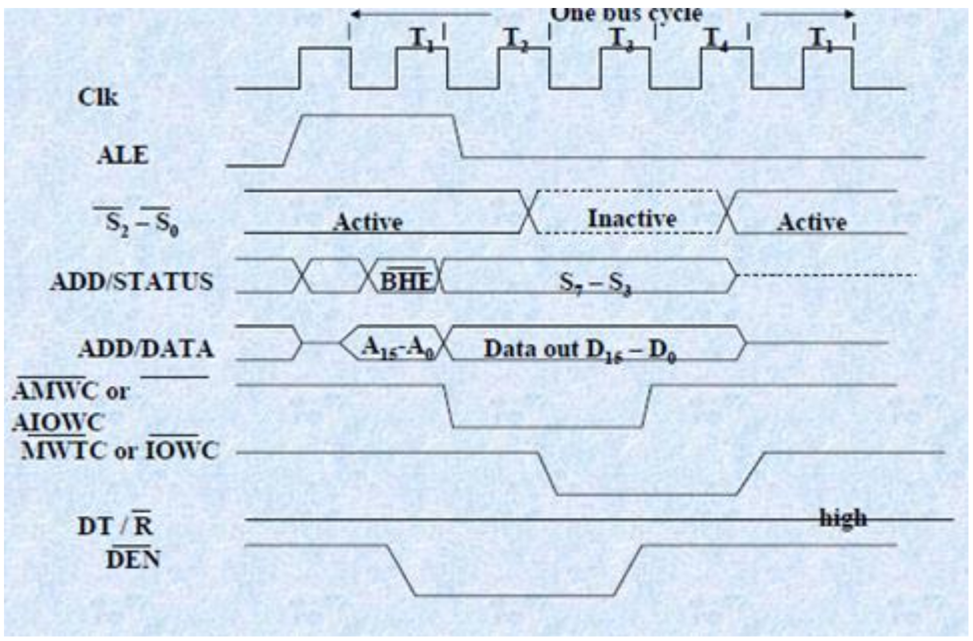


Figure: Memory write timing diagram in maximum mode

IO programming

With example explain the input output program concepts in 8086.

On the 8086, all programmed communications with the I/O ports is done by the IN and OUT instructions defined in Fig. 6-2.

✓ IN and OUT instructions

Name	Mnemonic and Format	Description
Input		
Long form, byte	IN AL, PORT	(AL) <- (PORT)
Long form, word	IN AX, PORT	(AX) <- (PORT+1: PORT)
Short form, byte	IN AL, DX	(AL) <- ((DX))
Short form, word	IN AX, DX	(AX) <- ((DX) + 1: (DX))
Output		
Long form, byte	OUT PORT, AL	(PORT) <- (AL)
Long form, word	OUT PORT, AX	(PORT+1: PORT) <- (AX)
Short form, byte	OUT DX, AL	((DX)) <- (AL)
Short form, word	OUT DX, AX	((DX)+1: (DX)) <- (AX)

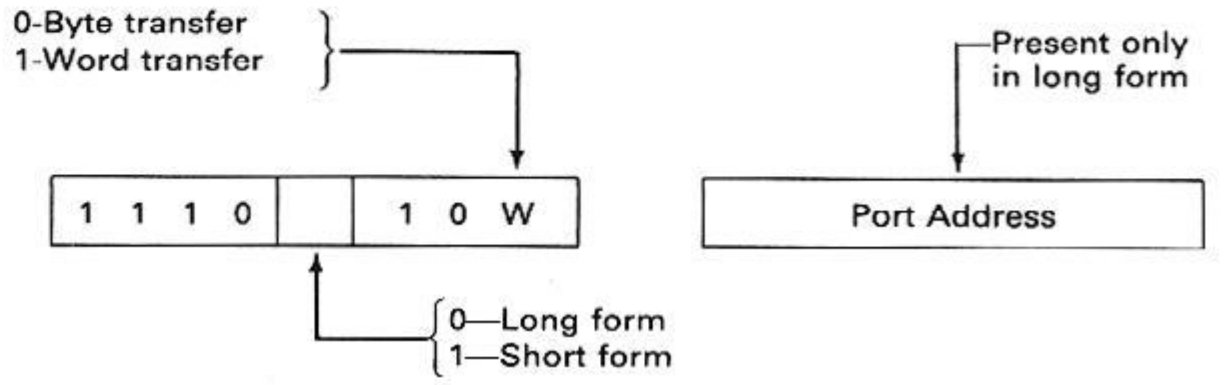
Note: PORT is a constant ranging from 0 to 255

Flags: No flags are affected

Addressing modes: Operands are limited as indicated above.

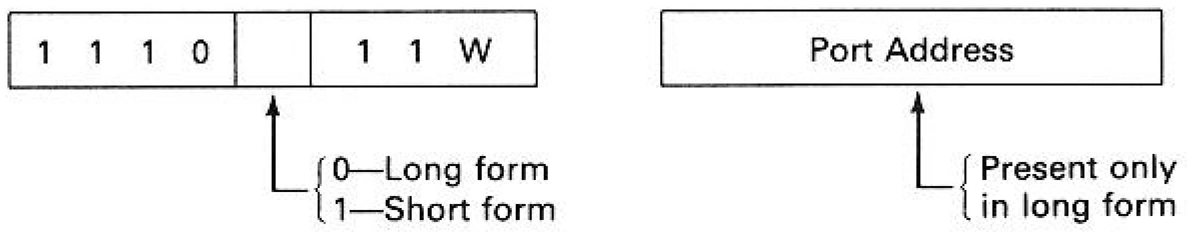
If the second operand is DX, then there is only one byte in the instruction and the contents of DX are used as the port address.

Unlike memory addressing, the contents of DX are not modified by any segment register. This allows variable access to I/O ports in the range 0 to 64K. The machine language code for the IN instruction is:



Although AL or AX is implied as the first operand in an IN instruction, either AL or AX must be specified so that the assembler can determine the W-bit.

Similar comments apply to the OUT instruction except that for it the port address is the destination and is therefore indicated by the first operand, and the second operand is either AL or AX. Its machine code is:



Note that if the long form of the IN or OUT instruction is used the port address must be in the range 0000 to 00FF, but for the short form it can be any address in the range 0000 to FFFF (i.e. any address in the I/O address space). Neither IN nor OUT affects the flags.

The IN instruction may be used to input data from a data buffer register or the status from a status register. The instructions

```
IN AX, 28H
MOV DATA_WORD, AX
```

would move the word in the ports whose address are 0028 and 0029 to the memory location DATA_WORD.

PROGRAMMED I/O

Programmed I/O consists of continually examining the status of an interface and performing an I/O operation with the interface when its status indicates that it has data to be input or its data-out buffer register is ready to receive data from the CPU.

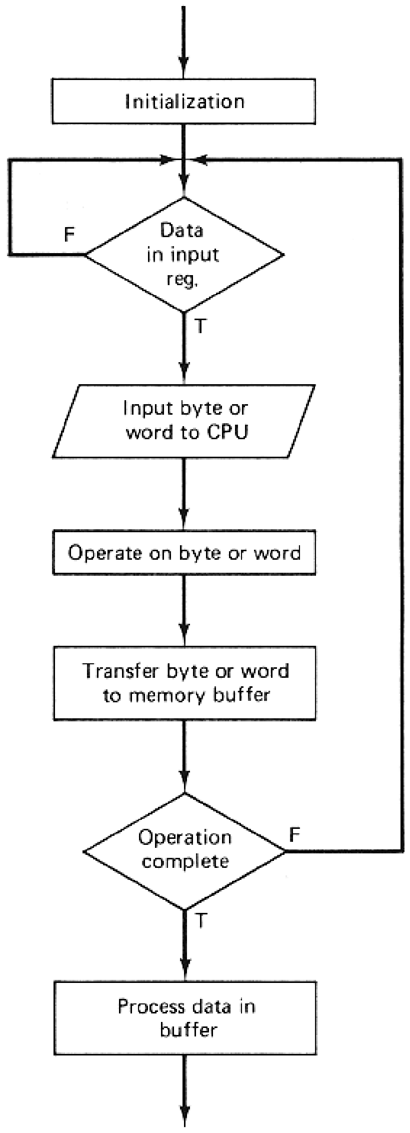


Figure 6-4 Programmed input.

As a more complete example, suppose a line of characters is to be input from a terminal to an 82-byte array beginning at BUFFER until a carriage return is encountered or more then 80 characters are input. If a carriage return is not found in the first 81 characters then the message "BUFFER OVERFLOW" is to be output to the terminal; otherwise, a line feed is to be automatically appended to the carriage return.

Because the ASCII code is a 7-bit code, the eighth bit, bit 7, is often used as parity bit during the transmission from the terminal. Let us assume that bit 7 is set according to even parity and if an odd parity byte is detected, a branch is to be made to ERROR. If there is no parity error, bit 7 is to be cleared before the byte is transferred to the memory buffer.

INTERRUPT I/O

Even though programmed I/O is conceptually simple, it can waste a considerable amount of time while waiting for ready bits to become active. In the above example, if the person typing on the terminal could type 10 characters per second and only 10 μs is required for the computer to input each character, then approximately

$$\frac{99,990}{100,000} \times 100\% = 99.99\%$$

of the time is not being utilized.

Before an 8086 interrupt sequence can begin, the currently executing instruction must be completed unless the current instruction is a HLT or WAIT instruction.

For a prefixed instruction, because the prefix is considered as part of the instruction, the interrupt request is not recognized between the prefix and the instruction.

In the case of the REP instruction, the interrupt request is recognized after the primitive operation following the REP is completed, and the return address is the location of the REP prefix.

For MOV and POP instructions in which the destination is a segment register, an interrupt request is not recognized until after the instruction following the MOV or POP instruction is executed.

For the 8086, once the interrupt request has been recognized, the interrupt sequence consists of:

1. Establishing a type N.
2. Pushing the current contents of the PSW, CS and IP onto the stack (in that order).
3. Clearing the IF and TF flags.
4. Putting the contents of the memory location 4*N into the IP and the contents of 4*N+2 into the CS.

Thus, an interrupt causes the normal program sequence to be suspended and a branch to be made to the location indicated by the double word beginning at four times the type (i.e. the *interrupt pointer*). Control can be returned to the point at which the interrupt occurred by placing an IRET instruction at the end of the *interrupt routine*.

It was mentioned that there are two classes of interrupts, internal and external interrupts, with external interrupts being caused by a signal being sent to the CPU through one of its pins, which for the 8086 is either the NMI pin or the INTR pin.

An interrupt initiated by a signal on the NMI pin is called a *nonmaskable interrupt* and will cause a type 2 interrupt regardless of the setting of the IF flag. Nonmaskable interrupt signals are normally caused by circuits for detecting catastrophic events.

An interrupt on the INTR pin is masked by the IF flag so that this flag is 0 the interrupt is not recognized until IF returns to 1.

When IF=1 and a maskable external interrupt occures, the CPU will return an acknowledgment signal to the device interface through its /INTA pin and initiate the interrupt sequence.

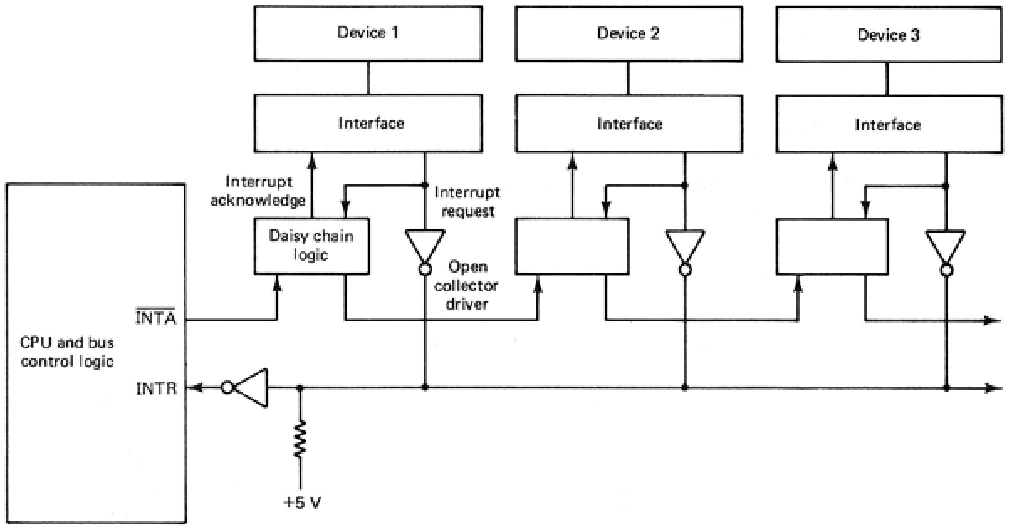
The acknowledgment signal will cause the interface that sent the interrupt signal to send to the CPU (over the data bus) the byte which specifies the type and hence the address of the interrupt pointer. The pointer, in turn, supplies the beginning address of the interrupt routine.

There are several ways of combining with interrupt I/O, some involving only software, some only hardware, and some a combination of the two. Let us consider the following means of giving priority to an interrupt system:

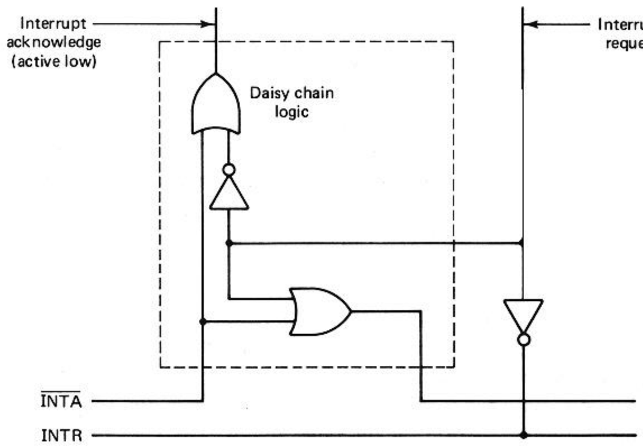
1. Polling
2. Daisy chaining
3. Interrupt priority management hardware

By putting a program sequence (similar to the one in Fig.6-7) at the beginning of the interrupt routine, the priority of the interfaces could be established by the order in which they are polled by the sequence.

Daisy chaining is a simple hardware means of attaining a priority scheme. It consists of associating a logic circuit with each interface and passing the interrupt acknowledge signal through these circuits as shown in Fig.(a). The details of daisy chain logic are shown in Fig.6-14(b). The priority of an interface is determined by its position on the daisy chain. The closer it is to the CPU the higher its priority.



(a) Daisy chain



(b) Logic

Figure 6-14 Daisy chain arrangement.

BLOCK TRANSFERS AND DMA

The activity involved in transferring a byte or word over the system bus is called a *bus cycle*. The execution of an instruction may require more than one bus cycle. For example the instruction:

MOV AL, TOTAL
would use a bus cycle to bring in the contents of TOTAL in addition to the cycle needed to fetch the instruction.

During any given bus cycle one of the system components connected to the system bus is given control of the bus. This component is said to be the *master* during that cycle and the component it is communicating with is said to be the *slave*.

The 8086 receives bus requests through its HOLD pin and issues grants from its hold acknowledge (HLDA) pin. A request is made when a potential master sends a 1 to the HOLD pin. Normally, after the current bus cycle is complete the 8086 will respond by putting a 1 on the HLDA pin.

During a block input byte transfer the following sequence occurs as the datum is sent from the interface to the memory:

1. The interface sends the controller a request for DMA service
2. The controller gains control of the bus
3. The contents of the address register are put on the address bus
4. The controller sends the interface a DMA acknowledgment which tells the interface to put data on the data bus (For an output it signals the interface to latch the next data placed on the bus)
5. The data byte is transferred to the memory location indicated by the address bus
6. The controller relinquishes the bus
7. The address register is incremented by 1
8. The byte count register is decremented by 1
9. If the byte count register is nonzero return to step 1; otherwise stop

The controller/interface design shows bidirectional address lines connected to the controller and only unidirectional address lines going to the interface.

Multiprocessor Systems

Explain the different configurations of multiprocessor systems. (May 2008)

Multiprocessor Systems refer to the use of multiple processors that execute instructions simultaneously and communicate using mailboxes and semaphores Maximum mode of 8086 is designed to implement 3 basic multiprocessor configurations:
1. Coprocessor (8087)
2. Closely coupled (dedicated I/O processor: 8089)
3. Loosely coupled (Multi bus)

Coprocessors and closely coupled configurations are similar - both the CPU and the external processor share:
- ✓ Memory
- ✓ I/O system
- ✓ Bus & bus control logic
- ✓ Clock generator

Multiprocessor configuration

Discuss about the multiprocessor system of 8086.

Explain multiprocessor system. (June 2016, Dec 2016)

Introduction:

Multiprocessor Systems refer to the use of multiple processors that execute instructions simultaneously and communicate using mailboxes and semaphores.

Maximum mode of 8086 is designed to implement 3 basic multiprocessor configurations:
1. Coprocessor (8087)
2. Closely coupled (8089)
3. Loosely coupled (Multibus)

Need for Multiprocessor system

1. Due to limited data width and lack of floating point arithmetic instructions, 8086 requires many instructions for computing even single floating point operations. For this Numeric data processor 8087 is used.
2. Some processor like DMA processor can take care of low level operations , while the 8086 CPU execute high level operations.

Advantages of Multiprocessor

- Easy to add more processor for expansion as per requirement
- When failure occurs, it is easier to replace the faulty processor
- Avoiding the expense of unneeded capabilities of a centralized system by combining several low cost processor.

6.Explain how co processor works and interacts with 8086 . (June 2016)

<u>Coprocessor configuration</u>

Coprocessors and closely coupled configurations are similar in that both the CPU and the external processor share:

- Memory
- I/O system
- Bus & bus control logic
- Clock generator

WAIT instruction allows the processor to synchronize itself with external hardware, eg., waiting for 8087 math co-processor. When the CPU executes WAIT waiting state.

TEST input is asserted (low), the waiting state is completed and execution will resume. ESC instruction: ESC opcode, operand, opcode: immediate value recognizable to a coprocessor as an instruction opcode

Coprocessor cannot take control of the bus, it does everything through the CPU.

- 8089 shares CPU and clock and bus control logic
- It communication with host CPU is by the way of shared memory

- The host sets up a message (command) in memory
- The independent processor interrupts host on completion.

Co processor adds instruction to the instruction set. An instruction to be executed by the co- processor is indicated by an escape **(ESC)** prefix or instruction.

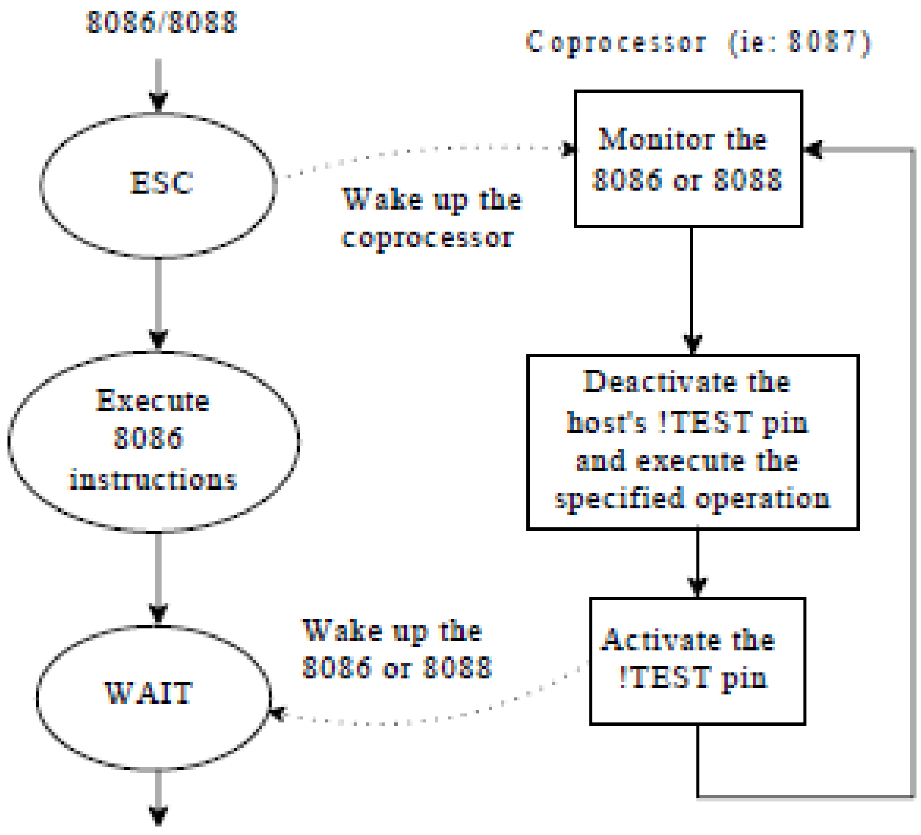


Figure: Flow diagram of coprocessor

The steps to be followed during the program execution of co processor are

1. The 8086 fetches the instruction
2. The co processor monitors the instruction sequence and captures its own instructions.
3. The ESC is decoded by the CPU and coprocessor simultaneously.
4. The CPU computes the 20 bit address of memory operand and does a dummy read. The co processor captures the address of the data and obtains control of the bus to load or store as needed.
5. The co processor sends BUSY (high) to the TEST pin
6. The CPU goes to the next instruction and if this is an 8086 instruction, the CPU and coprocessor execute in parallel.

7. If another coprocessor instruction occurs, the 8086 must wait until BUSY goes low.ie TEST pin become active. To implement this, a WAIT instruction is put in front of most 8087 instructions by the assembler.
8. The WAIT instruction does the operations ie wait until the TEST pin is active.
9. The co processor also makes use of Queue status.

****

7. Explain the closely coupled configuration of 8086 with example

. **Closely Coupled Configuration:**

The main difference between co processor and closely coupled configuration is no special instructio such as WAIT and ESC is used. The communication between 8086 and independent processor is done throug memory space.

NOTE: Closely Coupled processor may take control of the bus independently. Two 8086's cannot be closely coupled.

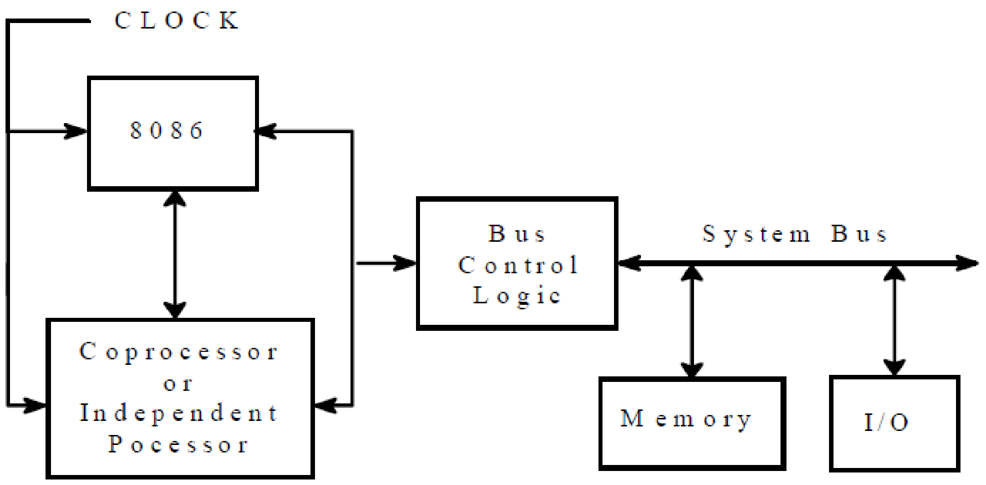


Figure: closely coupled configuration

The 8086 sets up a message in memory and wakes up independent processor by sendin command to one of its ports. The independent processor then accesses the memory to execute the task i parallel with the 8086.When task is completed the external processor informs the 8086 about the completion o task by using either a status bit or an interrupt request.

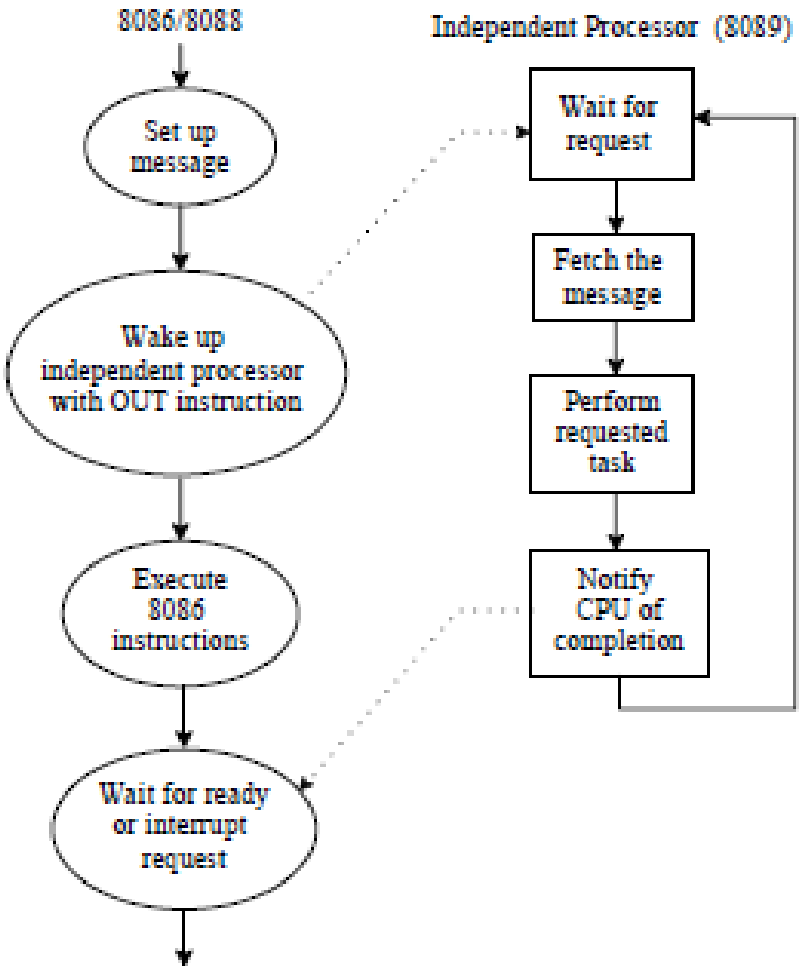

8086/8088
Set up message
Wake up independent processor with OUT instruction
Execute 8086 instructions
Wait for ready or interrupt request
Independent Processor (8089)
Wait for request
Fetch the message
Perform requested task
Notify CPU of completion

Figure:**Interaction between 8086 and 8089**

****

8. Write brief note on 8086 loosely coupled system configuration. (April 2006, May 2017)

<u>Loosely Coupled Configuration</u>:

- In loosely coupled configuration a number of modules of 8086 can be interfaced through a common system bus to work as a multiprocessor system.
- Each module in the loosely coupled configuration is an independent microprocessor based system with its own clock source, and its own memory and 1 0 devices interfaced through a local bus.
- Each module can also be a closely coupled configuration of a processor or coprocessor. The block diagram of a loosely coupled configuration of 8086 is shown in figure

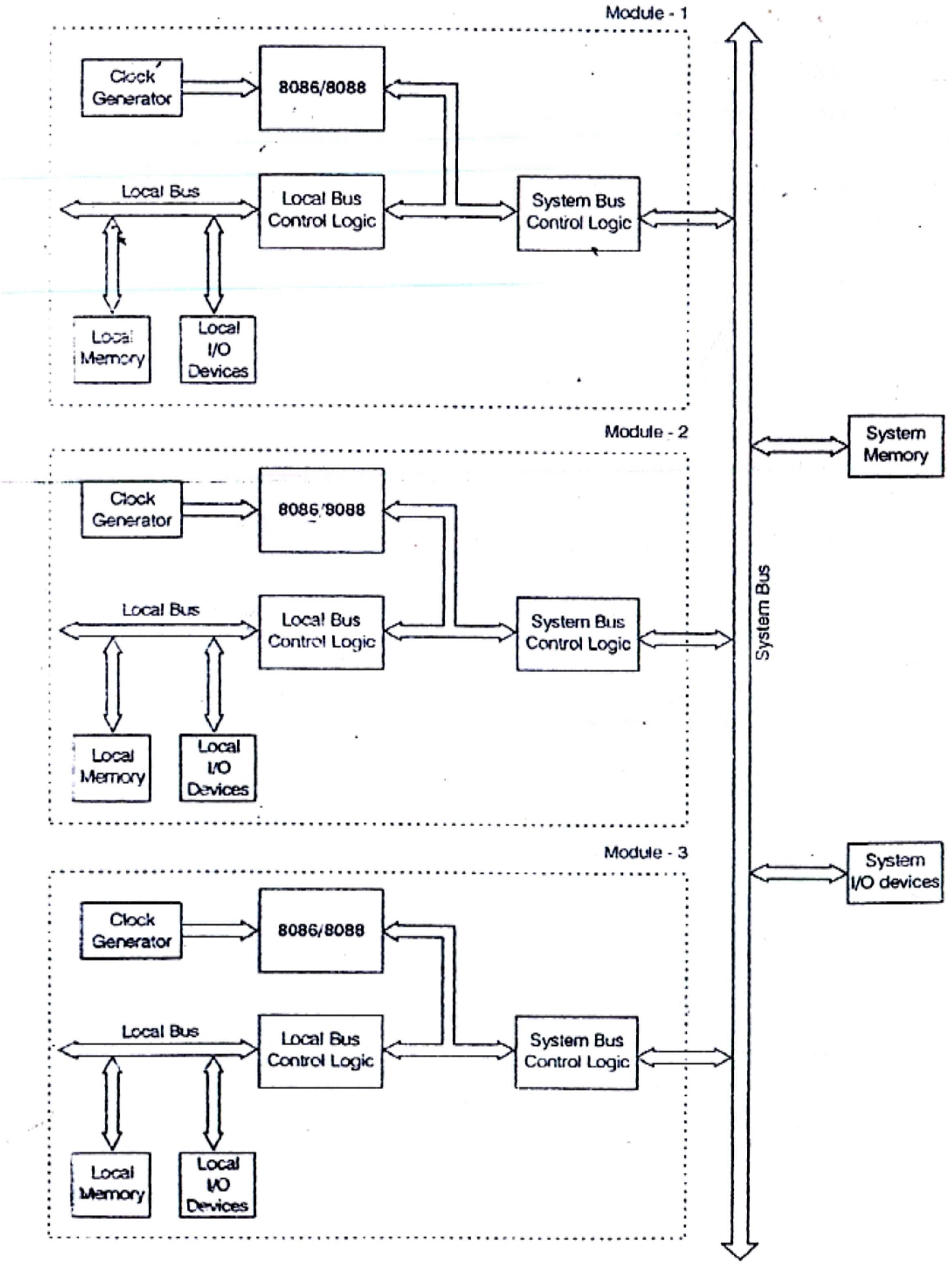


Fig loosely coupled configuration

Advantages:

1. Better system throughput by having more than one processor.
2. The system can be expanded in modular form. Each processor is an independent unit and normally on a separate PC board. One can be added or removed without affecting the others in the system.

3. A failure in one module normally does not affect the breakdown of the entire system and faulty module can be easily detected and replaced.

4. Each processor may have its own local bus to access dedicated memory or I/**O** devices so that a greater degree of parallel processing can be achieved

Disadvantages

1. **Bus Arbitration (contention**): Make sure that only 1 processor can access the bus at any given time
2. It must synchronize local and system clocks for synchronous transfer
3. It requires control chips to tie into the system bus.

****

9.Explain the basic bus access control and arbitration schemes used in multiprocessor systems. (dec 2008)

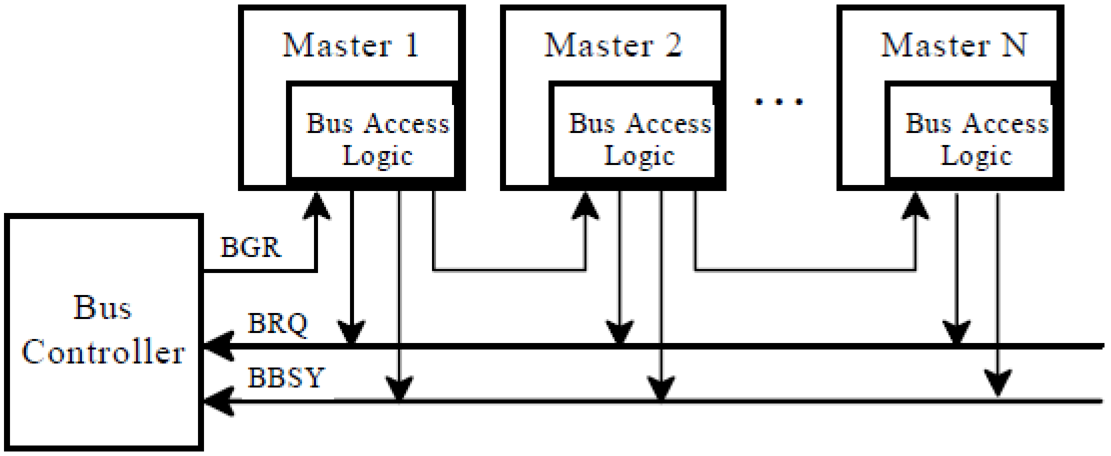
Bus allocation schemes:

- It needs some kind of priority allocation.
- It output a Bus Request (BRQ) to request the bus and BRQ line goes to some controller.
- The CPU input a Bus Grant (BGR) to gain access to bus
- The Bus access logic output a Bus Busy >BBSY= signal to hold the bus.
- To allocate the bus various methods are available.They are
  - **Daisy Chaining**
  - **Polling**
  - **Independent Priority**

Daisy Chaining:

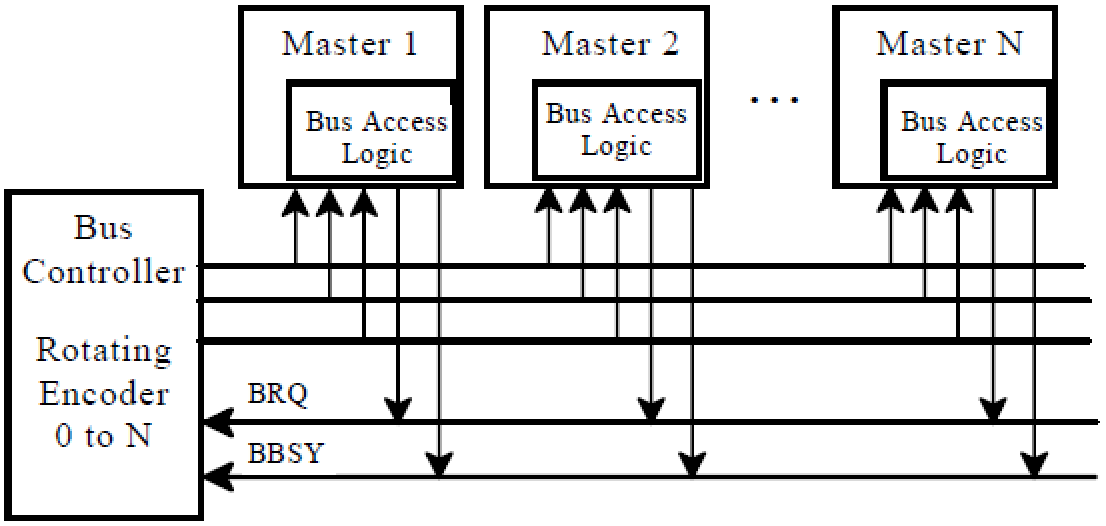
Need a bus controller to monitor bus busy and bus request signals

- It does not require any priority resolving network, rather the priorities of all the devices are essentially assumed to be in sequence.
- All the masters use a single bus request line for requesting the bus access.
- The controller sends a bus grant signal, in response to the request, if the busy signal is inactive when the bus is free.
- The bus grant pulse goes to each of the masters in the sequence till it reaches a requesting master.
- The master then receives the grant signal, activates the busy line and gains the control of the bus.
- The priority is decided by the position of the requesting master in the sequence.



<u>Polling</u>:

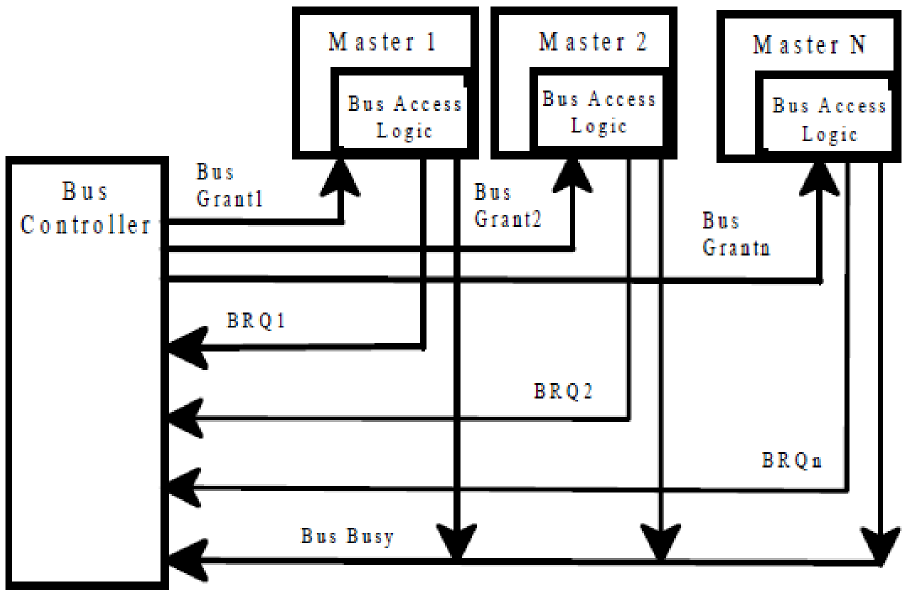
- In polling schemes, a set of address lines is driven by the controller to address each of the masters in sequence.
- When a bus request is received from a device by the controller, it generates the address on the address lines.
- If the generated address matches with that of the requesting masters, the controller activates the BUSY line.



<u>Independent Priority</u>

- In independent priority scheme each master has a pair of Bus request and Bus grant line and each pair has a priority assigned to it.
- The built in priority decoder within the controller selects the highest priority request a asserts the corresponding bus grant signal.
- Synchronization of the clocks must be performed once a Master is recognized.
- Master will receive a common clock from one side and pass it to the controller which will derive a clock for transfer.

- Due to separate pairs of bus request and bus grant signals, arbitration is fast.



<u>Introduction to Advanced processors: 80286 Microprocessor</u>

<u>Salient Features of 80286</u>

✓ The 80286 is the first member of the family of advanced microprocessors with memory management and protection abilities. The 80286 CPU, with its 24-bit address bus is able to address 16 Mbytes of physical memory. Various versions of 80286 are available that runs on 12.5 MHz, 10 MHz and 8 MHz clock frequencies. 80286 is upwardly compatible with 8086 in terms of instruction set.

✓ 80286 has two operating modes namely real address mode and virtual address mode. In real address mode, the 80286 can address upto 1Mb of physical memory address like 8086. In virtual address mode, it can address up to 16 Mb of physical memory address space and 1 GB of virtual memory address space.

✓ The instruction set of 80286 includes the instructions of 8086 and 80186. 80286 has some extra instructions to support operating system and memory management. In real address mode, the 80286 is object code compatible with 8086. In protected virtual address mode, it is source code compatible with 8086. The performance of 80286 is five times faster than the standard 8086.

Need for Memory Management

The part of main memory in which the operating system and other system programs are stored is not accessible to the users. It is required to ensure the smooth execution of the running process and also to ensure their protection. The memory management which is an important task of the operating system is supported by a hardware unit called memory management unit.

Swapping in of the Program

Fetching of the application program from the secondary memory and placing it in the physical memory for execution by the CPU.

Swapping out of the executable Program

Saving a portion of the program or important results required for further execution back to the secondary memory to make the program memory free for further execution of another required portion of the program.

<u>Concept of Virtual Memory</u>

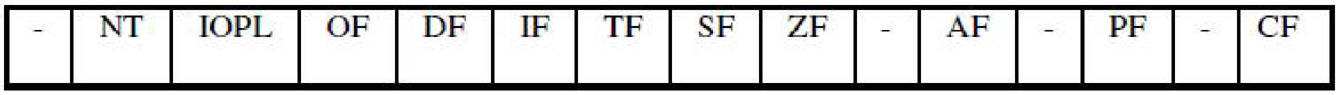
Large application programs requiring memory much more than the physically available 16Mbytes of memory, may be executed by diving it into smaller segments. Thus for the user, there exists a very large logical memory space which is not actually available. Thus there exists a virtual memory which does not exist physically in a system. This complete process of virtual memory management is taken care of by the 80286 CPU and the supporting operating system.

<u>Internal Architecture of 80286</u>

<u>Register Organization of 80286</u>

The 80286 CPU contains almost the same set of registers, as in 8086, namely
1. Eight 16-bit general purpose registers
2. Four 16-bit segment registers
3. Status and control registers
4.Instruction Pointer

The flag register reflects the results of logical and arithmetic instructions.



-	NT	IOPL	OF	DF	IF	TF	SF	ZF	-	AF	-	PF	-	CF

Fig. 80286 Flag Register

D_2, D_4, D_6, D_7 and D_{11} are called as status flag bits. The bits D_8 (TF) and D_9 (IF) are used for controlling machine operation and thus they are called control flags. The additional fields available in 80286 flag registers are:
1. IOPL - I/O Privilege Field (bits D12 and D13)
2. NT - Nested Task flag (bit D14)
3. PE - Protection Enable (bit D16)
4. MP - Monitor Processor Extension (bit D17)
5. EM - Processor Extension Emulator (bit D18)
6. TS – Task Switch (bit D19)

Protection Enable flag places the 80286 in protected mode, if set. This can only be cleared by resetting the CPU. If the Monitor Processor Extension flag is set, allows WAIT instruction to generate a processor extension not present exception.

Processor Extension Emulator flag if set, causes a processor extension absent exception and permits the emulation of processor extension by the CPU.

Task Switch flag if set, indicates the next instruction using extension will generate exception 7, permitting the CPU to test whether the current processor extension is for the current task.

Machine Status Word (MSW)

The machine status word consists of four flags – PE, MO, EM and TS of the four lower order bits D19 to D16 of the upper word of the flag register. The LMSW and SMSW instructions are available in the instruction set of 80286 to write and read the MSW in real address mode.

<u>Internal Block Diagram of 80286</u>

The CPU contain four functional blocks
1. Address Unit (AU), 2. Bus Init (BU)
3. Instruction Unit (IU), 4. Execution Unit (EU)

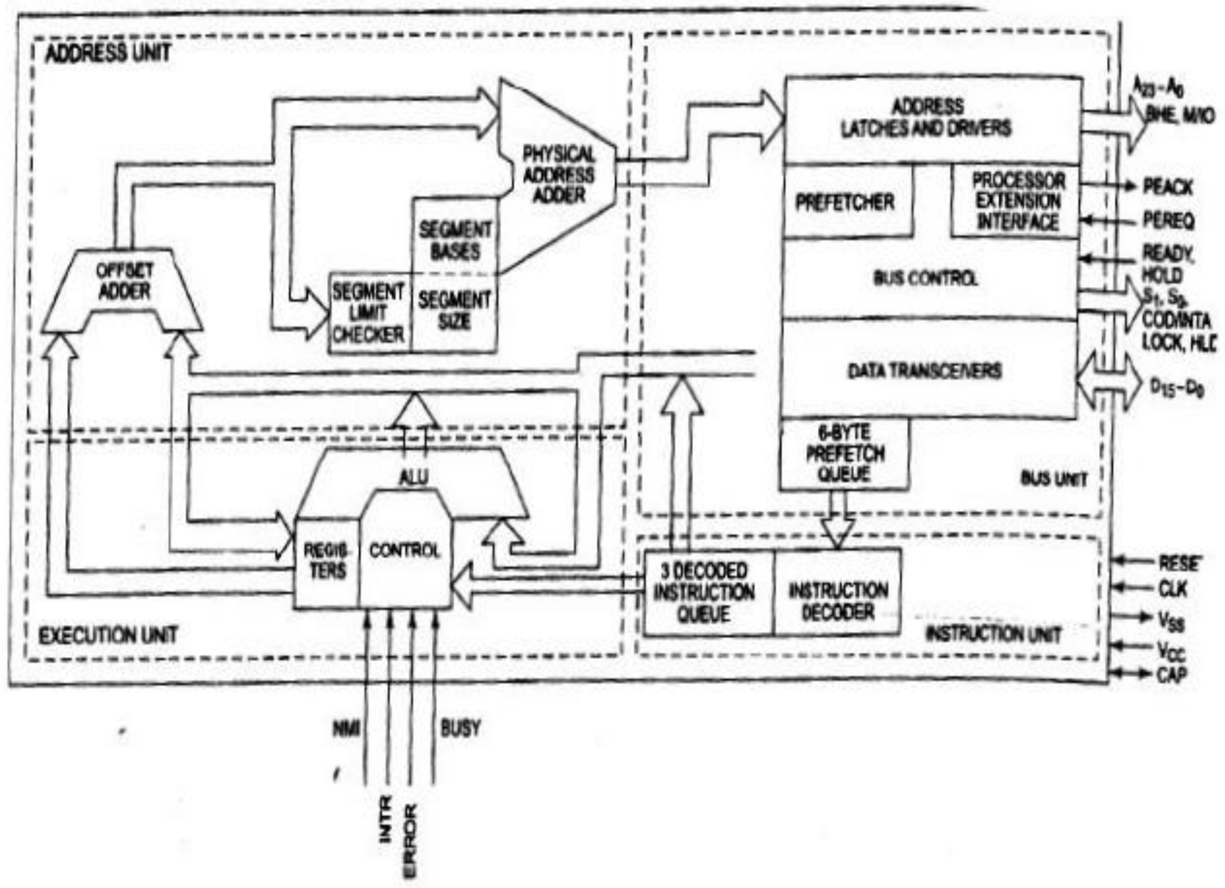
The address unit is responsible for calculating the physical address of instructions and data that the CPU wants to access. Also the address lines derived by this unit may be used to address different peripherals. The physical address computed by the address unit is handed over to the bus unit (BU) of the CPU. Major function of the bus unit is to fetch instruction bytes from the memory. Instructions are fetched in advance and stored in a queue to enable faster execution of the instructions.

The bus unit also contains a bus control module that controls the prefetcher module. These prefetched instructions are arranged in a 6-byte instructions queue. The 6-byte prefetch queue forwards the instructions arranged in it to the **instruction unit** (IU).

The instruction unit accepts instructions from the prefetch queue and an instruction decoder decodes them one by one. The decoded instructions are latched onto a decoded instruction queue. The output of the decoding circuit drives a control circuit in the **execution unit,** which is responsible for executing the instructions received from decoded instruction queue.

The decoded instruction queue sends the data part of the instruction over the data bus. The EU contains the register bank used for storing the data as scratch pad, or used as special purpose registers. The ALU, the heart of the EU, carries out all the arithmetic and logical operations and sends the results over the data bus or back to the register bank.

8.2.2 Internal Block Diagram of 80286



Interrupts of 80286

The Interrupts of 80286 may be divided into three categories,

1. External or hardware interrupts
2. INT instruction or software interrupts
3. Interrupts generated internally by exceptions

While executing an instruction, the CPU may sometimes be confronted with a special situation because of which further execution is not permitted. While trying to execute a divide by zero instruction, the CPU detects a major error and stops further execution.

In this case, we say that an exception has been generated. In other words, an instruction exception is an unusual situation encountered during execution of an instruction that stops further execution. The return address from an exception, in most of the cases, points to the instruction that caused the exception. As in the case of 8086, the interrupt vector table of 80286 requires 1Kbytes of space for storing 256, four-byte pointers to point to the corresponding 256 interrupt service routines (lSR).

Each pointer contains a 16-bit offset followed by a 16-bit segment selector to point to a particular ISR. The calculation of vector pointer address in the interrupt vector table from the (8-bit) INT type is exactly similar to 8086. Like 8086, the 80286 supports the software interrupts of type 0 (INT 00) to type FFH (INT FFH).

Maskable Interrupt INTR: This is a maskable interrupt input pin of which the INT type is to be provided by an external circuit like an interrupt controller. The other functional details of this interrupt pin are exactly similar to the INTR input of 8086.

Non-Maskable Interrupt NMI: It has higher priority than the INTR interrupt. Whenever this interrupt is received, a vector value of 02 is supplied internally to calculate the pointer to the interrupt vector table. Once the CPU responds to a NMI request, it does not serve any other interrupt request (including NMI). Further it does not serve the processor extension (coprocessor) segment overrun interrupt, till either it executes IRET or it is reset. To start with, this clears the IF flag which is set again with the execution of IRET, i.e. return from interrupt.

Single Step Interrupt

As in 8086, this is an internal interrupt that comes into action, if *trap* flag (TF) of 80286 is set. The CPU stops the execution after each instruction cycle so that the register contents (including flag register), the program status word and memory, etc. may be examined at the end of each instruction execution. This interrupt is useful for troubleshooting the software. An interrupt vector type 01 is reserved for this interrupt.

Interrupt Priorities:

If more than one interrupt signals occur simultaneously, they are processed according to their priorities as shown below:

Order	Interrupt
1	Interrupt exception
2	Single step
3	NMI
4	Processor extension segment overrun
5	INTR
6	INT instruction

FUNCTION	Interrupt Number
Divide error exception	0
Single step interrupt	1
NMI interrupt	2
Breakpoint interrupt	3
INTO detected overflow exception	4
BOUND range exceeded exception	5
Invalid opcode exception	6
Processor extension not available exception	7
Intel reserved, do not use	8-15
Processor extension error interrupt	16
Intel reserved, do not use	17-31
User defined	32-255

Signal Description of 80286

CLK: This is the system clock input pin. The clock frequency applied at this pin is divided by two internally and is used for deriving fundamental timings for basic operations of the circuit. The clock is generated using 8284 clock generator.

D_{15}-D_0: These are sixteen bidirectional data bus lines. **A_{23}-A_0:** These are the physical address output lines used to address memory or I/O devices. The address lines A23 - A16 are zero during I/O transfers

BHE: This output signal, as in 8086, indicates that there is a transfer on the higher byte of the data bus (D15 – D8) .

S1 , **S0:** These are the active-low status output signals which indicate initiation of a bus cycle and with M/IO and COD/INTA, they define the type of the bus cycle.

M/ IO: This output line differentiates memory operations from I/O operations. If this signal is it “0” indicates that an I/O cycle or INTA cycle is in process and if it is “1” it indicates that a memory or a HALT cycle is in progress.

COD/ INTA: This output signal, in combination with M/ IO signal and S1 , S0 distinguishes different memory, I/O and INTA cycles.

LOCK: This active-low output pin is used to prevent the other masters from gaining the control of the bus for the current and the following bus cycles. This pin is activated by a "LOCK" instruction prefix, or automatically by hardware during XCHG, interrupt acknowledge or descriptor table access

READY This active-low input pin is used to insert wait states in a bus cycle, for interfacing low speed peripherals. This signal is neglected during HLDA cycle.

HOLD and HLDA This pair of pins is used by external bus masters to request for the control of the system bus (HOLD) and to check whether the main processor has granted the control (HLDA) or not, in the same way as it was in 8086.

INTR: Through this active high input, an external device requests 80286 to suspend the current instruction execution and serve the interrupt request. Its function is exactly similar to that of INTR pin of 8086.

NMI: The Non-Maskable Interrupt request is an active-high, edge-triggered input that is equivalent to an INTR signal of type 2. No acknowledge cycles are needed to be carried out.

PEREG and **PEACK** (**Processor Extension Request and Acknowledgement**)

Processor extension refers to coprocessor (80287 in case of 80286 CPU). This pair of pins extends the memory management and protection capabilities of 80286 to the processor extension 80287. The PEREQ input requests the 80286 to perform a data operand transfer for a processor extension. The PEACK active-low output indicates to the processor extension that the requested operand is being transferred.

BUSY and **ERROR:** Processor extension BUSY and ERROR active-low input signals indicate the operating conditions of a processor extension to 80286. The BUSY goes low, indicating 80286 to suspend the execution and wait until the BUSY become inactive.

In this duration, the processor extension is busy with its allotted job. Once the job is completed the processor extension drives the BUSY input high indicating 80286 to continue with the program execution. An active ERROR signal causes the 80286 to perform the processor extension interrupt while executing the WAIT and ESC instructions. The active ERROR signal indicates to 80286 that the processor extension has committed a mistake and hence it is reactivating the processor extension interrupt.

CAP: A 0.047 μf, 12V capacitor must be connected between this input pin and ground to filter the output of the internal substrate bias generator. For correct operation of 80286 the capacitor must be charged to its operating voltage. Till this capacitor charges to its full capacity, the 80286 may be kept stuck to reset to avoid any spurious activity.

V_{ss}: This pin is a system ground pin of 80286.

V_{cc}: This pin is used to apply +5V power supply voltage to the internal circuit of 80286. RESET The active-high RESET input clears the internal logic of 80286, and reinitializes it.

RESET The active-high reset input pulse width should be at least 16 clock cycles. The 80286 requires at least 38 clock cycles after the trailing edge of the RESET input signal, before it makes the first opcode fetch cycle.

<u>Real Address Mode</u>

• Act as a fast 8086
• Instruction set is upwardly compatible
• It address only 1 M byte of physical memory using A_0-A_{19}.
• In real addressing mode of operation of 80286, it just acts as a fast 8086. The instruction set is upward compatible with that of 8086.

The 80286 addresses only 1Mbytes of physical memory using A_0- A_{19}. The lines A_{20}-A_{23} are not used by the internal circuit of 80286 in this mode. In real address mode, while addressing the physical memory, the 80286 uses BHE along with A_0- A_{19}. The 20-bit physical address is again formed in the same way as that in 8086.

The contents of segment registers are used as segment base addresses. The other registers, depending upon the addressing mode, contain the offset addresses. Because of extra pipelining and other circuit level improvements, in real address mode also, the 80286 operates at a much faster rate than 8086, although functionally they work in an identical fashion. As in 8086, the physical memory is organized in terms of segments of 64Kbyte maximum size.

An exception is generated, if the segment size limit is exceeded by the instruction or the data. The overlapping of physical memory segments is allowed to minimize the memory requirements for a task. The 80286 reserves two fixed areas of physical memory for system initialization and interrupt vector table. In the real mode the first 1Kbyte of memory starting from address 0000H to 003FFH is reserved for interrupt vector table. Also the addresses from FFFF0H to FFFFFH are reserved for system initialization.

The program execution starts from FFFFH after reset and initialization. The interrupt vector table of 80286 is organized in the same way as that of 8086. Some of the interrupt types are reserved for exceptions, single-stepping and processor extension segment overrun, etc

When the 80286 is reset, it always starts the execution in real address mode. In real address mode, it performs the following functions: it initializes the IP and other registers of 80286, it prepares for entering the protected virtual address mode.

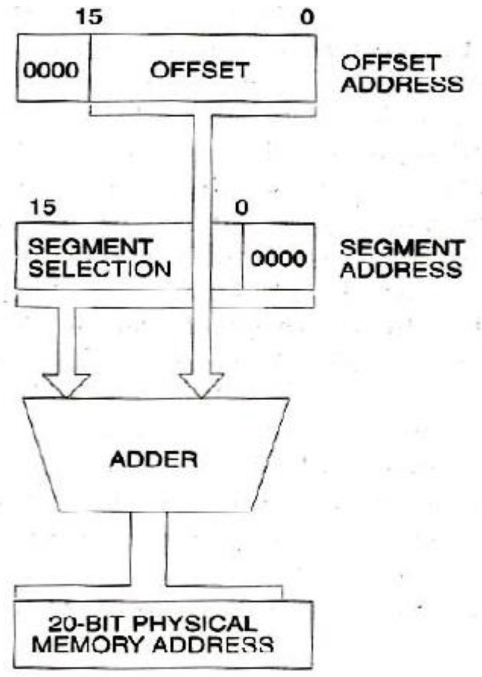


Fig. Real Address Mode Address Calculation

Protected Virtual Address Mode (PVAM)

80286 is the first processor to support the concepts of virtual memory and memory management. The virtual memory does not exist physically it still appears to be available within the system. The concept of VM is implemented using Physical memory that the CPU can directly access and secondary memory that is used as a storage for data and program, which are stored in secondary memory initially.

The Segment of the program or data required for actual execution at that instant is fetched from the secondary memory into physical memory. After the execution of this fetched segment, the next segment required for further execution is again fetched from the secondary memory, while the results of the executed segment are stored back into the secondary memory for further references. This continues till the complete program is executed.

During the execution the partial results of the previously executed portions are again fetched into the physical memory, if required for further execution. The procedure of fetching the chosen program segments or data from the secondary storage into physical memory is called *swapping.* The procedure of storing back the partial results or data back on the secondary storage is called *unswapping.* The virtual memory is allotted per task.

The 80286 is able to address 1 G byte (2^{30} bytes) of virtual memory per task. The complete virtual memory is mapped on to the 16Mbyte physical memory. If a program larger than 16Mbyte is stored on the hard disk and is to be executed, if it is fetched in terms of data or program segments of less than 16Mbyte in size into the program memory by swapping sequentially as per sequence of execution.

Whenever the portion of a program is required for execution by the CPU, it is fetched from the secondary memory and placed in the physical memory is called *swapping in* of the program. A portion of the program or important partial results required for further execution, may be saved back on secondary storage to make the PM free for further execution of another required portion of the program is called *swapping out* of the executable program.

UNIT III
I/O INTERFACING

Memory Interfacing and I/O interfacing - Parallel communication interface – Serial communication interface – D/A and A/D Interface - Timer – Keyboard /display controller – Interrupt controller – DMA controller – Programming and applications Case studies: Traffic Light control, LED display , LCD display, Keyboard display interface and Alarm Controller.

1. Explain in detail about Memory Interfacing and I/O interfacing of 8086.

8086 memory is divided into two memory banks and each memory bank size is 512K X 8 bits (Shown in fig-1)

• Low-bank holds even addressed bytes 00000H through FFFFEH
• High-bank holds odd addressed bytes 00001H through FFFFFH
• Address/data bus is demultiplexed.
• Input bus: 20-bit address bus (A_{19} through A_0), and BHE*
  A1-A19, address lines select storage location
  If A0 = 0 enables low memory bank
  If BHE* = 0 enables high memory bank
• Input / Output bus: 16-bit data bus (D_{15} through D_0)

D7-D0	: Even addressed byte accesses
D15-D8	: Odd addressed byte accesses
D15-D0	: Word accesses

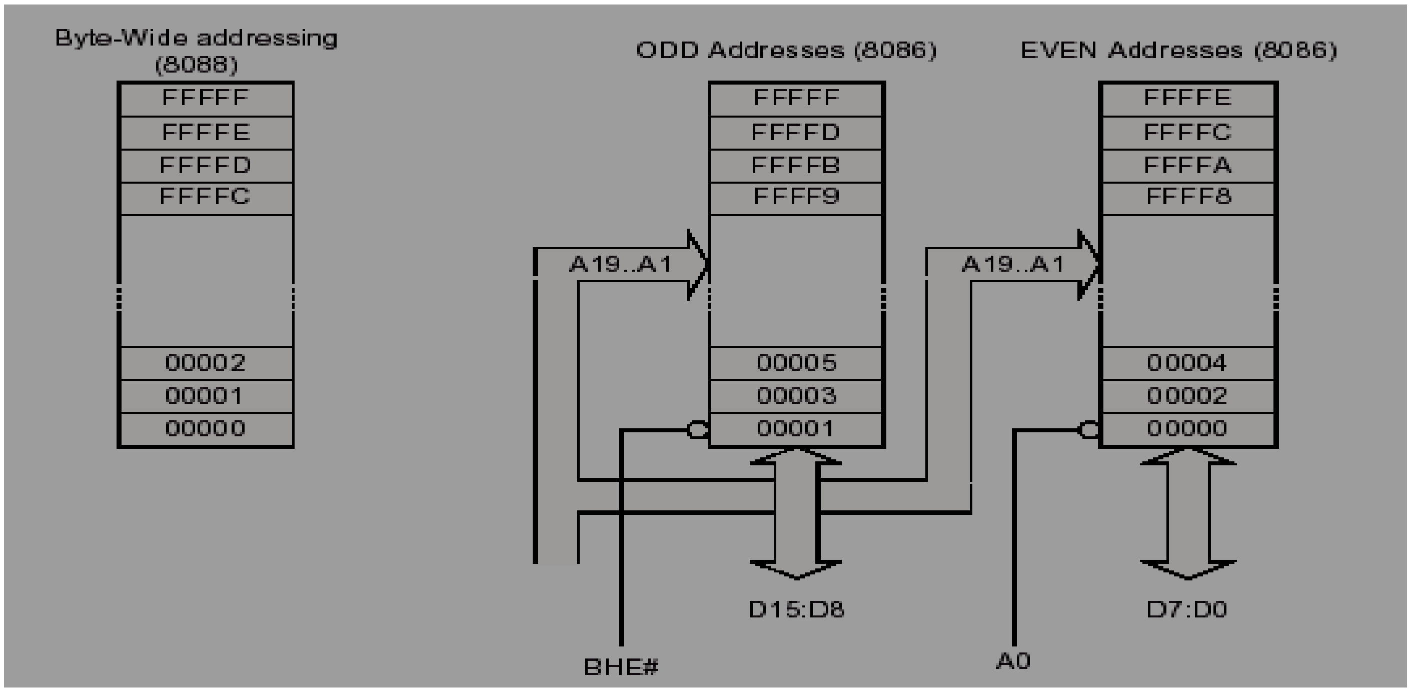


Fig-1: Memory *Hardware organization of address space*

A type of data writes that may take place:

• Byte to a storage location in the upper (odd) bank
• Byte to a storage location in the lower (even) bank
• Word to storage locations in both banks
• Write control logic must decode A0L, BHEL* and MWTC* to produce independent write signals WRU* and WRL* (Shown in fig -2)

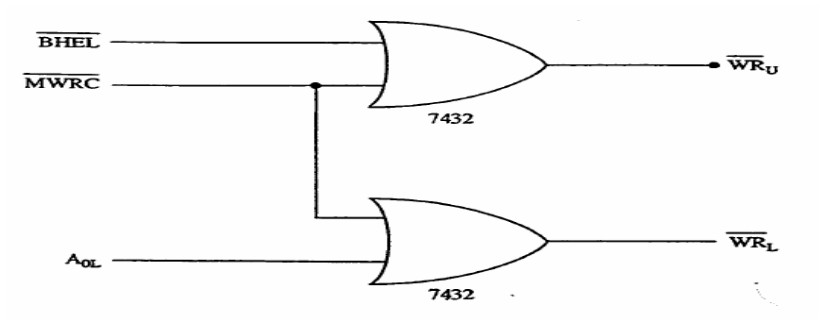


Fig -2: 2-input OR gate solution for decoding write control signals

• MWTC* =0 enables both gates

BHEL*	**A_{OL}**	**WRU***	**WRL***	**Bank selection**
0	0	0	0	Both banks enabled
1	0	1	0	Lower (even) bank enabled
0	1	0	1	Upper (odd) bank enabled

• All accesses take a minimum of one bus cycle of duration
@5MHz—800ns
@8MHz—500ns

During all memory accesses one of three bus cycle status code are output by the MPU (Microprocessor Unit)
• Opcode fetch
• Read memory
• Write memory

• 8288 decodes to produce appropriate control / command signals

• MRDC*	Opcode fetch/memory read
• MWTC*	Memory write
• AMWC*	Advanced memory write

Building blocks of the maximum mode 8086 memory interface

It has the following blocks shown in fig-3.:
• 8288 bus controller
• Address bus latch
• Address decoder
• Data bus transceiver/buffer
• Bank read control logic

• Bank write control logic
• Memory subsystem

Parts of address applied to address inputs of memory subsystem, address decoder, and read/ write control logic **banks**

- Bank selection is accomplished in two ways:
  - separate write signal is developed to select a write to each bank of the memory
  - separate decoders are used for each bank
- The first technique is by far the least costly approach to memory interface.
- The second technique is only used in a system that must achieve most efficient use of the power supply.

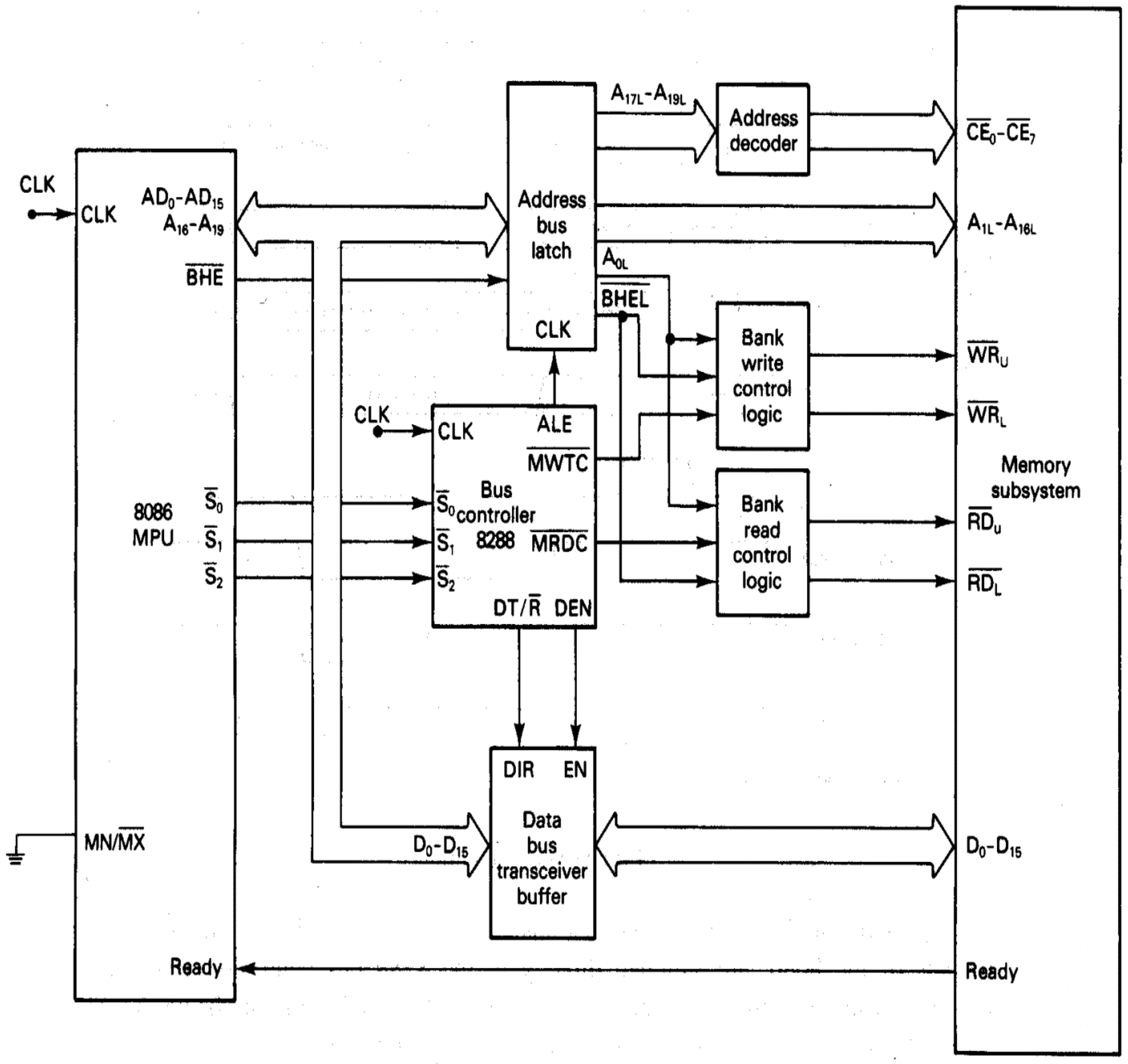


Fig-3 : Maximum mode of 8086 memory interface

I/O Interfacing

- To communicate with the outside world, microprocessor use Peripherals, I/O Devices such as keyboards, A/D converters, input devices and output devices such as CRT, Printers etc.,

- These input and output devices are called Peripherals or I/Os.
- Peripherals are connected to the microprocessor through electronic circuits known as interfacing circuits.
- These interfacing circuits convert 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 microprocessor into the desired peripheral format.

There are two schemes (**Interfacing Configurations)** for the allocation of addresses to memories and input/output devices. They are

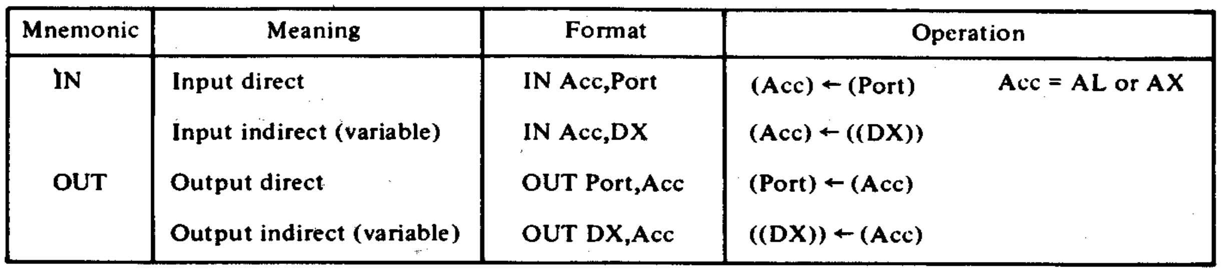
Interfacing Configurations

1. Memory mapped I/O
2. I/O mapped I/O (Isolated I/O)

Isolated I/O: It uses I/O instructions (IN & OUT) and it has its own address space for I/O ports (0000H-FFFFH), isolated from the memory address space.

Memory mapped I/O: uses memory reference instructions (e.g. MOV). So address space is shared between memory and I/O.

- Memory-mapped I/O **does not use** the **IN** or **OUT** instructions.
- It uses **any instruction** that transfers data between the microprocessor and memory.
  - treated as a memory location in memory map
- Same as interfacing 8086 memory in Minimum Mode and Maximum Mode
- I/O devices are treated **separately** from memory
- Address 0000 to 00FF is referred to **page 0**.
- Special instructions exist for this address range
- **Advantage:** Any memory transfer instruction can access the I/O device.
- **Disadvantage:** A portion of memory system is used as the I/O map and reduces memory available to applications



Mnemonic	Meaning	Format	Operation
IN	Input direct	IN Acc,Port	(Acc) ← (Port) Acc = AL or AX
	Input indirect (variable)	IN Acc,DX	(Acc) ← ((DX))
OUT	Output direct	OUT Port,Acc	(Port) ← (Acc)
	Output indirect (variable)	OUT DX,Acc	((DX)) ← (Acc)

Memory mapped I/O

In this type of I/O interfacing, the 8086 uses 20 address lines to identify an I/O device. The I/O device is connected as memory device.

The 8086 uses same control signals and instructions to access I/O. RD and WR signals are activated indicating memory bus cycle.

I/O mapped I/O:

8086 has special instructions IN and OUT to transfer data through the input/output ports in I/O mapped I/O system.

The IN instruction copies data from an input port to the Accumulator. The OUT instruction copies a byte from AL or a word from AX to the specified port.

The M/IO signal is always low when 8086 is executing these instructions. Address of I/O device is 8-bit or 16-bit long.

Program to operate in I/O mode.

- **To write the data 00H into Output port 62H:**

```
MOV  AL,00H
OUT  62H,AL
      or
MOV  AL,00H
MOV  DX,62H
OUT   DX,AL
```

- **To read a byte from Input port address 71H:**

```
IN  AL,71H
      or
MOV  DX,71H
IN  AL,DX
```

Minimum mode interface

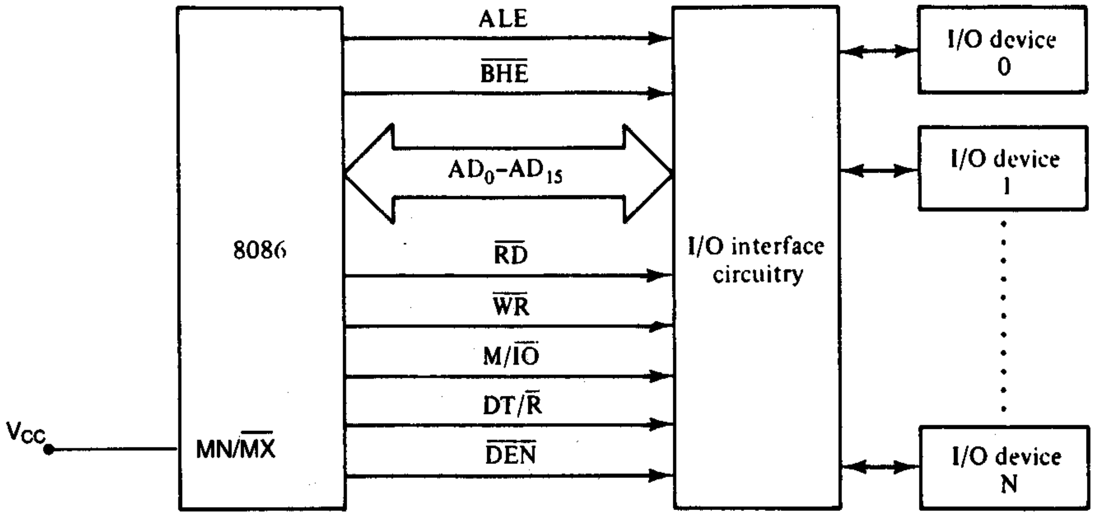


Fig-4: I/O interfacing with minimum mode

Maximum mode interface

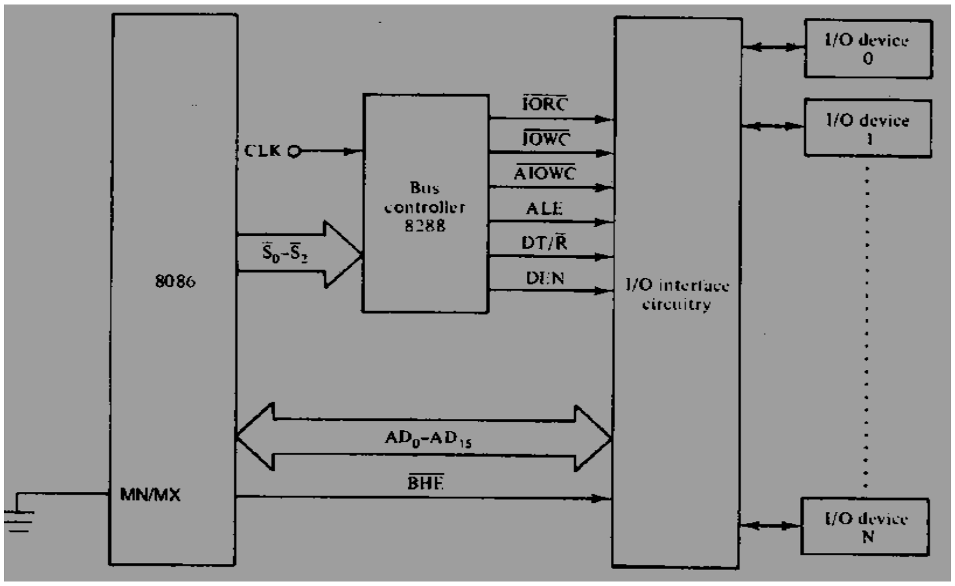


Fig-5: I/O interfacing with maximum mode

2. **Describe the internal block diagram of 8255 (December 2010) (or)**
   Parallel communication interface (8255)
   (Programmable peripheral interface)

- The 8255 is a general purpose programmable I/O device used for parallel data transfer.
- It has 24 I/O programmable pins which can be grouped into three 8 bit parallel ports of Port A , Port B and Port C. It is TTL compatible.
- The eight bit ports of PORT C can be used as individual bits or be grouped into two 4 bit ports. C_{upper} (C_u) and C_{Lower} (C_L).
- The functions of 8255 are classified according to two modes. The Bit Set/Reset mode and the I/O mode. The BSR mode is used to set or reset the bits in Port C.
- The 8-bit data bus buffer is controlled by the read/write control logic. The read/write control logic manages all of the internal and external transfers of both data and control words.
- RD , WR , A_1, A_0 and RESET are the inputs provided by the microprocessor to the READ/ WRITE control logic of 8255.
- The 8-bit, 3-state bidirectional buffer is used to interface the 8255 internal data bus with the external system data bus.
- This buffer receives or transmits data based on the execution of input or output instructions by the microprocessor. The control word is also transferred through the buffer.

Functions of Pin:
The signal descriptions of 8255 are briefly presented as follows:

• **PA_7-PA_0**: These are eight port A lines that acts as either latched output or buffered input lines depending upon the control word loaded into the control word register.

• **PC_7-PC_4** : Upper nibble of port C lines. They may act as either output latches or input buffers lines.

• This port also can be used for generation of handshake lines in mode 1 or mode 2.

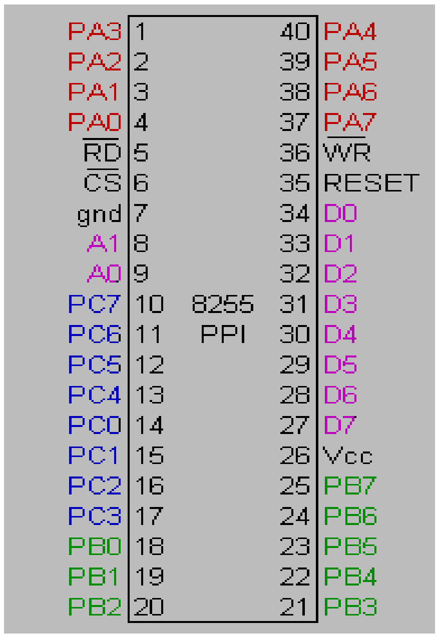
• **PC_3-PC_0** : Lower nibble of port C lines. They may act as either output latches or input buffers lines.

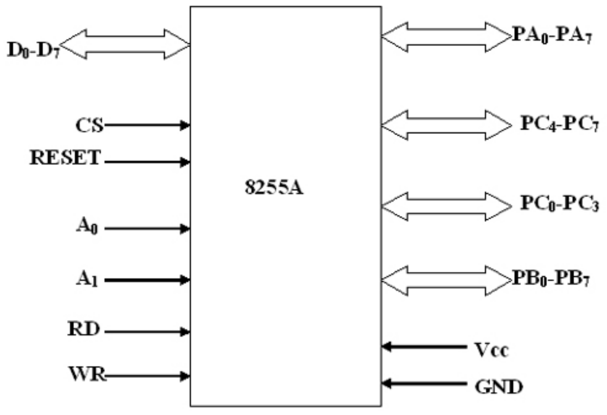
• This port also can be used for generation of handshake lines in mode 1 or mode 2.

- **PB_0-PB_7** : These are eight port B lines which are used as latched output lines or buffered input lines in the same way as port A.
- **A_1-A_0** : These are the address input lines and are driven by the microprocessor.
  - In case of 8086 systems, if the 8255 is to be interfaced with lower order data bus, the A_0 and A_1 pins of 8255 are connected with A_1 and A_2 of 8086 respectively.
  - These address lines A_1- A_0 are used for addressing any one of the four registers, i.e. three ports and a control word register as given in table below.

A1	**A0**	**Select**
0	**0**	**P_A**
0	**1**	**P_B**
1	**0**	**P_C**
1	**1**	**Control register**

- **RD** : This is the input line driven by the microprocessor and should be low to indicate read operation to 8255.
- **WR** : This is an input line driven by the microprocessor. A low on this line indicates write operation.
- **CS** : This is a chip select line. If this line goes low, it enables the 8255 to respond to RD and WR Signals.
- **D_0-D_7** : These are the data bus lines carry data or control word to/from the microprocessor.
- **RESET** : A logic high on this line, clears the control word register of 8255. All ports are set as input ports by default after reset.





Signals of 8255

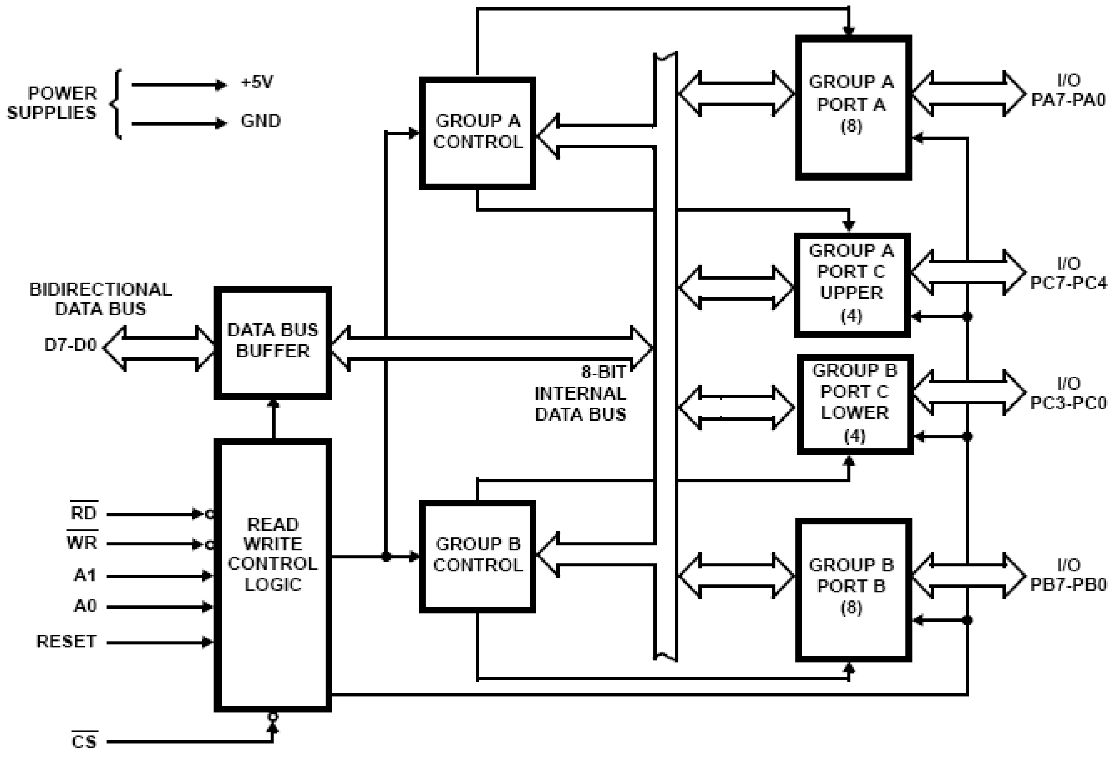


Fig-6: Pin diagram &Block diagram of 8255 Programmable Peripheral interface.

The 8255 consists of four sections namely,

- Data bus buffer
- Read/write control logic
- Group A control
- Group B control

Data Bus buffer:

- It is an 8-bit bidirectional Data bus.
- Used to interface between 8255 data bus with system bus.
- The internal data bus and Outer pins D_0-D_7 pins are connected in internally.
- The direction of data buffer is decided by Read/Control Logic.

Read/Write Control Logic:

- This is getting the input signals from control bus and Address bus
- Control signal are RD and WR.
- Address signals are A0, A1and CS.
- 8255 operation is enabled or disabled by CS.

Group A and Group B control:

- Group A and B get the Control Signals from CPU and send the command to the individual control blocks.
- Group A send the control signal to port A and Port C (Upper) PC7-PC4.
- Group B send the control signal to port B and Port C (Lower) PC3-PC0.

PORT A:

- This is an 8-bit buffered I/O latch.
- It can be programmed by mode 0 , mode 1 & mode 2 **.**

PORT B:

- This is an 8-bit buffer I/O latch.

- It can be programmed by mode 0 and mode 1.

PORT C:

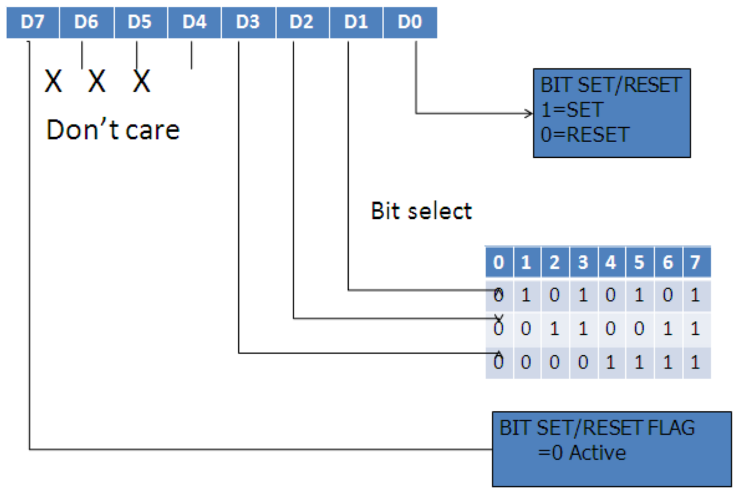
- This is an 8-bit Unlatched buffer Input and an Output latch.
- It is divided into two parts as Port C (Upper) PC7-PC4 & Port C (Lower) PC3-PC0
- It can be programmed by bit set/reset operation.

CONTROL WORD FORMATS:

There are two control word formats i) BSR mode ii) Input / Output mode

FOR BIT SET/RESET MODE:

- This is bit set/reset control word format.



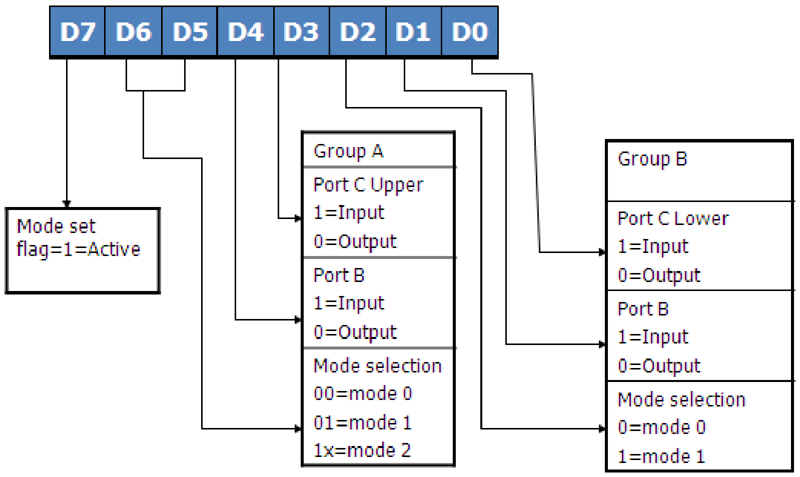
- PC0-PC7 is set or reset as per the status of D0.
- A BSR word is written for each bit of Port C

Example:

- PC3 is Set then control register will be 0XXX0111.
- PC4 is Reset then control register will be 0XXX01000.
- X is a don't care.

FOR I/O MODE

The mode format for I/O as shown in figure



- The control word for both Mode 1 and Mode 2 are same.

- Bit D7 is used for specifying whether word loaded in to Bit set/reset mode or Mode definition word.
- D7=1=Mode definition mode.
- D7=0=Bit set/Reset mode.

Steps to communicate with peripherals through the 8255:

1. Determine the addresses of Port A, port B, port C and control register according to the chip select logic and address lines A0 and A1.
2. Write a control word in the control register.
3. Write I/O instructions to communicate with peripherals through ports A, B and C.

Operation modes

BIT SET/RESET MODE:

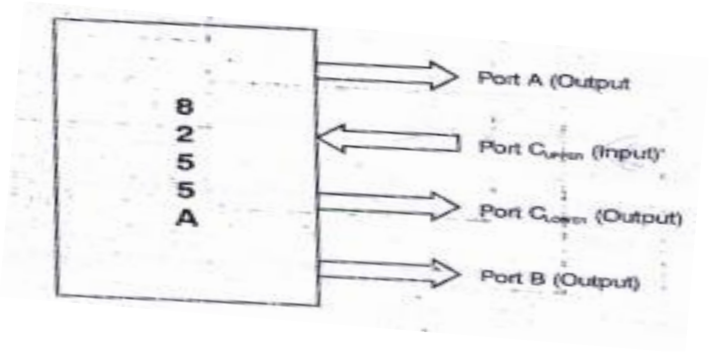
- The PORT C can be Set or Reset by sending OUT instruction to the CONTROL registers.
- In BSR mode, individual bits of Port C can be used for applications such as on/off switch.
- The control word sets or resets one bit at a time.
- BSR control word does not alter any previously transmitted control word with bit D7=1. Thus the I/O operations of Port A and Port B are not affected by a BSR control word.

I/O MODES:

- The I/O mode is divided into three modes as mode 0, mode 1, and mode 2.
    - Mode 0 – Basic I/O mode
    - Mode 1 – strobbed I/O mode
    - Mode 2 – Bidirectional data transfer mode

MODE 0 (Simple input / Output):

- In this mode , port A and port B are used as two simple 8 bit I/O ports and port C as two 4 bit ports used as individually (Simply).



Features:

- Outputs are latched, Inputs are buffered.
- Ports do not have Handshake or interrupt capability.

MODE 1 : (Input/output with Hand shake)

- In this mode, input or output is transferred by hand shaking Signals. The handshaking signals are exchanged between the microprocessor and peripherals.

The features of this mode include the following

1. Two ports (A and B) function as 8 bit I/O ports .They can be configured either as input or output ports.
2. Each port uses 3 lines from port C as handshake signals. The remaining 2 lines of PORT C can be used for simple I/O operations.
3. Input and outputs data are latched.
4. Interrupt logic is supported.

In 8255, the specific lines from PORT C used for handshake signals vary according to the I/O function of a port. Therefore input and output functions in Mode 1 are discussed separately.

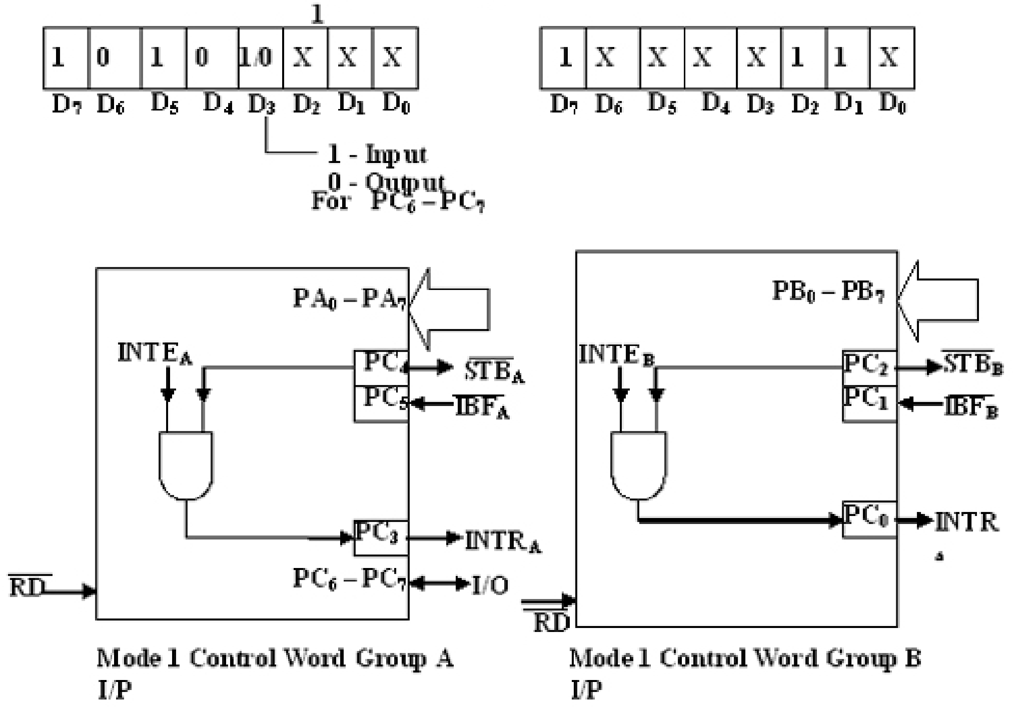
Input control signal definitions (Mode 1):

- **STB(Strobe input**) – If this line falls to logic low level, the data available at 8-bit input port is loaded into input latches.
- **IBF (Input buffer full)** – If this signal rises to logic 1, it indicates that data has been loaded into latches, i.e. it works as an acknowledgement.
- **INTR (Interrupt request)** – This active high output signal can be used to interrupt the CPU whenever an input device requests the service. INTR is set by a high STB pin and a high at IBF pin.

INTE is an internal flag that can be controlled by the bit set/reset mode of either PC4(INTEA) or PC2(INTEB) as shown in fig.

INTR is reset by a falling edge of RD input. Thus an external input device can be request the service of the processor by putting the data on the bus and sending the strobe signal.

Input control signal definitions in Mode 1



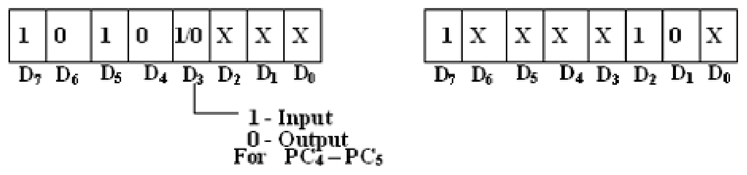
Output control signal definitions (Mode 1) :

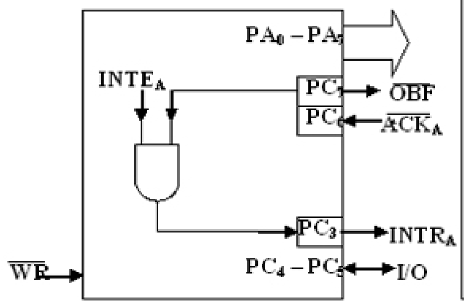
- **OBF (Output buffer full)** – When this signal falls to low, indicates that CPU has written data to the specified output port.

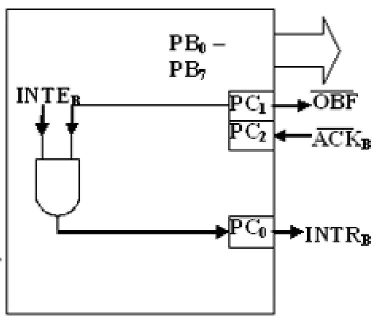
• **ACK (Acknowledge input**) – ACK signal acts as an acknowledgement to be given by an output device. ACK signal, whenever low, informs the CPU that the data transferred by the CPU to the output device through the port is received by the output device.

• **INTR (Interrupt request**) – Thus an output signal that can be used to interrupt the CPU when an output device acknowledges the data received from the CPU.

Output control signal definitions Mode 1







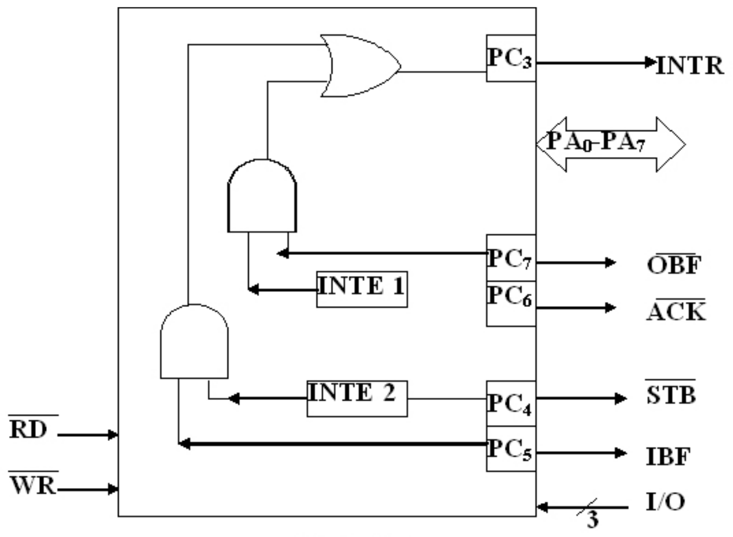
Mode 1 Control Word Group A

Mode 1 Control Word Group B

MODE 2: Bi-directional I/O data transfer:

- This mode allows bidirectional data transfer over a single 8-bit data bus using handshake signals.
- In this mode, Port A can be configured as the bidirectional port and Port B is either in Mode 0 or Mode 1.
- Port A uses 5 signals from Port C as handshake signals for data transfer. The remaining 3 signals from Port C can be used either as simple I/O or as handshake for Port B.

.



Mode 2 pins

****

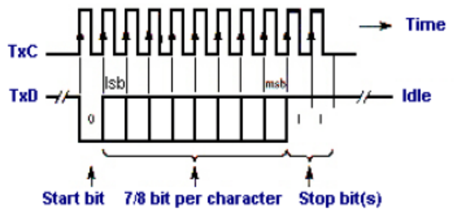
3.Explain in detail about Serial Communication Interface

Serial I/O Interfacing:

The MPU (Microprocessor Unit) selects the peripheral through chip select and uses the control signals. Read to receive data and write to transmit data.

Transmission format:

- In **synchronous format**, receiver and transmitter are synchronized with the same clock and a block of characters are transmitted along with the synchronization information. This format is generally used for high speed transmission (more than 20 Kbits/second)
- The **asynchronous format** is character oriented. Each character carries the information of the start and stop bits. Transmission starts with one start bit (low) followed by a character , and one or two stop bits (high). It is used in low speed transmission less than 20Kbits/second.



Communication Modes:

Simplex - Data are transmitted in only one direction.

Example: Transmission from a microcomputer to a printer.

Duplex - Data flow in both direction

- **Half Duplex** - If the transmission goes one way at a time it is called half duplex. Ex.: walky-talky
- **Full Duplex** – If both transmitting and receiving signals goes simultaneously, it is called full duplex. Example: Transmission between computers.

Rate of transmission

The rate at which the bits are transmitted is called bits/second or Baud rate

For example 1200 baud = 1200 bits/second

It indicates1200 bits are transmitted in a second. For 1 bit it takes 1/1200 =0.83 ms.

Programmable serial Communication Interface (8251):
Programmable serial interface

The 8251 is a programmable USART (Universal Synchronous Asynchronous Receiver Transmitter) is designed for Synchronous and Asynchronous serial communication packaged in a 28 pin DIP.

The 8251 receives parallel data from the CPU and transmits serial data after conversion. This device also receives serial data from the outside and transmits parallel data to the CPU after conversion.

Features of 8251

- Supports both synchronous and asynchronous modes of operation
- Synchronous baud rate – 0 to 64 K baud
- Asynchronous baud rate – 0 to 19.2 K baud
- Contains full duplex double buffered system
- Provides error detection to detect parity and framing errors
- 28-pin DIP package, TTL compatible
- Single +5V supply

Block diagram of 8251:

- Intel 8251 A is a programmable Serial Communication interface IC. It is available in 28 pin Dual-In-Line package. It is used for synchronous & asynchronous serial data communication. The functional block diagram is shown below. It consists of 5 sections namely,
  - Data bus buffer
  - Read/Write control logic
  - Modem control
  - Transmitter section
  - Receiver section

Data Bus Buffer:

- It is used to temporarily store the data which is to be transmitted (or) received. It consists of D_0 - D_7 signals.

Read/Write Control Logic:

- It consists of 3 registers namely data bus buffer, control register and status register.
- Reset, CLK, C/D, RD, WR, CS signals are associated with this block, If C/D is high, the control register is selected for writing control word.
- If C/D is low, then, the data buffer is selected for read/write operation.
- CS signal means chip select signal. It is generated by using unused address lines of processor. If it is low, then the chip is activated. If Reset signal is high, then 8251 is forced to enter into the idle mode.
- CLK signal is used for 8251 to communicate with CPU.
- RD and WR signal are used for read & write operations.

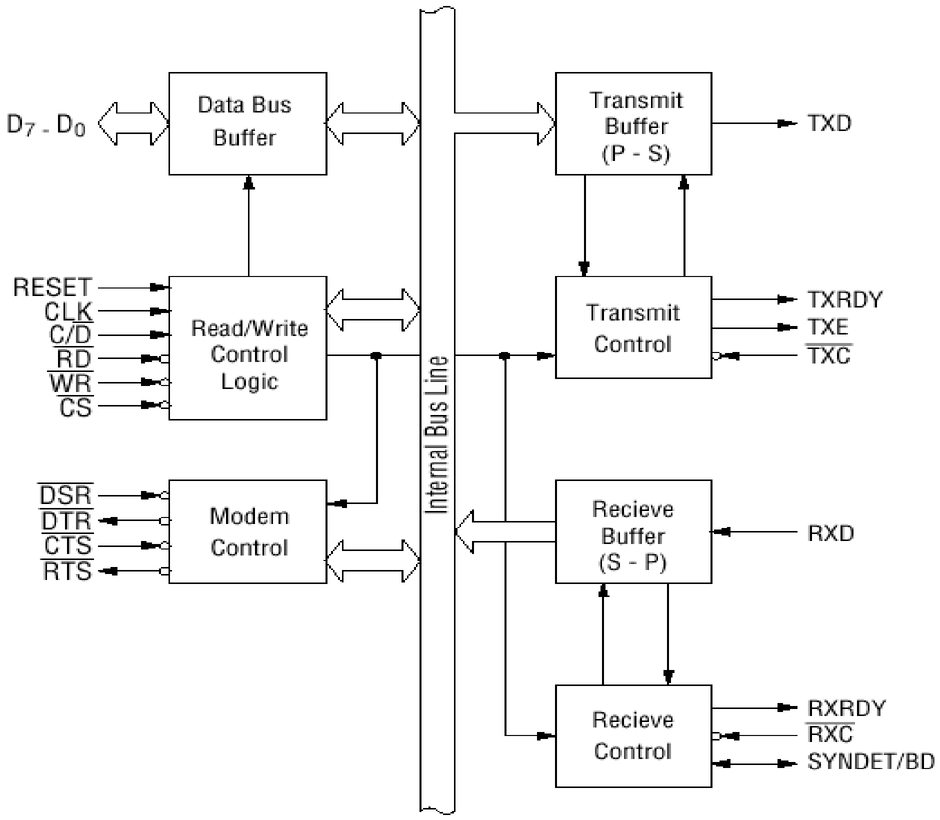


Fig: BLOCK DIAGRAM OF 8251

MODEM Control

- This block is used to interface MODEM to 8251. It is used to provide data communication through MODEM over the telephone cable.

Transmitter Section

- The data which is to be transmitted is given by using D0 - D7 signals to the data bus buffer. Then, the data is transferred to the Transmit Buffer. Here, the parallel data is converted to the serial data. It is transmitted by using the signal TXD.
- This section consists of 2 registers namely transmit buffer register & output register. Transmit buffer is used to hold the 8-bit data & output register is used to convert parallel data into serial data. If output register is empty, then the data is transferred from buffer register to output register.
- If buffer register is empty, then TXRDY signal is asserted high. If output register is empty, then TXEMPTY signal is asserted high.
- TXC signal is used to control the rate of transmission.

Receiver Section

- This section receives serial data from the signal RXD and converts that data into parallel data. It consists of 2 registers namely input register & buffer register.
- Input register receives the serial data & convert it into parallel. Buffer register is used to hold the previous converted data. If input register loads parallel data into buffer register, then, the RXRDY signal is asserted high.

- If RXD signal is low for a half of bit time, then it is assumed as start bit. So, following bits are loaded into the buffer register.
- If RXC signal is used to control the rate of reception.
- During synchronous mode, the signal SYNDET/BRKDET is used to indicate the reception of synchronous character.
- During asynchronous mode, SYNDET/BRKDET signal is used to indicate the break in the data transmission.

Pin Description:

D_0 to D_7 (Data bus Buffer)

- This is bidirectional data bus which receives control word and transmits data from the CPU and sends status words and received data to CPU.

RESET (Input terminal)

- A "High" on this input forces the 8251 into "reset status." The device waits for the writing of "mode instruction."

CLK (Input terminal)

- CLK signal is used to generate internal device timing. CLK signal is independent of RxC or TxC.

WR (Write)

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

RD (Read)

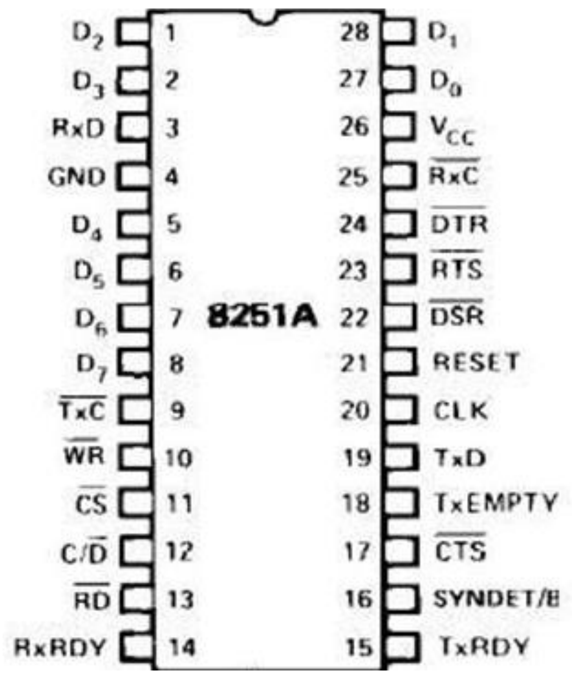
- This is the "active low" input terminal which receives a signal for reading receive data and status words from the 8251.

C/D (Control/Data)

- If C/D = low, data will be accessed. If C/D = high, command word or status word will be accessed.

CS (Chip Select)

- This is the "active low" input terminal which selects the 8251 at low level when the CPU accesses.



SYNDET/BD (Input or output terminal)

- This is a terminal whose function changes according to mode. In "internal synchronous mode." this terminal is at high level, if sync characters are received and synchronized.
- **DSR (Data set ready)**
  This is an input port for MODEM interface. This is normally used to check if the Data set is ready when communicating with a modem.
- **DTR (Data terminal ready)**
  This is an output port for MODEM interface. It is used to indicate that the device is ready to accept data when the 8251 is communicating with a modem.
- **CTS (clear to send)**
  This is an input terminal for MODEM interface which is used for controlling a transmit circuit. Data is transmittable if the terminal is at low level.
- **RTS (Request to send data)**
  This is an output port for MODEM interface. It is used to indicate the MODEM that the receiver is ready to receive a data byte from the MODEM.

Control Register

The 16 bit register for a control word consists of two independent bytes. The first byte is called the mode instruction and the second byte is called the command instruction. This register can be accessed as an output port when the C/D pin is high**.**

Status Register

This input register checks the ready status of a peripheral. This register is addressed as an input port when the C /D is high. It has the same port address as the control register.

4. Draw the block diagram of 8279 Keyboard/Display controller and explain how to interface the Hex Key Pad and 7-segment LEDs using 8279. (April 2010)

- It simultaneously drives the display of a system and interfaces a keyboard with the microprocessor.
- The keyboard display interface scans the keyboard to identify if any keys has been pressed and sends the code of the pressed key to the microprocessor.
- It also transmits the data received from microprocessor to the display device.

PIN DIAGRAM OF 8279:

DATA BUS (D7-D0)

- All data and commands between the microprocessor and 8279 are transmitted on these lines.

RD (read):

- Microprocessor reads the data/ status from 8279.

WR (write):

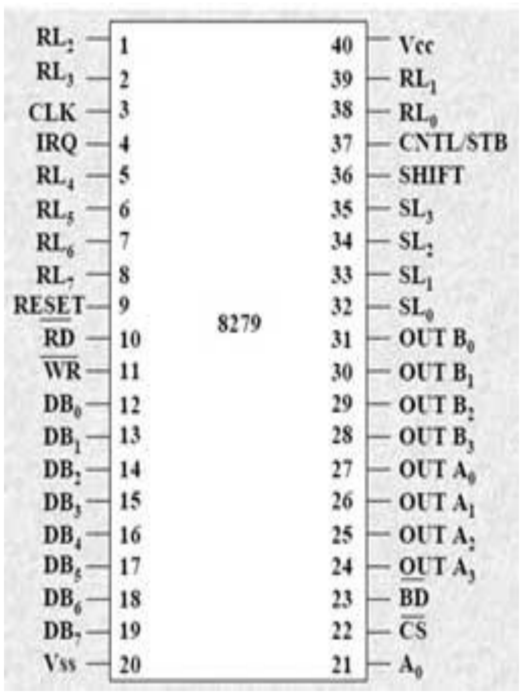
- Microprocessor writes the data to 8279

A0:

- A high signal on this line indicates that the word is a command or status. A low signal indicates the data.

RESET:

- High signal in this pin resets the 8279. After being reset, the 8279 is placed in the following modes 16 x 8 – bit character display – left entry
- Two key lock out



CS (Chip Select):

- A low signal on this input pin enables the communication between 8279 and the microprocessor.

IRQ (Interrupt Request):

- The interrupt line goes low with each FIFO/sensor RAM reads and returns high if there still information in the RAM

SL0-SL3:

- The scan lines which are used to scan the key switch or sensor matrix and the displays digits. These lines can be either encoded (1 of 16) or decoded (1 of 4)

RL0-RL7:

- Input return lines which are connected to the scan lines through the keys or sensor switches.

SHIFT:

- It has an active internal pull-up to keep it high until a switch closure pulls it low.

CNTL/STB:

- For keyboard mode, this line is used as a control input and stored like status on a key closure.
- The line is also the strobed line to enter the data into the FIFO in the strobed input.

OUT A0 – OUT A3, OUT B0 – OUT B3:

- These two ports are the outputs for the 16x4 display refresh registers. These two ports may also be considered as one 8 – bit port. The two 4 – bit ports may be blanked independently.

BD:

- This output is used to blank the display digit switching or by a display banking command.

BLOCK DIAGRAM OF 8279:

The 8279 has the following four sections.

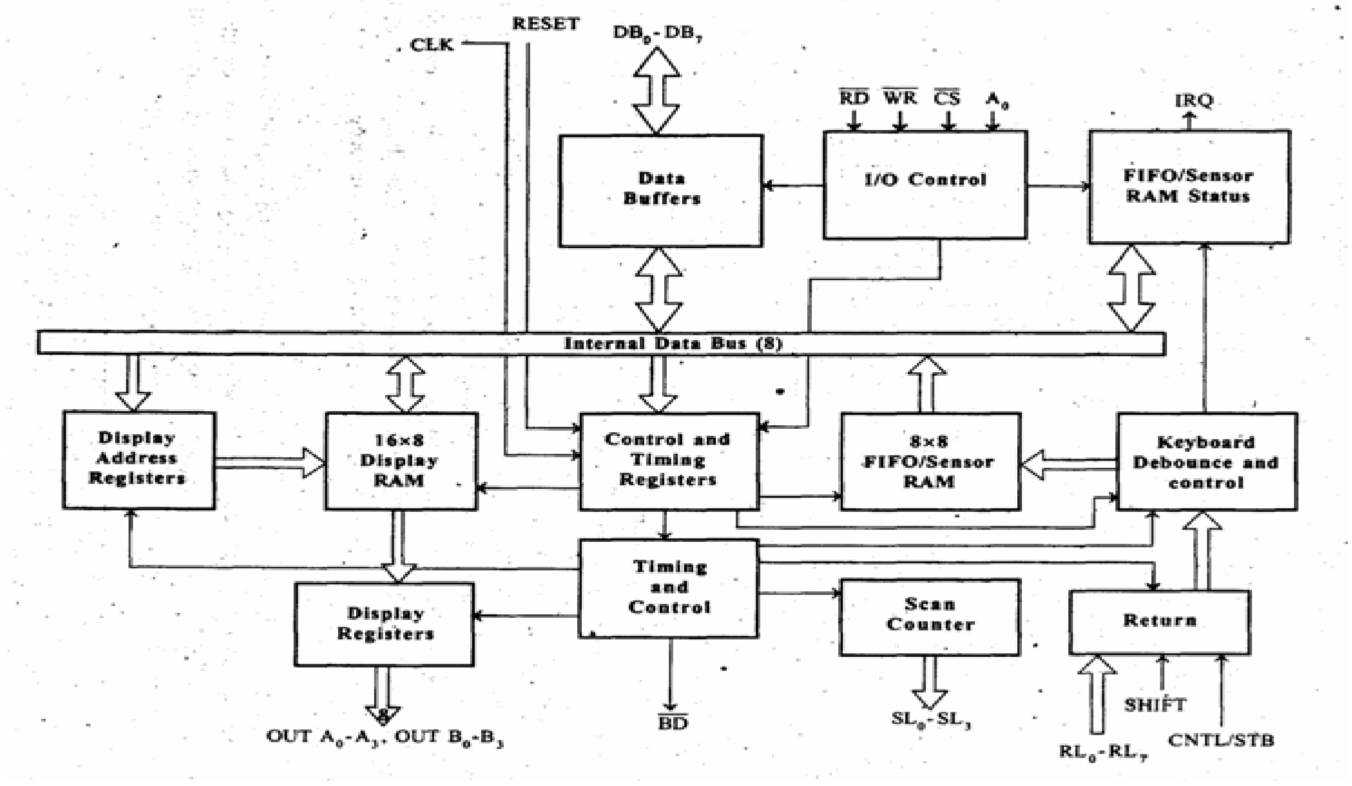
- CPU interface section
- Keyboard section
- Scan section
- Display section

CPU INTERFACE SECTION:

- This section has bi-directional data buffer (DB0 –DB7), I/O control lines (RD, WR, CS, A0) and Interrupt Request lines (IRQ).
- The A0 signal determines whether transmit/receive control word or data is used.

An active high in line IRQ is generated to interrupt the microprocessor whenever the data is available.

A0	RD	WR	Operation
0	0	0	MPU writes the data is 8279
0	0	1	MPU reads the data from 8279
1	1	0	MPU writes control word to 8279
1	0	1	MPU read status word from 8279



KEYBOARD SECTION:

- This section has keyboard debounce & control, 8X8 FIFO/sensor RAM, 8 return lines (RL0 – RL7) and CNTL/STB and shift lines.
- In the keyboard debounce and control unit, keys are automatically debounced and the keyboard can be operated in two modes.
    - Two keys lock out
    - N – key roll over

Two-Key lockout mode:

- If two keys are depressed within the debounce cycle, it is a simultaneous depression. Neither key will be recognized until one of the key is released. The final key released will be recognized and entered.

N-Key Rollover mode:

- In this mode, each key depression is treated independently. If simultaneous depression occurs, then keys are recognized and entered according to the order the keyboard scan found them.
- The 8X8 FIFO/sensor RAM consists of 8 registers that are used to store eight keyboard entries.
- The return lines (RL0-RL7) are connected to eight columns of keyboard.
- The status of shift and CNTL/STB lines are stored along with the key closure.

SCAN SECTION:

- This section has scan counter and four scan lines (SL0 – SL3).
- These lines are decoded (by using 4 to 16 decoder) to generate 16 scan lines.
- Generally SL0 – SL3 are connected with the rows of a matrix keyboard.

DISPLAY SECTION:

- This section has two groups of outputs lines A0 – A3 and B0 – B3. These lines are used to send data to display drivers.
- BD line is used blank the display. It also has 16X8 displays RAM.

<u>Modes of operations of 8279</u>

1. Input (Keyboard) modes
2. Output (Display) modes

Keyboard modes

- **Scanned keyboard mode with N key rollover**

In this mode, each key depression is treated independently. When a key is pressed, the debounce circuit waits for 2 keyboards scans and then checks whether the key is still depressed. If it is still depressed, the code is entered in FIFO RAM

- **Scanned keyboard mode with 2 key lock out.**

It Prevents 2 keys from being recognized if pressed simultaneously. If two keys are pressed within a debounce cycle (simultaneously), no key is recognized till one of them remains closed, and the other is released. The last key that remains depressed is considered as single valid key depression.

Display modes:

Left entry mode
The data is entered from the left side of the display unit.
Right entry mode
The first entry to be displayed is entered on the rightmost display.

Programming the Keyboard Interface :

- Before any keystroke is detected, the 8279 must be programmed.
- The first 3 bits of the number sent to the control port (11H) select one of the 8 different control words.

Command Words of 8279

a) Keyboard Display mode set

The format of the command word is to select different modes of operation of 8279

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_0
0	0	D	D	D	K	K	K	1

D_7	D_6	D_5	*Function*	*Purpose*
0	0	0	Mode set	Selects the number of display positions, type of key scan...
0	0	1	Clock	Programs internal clk, sets scan and debounce times.
0	1	0	Read FIFO	Selects type of FIFO read and address of the read.
0	1	1	Read Display	Selects type of display read and address of the read.
1	0	0	Write Display	Selects type of write and the address of the write.
1	0	1	Display write inhibit	Allows half-bytes to be blanked.
1	1	0	Clear	Clears the display or FIFO
1	1	1	End interrupt	Clears the IRQ signal to the microprocessor.

D	D	Display modes
0	0	Eight 8-bit character Left entry
0	1	Sixteen 8-bit character left entry
1	0	Eight 8-bit character Right entry
1	1	Sixteen 8-bit character Right entry

K	K	K	Keyboard modes
0	0	0	Encoded Scan, 2 key lockout (Default after reset)
0	0	1	Decoded Scan, 2 key lockout
0	1	0	Encoded Scan, N- key Roll over
0	1	1	Decoded Scan, N- key Roll over
1	0	0	Encode Scan, N- key Roll over
1	0	1	Decoded Scan, N- key Roll over
1	1	0	Strobed Input Encoded Scan
1	1	1	Strobed Input Decoded Scan

Control Word Description

First three bits given below select one of 8 control registers (opcode)

- ***<u>000DDMMM</u>***

***Mode set*: Opcode 000.**
DD sets displays mode.
MMM sets keyboard mode
DD field selects either**:**

- 8- or 16-digit display
- Whether new data are entered to the rightmost or leftmost display position.

b)Programmable clock
The clock for operation of 8279 is obtained by dividing the external clock input signal by a programmable constant called prescaler**.**

- **<u>001PPPPP</u>**

- The clock command word programs the internal clock driver**.**
- **The code PPPPP,** is a prescalar that divides the clock input pin (CLK) to achieve the desired
- operating frequency, e.g. 100 KHz requires 01010_2 .

(c)Read FIFO/Sensor RAM

- **<u>010 AI X AAA</u>**

The read FIFO control word selects the address (AAA) of a keystroke from the FIFO buffer (000 to 111).
X - don't care
AI selects auto-increment for the address

d) Read Display RAM
This command enables a programmer to read the display RAM data.

- **<u>011 AI AAAA</u>**

The display read control word selects the 4 bit address AAAA points to the 16 byte display RAM position that is to be read.
AI selects auto-increment for the address.

e) Write Display RAM

- **<u>100 AI AAAA</u>**

The display write control word selects the 4 bit address AAAA points to the 16 byte display RAM positions that is to be written.
Display. Z selects auto-increment so subsequent writes go to subsequent display positions.

f)Display with inhibit blanking

- **<u>1010WWBB</u>**

The display write inhibit control word inhibits writing to either the leftmost 4 bits of the display (left W) or rightmost 4 bits (right W).
BB works similarly except that they blank (turn off) half of the output pins.

g) Clear Display RAM

- **<u>1100CCFA</u>**

The clear control word clears the display, FIFO or both
Bit F clears FIFO and the display RAM status, and sets address pointer to 000.
If CC are 00 or 01, all display RAM locations become 00000000.
If CC is 10, --> 00100000, if CC is 11, --> 11111111.

h) End Interrupt/Error mode set

- **<u>1110E000</u>**

  End of Interrupt control word is issued to clear IRQ pin to zero in sensor matrix mode

- Clock must be programmed first. If 3.0 MHz drives CLK input, PPPPP is programmed to 30 or 11110_2.
- Keyboard type is programmed next. The previous example illustrates an encoded keyboard, external decoder used to drive matrix.
- Program the operation of the FIFO.Once programmed never reprogrammed done, until a procedure is needed to read prior keyboard codes .

  To determine if a character has been typed, the FIFO status register is checked.

When the control port is addressed by the IN instruction, the contents of the FIFO status word is copied into register AL:

5.Draw the functional block diagram of 8254 timer and explain the different modes of operation. (April 2010)(Nov/Dec-2013)

Programmable Interval Timer: 8254

The 8254 is a programmable interval timer/counter is used for the generation of accurate time delays ,controlling real-time events such as real-time clock, events counter, and motor speed and direction control under software control.

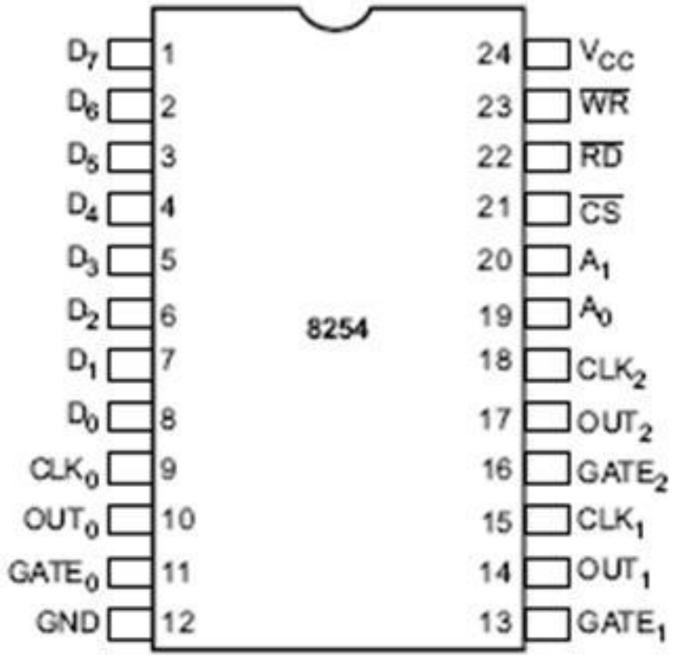
After the desired delay, the 8254 will interrupt the CPU. This makes microprocessor to be free the tasks related to the counting process and can execute the programs in memory, while the timer device may perform the counting tasks. This minimize the Software overhead on the microprocessor.

It consists of three independent 16-bit programmable counters (timers),each with capable of counting in binary or BCD with a maximum frequency of 10MHz.

Some of the other counter/timer functions common to microcomputers which can be implemented with the 8254 are:

- Real time clock
- Event-counter
- Digital one-shot
- Programmable rate generator
- Square wave generator
- Binary rate multiplier

PIN DIAGRAM OF 8254:



PIN DESCRIPTION:

A1	A2	SELECTION
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Counter 3

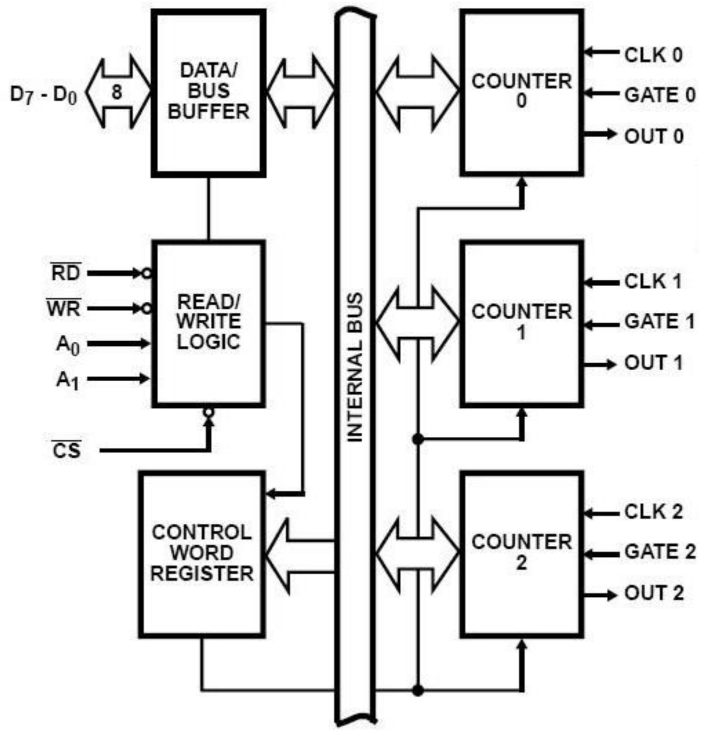
BLOCK DIAGRAM OF 8254:

DATA BUS BUFFER:

- This 3- state, bi-directional, 8-bit buffer is used to interface the 8254 to the system bus.

READ/WRITE LOGIC:

- The Read/Write logic accepts inputs from the system bus and generates control signals for the other functional blocks of the 8254.
- A1 and A0 select one of the three contents counters or the control word register to be read from/written into.
- A “low” on the RD input tells the 8254 that the CPU is reading one of the counters.
- A “low” on the WR input tells the 8254 that the CPU is writing either a control word or an initial count.
- Both RD and WR are qualified by CS; RD and WR are ignored unless than 8254 has been selected by holding CS low.



CONTROL WORD REGISTER:

- The control word register is selected by the read/write logic when A1, A0=11.
- If the CPU then does a write operation to the 8254, the data is stored in the control word register and is interpreted as a control word used to define the operation of the counters.
- The control word register can only be written to; status information is available with the Read-Back command.

COUNTER 0, COUNTER 1, COUNTER 2:

- Each is a 16 bit down counter
- The counters are fully independent. Each counter may operate in a different mode.
- Each counter has a separate clock input, count enable (gate) input lines and output lines.
- The control word register is not a part of the counter itself, but its contents determine how the counter operates.

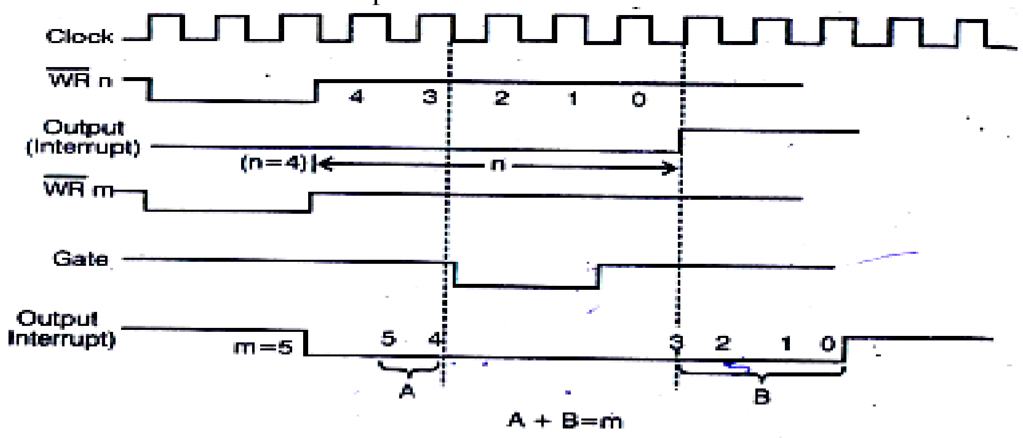
OPERATIONAL MODES OF 8254:

- The 8254 can operate in six operating modes. They are,

Mode 0: Interrupt on Terminal count:

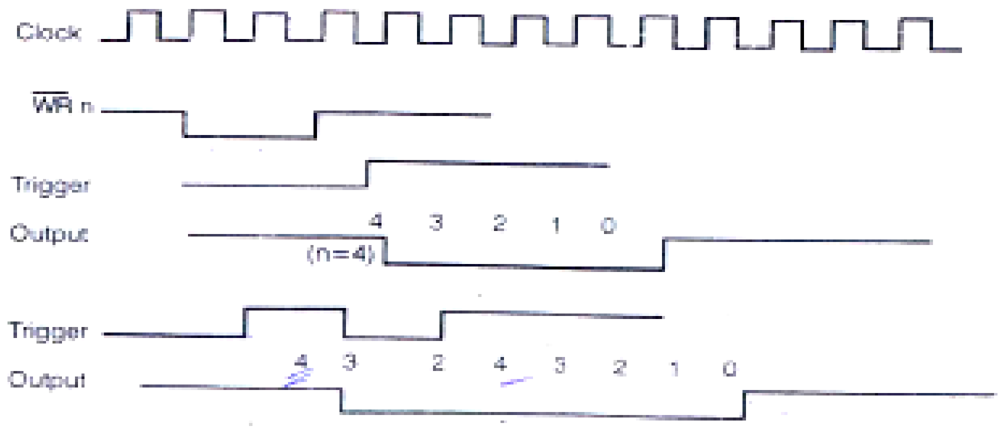
- Mode 0 is typically used for event countering.
- After the control word is written OUT is initially low, and will remain low until the counter reaches zero.
- OUT then goes high and remains high until a new count or a new mode 0 control word is written into the counter.

  - GATE = 1 enables counting;
  - GATE = 0 disables counting. GATE has no effect on OUT.
- After the control word and initial count (say n=4, m=5) are written to a counter, the initial count will be loaded on the next CLK pulse.
- This CLK pulse does not decrement the count. So far an initial count of N, OUT does not go high until N+1 CLK pulses after the initial count is written.
- This mode can be used as an interrupt.



Mode 1 – Hardware Retriggerable one-shot:

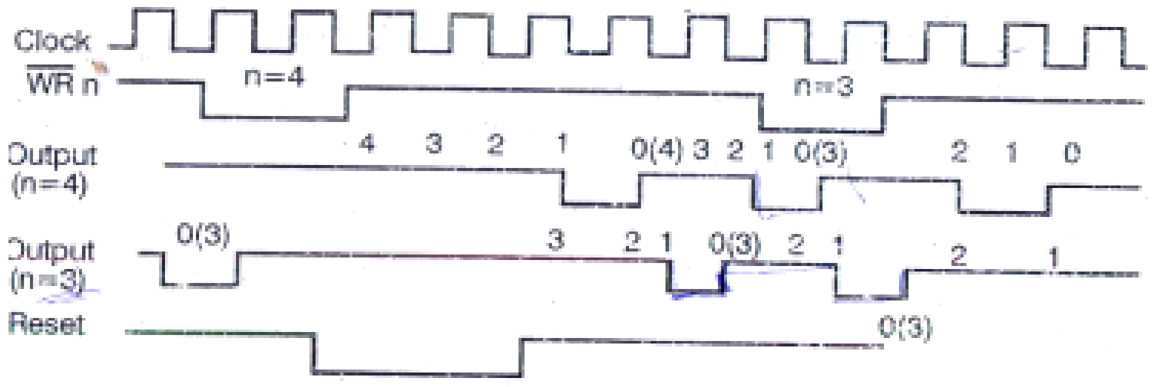
- OUT will be initially high. OUT will go low on the CLK pulse following a trigger to begin the one-shot pulse, and will remain low until the counter reaches zero.
- OUT will then go high and remain high until the CLK pulse after the next trigger. Thus generating a one- shot pulse.
- After writing the control word and initial count, the counter is armed. A trigger results in loading the counter and setting OUT low on the next CLK pulse, the starting the one-shot pulse.
- An initial count of N will result is a one-shot pulse 'N' CLK cycles in duration.



Mode 2: Rate generator

- This mode function like a device – by – N counter
- It is typically used to generate a real time clock interrupt.
- OUT will initially be high. When the initial count has decremented to 1, OUT goes low for one CLK pulse.

- Count and the process are repeated.



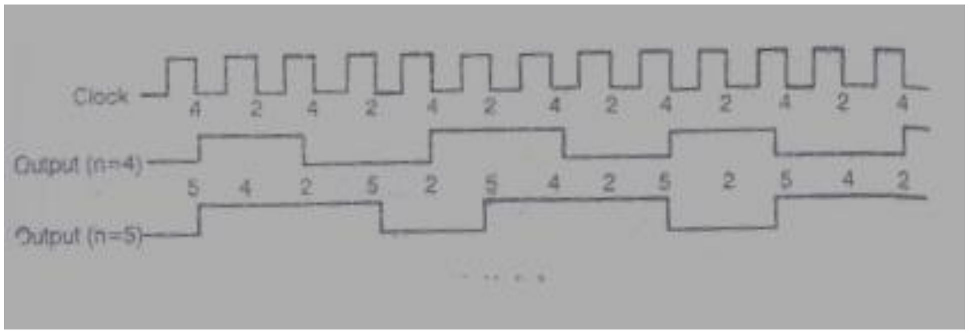
- **Mode 3: Square wave mode:**
- Mode 3 is typically used for baud rate generation.
- Mode 3 is similar to mode 2 except for the duty cycle of OUT, OUT will initially be high.
- When half the initial count has expired, OUT goes low for the reminder of the count.
- Mode 3 is periodic; the sequence above is repeated indefinitely.
- An initial count of N results in a square wave with a period of N CLK cycles.
- Mode 3 is implemented as follows:

EVEN COUNTS:

- OUT is initially high. The initial count is loaded on 1 CLK pulse and then is decremented by two on succeeding CLK pulses.
- When the count expires OUT changes value and the counter is reloaded with the initial count.
- The above process is repeated indefinitely.

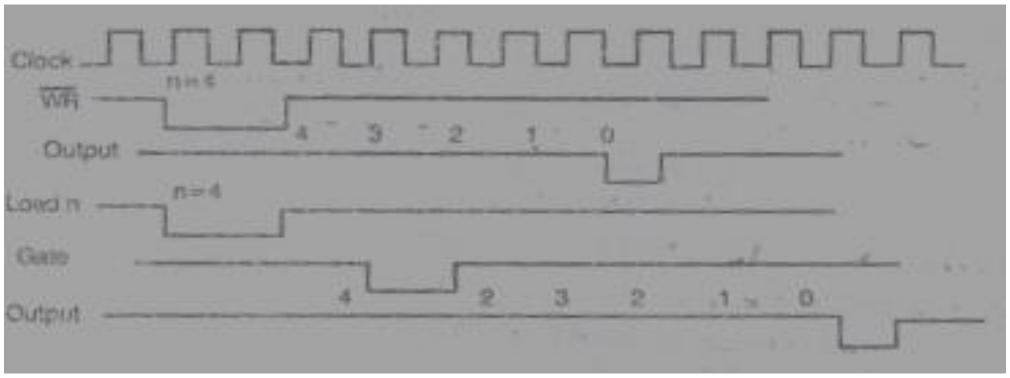
ODD COUNTS:

- For odd counts, OUT will be high for (N+1)/2 counts and low for (N-1)/2 counts.



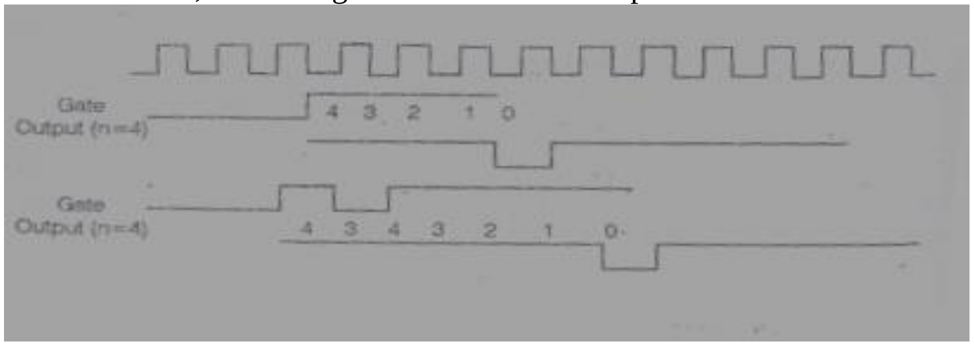
Mode 4: Software triggered Strobe

- The output goes high on setting the mode.
- After terminal count, the output goes low for one clock period and then goes high again.
- In this mode the OUT is initially high; it goes low for clock period at the end of the count.
- The count must be reloaded for subsequent outputs.

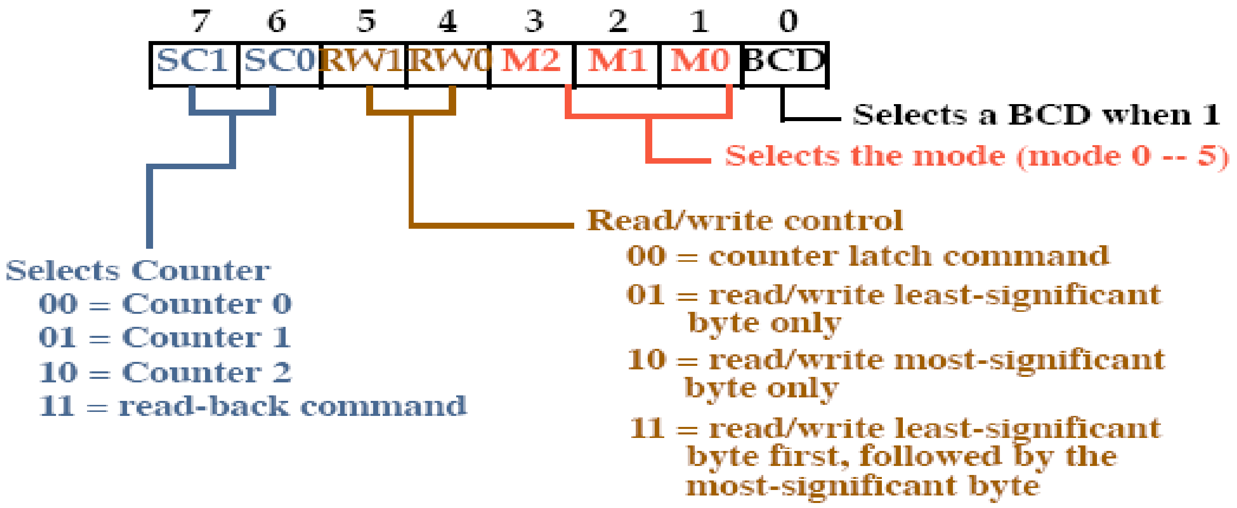


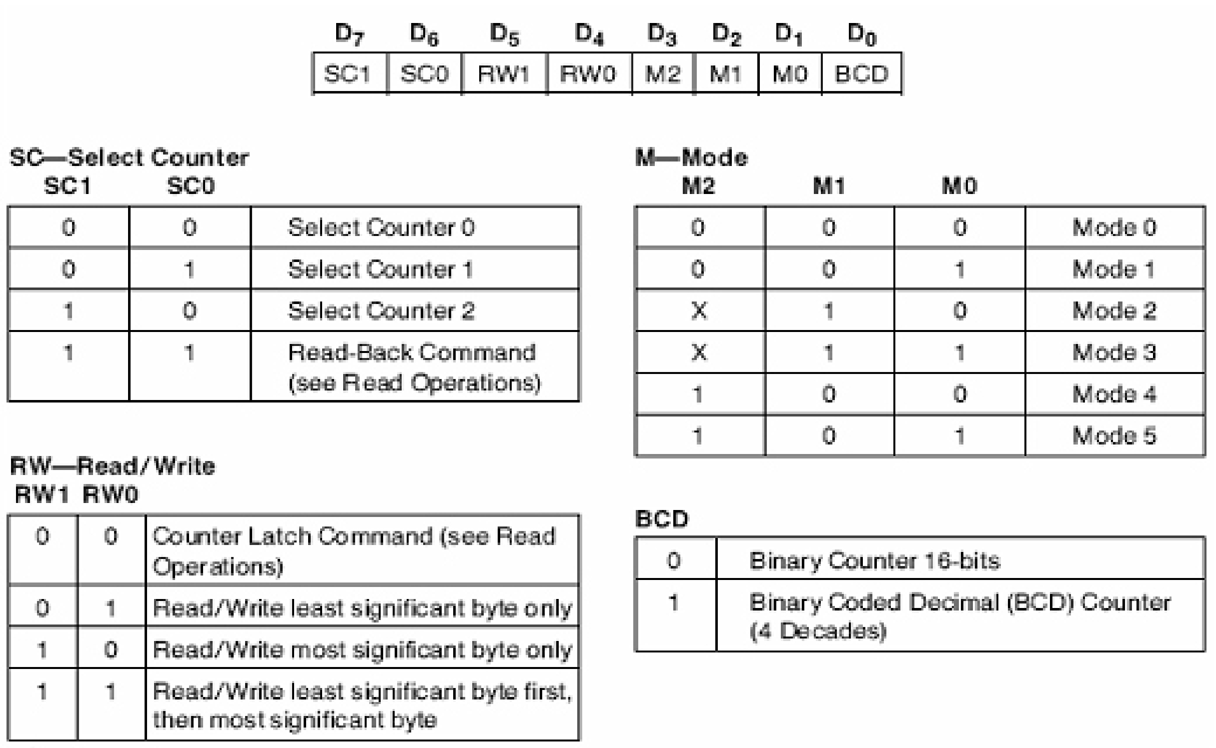
Mode 5: hardware triggered strobe

- This mode is similar to mode 4, but a trigger at the gate initiates the counting.
- This mode is similar to mode 4, except that it is triggered by the rising pulse at the gate.
- Initially the OUT is high and when the gate pulse is triggered from low to high, the count begins, at the end of the count; the OUT goes low for one clock period.



Command word of 8254


7 6 5 4 3 2 1 0
SC1 SC0 RW1 RW0 M2 M1 M0 BCD
Selects a BCD when 1
Selects the mode (mode 0 -- 5)
Read/write control
00 = counter latch command
01 = read/write least-significant byte only
10 = read/write most-significant byte only
11 = read/write least-significant byte first, followed by the most-significant byte
Selects Counter
00 = Counter 0
01 = Counter 1
10 = Counter 2
11 = read-back command



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC—Select Counter

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Read-Back Command (see Read Operations)

M—Mode

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

RW—Read/Write

RW1	RW0	
0	0	Counter Latch Command (see Read Operations)
0	1	Read/Write least significant byte only
1	0	Read/Write most significant byte only
1	1	Read/Write least significant byte first, then most significant byte

BCD

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

NOTE: Don't care bits (X) should be 0 to insure compatibility with future Intel products.

Figure 7. Control Word Format

Each counter may be programmed with a count of 1 to FFFFH. Minimum count is 1 all modes except 2 and 3 with minimum count of 2.Each counter has a program control word used to select the way the counter operates. If two bytes are programmed, then the first byte (LSB) stops the count, and the second byte (MSB) starts the counter with the new count.

6.Discuss in detail about Programming and interfacing 8253

There may be two types of write operations in 8253

i) Writing control word into a control word register
ii) Writing a count value into a count register.
iii) The control word register accepts data from the data buffer and initializes the counter as required.
iv) The control word register contents are used for
    a) Initializing operating modes(Mode 0 to Mode 4)
    b) Selection of counters (Counter0 to counter3)
    c) Choosing binary/BCD counters
    d) Loading the counter register

<u>Read Operations</u>

There are three possible methods for reading the counters:

- a simple read operation
- the Counter Latch Command
- the Read-Back Command

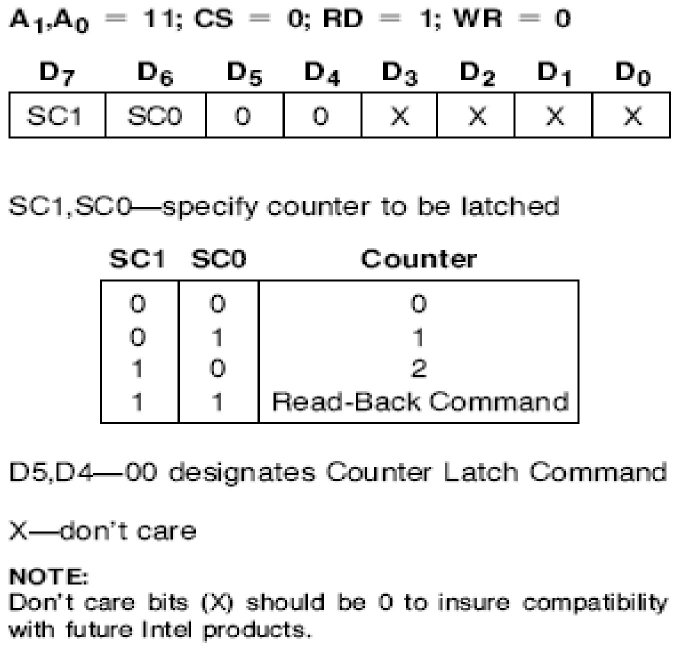
Simple read operation :

- The Counter which is selected with the A1, A0 inputs, the CLK input of the selected Counter must be inhibited by using either the GATE input or external logic. Otherwise, the count may be in the process of changing when it is read, giving an undefined result

Counter Latch Command:

- SC0, SC1 bits select one of the three Counters
- Two other bits, D5 and D4, distinguish this command from a Control Word

If a Counter is latched and then, sometime later, latched again before the count is read, the second Counter Latch Command is ignored. The count read will be the count at the time the first Counter Latch Command was issued.


A_1,A_0 = 11; CS = 0; RD = 1; WR = 0

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,SC0—specify counter to be latched

SC1	SC0	Counter
0	0	0
0	1	1
1	0	2
1	1	Read-Back Command

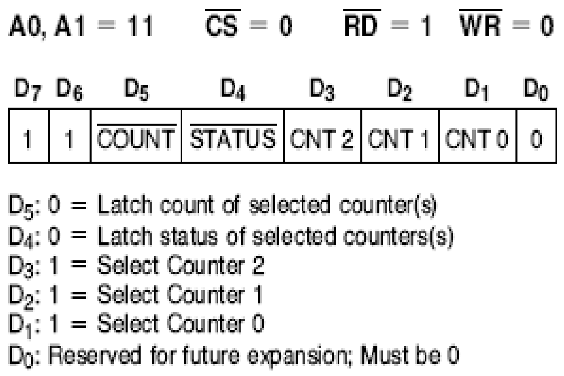
D5,D4—00 designates Counter Latch Command

X—don't care

NOTE:
Don't care bits (X) should be 0 to insure compatibility with future Intel products.

Read-back control command

- The read-back control, word is used when it is necessary for the contents of more than one counter to be read at a same time.
- Count : logic 0, select one of the Counter to be latched
- Status : logic 0, Status must be latched to be read status of a counter is accessed by a read from that counter

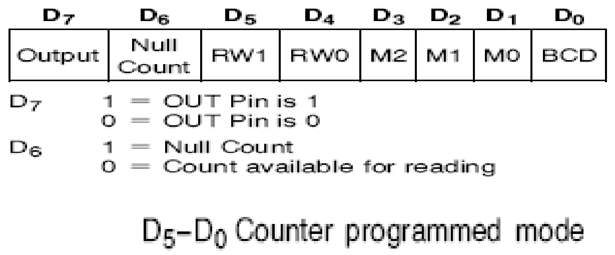

A0, A1 = 11 $\overline{CS}$ = 0 $\overline{RD}$ = 1 $\overline{WR}$ = 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	$\overline{\text{COUNT}}$	$\overline{\text{STATUS}}$	CNT 2	CNT 1	CNT 0	0

D_5: 0 = Latch count of selected counter(s)
D_4: 0 = Latch status of selected counters(s)
D_3: 1 = Select Counter 2
D_2: 1 = Select Counter 1
D_1: 1 = Select Counter 0
D_0: Reserved for future expansion; Must be 0

Status register:

- shows the state of the output pin
- check the counter is in NULL state (0) or not
- how the counter is programmed



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Output	Null Count	RW1	RW0	M2	M1	M0	BCD

D_7	1 = OUT Pin is 1	
	0 = OUT Pin is 0	
D_6	1 = Null Count	
	0 = Count available for reading	

D_5–D_0 Counter programmed mode

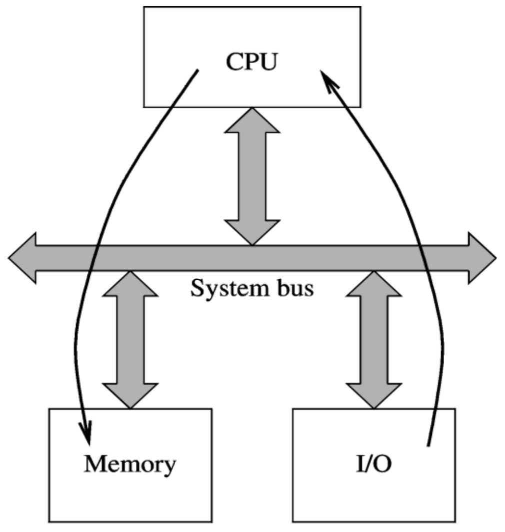
****

7. Explain in detail about Direct Memory Access (DMA Controller 8257)

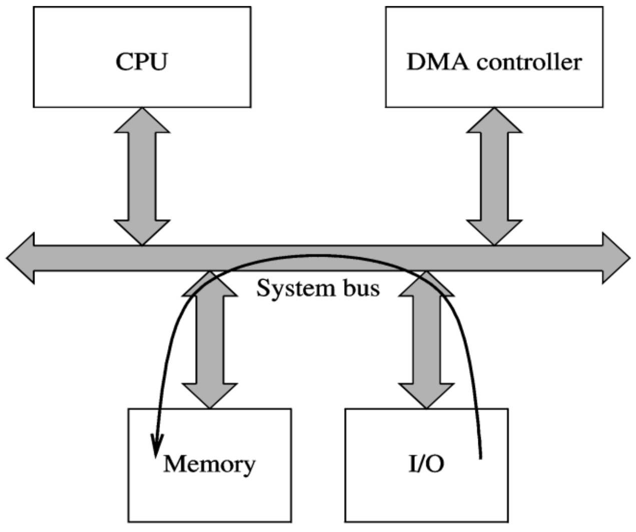
Direct memory access (**DMA**) or DMA mode of data transfer is the fastest amongst all the modes of data transfer. In this mode, the device may transfer data directly to/from memory without any interference from the CPU.

THE **DMA controller (8257)** allows certain hardware subsystems to read/write data to/from memory without microprocessor intervention, allowing the processor to do other work.

The device requests the CPU (through a DMA controller) to hold its data, address and control bus, so that the device may transfer data directly to/from memory. The DMA data transfer is initiated only after receiving HLDA signal from the CPU. For facilitating DMA type of data transfer between several devices, a DMA controller may be used.



(a) Programmed I/O transfer



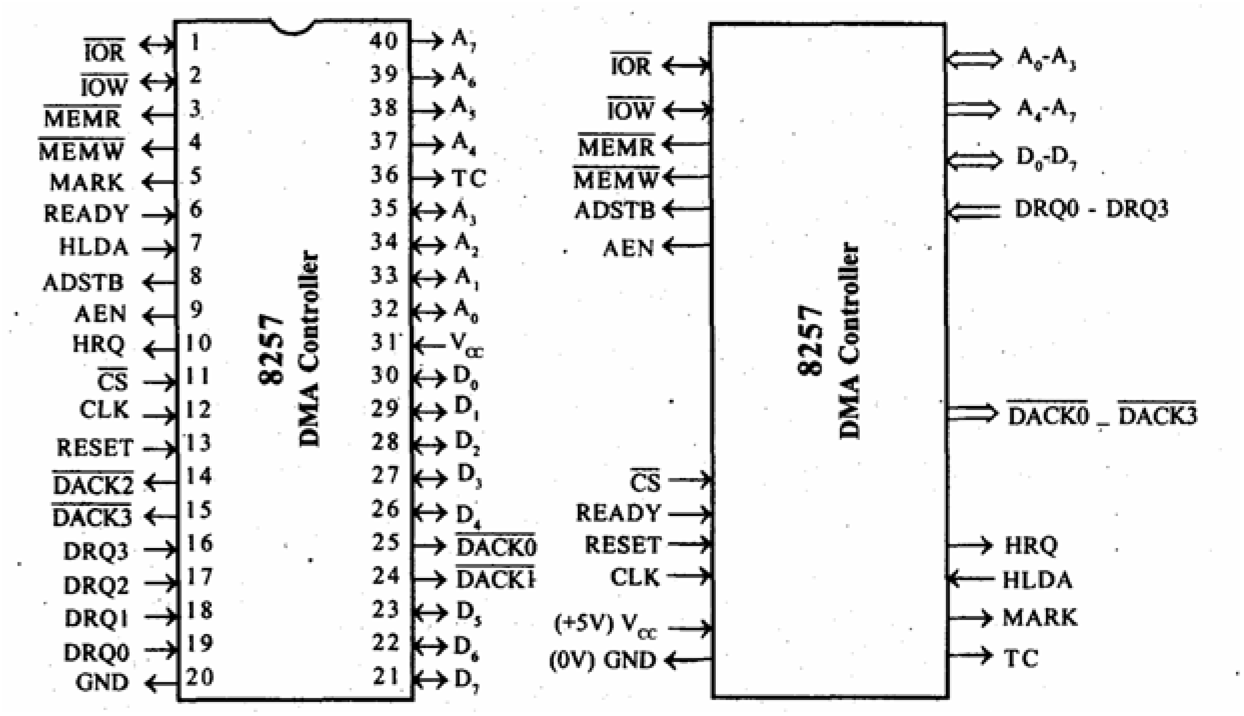
(b) DMA transfer

It is used in disk controllers, video/sound cards etc, or between memory locations. Typically, the CPU initiates DMA transfer, does other operations while the transfer is in progress, and receives an interrupt from the DMA controller once the operation is complete.

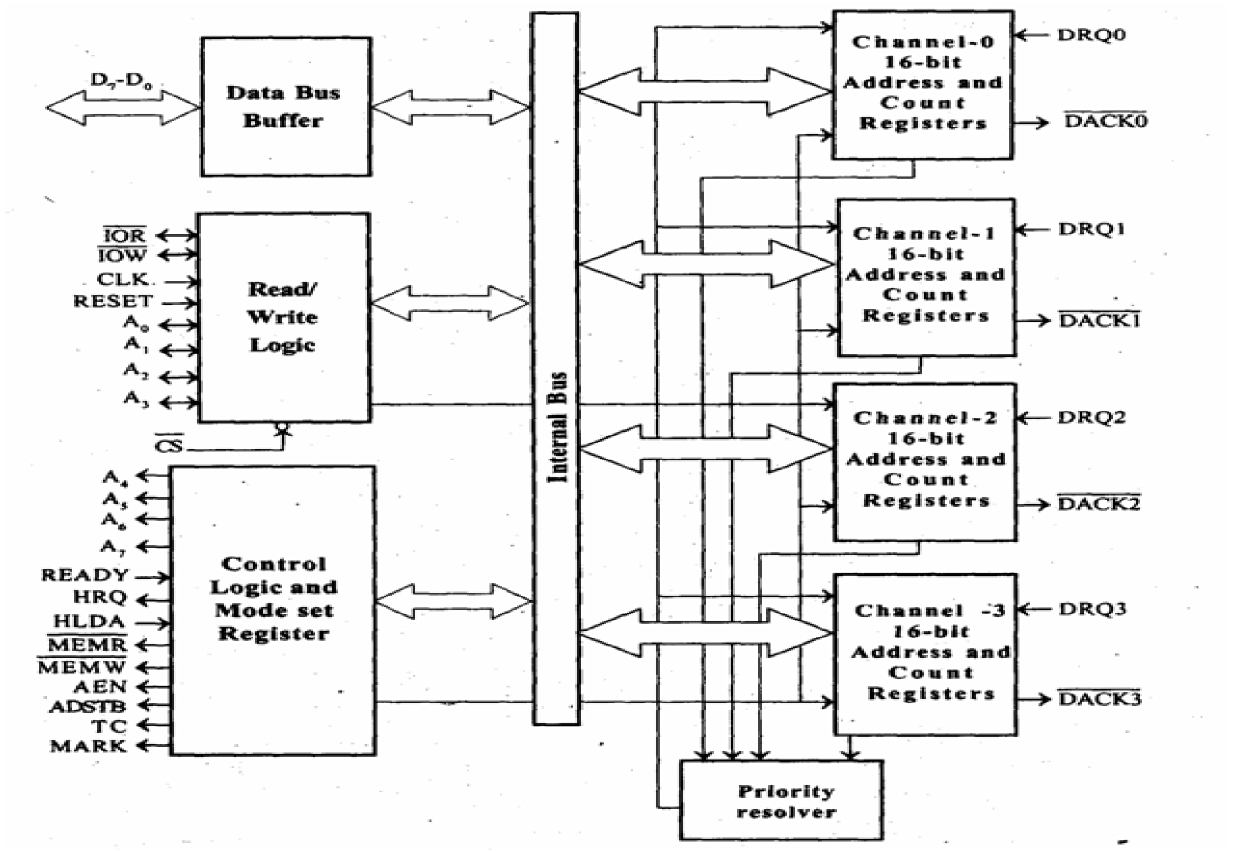
It contains Five main Blocks.

1. **Data bus buffer**
2. **Read/Control logic**
3. **Control logic block**
4. **Priority resolver**
5. **DMA channels.**

Pin diagram of 8257:



Block diagram of 8257:


D7-D0
Data Bus Buffer
Read/ Write Logic
Control Logic and Mode set Register
Internal Bus
Channel-0 16-bit Address and Count Registers
Channel-1 16-bit Address and Count Registers
Channel-2 16-bit Address and Count Registers
Channel -3 16-bit Address and Count Registers
Priority resolver
DRQ0
DACK0
DRQ1
DACK1
DRQ2
DACK2
DRQ3
DACK3
IOR
IOW
CLK
RESET
A0
A1
A2
A3
CS
A4
A5
A6
A7
READY
HRQ
HLDA
MEMR
MEMW
AEN
ADSTB
TC
MARK

<u>DATA BUS BUFFER:</u>

- It contains tri-state, 8 bit bi-directional buffer.
- **Slave mode**, it transfers data between microprocessor and internal data bus.
- **Master mode**, the outputs A8-A15 bits of memory address on data lines (Unidirectional).

READ/CONTROL LOGIC:

- It controls all internal Read/Write operation.
- Slave mode ,it accepts address bits and control signal from microprocessor.
- Master mode, it generates address bits and control signal.

Control logic block:

It contains ,

1. Control logic
2. Mode set register and
3. Status Register.

CONTROL LOGIC:

Master mode,

It control the sequence of DMA operation during all DMA cycles.

It generates address and control signals.

It increments 16 bit address and decrement 14 bit counter registers.

It activate a HRQ signal on DMA channel Request.

Slave mode it is disabled.

D_0 –D7

- it is a bidirectional ,tri state ,Buffered ,Multiplexed data (D0-D7)and (A8-A15).
- In the slave mode it is a bidirectional (Data is moving).

In the Master mode it is a unidirectional (Address is moving)

IOR

- It is active low, tri-state, buffered, Bidirectional lines.
- **In the slave mode it function as a input line**. IOR signal is generated by microprocessor to read the contents 8257 registers.
- **In the master mode it function as a output line.** IOR signal is generated by 8257 during write cycle

IOW

- It is active low, tri-state ,buffered ,Bidirectional control lines.
- **In the slave mode it function as a input line**. IOR signal is generated by microprocessor to write the contents 8257 registers.
- **In the master mode it function as a output line.** IOR signal is generated by 8257 during read cycle

CLK:

- It is the input line, connected with TTL clock generator.
- This signal is ignored in slave mode.

RESET:

- Used to clear mode set registers and status registers

A0-A3:

These are the tri-state, buffer, bidirectional address lines.

In slave mode, these lines are used as address inputs lines and internally decoded to access the internal registers**.**
In master mode, these lines are used as address outputs lines, A0-A3 bits of memory address on the lines.

- It is active low, Chip select input line.
- In the slave mode, it is used to select the chip.
- In the master mode, it is ignored**.**

A4-A7:

These are the tristate, buffer, output address lines.
In slave mode ,these lines are used as address input lines.
In master mode, these lines are used as address outputs lines, A0-A3 bits of memory address on the lines.

READY:

- It is an asynchronous input line.
- **In master mode,**

When ready is high it receives the signal.
When ready is low, it adds wait state between S1 and S3

- **In slave mode,** this signal is ignored.

HRQ:
It is used to **receiving the hold request** signal from the output device.
HLDA:

- It is acknowledgment signal from microprocessor.

MEMR:

- It is active low, tristate, Buffered control output line.
- In slave mode, it is tristated.
- In master mode, it activated during DMA read cycle.

MEMW:

- It is active low, tristate, Buffered control input line.
- In slave mode, it is tristated.
- In master mode, it activated during DMA write cycle.

AEN (Address enable):

- It is a control output line.
- In master mode ,it is high
- In slave mode ,it is low
- Used it isolate the system address, data, and control lines.

ADSTB: (Address Strobe)

- It is a control output line.
- Used to split data and address line.
- It is working in master mode only.
- In slave mode it is ignore.

TC (Terminal Count):

- It is a status of output line.
- It is **activated in master mode only.**
- It is high, it selected the peripheral.
- It is low, it is free and looking for a new peripheral.

MARK:

- It is a **modulo 128 MARK output line**.
- It is activated in master mode only.
- It goes high, after **transferring every 128 bytes of data block.**

DMA controller

- A DMA controller is capable of becoming the bus master and supervising a transfer between an I/O or mass storage interface and memory. While making a transfer, it must be able to place memory address on the bus and send and receive handshaking signals in a manner similar to that of the bus control logic. The purpose of a DMA controller is to perform a sequence of transfers (ie a block transfer) by stealing bus cycles.
- A DMA controller is designed to service one or more I/O mass storage interfaces, and each interface is connected to the controller by a set of conductors. A portion of a DMA controller for servicing a single interface is called a channel.
- The general organization of a one channel DMA controller and its principal connection is shown in figure. In addition to the usual control and status registers, each channel must contain an address register and a byte (or word) count register.
- Initializing the controller consists of filling these registers with the beginning (or ending) address of the memory array that is to be used as a buffer and the number of bytes (words) to be transferred .For an input to memory, each time the interface has data to transfer it makes a DMA request. The controller then makes a bus request and when it receives a bus grant, it puts the contents of the address register on the address bus, sends an acknowledgement back to the interface, and issues I/O read and memory write signals. The interface then puts the data on the data bus and drops its request. When the memory accepts the data it returns a ready signal to the controller, which then increments (or decrements) the address register, decrements the byte (word) count, and drops its bus request.
- Upon the count reaching zero, the process stops and a signal is sent to the processor as an interrupt request or to the interface to notify it that the transfers have terminated. An output is similarly executed except that the controller issues I/O write and memory read signals and the data are transferred in the other direction.

DRQ0-DRQ3 (DMA Request):

- These are the asynchronous peripheral request input signal.
- The request signals are generated by external peripheral device.

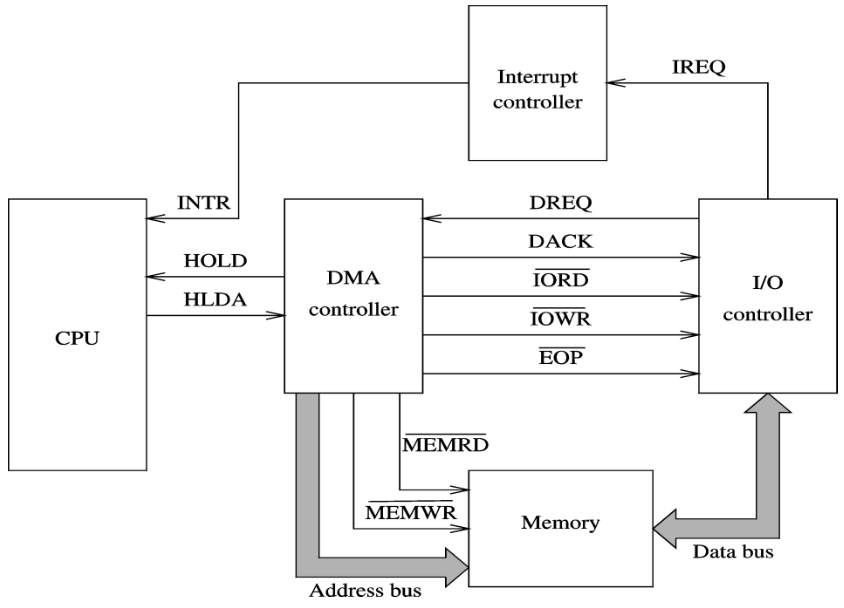
DACK0-DACK3:

- These are the active low DMA acknowledge output lines.
- Low level indicate that, peripheral is selected for giving the information (DMA cycle).

In master mode it is used for chip select

HLDA becomes active to indicate the processor has placed its buses at high-impedance state as can be seen in the timing diagram, there are a few clock cycles between the time that HOLD changes and until HLDA changes

HLDA output is a signal to the requesting device that the processor has relinquished control of its memory and I/O space one could call HOLD input a DMA request input and HLDA output a DMA grant signal



Steps in a DMA operation

- Processor initiates the DMA controller gives device number, memory buffer pointer, called ***channel initialization***
- Once initialized, it is ready for data transfer.
- When ready, I/O device informs the DMA controller .DMA controller starts the data transfer process
- Obtains bus by going through bus arbitration
- Places memory address and appropriate control signals
- Completes transfer and releases the bus
- Updates memory address and count value
- If more to read, loops back to repeat the process
- Notify the processor when done typically uses an interrupt

<u>Modes of DMA operation</u>

Each channel may be put in one of four modes, with its current mode being determined by bits 7 and6 of the channel's mode register. The four possible modes are

Single transfer mode (01)
After each transfer the controller will release the bus to the processor for at least one by cycle, but will immediately begin testing for DREQ inputs and proceed to steal another cycle as soon as a DREQ line becomes active.

Block transfer mode (10)
DREQ need only be active until DACK becomes active, after which the bus is not released until the entire block of data has been transferred.

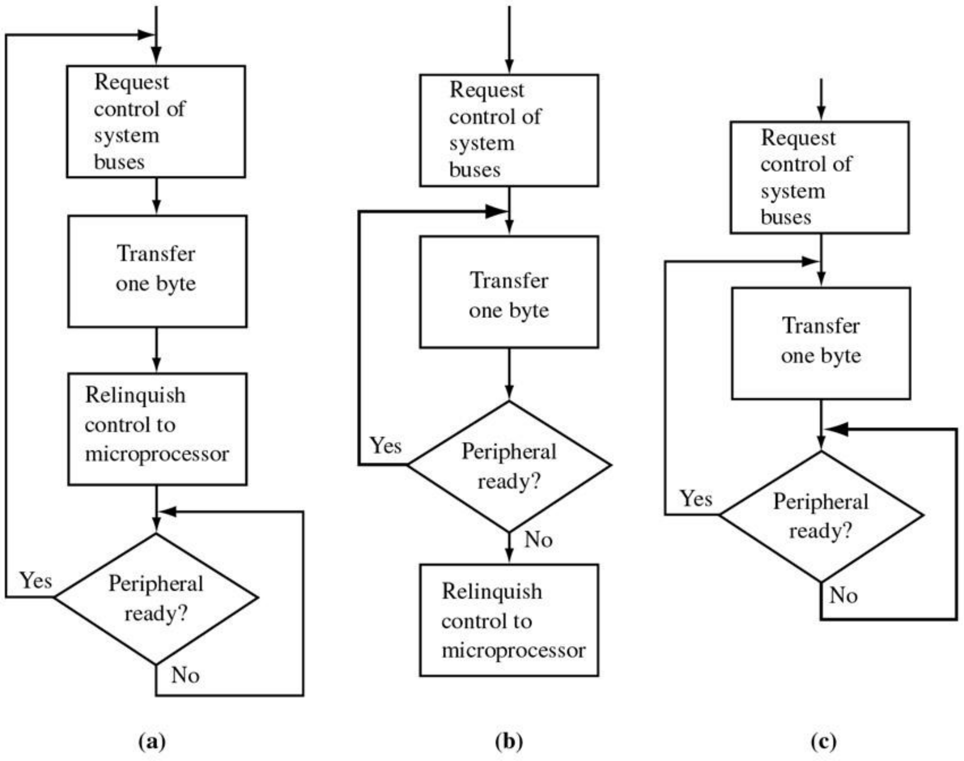
Demand Transfer mode(00)
This is similar to the block mode except that DREQ is tested after each transfer. If DREQ is inactive, transfers are suspended until DREQ once again becomes active, at which time the block transfer continues from the point at which it was suspended. This allows the interface to stop the transfer in the event that its device cannot keep up.

Cascade Mode (11)
In this mode 8237s may be cascaded so that more than four channels can be included in the DMA subsystem. In cascading the controllers, those in the second level are connected to those in the first level by joining HRQ to DREQ and HLDA to DACK, To conserve space, this mode will not be considered further.

In this mode
Single-cycle mode: DMA data transfer is done one byte at a time

***Burst-mode**:* DMA transfer is finished when all data has been moved

a) Byte **b) Burst** **c) Block**



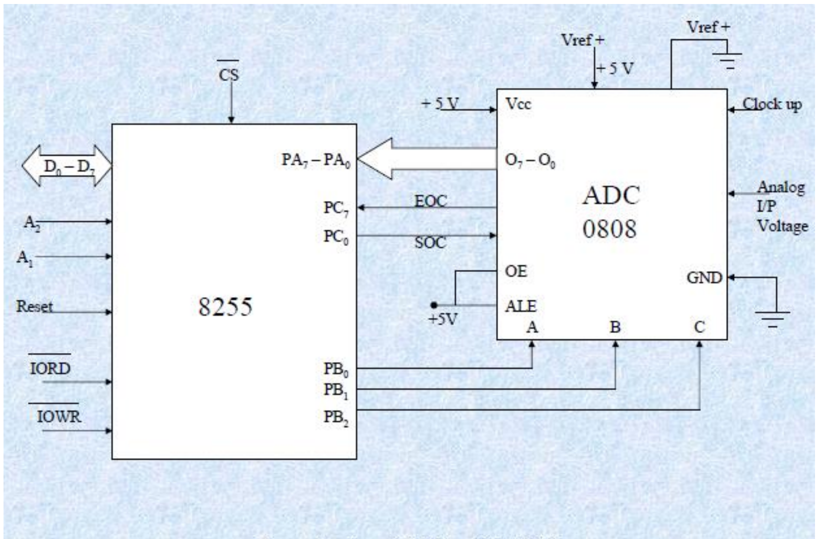
Request control of system buses
Transfer one byte
Relinquish control to microprocessor
Peripheral ready?
Yes
No
(a)
Request control of system buses
Transfer one byte
Peripheral ready?
Yes
No
Relinquish control to microprocessor
(b)
Request control of system buses
Transfer one byte
Peripheral ready?
Yes
No
(c)

8. Write in detail about Analog to digital conversion (ADC)

- The process of analog to digital conversion is a slow process, and the microprocessor has to wait for the digital data till the conversion is over. After the conversion is over, the ADC sends end of conversion EOC signal to inform the microprocessor that the conversion is over and the result is ready at the output buffer of the ADC. These tasks of issuing an SOC pulse to ADC, reading EOC signal from the ADC and reading the digital output of the ADC are carried out by the CPU using 8255 I/O ports.
- The time taken by the ADC from the active edge of SOC pulse till the active edge of EOC signal is called as the conversion delay of the ADC.
- It may range anywhere from a few microseconds in case of fast ADC to even a few hundred milliseconds in case of slow ADCs.
- The available ADC in the market use different conversion techniques for conversion of analog signal to digitals. Successive approximation techniques and dual slope integration techniques are the most popular techniques used in the integrated ADC chip.

General algorithm for ADC interfacing contains the following steps:

1. Ensure the stability of analog input, applied to the ADC.
2. Issue start of conversion (SOC) pulse to ADC
3. Read end of conversion signal to mark the end of conversion processes.
4. Read digital data output of the ADC as equivalent digital output.



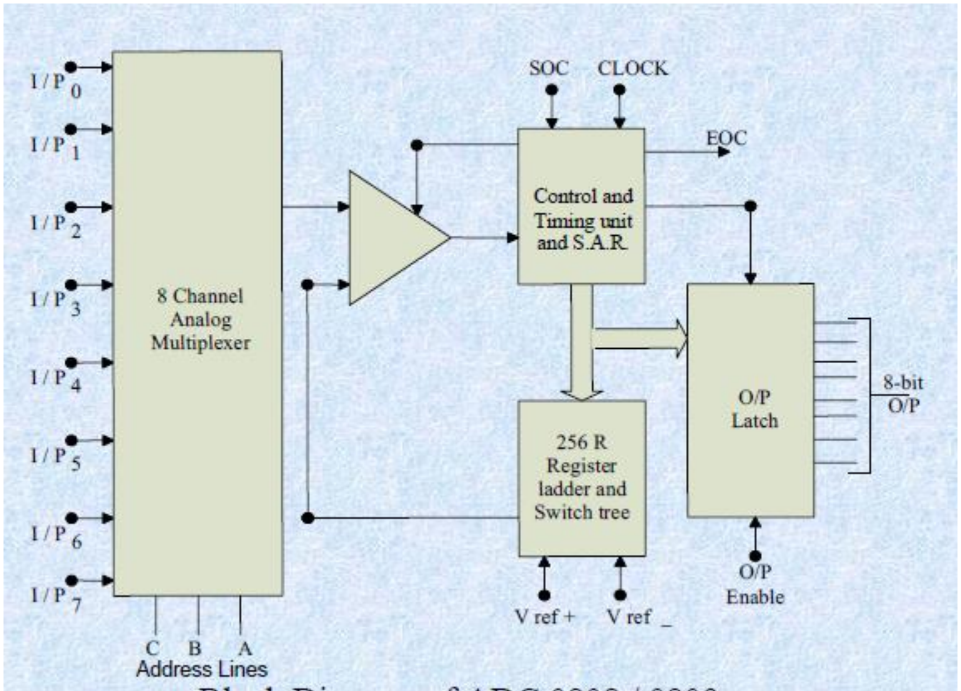
Interfacing 0808 with 8086

- Analog input voltage must be constant at the input of the ADC right from the start of conversion till the end of the conversion to get correct results. This may be ensured by a sample and hold circuit which samples the analog signal and holds it constant for specific time duration.
- The microprocessor may issue a hold signal to the sample and hold circuit. If the applied input changes before the complete conversion process is over, the digital equivalent of the analog input calculated by the ADC may not be correct.

ADC 0808/0809 :

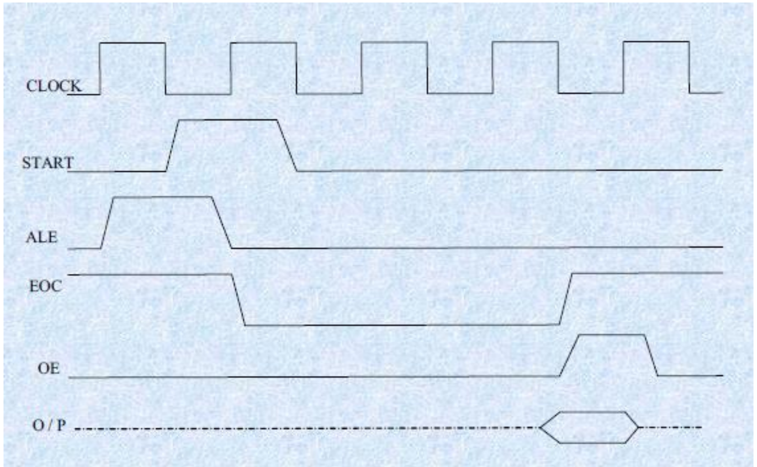
- The analog to digital converter chips 0808 and 0809 are 8-bit CMOS, successive approximation converters. This technique is one of the fast techniques for analog to digital conversion.
- The conversion delay is 100µs at a clock frequency of 640 KHz, which is quite low as compared to other converters.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

circuit.If one needs a sample and hold circuit for the conversion of fast signal into equivalent digital quantities, it has to be externally connected at each of the analog inputs.



Block Diagram of ADC 0808 / 0809

• Vcc	Supply pins +5V
• GND	GND
• Vref+	Reference voltage positive +5 Volts maximum.
• Vref_	Reference voltage negative 0Volts sminimum
• I/P0–I/P7	Analog inputs
• ADD A,B,C	Address lines for selecting analog inputs.
• O7 – O0	Digital 8-bit output with O7 MSB and O0 LSB
• SOC	Start of conversion signal pin
• EOC	End of conversion signal pin
• OE	Output latch enable pin, if high enables output
• CLK	Clock input for ADC



Timing Diagram of ADC 0808

Example: Interfacing ADC 0808 with 8086 using 8255 ports. Use port A of 8255 for transferring digital data output of ADC to the CPU and port C for control signals. Assume that an analog input is present at I/P2 of the ADC and a clock input of suitable frequency is available for ADC.

• **Solution**: The analog input I/P2 is used and therefore address pins A,B,C should be 0,1,0 respectively to select I/P2. The OE and ALE pins are already kept at +5V to select the ADC and enable the outputs. Port C upper acts as the input port to receive the EOC signal while port C
lower acts as the output port to send SOC to the ADC.

Port A acts as a 8-bit input data port to receive the digital data output from the ADC. The 8255 control word is written as follows:

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	1	0	0	0

• The required ALP is as follows:

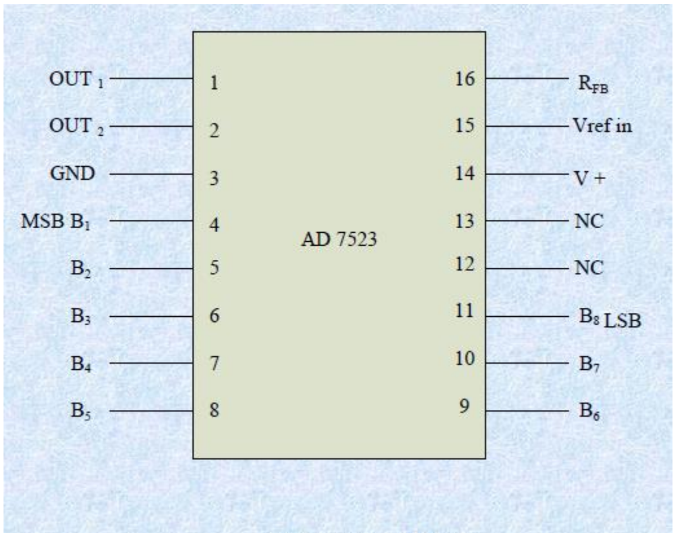
```
MOV AL, 98h          ;initialise 8255 as
OUT CWR, AL          ;discussed above.
MOV AL, 02h          ;Select I/P2 as analog
OUT Port B, AL        ;input.

MOV AL, 00h          ;Give start of conversion
OUT Port C, AL       ; pulse to the ADC
MOV AL, 01h
OUT Port C, AL
MOV AL, 00h
OUT Port C, AL
WAIT: IN AL, Port C  ;Check for EOC by
RCR                   ; reading port C upper and
JNC WAIT              ;rotating through carry.
IN AL, Port A        ;If EOC, read digital equivalent
                     ;in AL
HLT                   ;Stop
```

****

9. Explain in detail about Interfacing Digital to Analog Converters

- The digital to analog converters convert binary number into their equivalent voltages. The DAC find applications in areas like digitally controlled gains, motors speed controls, programmable gain amplifiers etc.
- AD7523 8-bit Multiplying DAC : This is a 16 pin DIP, multiplying digital to analog converter, containing R-2R ladder for D-A conversion along with single pole double thrown NMOS switches to connect the digital inputs to the ladder.



Pin Diagram of AD 7523

- The pin diagram of AD7523 is shown in fig the supply range is from +5V to +15V, while Vref may be any where between -10V to +10V. The maximum analog output voltage will be any where between -10V to +10V, when all the digital inputs are at logic high state.
- Usually a zener is connected between OUT1 and OUT2 to save the DAC from negative transients. An operational amplifier is used as a current to voltage converter at the output of AD to convert the current output of AD to a proportional output voltage.
  It also offers additional drive capability to the DAC output.An external feedback resistor acts to control the gain. One may not connect any external feedback resistor, if no gain control is required.
- ***EXAMPLE***: Interfacing DAC AD7523 with an 8086 CPU running at 8MHZ and write an assembly language program to generate a saw tooth waveform of period 1ms with Vmax 5V.
- Solution: Fig shows the interfacing circuit of AD 74523 with 8086 using 8255. program gives an

ALP to generate a saw tooth waveform using circuit.

```
ASSUME CS:CODE
CODE SEGMENT
START: MOV AL,80h ;make all ports output
OUT CW, AL
AGAIN: MOV AL,00h ;start voltage for ramp
BACK : OUT PA, AL
INC AL
CMP AL, 0FFh
JB BACK
JMP AGAIN
CODE ENDS
END START
```

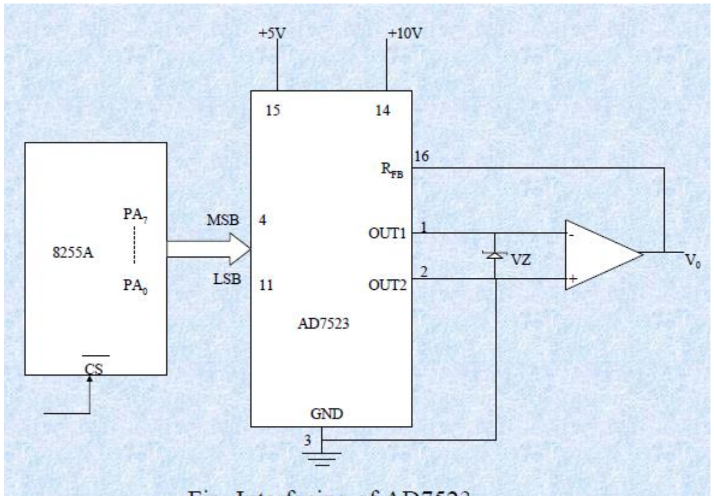


Fig: Interfacing of AD7523

- In the above program, port A is initialized as the output port for sending the digital data as input to DAC. The ramp starts from the 0V (analog), hence AL starts with 00H. To increment the ramp, the content of AL is increased during each execution of loop till it reaches F2H.
- After that the saw tooth wave again starts from 00H, i.e. 0V (analog) and the procedure is repeated. The ramp period given by this program is precisely 1.000625 ms. Here the count F2H has been calculated by dividing the required delay of 1ms by the time required for the execution of the loop once. The ramp slope can be controlled by calling a controllable delay after the OUT instruction

****

10. Draw the block diagram of 8259A and explain how to program 8259A (April 2010).

Programmable Interrupt controller (8259)

Introduction:

For applications where we have interrupts from multiple source, we use an external device called a ***priority interrupt controller*** (PIC) to the interrupt signals into a single interrupt input on the processor.

It accepts requests from the peripheral equipment, determines which of the incoming requests is of the highest importance (priority), ascertains whether the incoming request has a higher priority value than the level currently being serviced, and issues an interrupt to the CPU based on this determination.

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

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

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

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

• **Interrupt Control Logic**: This block manages the interrupt and interrupt acknowledge signals to be sent to the CPU for serving one of the eight interrupt requests. This also accepts the interrupt acknowledge (INTA) signal from CPU that causes the 8259A to release vector address on to the data bus.
• **Data Bus Buffer** : This tristate bidirectional buffer interfaces internal 8259A bus to the microprocessor system data bus. Control words, status and vector information pass through data buffer during read or write operations.
• **Read/Write Control Logic**: This circuit accepts and decodes commands from the CPU. This block also allows the status of the 8259A to be transferred on to the data bus.
• **Cascade Buffer/Comparator**: This block stores and compares the ID's all the 8259A used in system. The three I/O pins CASO-2 are outputs when the 8259A is used as a master. The same pins act as inputs when the 8259A is in slave mode. The 8259A in master mode sends the ID of the interrupting slave device on these lines. The slave thus selected, will send its preprogrammed vector address on the data bus during the next INTA pulse.
• **CS**: This is an active-low chip select signal for enabling RD and WR operations of 8259A. INTA function is independent of CS.
• **WR:** This pin is an active-low write enable input to 8259A. This enables it to accept command words from CPU.
• **RD:** This is an active-low read enable input to 8259A. A low on this line enables 8259A to release status onto the data bus of CPU.
• **D0-D7** : These pins from a bidirectional data bus that carries 8-bit data either to control word or from status word registers. This also carries interrupt vector information.
• **CAS0 – CAS2 Cascade Lines:** A signal 8259A provides eight vectored interrupts. If more interrupts are required, the 8259A is used in cascade mode. In cascade mode, a master 8259A along with eight slaves 8259A can provide up to 64 vectored interrupt lines. These three lines act as select lines for addressing the slave 8259A.
• **PS/EN** : This pin is a dual purpose pin. When the chip is used in buffered mode, it can be used as buffered enable to control buffer transreceivers. If this is not used in buffered mode then the pin is used as input to designate whether the chip is used as a master (SP =1) or slave (SP = 0).
• **INT** : This pin goes high whenever a valid interrupt request is asserted. This is used to interrupt the CPU and is connected to the interrupt input of CPU.
• **IR0 – IR7 (Interrupt requests)** :These pins act as inputs to accept interrupt request to the CPU. In edge triggered mode, an interrupt service is requested by raising an IR pin from a low to a high state and holding it high until it is acknowledged, and just by latching it to high level, if used in level triggered mode.

A0

This input signal is used in conjunction with WR and RD signals to write commands into the various command registers, as well as reading the various status registers of the chip. This line can be tied directly to one of the address lines.

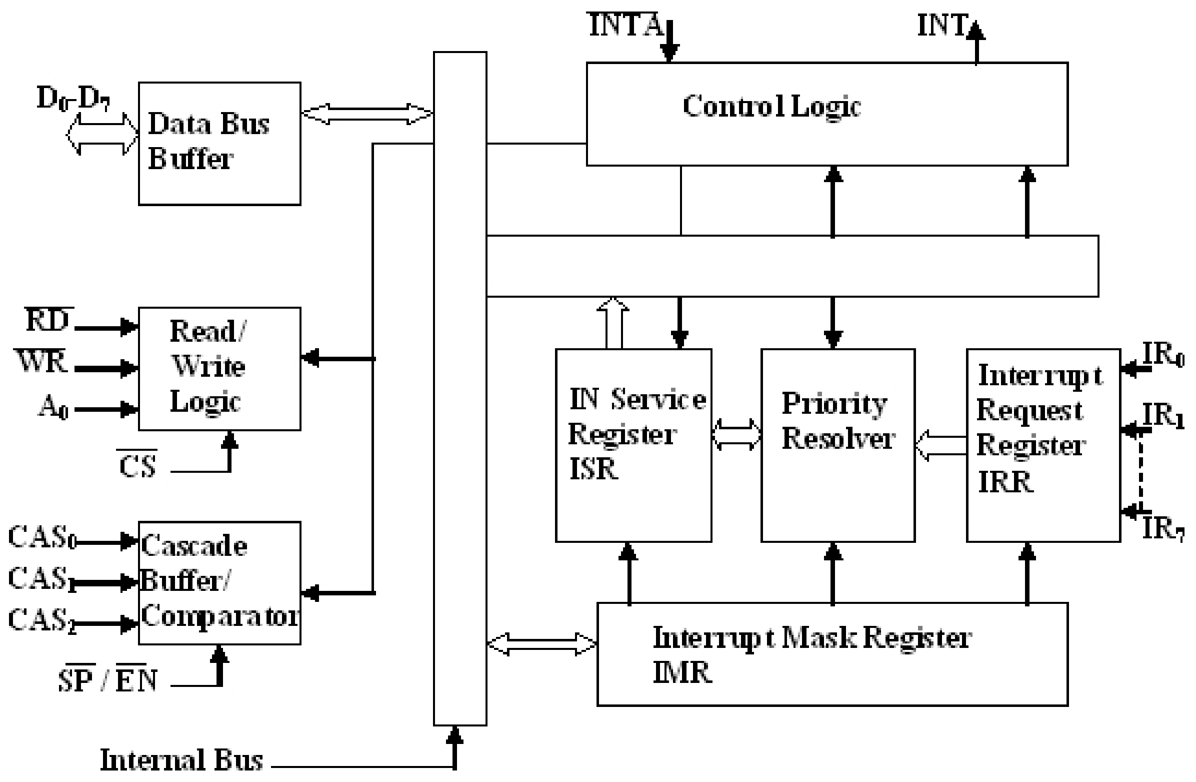


Fig:1 8259A Block Diagram

Interrupt Sequence in an 8086 system

The Interrupt sequence in an 8086-8259A system is described as follows:
1. One or more IR lines are raised high that set corresponding IRR bits.
2. 8259A resolves priority and sends an INT signal to CPU.
3. The CPU acknowledge with INTA pulse.
4. Upon receiving an INTA signal from the CPU, the highest priority ISR bit is set and the corresponding IRR bit is reset. The 8259 will also release a CALL instruction code (11001101) on to the 8 bit data bs through its D7 D0 pins.
5. The CALL instruction will initiate a second INTA pulse. During this period 8259A releases an 8-bit pointer on to a data bus from two more INTA pulses to be sent to the 8259 from the CPU group.
6.These two INTA pulses allow the 8259 to release its programmed subroutine address onto the data bits. The lower 8 bit address is released at the first INTA pulse and the higher 8 bit address is released at the second INTA pulse.
6. This completes the 3 byte CALL instruction released by the 8259. interrupt cycle. The ISR bit is reset at the end of the second INTA pulse if automatic end of interrupt (AEOI) mode is programmed. Otherwise ISR bit remains set until an appropriate EOI command is issued at the end of interrupt subroutine.

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

1. **Initialization Command Words (ICWs):**
Before normal operation can begin, each 8259A in the system must be brought to a starting pointed by a sequence of 2 to 4 bytes timed by WR pulses.

2. Operational Command Words (OCWs):

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

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

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

INITIALIZATION COMMAND WORDS (ICWS)

Initialization Command Words (ICW): Before it starts functioning, the 8259A must be initialized by writing two to four command words into the respective command word registers. These are called as initialized command words.

• If A0 = 0 and D4 = 1, the control word is recognized as ICW1. It contains the control bits for edge/level triggered mode, single/cascade mode, call address interval and whether ICW4 is required or not.

• If A0=1, the control word is recognized as ICW2. The ICW2 stores details regarding interrupt vector addresses. The initialization sequence of 8259A is described in form of a flow chart in fig 3 below.

• The bit functions of the ICW1 and ICW2 are self explanatory as shown in fig below.

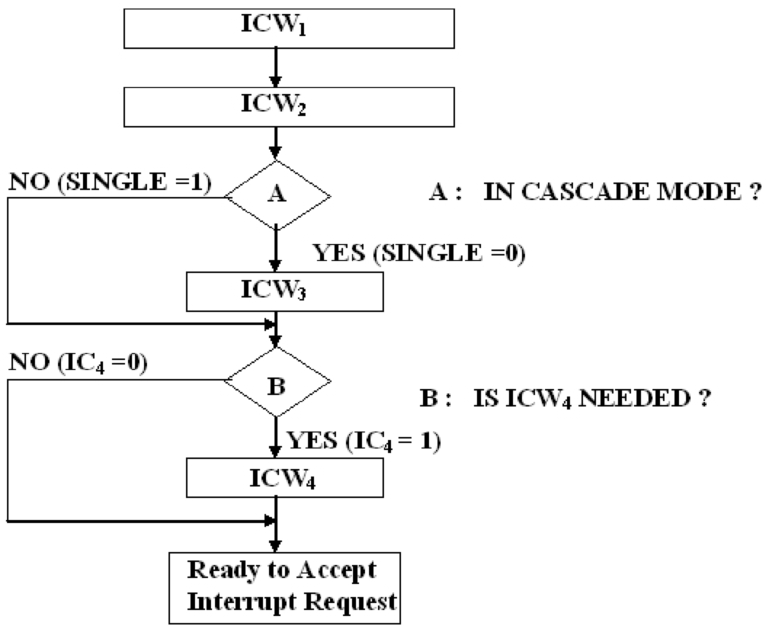
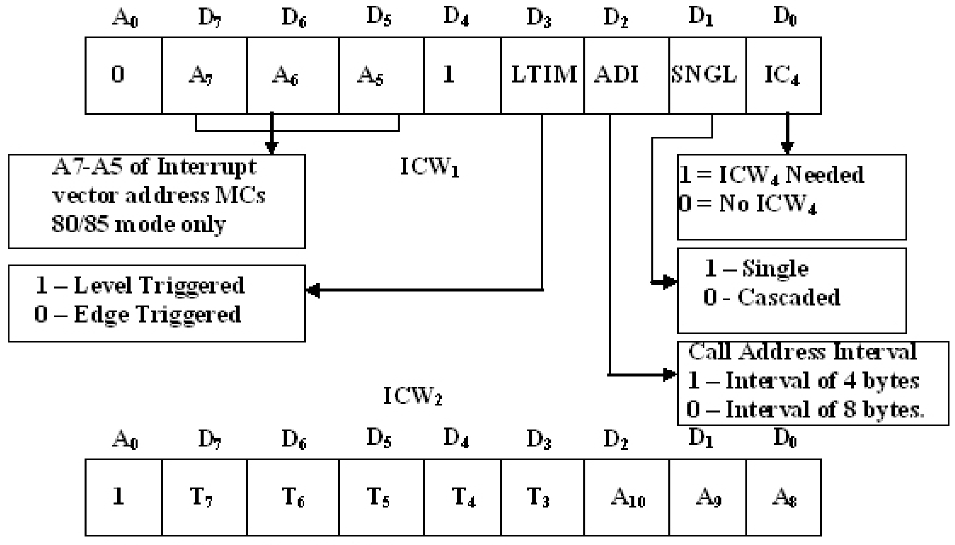


Fig 3: Initialisation Sequence of 8259A

ICW1 :

A write command issued to the 8259 with A_0 =0 and D4 =1 is interpreted as ICW1,which starts the initialization sequence It specifies

1.Single or Multiple 8259 s in the system
2.4 or 8 bit,interval between interrupt vector locations
3. The address bits A7 A5 of the CALL instruction
4. Edge triggered or Level triggered interrupts
5. ICW4 is needed or not


A0 D7 D6 D5 D4 D3 D2 D1 D0
0 A7 A6 A5 1 LTIM ADI SNGL IC4
A7-A5 of Interrupt vector address MCs 80/85 mode only
ICW1
1 = ICW4 Needed
0 = No ICW4
1 – Level Triggered
0 – Edge Triggered
1 – Single
0 - Cascaded
Call Address Interval
1 – Interval of 4 bytes
0 – Interval of 8 bytes.
ICW2
A0 D7 D6 D5 D4 D3 D2 D1 D0
1 T7 T6 T5 T4 T3 A10 A9 A8

- T_7 – T3 are A3 – A0 of interrupt address
- A_{10} – A_9, A_8 – Selected according to interrupt request level. They are not the address lines of Microprocessor
- A0 =1 selects ICW_2

Fig 4 : Instruction Command Words ICW_1 and ICW_2

Once ICW1 is loaded, the following initialization procedure is carried out internally.
a. The edge sense circuit is reset, i.e. by default 8259A interrupts are edge sensitive.
b. IMR is cleared.
c. IR7 input is assigned the lowest priority.
d. Slave mode address is set to 7.
e. Special mask mode is cleared and status read is set to IRR.
f. If IC4 = 0, all the functions of ICW4 are set to zero. Master/Slave bit in ICW4 is used in the buffered mode only.

In 8086 based system A15-A11 of the interrupt vector address are inserted in place of T7 – T3 respectively and the remaining three bits A8, A9, A10 are selected depending upon the interrupt level, i.e. from 000 to 111 for IR0 to IR7.

ICW1 and ICW2 are compulsory command words in initialization sequence of 8259A as is evident from fig, while ICW3 and ICW4 are optional.

The ICW3 is read only when there are more than one 8259A in the system, cascading is used (SNGL=0).The SNGL bit in ICW1 indicates whether the 8259A in the cascade mode or not.
The ICW3 loads an 8-bit slave register. The detailed functions are as follows.

In master mode [SP = 1 or in buffer mode M/S = 1 in ICW4], the 8-bit slave register will be set bit-wise to 1 for each slave in the system

Operation Command Words:

- Once 8259A is initialized using the previously discussed command words for initialisation, it is ready for its normal function, i.e. for accepting the interrupts but 8259A has its own way of handling the received interrupts called as modes of operation.

- These modes of operations can be selected by programming, i.e. writing three internal registers called as operation command words.
- In the three operation command words OCW1, OCW2 and OCW3 every bit corresponds to some operational feature of the mode selected, except for a few bits those are either 1 or 0. The three operation command words are shown in fig with the bit selection details.

OCW1

Issued with A0= 1,used to mask the interrupts. To enable all the IR lines, the command word is 00H**.**

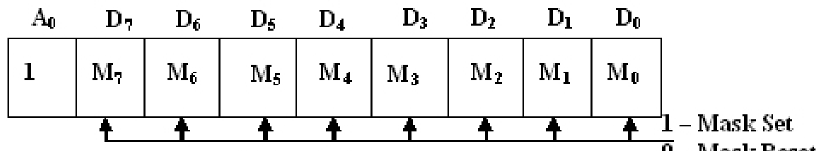


Fig (a) : OCW1

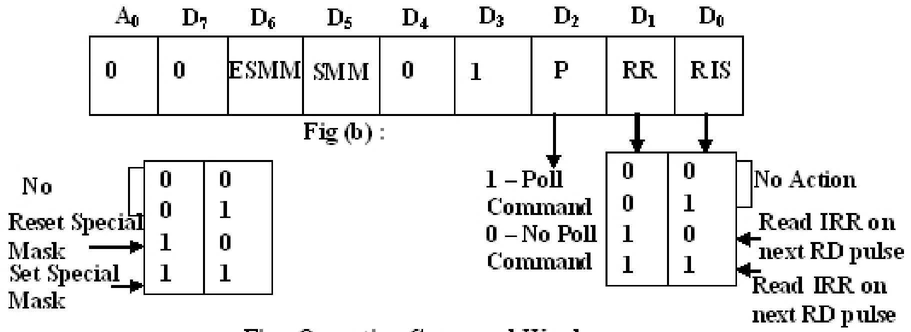

A0 D7 D6 D5 D4 D3 D2 D1 D0
0 0 ESMM SMM 0 1 P RR RIS
No
Reset Special Mask
Set Special Mask
1 – Poll Command
0 – No Poll Command
No Action
Read IRR on next RD pulse
Read IRR on next RD pulse

Fig (b) :

Fig : Operation Command Words

Fig (c) :OCW2

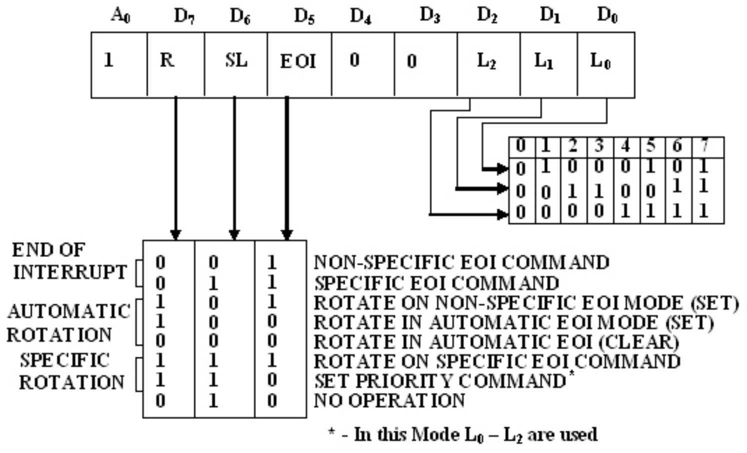

A0 D7 D6 D5 D4 D3 D2 D1 D0
1 R SL EOI 0 0 L2 L1 L0
END OF INTERRUPT
AUTOMATIC ROTATION
SPECIFIC ROTATION
NON-SPECIFIC EOI COMMAND
SPECIFIC EOI COMMAND
ROTATE ON NON-SPECIFIC EOI MODE (SET)
ROTATE IN AUTOMATIC EOI MODE (SET)
ROTATE IN AUTOMATIC EOI (CLEAR)
ROTATE ON SPECIFIC EOI COMMAND
SET PRIORITY COMMAND*
NO OPERATION
* - In this Mode L0 – L2 are used

Fig : Operation Command Word

- In OCW2 the three bits, R, SL and EOI control the end of interrupt, the rotate mode and their combinations as shown in fig below.
- The three bits L2, L1 and L0 in OCW2 determine the interrupt level to be selected for operation, if SL bit is active i.e. 1.
- The details of OCW2 are shown in fig.
- In operation command word 3 (OCW3), if the ESMM bit, i.e. enable special mask mode bit is set to 1, the SMM bit is neglected. If the SMM bit, i.e. special mask mode. When ESMM bit is 0 the SMM bit is neglected. If the SMM bit. i.e. special mask mode bit is 1, the 8259A will enter special mask mode provided ESMM=1.

- If ESMM=1 and SMM=0, the 8259A will return to the normal mask mode. The details of bits of OCW3 are given in fig along with their bit definitions.

<u>Priority Modes</u>

Fully Nested Mode

This mode is entered after initialization unless another mode is programmed. The interrupt requests are ordered in priority from 0 through 7. After the initialization sequence, IR0 has the highest prioirity and IR7 the lowest. Priorities can be changed. When an interrupt is acknowledged the highest priority request is determined and its vector placed on the bus.

0	1	2	3	4	5	6	7
IR0	IR1	IR2	IR3	IR4	IR5	IR6	IR7
↓							↓

Highest priority **Lowest Priority**

Now if IR3 is made highest priority the priorities for other interrupt will also be automatically changed.

5	6	7	0	1	2	3	4
IR0	IR1	IR2	IR3	IR4	IR5	IR6	IR7
		↓	↓				

Lowest priority **Highest priority**

End of Interrupt (EOI)

The In Service (IS) bit can be reset either automatically following the trailing edge of the last in sequence INTA pulse (when AEOI bit in ICW1 is set) or by a command word that must be issued to the 8259A before returning from a service routine (EOI command). An EOI command must be issued twice if in the Cascade mode, once for the master and once for the corresponding slave.

Automatic Rotation(equal Priority): This is used in the applications where all the interrupting devices are of equal priority.

• In this mode, an interrupt request IR level receives priority after it is served while the next device to be served gets the highest priority in sequence. Once all the devices are served like this, the first device again receives highest priority.

Specific rotation mode(Specific Priority)

The programmer can change the priorities by programming the bottom priority and the fixing all other priorities.ie if IR4 is programmed as the lowest priority, then IR% will have the highest one**.**

• **Automatic EOI Mode:** Till AEOI=1 in ICW4, the 8259A operates in AEOI mode.

Special mask mode

- In the special mask mode, when a mask bit is set in OCW1,it inhibits further interrupts at that level and enables interrupts from all other levels that are not masked. Thus any interrupts may be selectively enabled by loading the mask register.

Poll command

- Service to devices is achieved by software using a poll command. So INTA sequence is not needed. It is used to expand the number of priorities levels to more than 64.

12. Explain in detail about Traffic light Control

Traffic lights, which may also be known as stop lights, traffic lamps, traffic signals, signal lights, robots or semaphore, are signaling devices positioned at road intersections, pedestrian crossings and other locations to control competing flows of traffic

ABOUT THE COLORS OF TRAFFIC LIGHT CONTROL

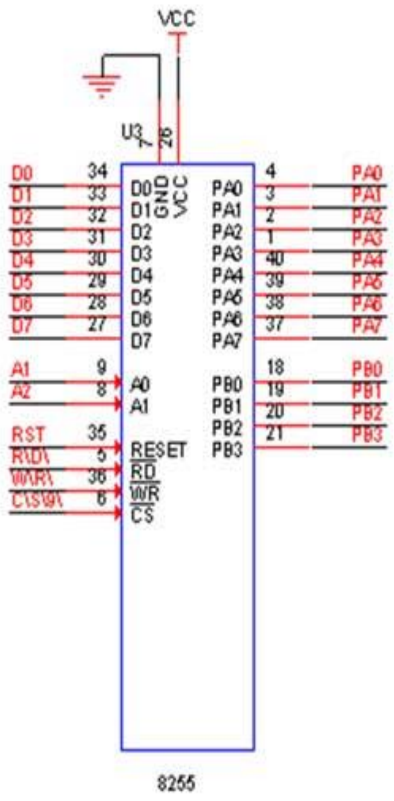
Traffic lights alternate the right of way of road users by displaying lights of a standard color (red, yellow/amber, and green), using a universal color code (and a precise sequence to enable comprehension by those who are color blind).

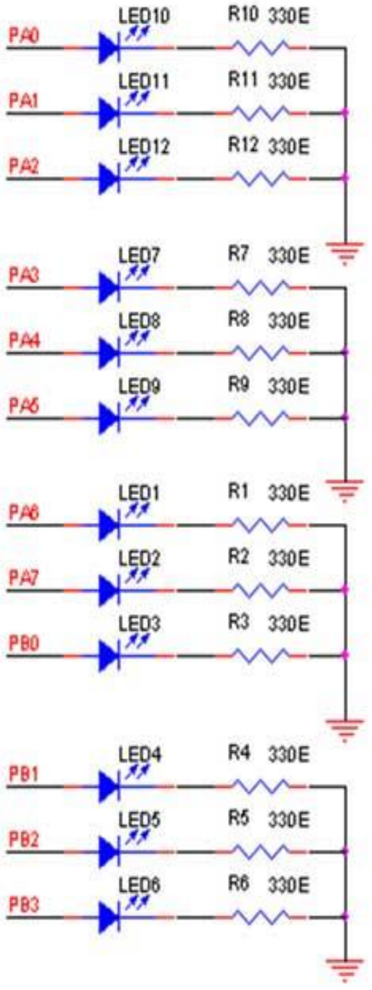
Illumination of the red signal prohibits any traffic from proceeding. Usually, the red light contains some orange in its hue, and the green light contains some blue, for the benefit of people with red-green color blindness, and "green" lights in many areas are in fact blue lenses on a yellow light (which together appear green).

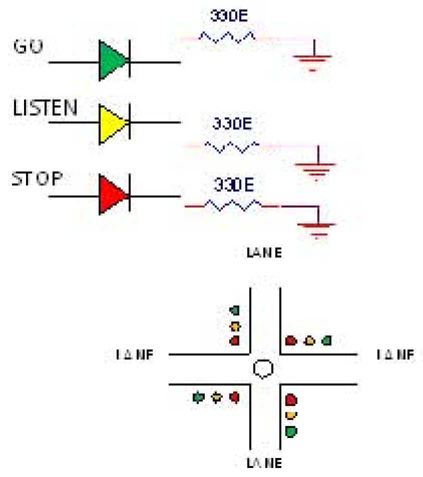
INTERFACING TRAFFIC LIGHT WITH 8086

The Traffic light controller section consists of 12 Nos. point LEDs arranged by 4 Lanes in Traffic light interface card. Each lane has Go (Green), Listen (Yellow) and Stop(Red) LED is being placed

PIN ASSIGNMENT WITH 8086


VCC
U3
GND
VCC
8255
D0 D1 D2 D3 D4 D5 D6 D7
A1 A2
RST
R\D\
W\R\
C\S\0\
RESET
RD
WR
CS
A0 A1
PA0 PA1 PA2 PA3 PA4 PA5 PA6 PA7
PB0 PB1 PB2 PB3


PA0 LED10 R10 330E
PA1 LED11 R11 330E
PA2 LED12 R12 330E
PA3 LED7 R7 330E
PA4 LED8 R8 330E
PA5 LED9 R9 330E
PA6 LED1 R1 330E
PA7 LED2 R2 330E
PB0 LED3 R3 330E
PB1 LED4 R4 330E
PB2 LED5 R5 330E
PB3 LED6 R6 330E

LAN Direction	8086 LINES	MODULES	Traffic Light Controller Card
SOUTH	PA.0	GO	 GO LISTEN STOP 330E 330E 330E LANE LANE LANE LANE
	PA.1	LISTEN	
	PA.2	STOP	
EAST	PA.3	GO	
	PA.4	LISTEN	
	PA.5	STOP	
NORTH	PA.6	GO	
	PA.7	LISTEN	
	PB.0	STOP	
WEST	PB.1	GO	Make high to - LED On Make low to – LED Off
	PB.2	LISTEN	
	PB.3	STOP	
	13-16	NC	
PWR	17,19	Vcc	Supply form MCU/MPU/FPGA Kits
	18,20	Gnd	

ASSEMBLY PROGRAM TO INTERFACE TRAFFIC LIGHT WITH 8086

MEMORY ADDRESS	OPCODE	MNEMONICS
1100	BB 00 11	START: MOV BX, 1200H
1103	B9 08 00	MOV CX, 0008H
1106	8A 07	MOV AL,[BX]
1108	BA 36 FF	MOV DX, CONTROL PORT
110B	EE	OUT DX, AL
110C	43	INC BX
110D	8A 07	NEXT:MOV AL,[BX]
110F	BA 30 FF	MOV DX, PORT A
1112	EE	OUT DX,AL
1113	43	INC BX
1114	8a 07	MOV AL,[BX]
1116	BA 32 FF	MOV DX,PORT B
1119	EE	OUT DX,AL
111A	E8 07 00	CALL DELAY
111D	E2 F1	INC BX
111E		LOOP NEXT
1120	EB E4	JMP START
1122	51	DELAY:PUSH CX
1124	B9 00 05	MOV CX,0005H
1126	BA FF FF	REPEAT:MOV DX,0FFFFH
1129	4A	LOOP2: DEC DX
112A	75 FD	JNZ LOOP2
112C	E2 F8	LOOP REPEAT
112E	59	POP CX
112F	C3	RET

LOOKUP TABLE

1200	80H
1201	21H,09H,10H,00H (SOUTH WAY)
1205	0CH,09H,80H,00H (EAST WAY)
1209	64H,08H,00H,04H (NOURTH WAY)
120D	24H,03H,02H,00H (WEST WAY)
1211	END

13. Discuss the following in detail

(i). Interfacing LED with 8086

(ii) Interfacing LED with 8086

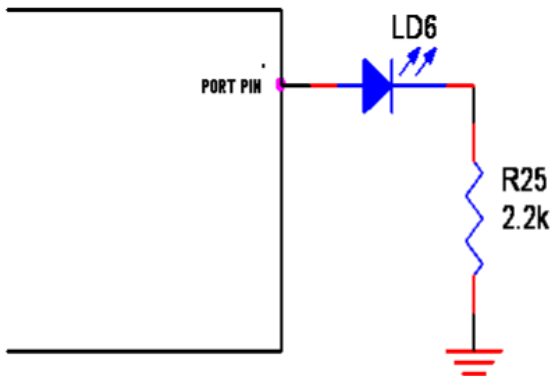
(iii) Keyboard interface

LED (LIGHT EMITTING DIODES)

Light Emitting Diodes (LED) is the most commonly used components, usually for displaying pins digital states. Typical uses of LEDs include alarm devices, timers and confirmation of user input such as a mouse click or keystroke.

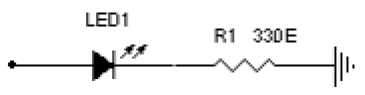
INTERFACING LED

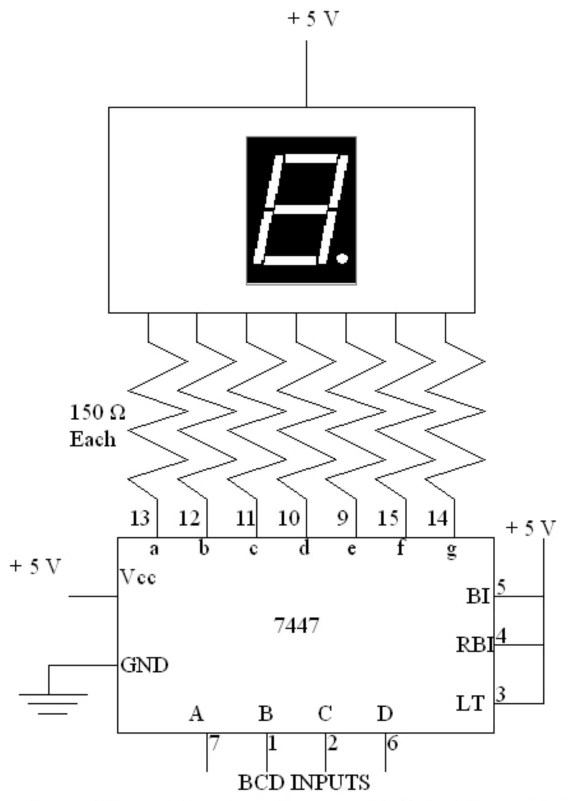
Fig. 1 shows how to interface the LED to microprocessor. As you can see the Anode is connected through a resistor to GND & the Cathode is connected to the Microprocessor pin. So when the Port Pin is HIGH the LED is OFF & when the Port Pin is LOW the LED is turned ON.



INTERFACING LED WITH 8086 We now want to flash a LED in 8086 Trainer Board. It works by turning ON a LED & then turning it OFF & then looping back to START. However the operating speed of microprocessor is very high

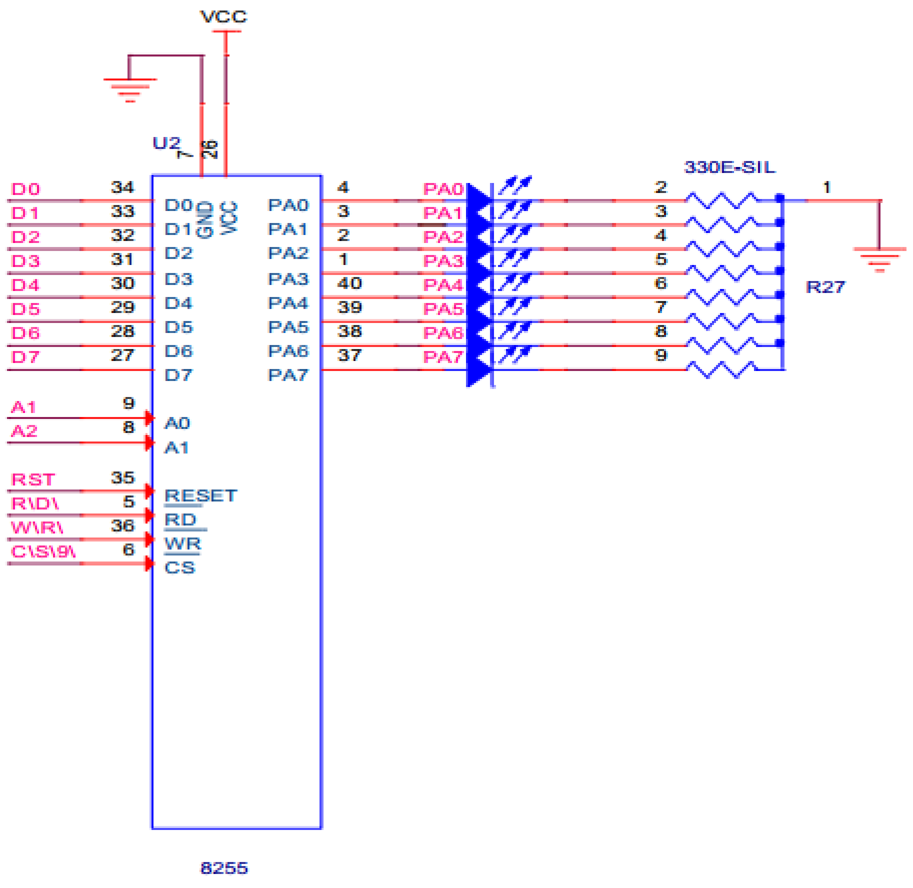
PIN ASSIGNMENT WITH 8086

	Point LEDs	8255 Lines	LED Selection
DIGITAL OUTPUTS	LD1	PA.0	
	LD2	PA.1	
	LD3	PA.2	
	LD4	PA.3	
	LD5	PA.4	
	LD6	PA.5	
	LD7	PA.6	
	LD8	PA.7	



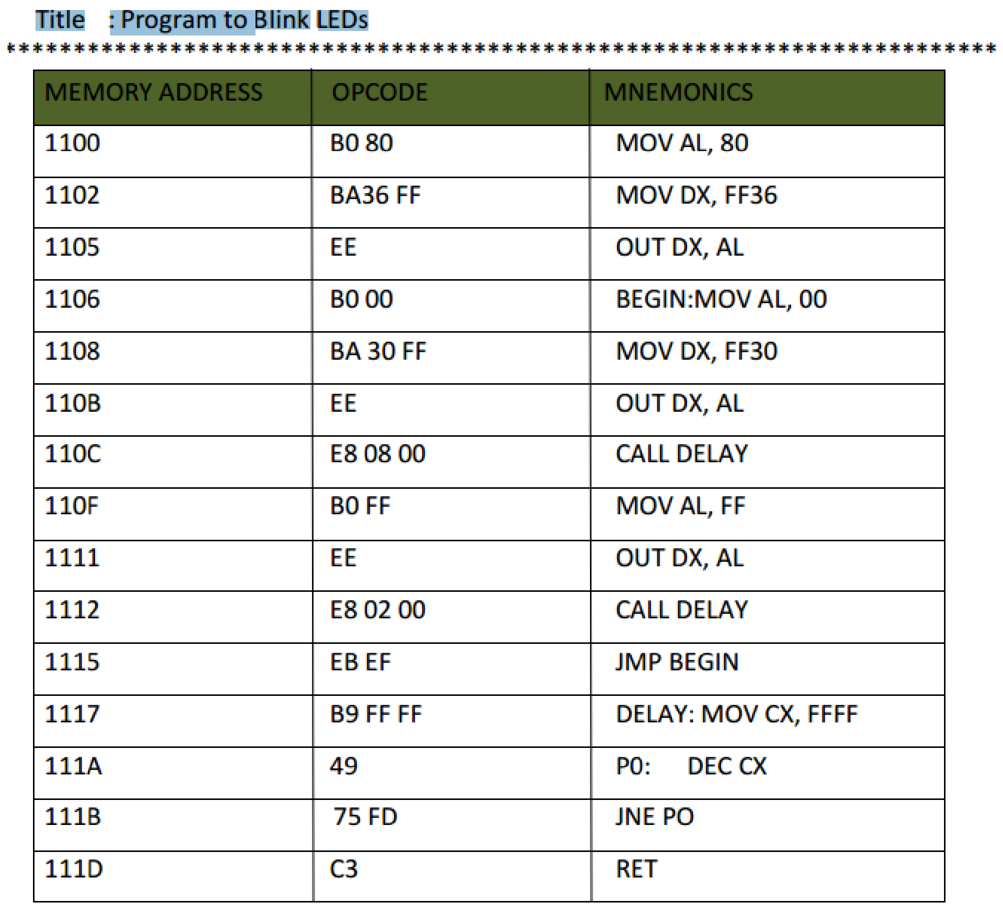
Circuit for driving single 7-segment LED display with 7447

CIRCUIT DIAGRAM TO INTERFACE LED WITH 8255



ASSEMBLY PROGRAM TO ON AND OFF LED USING 8086

Title : Program to Blink LEDs



Title : Program to Blink LEDs

****

MEMORY ADDRESS	OPCODE	MNEMONICS
1100	B0 80	MOV AL, 80
1102	BA36 FF	MOV DX, FF36
1105	EE	OUT DX, AL
1106	B0 00	BEGIN:MOV AL, 00
1108	BA 30 FF	MOV DX, FF30
110B	EE	OUT DX, AL
110C	E8 08 00	CALL DELAY
110F	B0 FF	MOV AL, FF
1111	EE	OUT DX, AL
1112	E8 02 00	CALL DELAY
1115	EB EF	JMP BEGIN
1117	B9 FF FF	DELAY: MOV CX, FFFF
111A	49	P0: DEC CX
111B	75 FD	JNE PO
111D	C3	RET

‘’’’

LCD INTERFACING

Liquid Crystal Display
Introduction:

Liquid Crystal displays are created by sandwiching a thin 10-12 µm layer of a liquid-crystal fluid between two glass plates. A transparent, electrically conductive film or backplane is put on the rear glass sheet. Transparent sections of conductive film in the shape of the desired characters are coated on the front glass plate.

When a voltage is applied between a segment and the backplane, an electric field is created in the region under the segment. This electric field changes the transmission of light through the region under the segment film.

There are two commonly available types of LCD

- **Dynamic scattering and field effect.**
- **Dynamic scattering types of LCD**: It scrambles the molecules where the field is present. This produces an etched-glass-looking light character on a dark background.

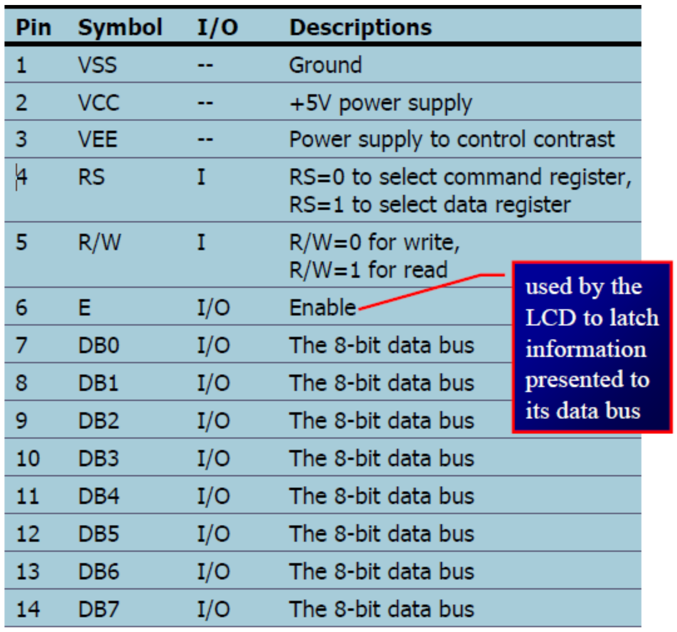
Field-effect types use polarization to absorb light where the electric field is present. This produces dark characters on a silver- gray background.

Most LCD’s require a voltage of 2 or 3 V between the backplane and a segment to turn on the segment. We cannot just connect the backplane to ground and drive the segment with the outputs of a TTL decoder. The reason for this is a steady dc voltage of more than about 50mV is applied between a segment and the To

prevent a dc buildup on the segments, the segment-drive signals for LCD must be square waves with a frequency of 30 to 150 Hz.
Even if you pulse the TTL decoder, it still will not work because the output low voltage of TTL devices is greater than 50mV. CMOS gates are often used to drive LCDs.

Advantages of LCD

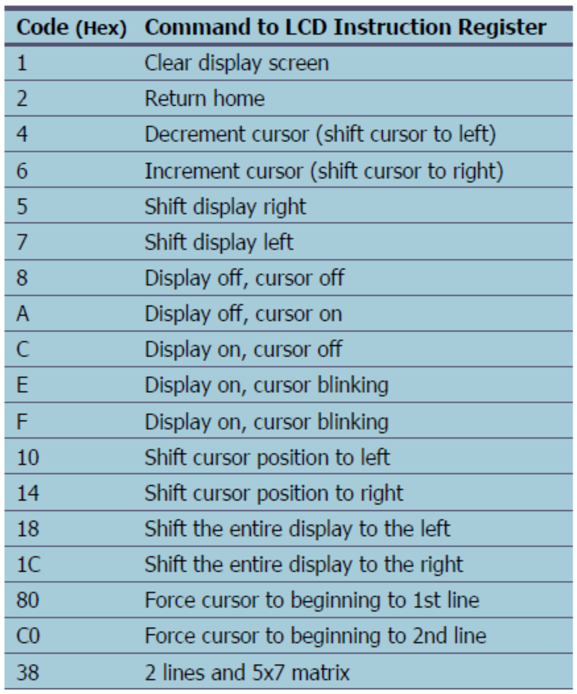
- LCD is finding widespread use replacing LEDs
- The declining prices of LCD
- The ability to display numbers, characters, and graphics
- Incorporation of a refreshing controller into the LCD, thereby relieving the CPU of the task of refreshing the LCD
- Ease of programming for characters and Graphics



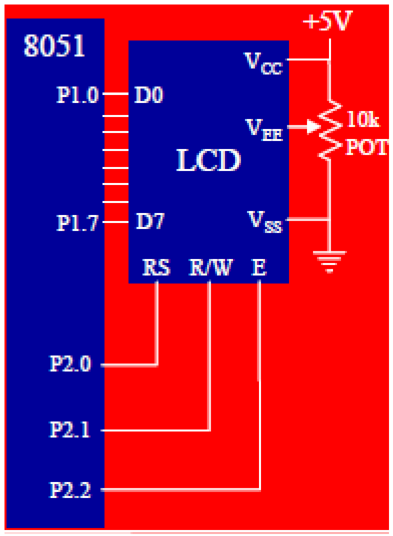
Pin	Symbol	I/O	Descriptions
1	VSS	--	Ground
2	VCC	--	+5V power supply
3	VEE	--	Power supply to control contrast
4	RS	I	RS=0 to select command register, RS=1 to select data register
5	R/W	I	R/W=0 for write, R/W=1 for read
6	E	I/O	Enable
7	DB0	I/O	The 8-bit data bus
8	DB1	I/O	The 8-bit data bus
9	DB2	I/O	The 8-bit data bus
10	DB3	I/O	The 8-bit data bus
11	DB4	I/O	The 8-bit data bus
12	DB5	I/O	The 8-bit data bus
13	DB6	I/O	The 8-bit data bus
14	DB7	I/O	The 8-bit data bus

used by the LCD to latch information presented to its data bus

LCD Command Codes



Code (Hex)	Command to LCD Instruction Register
1	Clear display screen
2	Return home
4	Decrement cursor (shift cursor to left)
6	Increment cursor (shift cursor to right)
5	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on
C	Display on, cursor off
E	Display on, cursor blinking
F	Display on, cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift the entire display to the left
1C	Shift the entire display to the right
80	Force cursor to beginning to 1st line
C0	Force cursor to beginning to 2nd line
38	2 lines and 5x7 matrix



To send any of the commands to the LCD, make pin RS=0. For data, make RS=1. Then send a high-to-low pulse to the E pin to enable the internal latch of the LCD. This is shown in the code below.

```
 ;calls a time delay before sending next data/command
;P1.0-P1.7 are connected to LCD data pins D0-D7
;P2.0 is connected to RS pin of LCD
;P2.1 is connected to R/W pin of LCD
;P2.2 is connected to E pin of LCD
ORG 0H
MOV A,#38H ;INIT. LCD 2 LINES, 5X7 MATRIX
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time
MOV A,#0EH ;display on, cursor on
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time
MOV A,#01 ;clear LCD
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time
MOV A,#06H ;shift cursor right
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time
MOV A,#84H ;cursor at line 1, pos. 4
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time
MOV A,#’N’ ;display letter N
ACALL DATAWRT ;call display subroutine
ACALL DELAY ;give LCD some time
MOV A,#’O’ ;display letter O
ACALL DATAWRT ;call display subroutine
AGAIN: SJMP AGAIN ;stay here
COMNWRT: ;send command to LCD
MOV P1,A ;copy reg A to port 1
CLR P2.0 ;RS=0 for command
CLR P2.1 ;R/W=0 for write
SETB P2.2 ;E=1 for high pulse
ACALL DELAY ;give LCD some time
CLR P2.2 ;E=0 for H-to-L pulse
RET
DATAWRT: ;write data to LCD
MOV P1,A ;copy reg A to port 1
SETB P2.0 ;RS=1 for data
CLR P2.1 ;R/W=0 for write
SETB P2.2 ;E=1 for high pulse
ACALL DELAY ;give LCD some time
CLR P2.2 ;E=0 for H-to-L pulse
RET
DELAY: MOV R3,#50 ;50 or higher for fast CPUs
HERE2: MOV R4,#255 ;R4 = 255
HERE: DJNZ R4,HERE ;stay until R4 becomes 0
DJNZ R3,HERE2
RET
END
```

```
;Check busy flag before sending data, command to LCD
;p1=data pin
;P2.0 connected to RS pin
;P2.1 connected to R/W pin
;P2.2 connected to E pin

ORG 0H
MOV A,#38H ;init. LCD 2 lines ,5x7 matrix
ACALL COMMAND ;issue command
MOV A,#0EH ;LCD on, cursor on
ACALL COMMAND ;issue command
MOV A,#01H ;clear LCD command
ACALL COMMAND ;issue command
MOV A,#06H ;shift cursor right
ACALL COMMAND ;issue command
MOV A,#86H ;cursor: line 1, pos. 6
ACALL COMMAND ;command subroutine
MOV A,#'N' ;display letter N
ACALL DATA_DISPLAY
MOV A,#'O' ;display letter O
ACALL DATA_DISPLAY
HERE:SJMP HERE ;STAY HERE
```

The Following fig shows how two CMOS gate outputs can be connected to drive an LCD segment and backplane.

- The off segment receives the same drive signal as the backplane. There is never any voltage between them, so no electric field is produced. The waveform for the on segment is 180 out of phase with the backplane signal, so the voltage between this segment and the backplane will always be +V.
- The logic for this signal, a square wave and its complement. To the driving gates, the segment-backplane sandwich appears as a somewhat leaky capacitor.
- The CMOS gates can be easily supply the current required to charge and discharge this small capacitance.
- Older inexpensive LCD displays turn on and off too slowly to be multiplexed the way we do LED display.
- At 0c some LCD may require as much as 0.5s to turn on or off. To interface to those types we use a non multiplexed driver device.
- More expensive LCD can turn on and off faster, so they are often multiplexed using a variety of techniques.
- In the following section we show you how to interface a non multiplexed LCD to a microprocessor such as SDK-86.
- Intersil ICM7211M can be connected to drive a 4-digit, non multiplexed, 7- segment LCD display.
- The 7211M input can be connected to port pins or directly to microcomputer bus.We have connected the CS inputs to the Y2 output of the 74LS138 port decoder.
- According to the truth table the device will then be addressable as ports with a base address of FF10H. SDK-86 system address lines A2 is connected to the digit-select input (DS2) and system address lines A1 is connected to the DS1 input. This gives digit 4 a system address of FF10H.

A8-A15	A5-A7	A4	A3	A2	A1	A0	M/IO	Y Output Selected	System Base Address	Device
1	0	0	0	X	X	0	0	00	F F 0 0	8259A #1
1	0	0	1	X	X	0	0	1	F F 0 8	8259A #2
1	0	1	0	X	X	0	0	2	F F 1 0	
1	0	1	1	X	X	0	0	3	F F 1 8	
1	0	0	0	X	X	1	0	4	F F 0 1	8254
1	0	0	1	X	X	1	0	5	F F 0 9	
1	0	1	0	X	X	1	0	6	F F 1 1	
1	0	1	1	X	X	1	0	7	F F 1 9	
ALL OTHER STATES								NONE		

Fig : Truth table for 74LS138 address decoder

Digit 3 will be addressed at FF12H, digit 2 at FF14H and digit 1 at FF16H.

- The data inputs are connected to the lower four lines of the SDK-86 data bus.The oscillator input is left open. To display a character on one of the digits, you simply keep the 4-bit hex code for that digit in the lower 4 bits of the AL register and output it to the system address for that digit.

- The ICM7211M converts the 4-bit hex code to the required 7-segment code.
- The rising edge of the CS input signal causes the 7-segment code to be latched in the output latches for the address digit.
- An internal oscillator automatically generates the segment and backplane drive waveforms as in fig . For interfacing with the LCD displays which can be multiplexed the Intersil ICM7233 can be use.

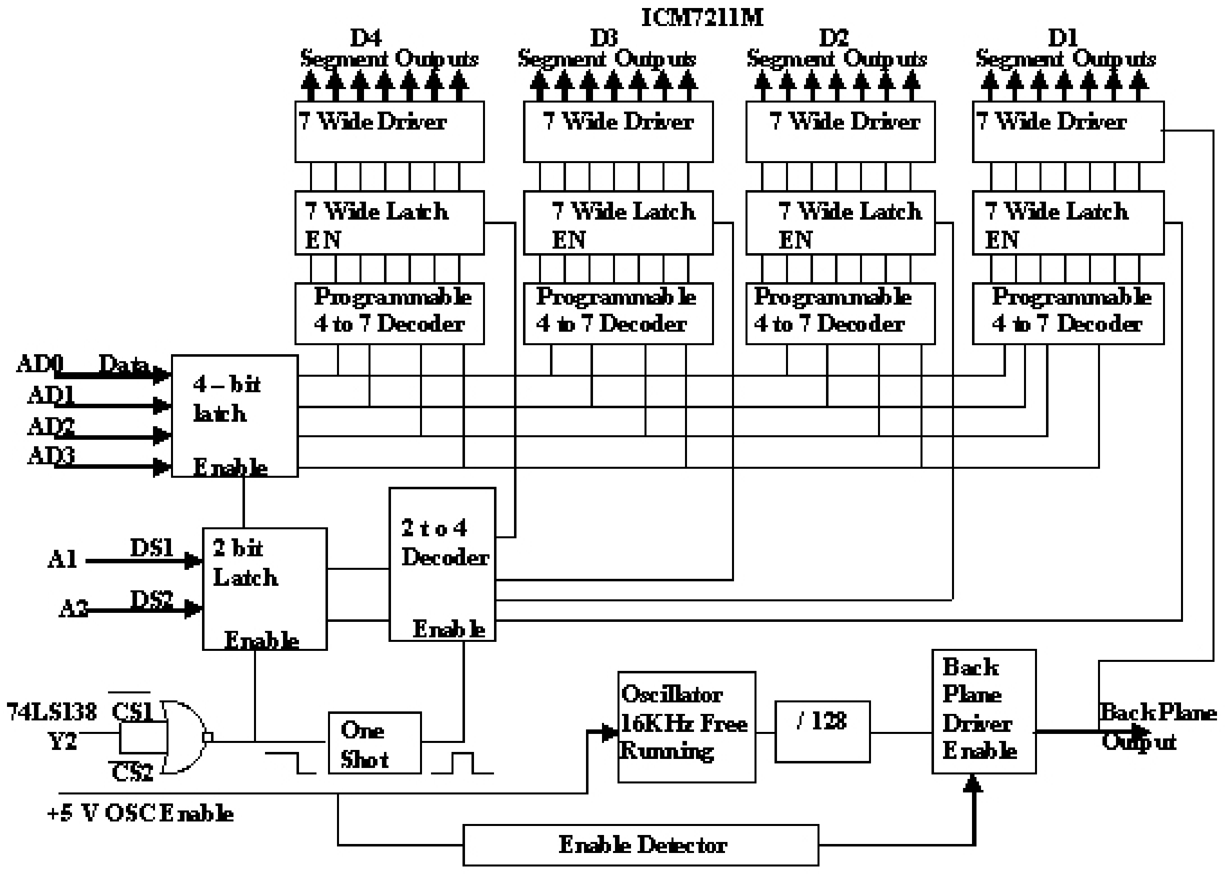


Fig : Circuit for interfacing four LCD digits to an SDK-86 bus using ICM7211M

KEYBOARD INTERFACING

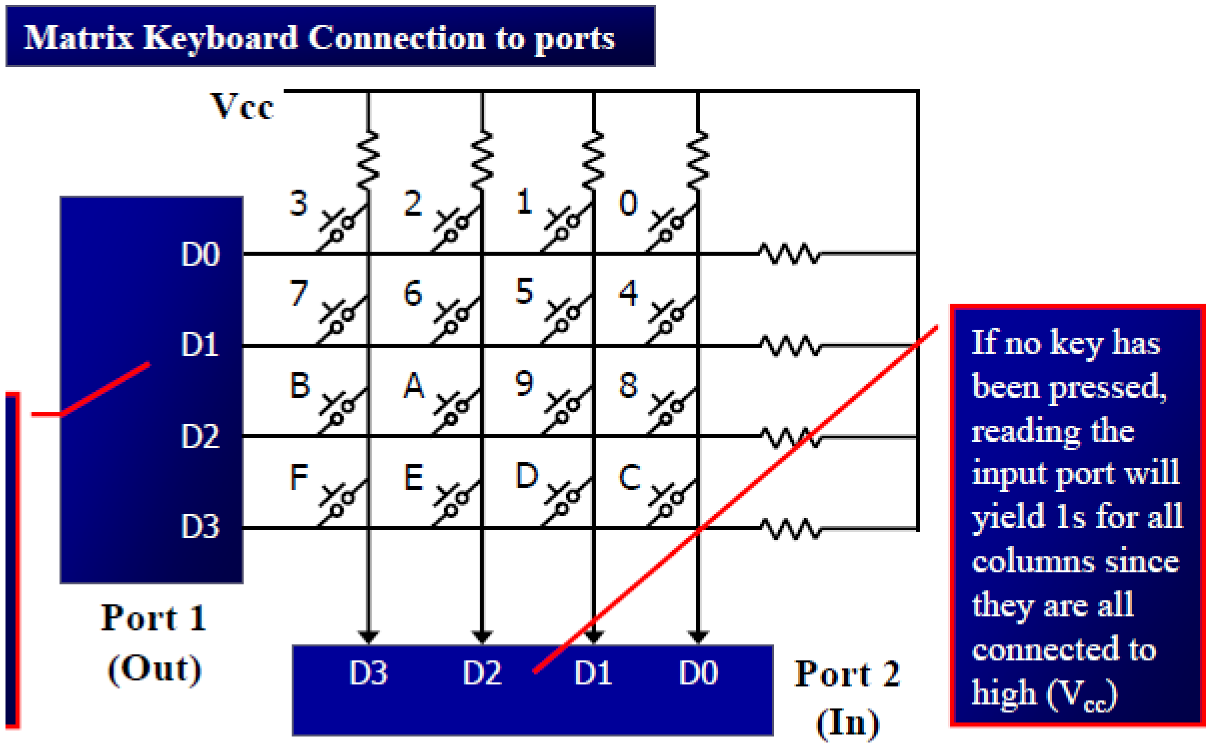
Keyboards are organized in a matrix of rows and columns

 The CPU accesses both rows and columns through ports. Therefore, with two 8-bit ports, an 8 x 8 matrix of keys can be connected to a microprocessor

 When a key is pressed, a row and a column make a contact, Otherwise, there is no connection between rows and columns

 In IBM PC keyboards, a single microcontroller takes care of hardware and software interfacing .A 4x4 matrix connected to two ports

 The rows are connected to an output port and the columns are connected to an input port KEYBOARD


Matrix Keyboard Connection to ports
Vcc
3
2
1
0
7
6
5
4
B
A
9
8
F
E
D
C
D0
D1
D2
D3
Port 1
(Out)
D3
D2
D1
D0
Port 2
(In)
If no key has been pressed, reading the input port will yield 1s for all columns since they are all connected to high (V_{cc})

It is the function of the microcontroller to scan the keyboard continuously to detect and identify the key pressed To detect a pressed key, the microcontroller grounds all rows by providing 0 to the output latch, then it reads the columns

- If the data read from columns is D3 – D0 = 1111, no key has been pressed and the process continues till key press is detected
- If one of the column bits has a zero, this means that a key press has occurred. For example, if D3 – D0 = 1101, this means that a key in the D1 column has been pressed
- After detecting a key press, microcontroller will go through the process of identifying the key Starting with the top row, the microcontroller grounds it by providing a low to row D0 only
- It reads the columns, if the data read is all 1s, no key in that row is activated and the process is moved to the next row
- It grounds the next row, reads the columns, and checks for any zero
- This process continues until the row is identified
- After identification of the row in which the key has been pressed
- Find out which column the pressed key belongs to
- Corresponding key is displayed.

‘’’

14.Explain in detail about alarm controller.

DS12887 RTC INTERFACING

This interfacing and programming of the DS12C887 real time clock (RTC) chip.

DS12887 RTC INTERFACING

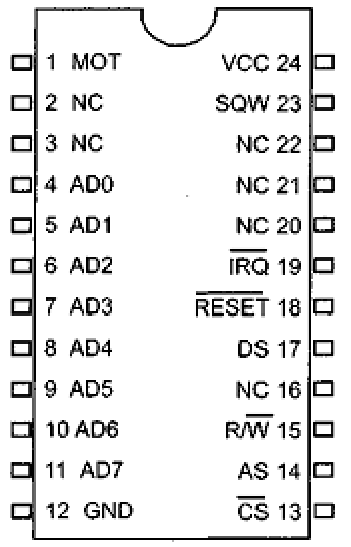


Figure 16-1. DS12887 RTC Chip

- ✓ The RTC provides accurate time and date for many applications.
- ✓ The RTC chip in the IBM PC provides time components of hour, minute and second, in addition to date / calendar components of year, month and day.
- ✓ It uses an internal battery, which keeps the time and date even when the power is off (over 10 years).
- ✓ One of the most widely used RTC chip is the DS 12887.
- ✓ It keeps track of "seconds, minutes, hours, days, day of week, date, month, and year with leap-year compensation valid up to year 2100″.
- ✓ The above information is provided in both binary (hex) and BCD formats. The DS 12887 supports both 12-hour and 24-hour clock modes with AM and PM in the 12-hour mode.
- ✓ It also supports the Daylight Savings Time option. The DS 12887 uses CMOS technology to keep the power consumption low and it has the designation DS12C887, where C is for CMOS. The DS12887 has a total of 128 bytes of nonvolatile RAM.
- ✓ It uses 14 bytes of RAM for clock/calendar and control registers, and the other 114 bytes of RAM are for general-purpose data storage.
- ✓ Next we describe the pins of the DS 12887. See Figure 16-1.

Vcc

Pin 24 provides external supply voltage to the chip. The external voltage source is +5V. When Voltage falls below the 3V level, the external source is switched off and the internal lithium battery provides power to the RTC.

GND

Pin 12 is the ground.

ADO-AD7

The multiplexed address/data pins provide both addresses and data to the chip. Addresses are latched into the DS 12887 on the falling edge of the AS (ALE) signal.

A simple way of connecting the DS 12887 to the 8051 is shown in Figure 16-2.

AS (ALE)

AS (address strobe) is an input pin. On the falling edge it will cause the addresses to be latched into the DS 12887. The AS pin is used for demultiplexing the address and data and is connected to the ALE pin of the 8051 chip.

MOT

This is an input pin that allows the choice between the Motorola and Intel microcontroller bus timings. The MOT pin is connected to GND for the Intel timing. That means when we connect DS 12887 to the 8051, MOT = GND.

DS

Data strobe or read is an input. When MOT = GND for Intel timing, the DS pin is called the RD (read) signal and is connected to the RD pin of the 8051.

R/W

Read/Write is an input pin. When MOT = GND for the Intel timing, the R/W pin is called the WR (write) signal and is connected to the WR pin of the 8051.

CS

Chip select is an input pin and an active low signal. When the DS 12887 is in write-protected state, all inputs are ignored.

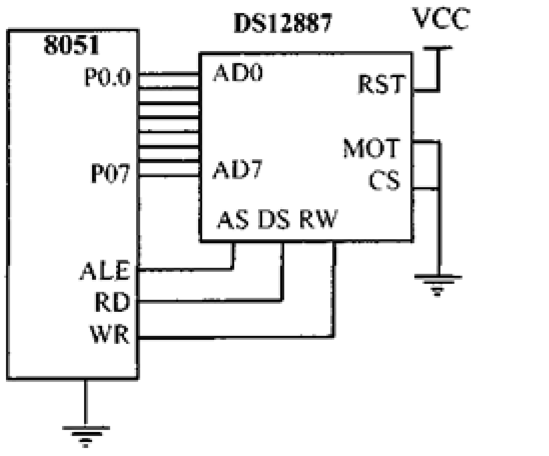


Figure 16-2. DS12887 Connection to 8051

IRQ

Interrupt request is an output pin and active low signal. To use IRQ, the interrupt-enable bits in register B must be set high. The interrupt feature of the DS12287 is discussed in Section 16.3.

SQW

Square wave is an output pin. We can program the DS 12887 to provide up to 15 different square waves. The frequency of the square wave is set by programming register A.

RESET

Pin 18 is the reset pin. It is an input and is active low (normally high). In most applications the reset pin is connected to the V_{cc} pin.

Address map of the DS12887

The DS12887 has a total of 128 bytes of RAM space with addresses 00 -7FH. The first ten locations, 00 – 09, are set aside for RTC values of time, calendar, and alarm data.

The next four bytes are used for the control and status registers. They are registers A, B, C, and D and are located at addresses 10-13 (OA – OD in hex).

The next 114 bytes from addresses OEH to 7FH are available for data storage. The entire 128 bytes of RAM are accessible directly for read or write except the following:

1. Registers C and D are read-only.
2. D7 bit of register A is read-only.
3. The high-order bit of the seconds byte is read-only.

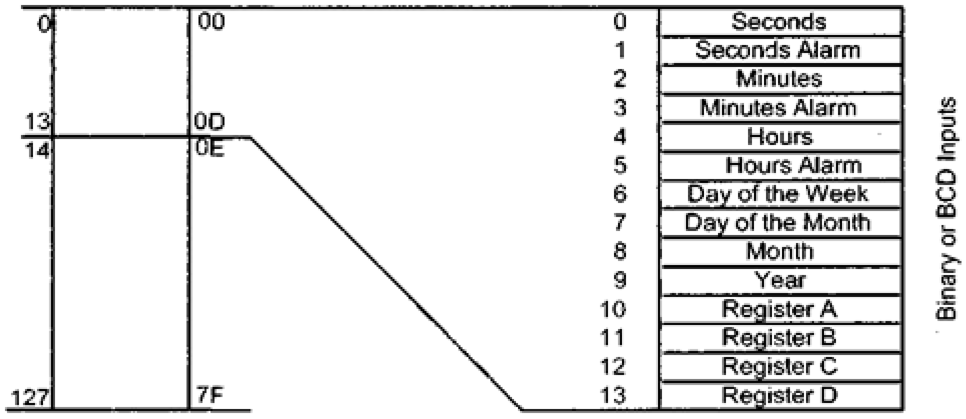


Figure 16-3. DS12887 Address Map

Figure 16.3 shows the address map of the DS 12887.

Time, calendar, and alarm address locations and modes

The byte addresses 0-9 are set aside for the time, calendar, and alarm data. Table 16-1 shows their address locations and modes. Notice the data is available in both binary (hex) and BCD formats.

Turning on the oscillator for the first time

The DS12887 is shipped with the internal oscillator turned off in order to save the lithium battery. We need to turn on the oscillator before we use the time keeping features of the DS 12887. To do that, bits D6 – D4 of register A must be set to value 010. See Figure 16-4 for details of register A.

The following code shows how to access the DS12887′s register A and is written for the Figure 16-2 connection. In Figure 16-2, the DS 12887 is using the external memory space of the 8051 and is mapped to address space of 00 – 7FH since CS = 0. See Chapter 14 for a discussion of external memory in the 8051. For the programs in this chapter, we use instruction “MOVX A, @RO” since the address is only 8-bit. In the case of a 16-bit address, we must use "MOVX A, @DPTR" as was shown in Chapter 14. Examine the following code to see how to access the DS12887 of Figure 16-2.

Table 16-1: DS12887 Address Location for Time, Calendar, and Alarm

Address Location	Function	Decimal Range	Data Mode Range Binary (hex)	BCD
0	Seconds	0 - 59	00 - 3B	00 - 59
1	Seconds Alarm	0 - 59	00 - 3B	00 - 59
2	Minutes	0 - 59	00 - 3B	00 - 59
3	Minutes Alarm	0 - 59	00 - 3B	00 - 59
4	Hours, 12-Hour Mode	1 - 12	01 - 0C AM	01 - 12 AM
	Hours, 12-Hour Mode	1 - 12	81 - 8C PM	81 - 92 PM
	Hours, 24-Hour Mode	0 - 23	0 - 17	0 - 23
5	Hours Alarm, 12-Hour	1 - 12	01 - 0C AM	01 - 12 AM
	Hours Alarm, 12-Hour	1 - 12	81 - 8C PM	81 - 92 PM
	Hours Alarm, 24-Hour	0 - 23	0 - 17	0 - 23
6	Day of the Week, Sun = 1	1 - 7	01 - 07	01 - 07
7	Day of the Month	1 - 31	01 - 1F	01 - 31
8	Month	1 - 12	01 - 0C	01 - 12
9	Year	0 - 99	00 - 63	00 - 99

```
ACALL DELAY_200ms ;RTC NEEDS 200ms AFTER POWER-UP
MOV  R0,#10         ;R0=0AH,Reg A address
MOV  A,#20H         ;010 in D6-D4 to turn on osc.
MOVX @R0,A          ;send it to Reg A of DS12887
```

UIP	DV2	DV1	DV0	RS3	RS2	RS1	RS0

UIP Update in progress. This is a read-only bit.

DV2	**DV1**	**DV0**	
0	1	0	will turn the oscillator on

RS3 RS2 RS1 RS0
Provides 14 different frequencies at the SQW pin. See Section 16.3 and the DS12887 data sheet.

Figure 16-4. Register A Bits for Turning on the DS12887's Oscillator

Setting the time

When we initialize the time or date, we need to set D7 of register B to 1. This will prevent any update at the middle of the initialization. After setting the time and date, we need to make D7 = 0 to make sure that the clock and time are updated. The update occurs once per second. The following code initializes the clock at 16:58:55 using the BCD mode and 24-hour clock mode with daylight savings time. See also Figure 16-5 for details of register B.

```
;------WAIT 200msec FOR RTC TO BE READY AFTER POWER-UP
     ACALL DELAY_200ms
;-----------TURNING ON THE RTC
     MOV   R0,#10     ;R0=0AH,Reg A address
     MOV   A,#20H     ;010 in D6-D4 to turn on osc.
     MOVX  @R0,A      ;send it to Reg A of DS12887
;-------------Setting the Time mode
     MOV   R0,#11     ;Reg B address
     MOV   A,#83H ;BCD,24hrs,Daylight saving,D7=1 No update
     MOVX  @R0,A      ;send it to Reg B
;---------Setting the Time
     MOV   R0,#0      ;point to seconds address
     MOV    A,#55H    ;seconds= 55H (BCD numbers need H)
     MOVX  @R0,A      ;set seconds
     MOV   R0,#02     ;point to minutes address
     MOV   A,#58H     ;minutes= 58
     MOVX  @R0,A      ;set minutes
     MOV   R0,#04     ;point to hours address
     MOV   A,#16H     ;hours=16
     MOVX  @R0,A      ;set hours
     MOV   R0,#11     ;Reg B address
     MOV   A,#03      ;D7=0 of reg B to allow update
     MOVX  @R0,A      ;send it to reg B
```

SET	PIE	AIE	UIE	SQWE	DM	24/12	DSE

SET	SET = 0: Clock is counting once per second and time and dates are updated SET = 1: Update is inhibited (during the initialization we must make SET = 1)
PIE	Periodic Interrupt Enable. See Section 16.3.
AIE	Alarm Interrupt Enable. The AIE = 1 will allow the IRQ to be asserted, when all three bytes of time (yy:mm:dd) are the same as the alarm bytes. See Section 16.3.
UIE	See the DS12887 data sheet
SQWE	Square wave enable: See Section 16.3
DM	Data mode. DM = 0: BCD data format and DM = 1: Binary (hex) data format
24/12	1 for 24-hour mode and 0 for 12-hour mode
DSE	Daylight Saving Enable. If 1, enables the daylight saving. (The first Sunday in April and the last Sunday of October)

Figure 16-5. Some Major Bits of Register B

Setting the date

The following shows how to set the date to October 19th, 2004. Notice that when we initialize time or date, we need to set D7 of register B to 1.

```
;------------TURNING ON THE RTC
          MOV   R0,#10      ;R0=0AH,Reg A address
          MOV   A,#20H      ;010 in D6-D4 to turn on osc
          MOVX  @R0,A       ;send it to Reg A of DS12887
;--------------Setting the Time mode
          MOV   R0,#11      ;Reg B address
          MOV   A,#83H      ;BCD,24 hrs, daylight saving
          MOVX  @R0,A       ;send it to Reg B
;----------Setting the DATE
          MOV   R0,#07      ;load pointer for DAY OF MONTH
          MOV   A,#19H      ; DAY=19H (BCD numbers need H)
          MOVX  @R0,A       ;set DAY OF MONTH
          ACALL DELAY       ;
          MOV   R0,#08      ;point to MONTH
          MOV   A,#10H      ;10=OCTOBER.
          MOVX  @R0,A       ;set MONTH
          ACALL DELAY       ;
          MOV   R0,#09      ;point to YEAR address
          MOV   A,#04       ;YEAR=04 FOR 2004
          MOVX  @R0,A       ;set YEAR to 2004
          ACALL DELAY
          MOV   R0,#11      ;Reg B address
          MOV   A,#03       ;D7=0 of reg B to allow update
          MOVX  @R0,A       ;send it to reg B
```

RTCs setting, reading, displaying time and date

The following is a complete Assembly code for setting, reading, and displaying the time and date. The times and dates are sent to the screen via the serial port after they are converted from BCD to ASCII.

```
;----RTCTIME.ASM: SETTING TIME,READING AND DISPLAYING IT
        ORG 0
        ACALL DELAY_200ms ;RTC needs 200ms upon power-up
    SERIAL PORT SET-UP
        MOV   TMOD,#20H
        MOV   SCON,#50H
        MOV   TH1,#-3     ;9600
        SETB  TR1
;------------TURNING ON THE RTC
        MOV   R0,#10      ;R0=0AH,Reg A address
        MOV   A,#20H      ;010 in D6-D4 to turn on osc.
        MOVX  @R0,A       ;send it to Reg A of DS12887
;--------------Setting the Time mode
        MOV   R0,#11      ;Reg B address
        MOV   A,#83H      ;BCD, 24 hrs, daylight saving
        MOVX  @R0,A       ;send it to Reg B
;----------Setting the DATE
        MOV   R0,#07      ;load pointer for DAY OF MONTH
        MOV   A,#24H      ; DAY=24H (BCD numbers need H)
        MOVX  @R0,A       ;set DAY OF MONTH
        ACALL DELAY       ;
        MOV   R0,#08      ;point to MONTH
        MOV   A,#10H      ; 10=OCTOBER.
        MOVX  @R0,A       ;set MONTH
        ACALL DELAY       ;
        MOV   R0,#09      ;point to YEAR address
        MOV   A,#04       ;YEAR=04 FOR 2004
        MOVX  @R0,A       ;set YEAR to 2004
        ACALL DELAY
        MOV   R0,#11      ;Reg B address
        MOV   A,#03       ;D7=0 of reg B to allow update
        MOVX  @R0,A       ;send it to reg B
·--------READ Time(HH:MM:SS), CONVERT IT AND DISPLAY IT
OV1:    MOV   A,#20H      ;ASCII for SPACE
        ACALL SERIAL
        MOV   R0,#4       ;point to HR loc
        MOVX  A,@R0       ;read hours
        ACALL DISPLAY
        MOV   A,#20H      ;send out SPACE
        ACALL SERIAL
        MOV   R0,#2       ;point to minute loc
        MOVX  A,@R0       ;read minute
        ACALL DISPLAY
        MOV   A,#20H      ;send out SPACE
        ACALL SERIAL
```

```
          MOV   R0,#0       ;point to seconds loc
          MOVX  A,@R0       ;read seconds
          ACALL DISPLAY
          MOV   A,#0AH      ;send out CR
          ACALL SERIAL
          MOV   A,#0DH      ;send LF
          ACALL SERIAL
          SJMP  OV1         ;read and display forever
;---------SMALL DELAY
DELAY:
          MOV   R7,#250
D1:       DJNZ  R7,D1
          RET
;-------------CONVERT BCD TO ASCII AND SEND IT TO SCREEN
DISPLAY:
          MOV   B,A
          SWAP  A
          ANL   A,#0FH
          ORL   A,#30H
          ACALL SERIAL
          MOV   A,B
          ANL   A,#0FH
          ORL   A,#30H
          ACALL SERIAL
          RET
;-----------
SERIAL:
          MOV   SBUF,A
S1:       JNB   TI,S1
          CLR   TI
          RET
;------------
          END
```

The following shows how to read and display the date. You can replace the time display portion of the above program with the program below.

```
;--------READ DATE(YYYY:MM:MM), CONVERT IT AND DSIPLAY IT
OV2:      MOV   A,#20H      ;ASCII SPACE
          ACALL SERIAL
          MOV   A,#'2'      ;SEND OUT 2 (for 20)
          ACALL SERIAL
          MOV   A,#'0'      ;SEND OUT 0 (for 20)
          ACALL SERIAL
          MOV   R0,#09      ;point to year loc
          MOVX  A,@R0       ;read year
          ACALL DISPLAY
          MOV   A,#':'      ;SEND OUT : for yyyy:mm
          ACALL SERIAL
          MOV   R0,#08      ;point to month loc
          MOVX  A,@R0       ;read month
```

```
ACALL DISPLAY
ACALL DELAY
MOV   A,#':'      ;SEND OUT : for mm:dd
ACALL SERIAL
MOV   R0,#07      ;point to DAY loc
MOVX  A,@R0       ;read day
ACALL DISPLAY
ACALL DELAY
MOV   A,#' '      ;send out SPACE
ACALL SERIAL
ACALL DELAY
MOV   A,#' '      ;send out SPACE
ACALL SERIAL
ACALL DELAY
MOV   A,#0AH      ;send out LF
ACALL SERIAL
MOV   A,#0DH      ;send CR
ACALL SERIAL
ACALL DELAY
LJMP OV2          ;display date forever
```

UNIT – IV MICROCONTROLLER

Architecture of 8051 – Special Function Registers(SFRs) - I/O Pins Ports and Circuits - Instruction set - Addressing modes - Assembly language programming.

4.1: Introduction:

Microcontroller:

- ✓ A Microcontroller is a single chip computer.
- ✓ A CPU with all the peripherals like RAM, ROM, I/O Ports, Timers, and ADCs etc. on the same chip.

  For Ex: Motorola 6811, Intel 8051, Zilog Z8 and PIC 16X etc…

Microprocessor:

- ✓ A CPU built into a single VLSI chip is called a microprocessor.
- ✓ It is a general-purpose device and additional external circuitry is added to make it a microcomputer.
- ✓ The microprocessor contains arithmetic logic unit (ALU), Control unit, Instruction register, Program counter (PC), clock circuit (internal or external), reset circuit (internal or external) and registers.
- ✓ But the microprocessor has no on chip I/O Ports, Timers, Memory etc.
- ✓ For example, Intel 8085 is an 8-bit microprocessor and Intel 8086/8088 a 16-bit microprocessor.
- ✓ The block diagram of the Microprocessor is shown in Fig.1

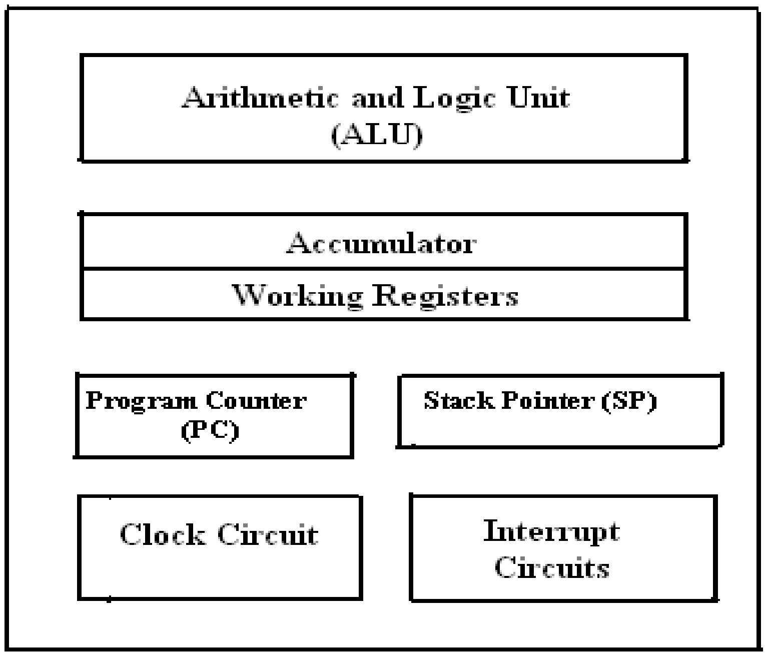


Fig.1: Block diagram of a Microprocessor.

MICROCONTROLLER :

- ✓ A microcontroller is an integrated single chip, which consists of CPU, RAM, EPROM/PROM/ROM, I/O ports, timers, interrupt controller.
- ✓ For example, Intel 8051 is 8-bit microcontroller and Intel 8096 is 16-bit microcontroller.
- ✓ The block diagram of Microcontroller is shown in Fig.2.

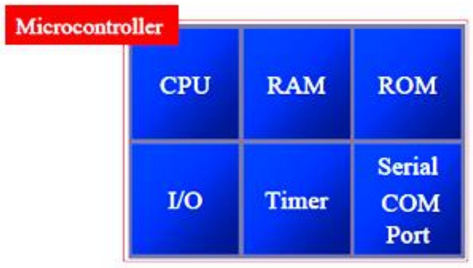


Fig.2.Block Diagram of a Microcontroller

Distinguish between Microprocessor and Microcontroller

S.No	**Microprocessor**	**Microcontroller**
1	A microprocessor is a general purpose device.	A microcontroller is a dedicated chip which is also called as single chip computer.
2	A microprocessor does not contain on chip I/O Ports, Timers, Memories etc.	A microcontroller includes RAM, ROM, serial and parallel interface, timers, interrupt circuitry in a single chip.
3	Microprocessor is used as the CPU in microcomputer system.	Microcontroller is used to perform control-oriented applications.
4	Microprocessor instructions are nibble or byte addressable	Microcontroller instructions are both bit addressable as well as byte addressable.
5	Microprocessor based system design is complex and expensive	Microcontroller based system design is simple and cost effective
6	The Instruction set of microprocessor is complex with large number of instructions.	The instruction sets are simple with less number of instructions.

****

4.2: INTEL 8051 MICRCONTROLLER:

Draw the architectural block diagram of 8051 microcontroller and explain. (NOV 2011, MAY 2010, NOV 2009, NOV2008, May 2008, MAY 2007, MAY 2006, NOV 2016, May 2016)

Features of 8051 Microcontroller:

The 8051 is an 8-bit Controller:

- ✓ The CPU can works on only 8 bits of data at a time
- ✓ The 8051 has
  - 128 bytes of RAM
  - 4K bytes of on-chip ROM
  - Two timers
  - One serial port
  - Four I/O ports, each 8 bits wide
  - 6 interrupt sources

ARCHITECTURE & BLOCK DIAGRAM OF 8051 MICROCONTROLLER:

- ✓ It has hardware architecture with RISC (Reduced Instruction Set Computer) concept.
- ✓ The block diagram of 8051 microcontroller is shown in Fig 3.
- ✓ 8051 has 8-bit ALU.
- ✓ ALU can perform all the 8-bit arithmetic and logical operations in one machine cycle.
- ✓ The ALU is associated with two registers A & B

A and B Registers:

- ✓ The A and B registers are special function registers.
- ✓ A & B registers hold the results of many arithmetic and logical operations of 8051.
- ✓ The A register is also called the **Accumulator.**
- ✓ A register is used as a general register to accumulate the results of a large number of instructions.
- ✓ By default, it is used for all mathematical operations and data transfer operations between CPU and external memory.
- ✓ The B register is mainly used for multiplication and division operations along with A register.
  - ▪ Ex: MUL AB : DIV AB.
- ✓ It has no other function other than as a store data.

R registers:

- ✓ "R" registers are a set of eight registers that are named R0, R1, etc. up to R7.
- ✓ These registers are used as auxiliary registers in many operations.
- ✓ The "R" registers are also used to temporarily store values.

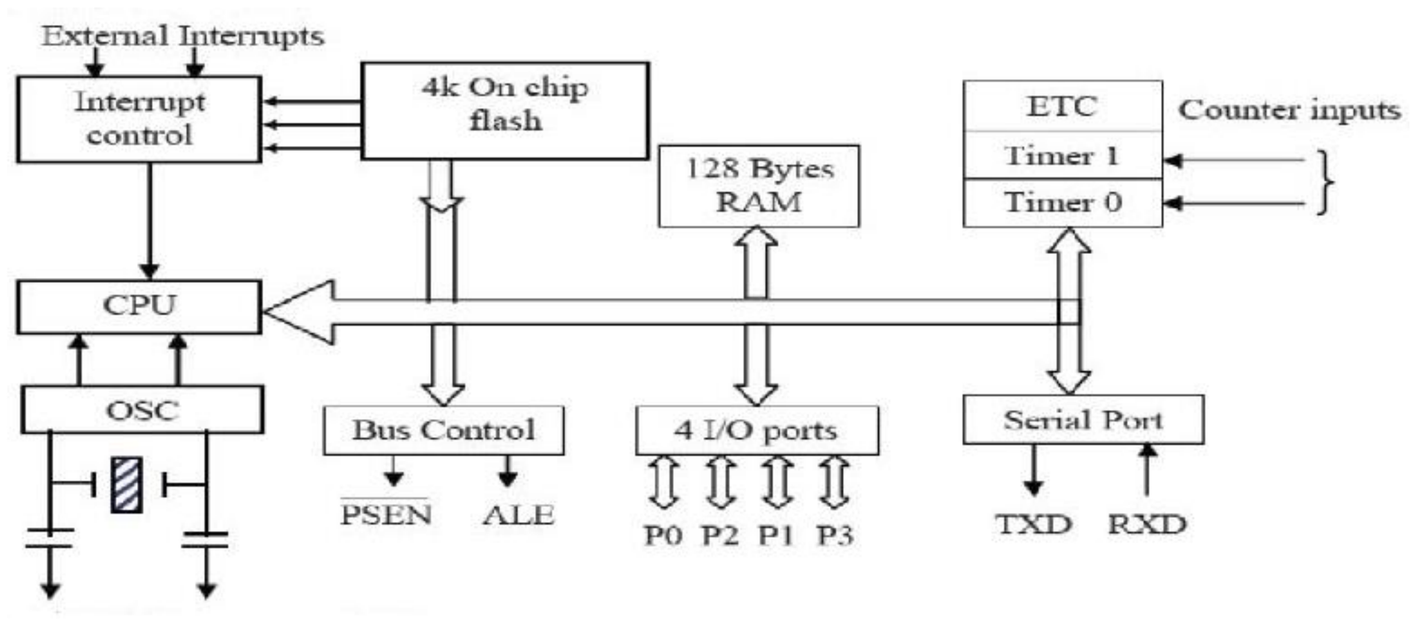


Fig.3. Block Diagram of 8051 Microcontroller

Program Counter (PC) :

- ✓ 8051 has a 16-bit program counter.
- ✓ The program counter holds address of the next instruction to be executed.
- ✓ After execution of one instruction, the program counter is incremented.

Data Pointer Register (DPTR):

- ✓ It is a 16-bit register which is the only user-accessible.
- ✓ DPTR is used to point the data. 8051 will access external memory at the address indicated by DPTR.
- ✓ DPTR can also be used as two 8-registers DPH and DPL.

Stack Pointer Register (SP) :

- ✓ It is an 8-bit register which stores the address of the stack top.
- ✓ When a value is pushed onto the stack, the 8051 first increments the value of SP and then stores the value.
- ✓ Similarly when a value is popped off the stack, the 8051 returns the value from the memory location indicated by SP and then decrements the value of SP.
- ✓ Since the SP is only 8-bit wide.
- ✓ It is incremented or decremented by two.
- ✓ SP is modified directly by the 8051 by six instructions: PUSH, POP, ACALL, LCALL, RET, and RETI.
- ✓ It is also used intrinsically whenever an interrupt is triggered.

Block Diagram

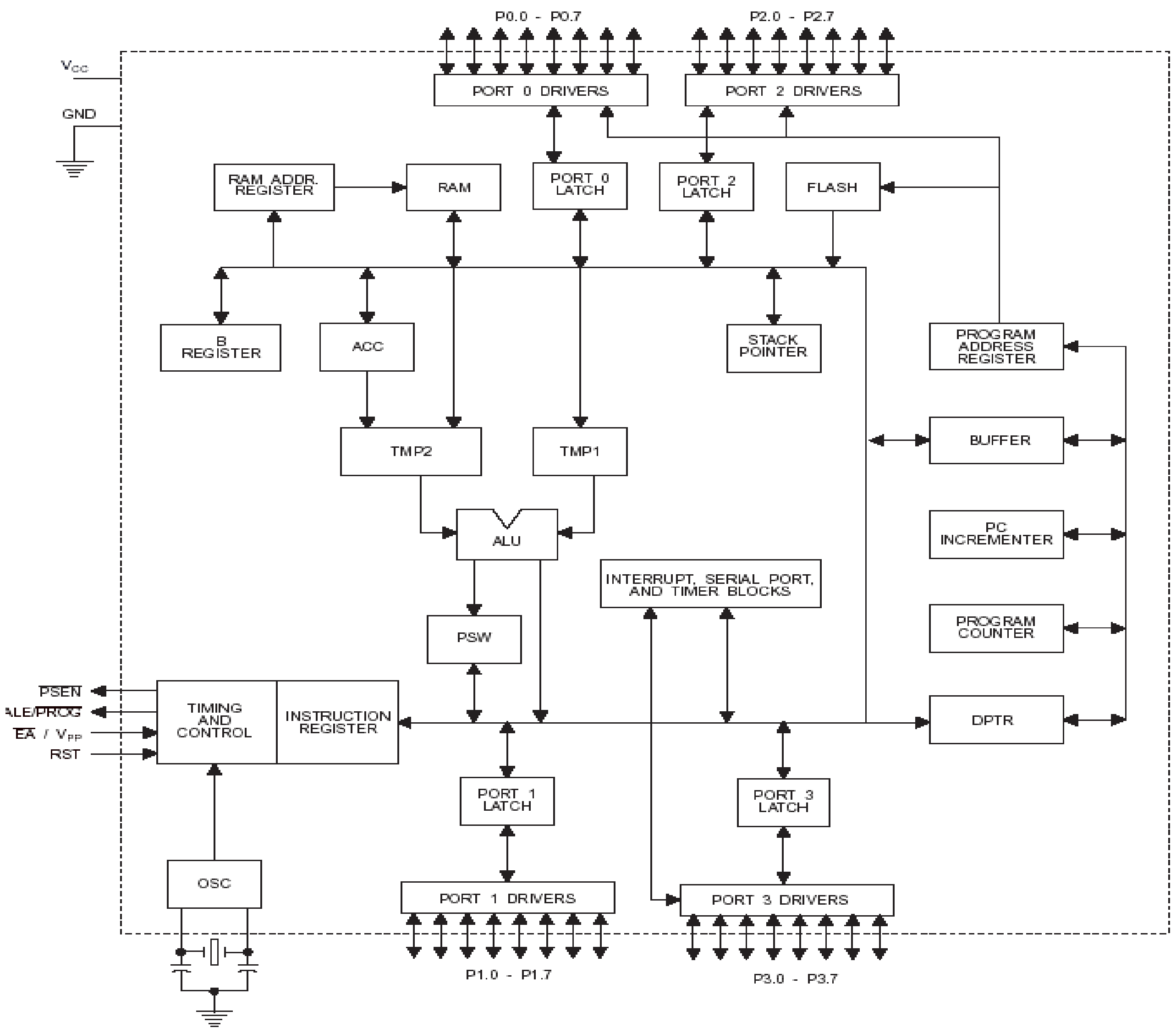


Fig 3a: Internal architecture diagram of 8051 Microcontroller

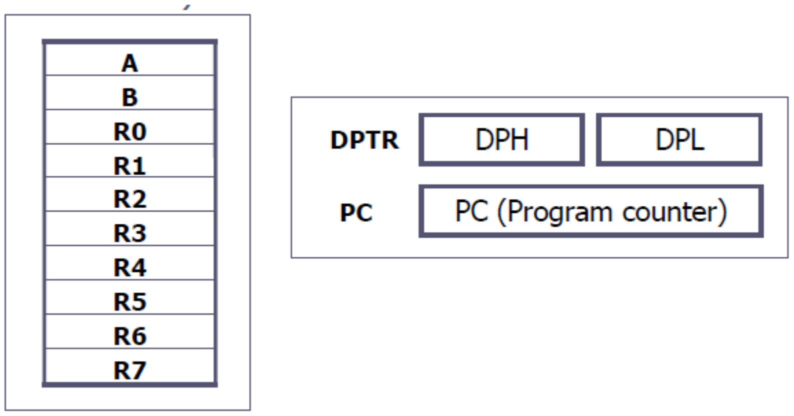


Fig: Structure of registers

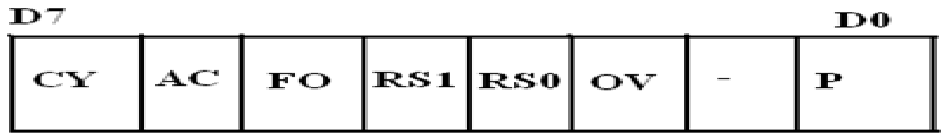
STACK in 8051 Microcontroller:

- ✓ The stack is a part of RAM used by the CPU to store information temporarily.
- ✓ This information may be either data or an address.
- ✓ The register used to access the stack is called the Stack pointer (SP).
- ✓ SP is an 8-bit register. So, it can take values of 00 to FF H.
- ✓ When the 8051 is powered up, the SP register contains the value 07.i.e the RAM location value 08 is the first location being used for the stack by the 8051 controller.
- ✓ There are two important instructions to handle stack. One is the PUSH and the other is the POP.
- ✓ The loading of data from CPU registers to the stack is done by PUSH.
- ✓ The loading of the contents of the stack back into a CPU register is done by POP.

Program Status Register (PSW):

Give PSW of 8051 and describe the use of each bit in PSW. (NOV 2015)

- ✓ The 8051 has an 8-bit PSW register which is also known as Flag register.
- ✓ In the 8-bit register only 6-bits are used by 8051.The two unused bits are user definable bits.
- ✓ In the 6-bits, four of them are conditional flags. They are Carry –CY, Auxiliary Carry-AC, Parity-P, and Overflow-OV.
- ✓ These flag bits indicate some conditions of result after an instruction was executed.



- ✓ The bits PSW3 and PSW4 are denoted as RS0 and RS1.
- ✓ These bits are used to select the bank registers of the RAM location.
- ✓ The meaning of various bits of PSW register is shown below.

CY	PSW.7	Carry Flag
AC	PSW.6	Auxiliary Carry Flag
FO	PSW.5	Flag 0 available for general purpose
RS1	PSW.4	Register Bank select bit 1
RS0	PSW.3	Register bank select bit 0
OV	PSW.2	Overflow flag
---	PSW.1	User definable flag
P	PSW.0	Parity flag .set/cleared by hardware.

- ✓ The selection of the register Banks and their addresses are given below.

RS1	RS0	Register Bank	Address
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

RAM & ROM:

- ✓ The 8051 microcontroller has 128 bytes of Internal RAM and 4KB of on chip ROM.
- ✓ The RAM is also known as Data memory and the ROM is known as program (Code) memory.
- ✓ Code memory holds program that is to be executed.
- ✓ Program Address Register holds address of the ROM/ Flash memory.
- ✓ Data Address Register holds address of the RAM.

I/O ports:

- ✓ The 8051 microcontroller has 4 parallel I/O ports, each of 8-bits.
- ✓ So, it provides 32 I/O lines for connecting the microcontroller to the peripherals.
- ✓ The four ports are P0 (Port 0), P1 (Port1), P2 (Port 2) and P3 (Port3).

****

4.3: Memory organization :

Explain in detail the internal memory organization of 8051 microcontroller (NOV 2014, May 2012, NOV 2011, NOV 2010, May 2010, MAY 2009, NOV 2008, NOV 2007)

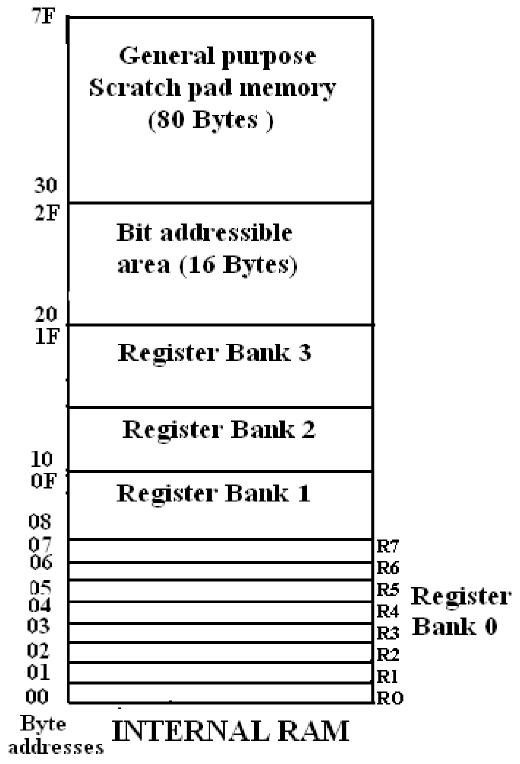
- ✓ The 8051 microcontroller has 128 bytes of Internal RAM and 4kB of on chip ROM.
- ✓ The RAM is also known as Data memory and the ROM is known as program (Code) memory.
- ✓ Code memory holds the actual 8051 program to be executed.
- ✓ In 8051, memory is limited to 64KB.
- ✓ Code memory may be found on-chip, as ROM or EPROM.
- ✓ It may also be stored completely off-chip in an external ROM / EPROM.
- ✓ The 8051 has only 128 bytes of Internal RAM but it supports 64KB of external RAM.
- ✓ Since the memory is off-chip, it is not as flexible for accessing and is also slower.

Structure of Internal RAM OF 8051(Data Memory) :

Explain the Data memory structure of 8051. (NOV 2011)

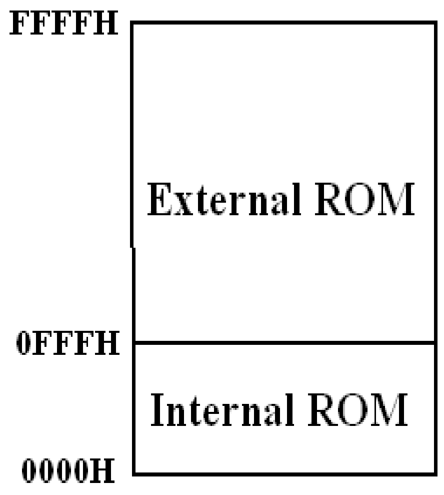
- ✓ Internal RAM is found on-chip on the 8051. So it is the fastest RAM available.
- ✓ It is flexible in terms of reading, writing and modifying its contents.
- ✓ Internal RAM is volatile.
- ✓ When the 8051 is reset, internal RAM is cleared.
- ✓ The 128 bytes of internal RAM is organized as below.
- ✓ Four register banks (Bank0, Bank1, Bank2 and Bank3) each of 8-bits (total 32 bytes).
- ✓ The default bank register is Bank0.
- ✓ The remaining Banks are selected with the help of RS0 and RS1 bits of PSW Register.
- ✓ 16 bytes of bit addressable area and
- ✓ 80 bytes of general purpose area (Scratch pad memory) of internal RAM as shown in the diagram below.
- ✓ This area is utilized by the microcontroller as a storage area for the operating stack.
- ✓ The 32 bytes of RAM from address 00 H to 1FH are used as working registers organized as four banks of eight registers each.
- ✓ The registers are named as R0-R7.
- ✓ Each register can be addressed by its name or by its RAM address.

For example: MOV A, R7 or MOV R7,#05H



Structure of Internal ROM (On –chip ROM / Program Memory / Code Memory):

- ✓ The 8051 microcontroller has 4KB of on chip ROM, but it can be extended up to 64KB.
- ✓ This ROM is also called program memory or code memory.
- ✓ The CODE segment is accessed using the program counter (PC) for opcode fetches and by DPTR for data.
- ✓ The external ROM is accessed when the EA pin is connected to ground or the contents of program counter exceeds 0FFFH.
- ✓ When the Internal ROM address is exceeded the 8051 automatically fetches the code bytes from the external program memory.



****

4.4: SPECIAL FUNCTION REGISTERS (SFRs)

Write the available special function registers in 8051. Explain each register with its format and functions. (April 2017, NOV 2015)

- ✓ In 8051 microcontroller, there are registers which uses the RAM addresses from 80h to FFh.
- ✓ They are used for certain specific operations. These registers are called Special Function Registers (SFRs).
- ✓ Most of SFRs are bit addressable and other few registers are byte addressable.
- ✓ In these SFRs, some of them are related to I/O ports (P0, P1, P2 and P3) and some of them are for control operations (TCON, SCON & PCON).
- ✓ Remaining are the auxiliary SFRs, that they don't directly configure the 8051.
- ✓ The list of SFRs and their functional names are given below.

- ✓ **The * indicates the bit addressable SFRs**

S.No	Symbol		Name of SFR	Address (Hex)
1	ACC*		Accumulator	**0E0**
2	B*		B-Register	**0F0**
3	PSW*		Program Status word register	**0DO**
4	SP		Stack Pointer Register	**81**
5	DPTR	DPL	Data pointer low byte	**82**
		DPH	Data pointer high byte	**83**
6	P0*		Port 0	**80**
	P1*		Port 1	**90**
8	P2*		Port 2	**0A**
9	P3*		Port 3	**0B**
10	IP*		Interrupt Priority control	**0B8**
11	IE*		Interrupt Enable control	**0A8**
12	TMOD		Timer mode register	**89**
13	TCON*		Timer control register	**88**
14	TH0		Timer 0 Higher byte	**8C**
15	TL0		Timer 0 Lower byte	**8A**
16	TH1		Timer 1Higher byte	**8D**
17	TL1		Timer 1 lower byte	**8B**
18	SCON*		Serial control register	**98**
19	SBUF		Serial buffer register	**99**
20	PCON		Power control register	**87**

Table: Special Function Registers

4.5: Input / Output (I/O) ports :

What are the I/O ports available in 8051 and explain? (MAY 2014, NOV 2012, MAY2010, NOV2009)
Enumerate about the ports available in 8051. (MAY 2014)
Explain parallel port architecture of 8051 microcontroller. (NOV 2012)
Explain each PORT circuitry available in 8051. (NOV 2007)

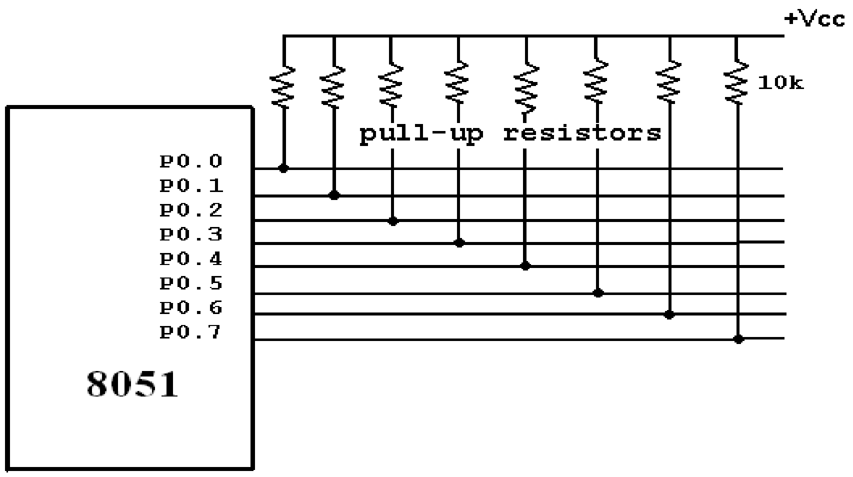
PARALLEL I /O PORTS :

- ✓ The 8051 microcontroller has 4 parallel I/O ports, each of 8-bits.
- ✓ So, it provides 32 I/O lines for connecting the microcontroller to the peripherals.
- ✓ The four ports are P0 (Port 0), P1 (Port1), P2 (Port 2) and P3 (Port3).
- ✓ When resetting, all the ports are output ports.
- ✓ In order to make them input, all the ports must be set. This is normally done by the instruction "SETB".
- ✓ Ex:

```
MOV A,#0FFH      ; A = FF
MOV P0, A        ; Make P0 an input port
```

PORT 0:

- ✓ Port 0 is an 8-bit I/O port with dual purpose.
- ✓ Port 0 does not have pull-up resistors internally, so pull-up resistors are to be connected externally as shown in the figure.



Dual role of port 0:

- ✓ If external memory is used, port 0 is used as lower order address/data bus (AD_0-AD_7), otherwise port 0 is used as input or output port.
- ✓ The 8051 multiplexes address and data through port 0 to reduce the pins.
- ✓ ALE indicates whether P0 has address or data.
- ✓ When ALE = 0, it provides data D0-D7, and when ALE =1, it provides address.

Port 0 circuit:

- ✓ Port-0 can be configured as a bidirectional I/O port or it can be used for lower order address/data lines.
- ✓ When control is '1', the port is used for address/data interfacing.
- ✓ When the control is '0', the port can be used as a bidirectional I/O port.

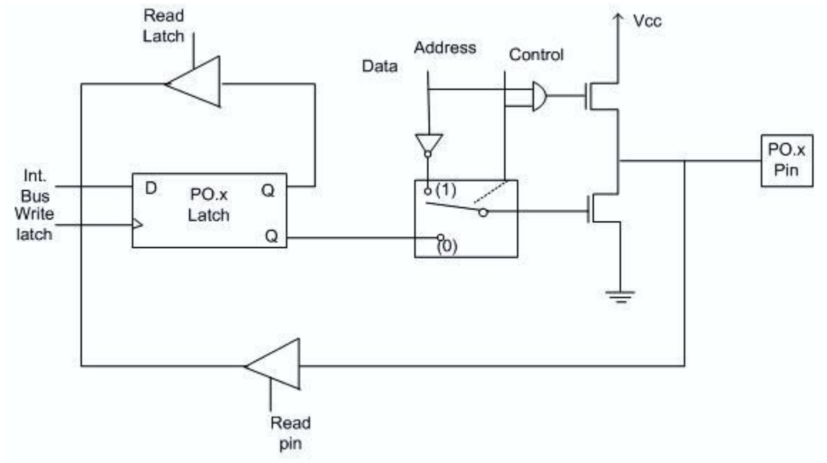


Fig: Port 0 Circuit

Port 1:

- ✓ Port 1 occupies a total of 8 pins. It has no dual application and acts only as I/O port.
- ✓ This port does not need any pull-up resistors because pull-up resistors connected internally.

Port 1 circuit:

- ✓ Port-1 does not have any alternate function i.e. it is dedicated solely for I/O interfacing.

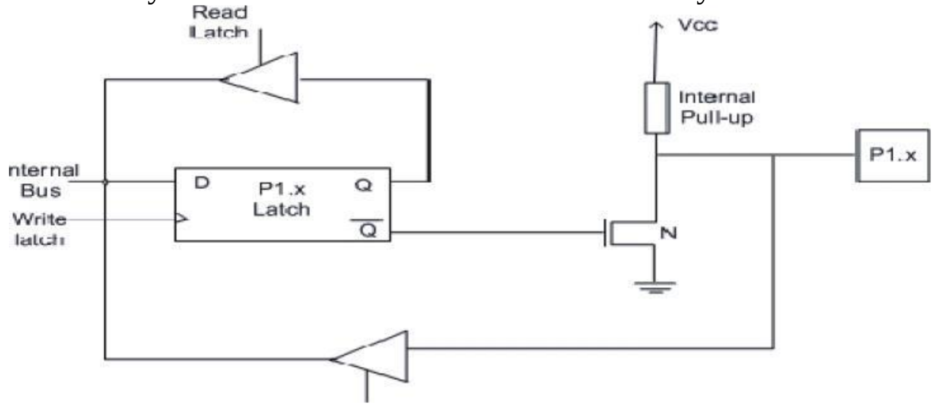


Fig: Port 1 Circuit

Port 2:

- ✓ Port 2 is an 8 bit parallel port. It can be used as input or output port.
- ✓ As this port is provided with internal pull-up resistors, it does not need any external pull-up resistors.

Dual role of port 2:

- ✓ Port 2 lines are also associated with the higher order address lines A8-A15.

- ✓ In 8051-based systems, port 2 is used along with P0 to provide the 16-bit address for the external memory.
- ✓ 8031/8051 is capable of accessing 64K bytes of external memory.
- ✓ When control is '1', the port is used for address interfacing.
- ✓ When the control is '0', the port can be used as a bidirectional I/O port.

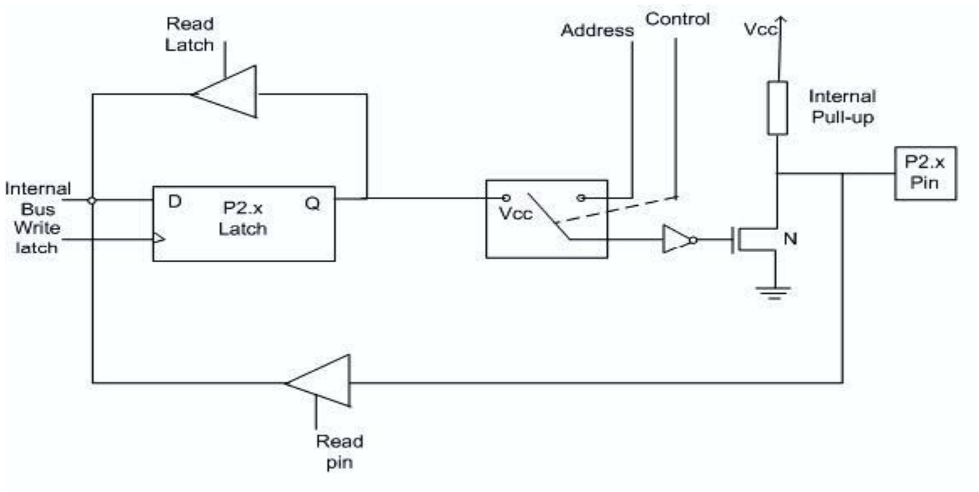


Fig: Port 2 Circuit

PORT 3:

Explain the use of port 3 of 8051 for interrupt signals. (NOV 2009)

- ✓ Port3 is an 8-bit parallel port with dual function.
- ✓ The port pins can be used for I/O operations as well as for control operations.
- ✓ Port 3 also do not need any external pull-up resistors as they are provided internally.
- ✓ The details of additional operations are given below in the table.

S.No	Port 3 bit	Pin No	Function
1	P3.0	10	RxD
2	P3.1	11	TxD
3	P3.2	12	$\overline{INT0}$
4	P3.3	13	$\overline{INT1}$
5	P3.4	14	T0
6	P3.5	15	T1
7	P3.6	16	$\overline{WR}$
8	P3.7	17	$\overline{RD}$

Table: PORT 3 alternate functions

Alternate Functions of Port 3:

- ✓ P3.0 and P3.1 are used for the RxD (Receive Data) and TxD (Transmit Data) as serial communications signals. Bits P3.2 and P3.3 are meant for external interrupts.
- ✓ Bits P3.4 and P3.5 are used for Timers 0 and Timer 1.
- ✓ P3.6 and P3.7 are used to provide the write and read signals of external memories connected in 8031/ 8051 based systems.

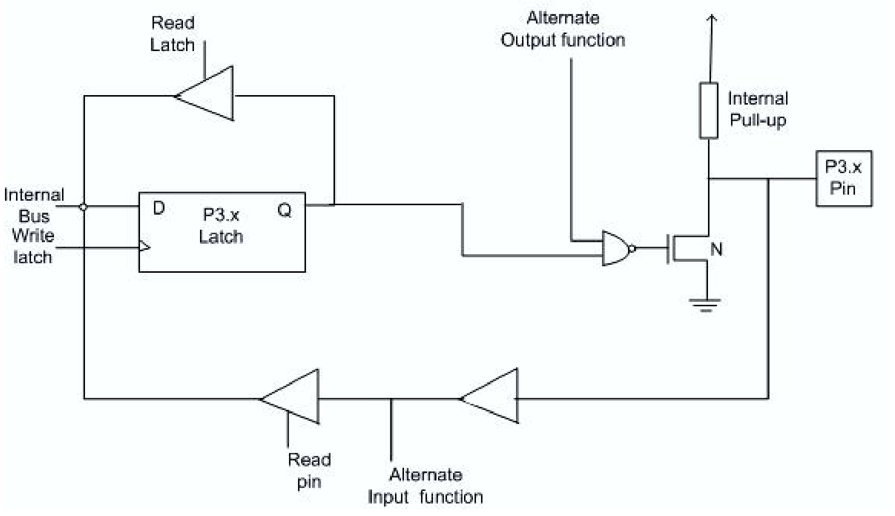


Fig: Port 3 Circuit

****

4.6: Interrupt of 8051 Microcontroller:

Explain interrupt structure of 8051 microcontroller. (NOV 2011, MAY 2009)

Interrupt Structure:

- ✓ An interrupt is an external or internal event that disturbs the microcontroller to inform it that a device needs its service.
- ✓ The program which is associated with the interrupt is called the **interrupt service routine** (ISR) or **interrupt handler**.
- ✓ Upon receiving the interrupt signal, the microcontroller finishes current operation and saves the PC on stack.
- ✓ Jumps to a fixed location in memory depending on type of interrupt.
- ✓ Starts to execute the interrupt service routine until RETI.

- ✓ Upon executing the RETI the microcontroller returns to the place where it was interrupted. Get pop PC from stack.
- ✓ The 8051 microcontroller has **FIVE** interrupts in addition to Reset. They are
  - Timer 0 overflow Interrupt
  - Timer 1 overflow Interrupt
  - External Interrupt 0(INT0)
  - External Interrupt 1(INT1)
  - Serial Port Interrupt
- ✓ Each interrupt has a specific place in code memory where program execution begins.
  - External Interrupt 0: 0003 H
  - Timer 0 overflow: 000B H
  - External Interrupt 1: 0013 H
  - Timer 1 overflow: 001B H
  - Serial Interrupt : 0023 H
- ✓ Upon reset all Interrupts are disabled & do not respond to the Microcontroller.
- ✓ These interrupts must be enabled by software. This is done by an 8-bit register called Interrupt Enable Register (IE).

Interrupt Enable Register :

EA	——	ET2	ES	ET1	EX1	ET0	EX0

- ▪ EA : Global enable/disable. To enable the interrupts, this bit must be set high.
- ▪ --- : Undefined-reserved for future use.
- ▪ ET2 : Enable /disable Timer 2 overflow interrupt.
- ▪ ES : Enable/disable Serial port interrupts.
- ▪ ET1 : Enable /disable Timer 1 overflow interrupt.
- ▪ EX1 : Enable/disable External interrupt1.
- ▪ ET0 : Enable /disable Timer 0 overflow interrupt.
- ▪ EX0 : Enable/disable External interrupt0

- ✓ Upon reset, the interrupts have the following priority from top to down. The interrupt with the highest PRIORITY gets serviced first.

  1. External interrupt 0 (INT0)
  2. Timer interrupt0 (TF0)
  3. External interrupt 1 (INT1)
  4. Timer interrupt1 (TF1)
  5. Serial communication (RI+TI)

- ✓ Priority can also be set to “high” or “low” by 8-bit IP register.

<u>Interrupt priority register:</u>

——	——	PT2	PS	PT1	PX1	PT0	PX0

- IP.7: reserved
- IP.6: reserved
- IP.5: Timer 2 interrupt priority bit (8052 only)
- IP.4: Serial port interrupt priority bit
- IP.3: Timer 1 interrupt priority bit
- IP.2: External interrupt 1 priority bit
- IP.1: Timer 0 interrupt priority bit
- IP.0: External interrupt 0 priority bit

****

<u>4.7: TIMERS in 8051 Microcontrollers:</u>

Explain in detail the timer of 8051 and their associated registers. (NOV 2009, MAY2009)

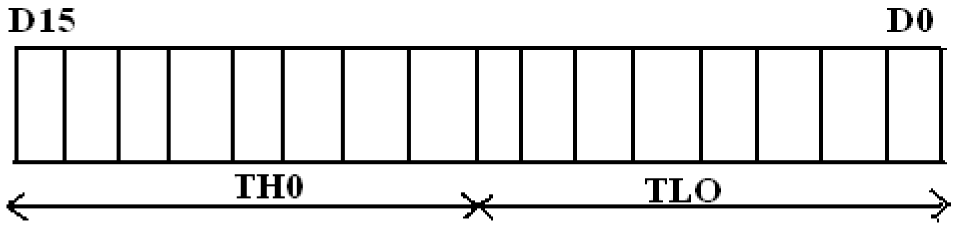
How are the timers of 8051 used to produce time delay in timer mode? (NOV 2011)

- ✓ The 8051 microcontroller has two 16-bit timers Timer 0 (T0) and Timer 1(T1), which can be used either to generate accurate time delays or as event counters.

- ✓ These timers are accessed as two 8-bit registers TLO, THO & TL1, TH1 because the 8051 microcontroller is 8-bit architecture.

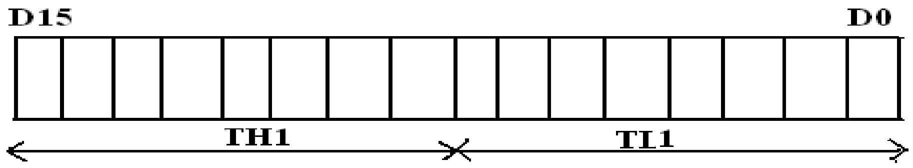
TIMER 0 :

- ✓ The Timer 0 is a 16-bit register and can be treated as two 8-bit registers (TL0 & TH0).
- ✓ These registers can be accessed similar to other registers like A, B or R1, R2, R3 etc…
- ✓ Ex : The instruction MOV TL0,#07; Moves the value 07 into lower byte of Timer0.



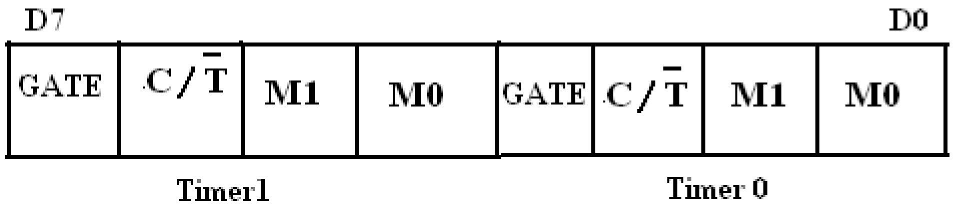
TIMER 1 :

- ✓ The Timer 1 is also a 16-bit register and can be treated as two 8-bit registers (TL1 & TH1).
- ✓ These registers can be accessed similar to any other registers like A, B or R1, R2, R3 etc…
- ✓ Ex : The instruction MOV TL1,#05: Moves the value 05 into lower byte of Timer1.



TMOD Register:

- ✓ The various operating modes of both the timers T0 and T1 are set by an 8-bit register called TMOD register**.**
- ✓ In this TMOD register the lower 4-bits are meant for Timer 0 and the higher 4-bits are meant for Timer1.



GATE:

- ✓ This bit is used to start or stop the timers by hardware.
- ✓ When GATE= 1, the timers can be started / stopped by the external sources.

- ✓ When GATE= 0, the timers can be started or stopped by software instructions like SETB TR0 or SETB TR1.

$C/\overline{T}$ **(Clock/Timer) :**

- ✓ This bit decides whether the timer is used as delay generator or event counter.
- ✓ When $C/\overline{T}$ **= 0,** the timer is used as delay generator and if $C/\overline{T}$ =1 the timer is used as an event counter.
- ✓ The clock source for the time delay is the crystal frequency of 8051.

M1, M0 (Mode):

- ✓ These two bits are the timer mode bits.
- ✓ The timers of the 8051 can be configured in three modes as Mode0, Mode1 and Mode2.
- ✓ The selection and operation of the modes is shown below.

S.No	M0	M1	Mode	Operation
1	0	0	0	**13-bit Timer mode.** 8-bit Timer/counter THx with TLx as 5-bit prescaler
2	0	1	1	**16-bit Timer mode**.16-bit timer /counter THx and TLx are cascaded. There is no prescaler
3	1	0	2	**8-bit auto reload.** 8-bit auto reload timer/counter. THx holds a value which is to be reloaded TLx each time it overflows
4	1	1	3	**13-bit Timer mode.** 8-bit Timer/counter THx with TLx as 5-bit prescaler

TCON (Timer control register)

- ✓ TCON (timer control) register is an 8-bit register. TCON register is a bit-addressable register.

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

- TF1: Timer 1 overflow flag.
- TR1: Timer 1 run control bit.
- TF0: Timer 0 overflow flag.
- TR0: Timer 0 run control bit.
- IE1: External interrupt 1 edge flag.
- IT1: External interrupt 1 type flag.

- IE0: External interrupt 0 edge flag.
- IT0: External interrupt 0 type flag.

****

4.8: PIN Diagram of 8051 Microcontroller:

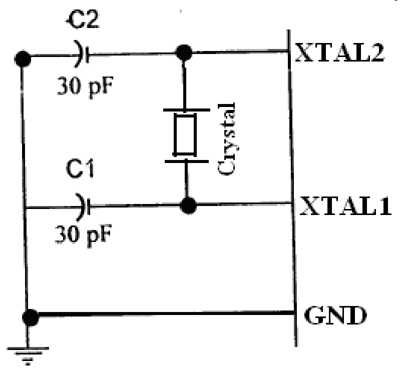
Explain Pin details of 8051 microcontroller. (MAY 2006)

Describe the functions of the following signals in 8051. RST, EA, PSEN and ALE. (NOV 2015)

- ✓ The 8051 microcontroller is available as a 40 pin DIP chip and it works at +5 volts DC.
- ✓ Among the 40 pins, a total of 32 pins are allotted for the four parallel ports P0, P1, P2 and P3 i.e each port occupies 8-pins.
- ✓ The remaining pins are VCC, GND, XTAL1, XTAL2, RST, EA ,PSEN.

XTAL1, XTAL2:

- ✓ These two pins are connected to Quartz crystal oscillator which runs the on-chip oscillator.
- ✓ The quartz crystal oscillator is connected to the two pins along with a capacitor of 30pF as shown in the circuit.
- ✓ If use a source other than the crystal oscillator, it will be connected to XTAL1 and XTAL2 is left unconnected.



RST:

- ✓ The RESET pin is an input pin and it is an active high pin.
- ✓ When a high pulse is applied to this pin, the microcontroller will reset and terminate all activities.
- ✓ Upon reset all the registers will reset to 0000 Value and SP register will reset to 0007 value.

$\overline{\text{EA}}$ (External Access):

- ✓ This pin is an active low pin.

- ✓ This pin is connected to ground when microcontroller is accessing the program code stored in the external memory.
- ✓ This pin is connected to Vcc when it is accessing the program code in the on chip memory.

$\overline{\text{PSEN}}$ (Program Store Enable):

- ✓ This is an output pin which is active low.
- ✓ When the microcontroller is accessing the program code stored in the external ROM, this pin is connected to the OE (Output Enable) pin of the ROM.

ALE (Address latch enable):

- ✓ This is an output pin, which is active high**.**
- ✓ This ALE pin will demultiplex the address and data bus.
- ✓ When the pin is high, the Address/ Data bus will act as address bus, otherwise the Address/ Data bus will act as Data bus.

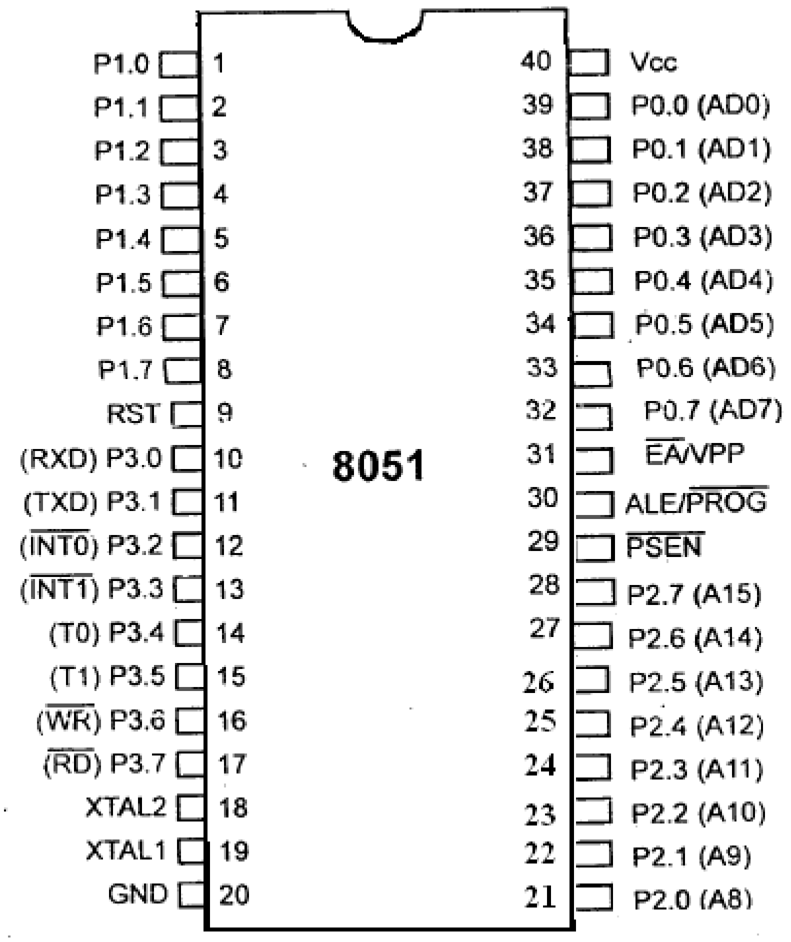


Figure: Pin diagram of 8051

P0.0- P0.7(AD0-AD7) :

- ✓ The port 0 pins multiplexed with Address/data pins.
- ✓ If the microcontroller is accessing external memory, these pins will act as address/data pins, otherwise they are used for Port 0 pins.

P2.0- P2.7 (A8-A15) :

- ✓ The port2 pins are multiplexed with the higher order address pins**.**
- ✓ When the microcontroller is accessing external memory, these pins provide the higher order address byte, otherwise they act as Port 2 pins.

P1.0- P1.7 :

- ✓ These 8-pins are dedicated to perform input or output port operations.

P3.0- P3.7:

- ✓ These 8-pins are meant for Port3 operations and also for some control operations like read, Write, Timer0, Timer1, INT0, INT1, RxD and TxD.

****

4.9: ADDRESSING MODES OF 8051 :

Explain different types addressing modes of 8051 microcontroller. (NOV 2008, NOV 2015, April 2017)

- ✓ The way in which the data operands are specified is known as the addressing modes. There are various methods of denoting the data operands in the instruction.
- ✓ The 8051 microcontroller supports 5 addressing modes. They are
  1. Immediate addressing mode
  2. Direct Addressing mode
  3. Register addressing mode
  4. Register indirect addressing mode
  5. Indexed addressing mode

Immediate addressing mode:

- ✓ The addressing mode in which the data operand is a constant and it is a part of the instruction itself is known as Immediate addressing mode.
- ✓ Normally the data must be preceded by a # sign.
- ✓ This addressing mode can be used to transfer the data into any of the registers including DPTR.

Examples:

- ▪ MOV A, # 27 H : The data (constant) 27 is moved to the accumulator register

- ADD R1, #45 H : Add the constant 45 to the contents of the accumulator
- MOV DPTR, # 8245H : Move the data 8245 into the data pointer register.

Direct addressing mode:

- ✓ In the addressing mode, the data operand is in the RAM location (00 -7FH) and the address of the data operand is given in the instruction.
- ✓ The direct addressing mode uses the lower 128 bytes of Internal RAM and the SFRs

Examples:

- MOV R1, 42H : Move the contents of RAM location 42 into R1 register
- MOV 49H, A : Move the contents of the accumulator into the RAM location 49.
- ADD A, 56H : Add the contents of the RAM location 56 to the accumulator

Register addressing mode:

- ✓ In the addressing mode, the data operands are available in the registers.

Examples:

- MOV A,R0 : Move the contents of the register R0 to the accumulator
- MOV P1, R2 :Move the contents of the R2 register into port 1
- MOV R5, R2 : This is invalid. The data transfer between the registers is not allowed.

Register Indirect addressing mode:

- ✓ In the addressing mode, a register is used as a pointer to the data memory block.

Examples:

- MOV A,@ R0 :Move the contents of RAM location whose address is in R0 into **A** (accumulator)
- MOV @ R1 , B : Move the contents of B into RAM location whose address is held by R1
- When R0 and R1 are used as pointers, they must be preceded by @ sign
- ✓ **Advantage: It makes accessing the data more dynamic than static as in the case of direct addressing mode.**

Indexed addressing mode:

- ✓ This addressing mode is used in accessing the data elements of lookup table entries, located in program ROM.

Example: MOVC A, @ A+DPTR

- The 16-bit register DPTR and register A are used to form the address of the data element stored in on-chip ROM.

****

<u>4.10: Instructions Set of 8051:</u>

<u>Discuss in detail the 8051 instruction set. (NOV 2008)</u>

4.10.1:Arithmetic instructions:

With example, explain arithmetic instructions in 8051 microcontroller. (NOV 2012)

- ✓ ADD
  - 8-bit addition between the accumulator (A) and a second operand.
  - The result is always in the accumulator.
  - The CY flag is set/reset appropriately.
- ✓ ADDC
  - 8-bit addition between the accumulator, a second operand and the previous value of the CY flag.
  - Useful for 16-bit addition in two steps.
  - The CY flag is set/reset appropriately.
- ✓ DAA
  - Decimal adjust the accumulator.
  - Format the accumulator into a proper 2 digit packed BCD number.
  - Operates only on the accumulator.
  - Works only after the ADD instruction.
- ✓ SUBB
  - Subtract with Borrow.
  - Subtract an operand and the previous value of a borrow (carry) flag from the accumulator.
  - A ← A - <operand> - CY.
  - The result is always saved in the accumulator.
  - The CY flag is set/reset appropriately.
- ✓ INC
  - Increment the operand by one.
  - The operand can be a register, a direct address, an indirect address, the data pointer.
- ✓ DEC
  - Decrement the operand by one.
  - The operand can be a register, a direct address, an indirect address.

- ✓ MUL AB / DIV AB
    - Multiply A by B and place result in A and B registers.
    - Divide A by B and place quotient in A register & remainder in B register.

4.10.2: Logical instructions in 8051.

- ✓ ANL : It performs AND logical operation between two operands.
    - ➢ Work on byte sized operands or the CY flag.
        - ANL A, Rn
        - ANL A, direct
        - ANL A, @Ri
        - ANL A, #data
        - ANL direct, A
        - ANL direct, #data
        - ANL C, bit
        - ANL C, /bit
- ✓ ORL: It performs OR logical operation between two operands.
    - ➢ Work on byte sized operands or the CY flag.
        - ORL A, Rn
        - ORL A, direct
        - ORL A, @Ri
        - ORL A, #data
- ✓ XRL
    - ➢ Works on bytes only.
        - XRL A, Rn
        - XRL A, direct
- ✓ CPL / CLR
    - ➢ Complement / Clear.
    - ➢ Work on the accumulator or a bit.
        - CLR P1.2
        - CPL Rn
- ✓ RL / RLC / RR / RRC
    - ➢ Rotate the accumulator.
        - RL and RR without the carry

- RLC and RRC rotate through the carry.
- SWAP A: Swap the upper and lower nibbles of the accumulator.

4.10.3: Data transfer instructions in 8051.

Briefly explain the data transfer instructions available in 8051 microcontroller. (NOV 2014)

MOV

- ➢ 8-bit data transfer for internal RAM and the SFR.
  - MOV A, Rn
  - MOV A, direct
  - MOV A, @Ri
  - MOV A, #data
  - MOV Rn, A
  - MOV Rn, direct
  - MOV Rn, #data
  - MOV direct, A
  - MOV direct, Rn
  - MOV direct, direct
  - MOV direct, @Ri
  - MOV direct, #data
  - MOV @Ri, A
  - MOV @Ri, direct
  - MOV @Ri, #data

- ✓ MOV
  - ➢ 1-bit data transfer involving the CY flag
    - MOV C, bit
    - MOV bit, C
- ✓ MOV
  - ➢ 16-bit data transfer involving the DPTR
    - MOV DPTR, #data
- ✓ MOVC
  - ➢ Move Code Byte
    - Load the accumulator with a byte from program memory.

- Must use indexed addressing
- MOVC A, @A+DPTR
- MOVC A, @A+PC

✓ MOVX

➢ Data transfer between the accumulator and a byte from external data memory.

- MOVX A, @Ri
- MOVX A, @DPTR
- MOVX @Ri, A
- MOVX @DPTR, A

✓ PUSH / POP

➢ Push and Pop a data byte onto the stack.

➢ The data byte is identified by a direct address from the internal RAM locations.

- PUSH DPL
- POP 40H

✓ XCH

➢ Exchange accumulator and a byte operand

- XCH A, Rn
- XCH A, direct
- XCH A, @Ri

✓ XCHD

➢ Exchange lower digit of accumulator with the lower digit of the memory location specified.

- XCHD A, @Ri
- The lower 4-bits of the accumulator are exchanged with the lower 4-bits of the internal memory location identified indirectly by the index register.
- The upper 4-bits of each are not modified.

4.10.4: Boolean (or) Bit manipulation instructions in 8051.

✓ This group of instructions is associated with the single-bit operations of the 8051.

✓ This group allows manipulating the individual bits of bit addressable registers and memory locations as well as the CY flag.

- The P, OV, and AC flags cannot be directly altered.

✓ This group includes:

- Set, clear, and, or complement, move.
- Conditional jumps.

✓ CLR

- Clear a bit or the CY flag.
- CLR P1.1
- CLR C

✓ SETB

- Set a bit or the CY flag.
- SETB A.2
- SETB C

✓ CPL

- Complement a bit or the CY flag.
- CPL 40H; Complement bit 40 of the bit addressable memory

✓ ORL / ANL

- OR / AND a bit with the CY flag.
- ORL C, 20H; OR bit 20 of bit addressable memory with the CY flag
- ANL C, 34H; AND bit 34 of bit addressable memory with the CY flag.

✓ MOV

- Data transfer between a bit and the CY flag.
- MOV C, 3FH; Copy the CY flag to bit 3F of the bit addressable memory.
- MOV P1.2, C; Copy the CY flag to bit 2 of P1.

✓ JC / JNC

- Jump to a relative address if CY is set / cleared.

✓ JB / JNB

- Jump to a relative address if a bit is set / cleared.
- JB ACC.2, <label>

✓ JBC - Jump to a relative address, if a bit is set and clear the bit.

4.10.5: Branching instructions:

With example, explain branching instructions in 8051 microcontroller. (May 2010, NOV 2012)

Explain the working of program control transfer instructions of 8051. (May 2012)

✓ The 8051 provides four different types of unconditional jump instructions:

➢ Short Jump – SJMP

- Uses an 8-bit signed offset relative to the 1st byte of the next instruction.
- Long Jump – LJMP
- Uses a 16-bit address.
- 3 byte instruction capable of referencing any location in the entire 64K of program memory.

➢ Absolute Jump – AJMP

- Uses an 11-bit address.
- 2 byte instruction
- The 11-bit address is substituted for the lower 11-bits of the PC to calculate the 16-bit address of the target.
- The location referenced must be within the 2K Byte memory.

➢ Indirect Jump – JMP

- JMP @A + DPTR

✓ The 8051 provides 2 forms for the CALL instruction:

➢ Absolute Call – ACALL

- Uses an 11-bit address similar to AJMP
- The subroutine must be within the same 2K page.

➢ Long Call – LCALL

- Uses a 16-bit address similar to LJMP
- The subroutine can be anywhere.

➢ Both forms push the 16-bit address of the next instruction on the stack and update the stack pointer.

✓ The 8051 provides 2 forms for the return instruction:

➢ Return from subroutine – RET

- Pop the return address from the stack and continue execution there.

➢ Return from Interrupt Service Routine – RETI

- Pop the return address from the stack.
- Continue execution at the address retrieved from the stack.
- The PSW is not automatically restored.

✓ The 8051 supports 5 different conditional jump instructions.

➢ ALL conditional jump instructions use an 8-bit signed offset.

➢ Jump on Zero – JZ / JNZ

- Jump if the A == 0 / A != 0
    - The check is done at the time of the instruction execution.

➢ Jump on Carry – JC / JNC

- Jump if the C flag is set / cleared.

➢ Jump on Bit – JB / JNB

- Jump if the specified bit is set / cleared.
- Any addressable bit can be specified.

➢ Jump if the Bit is set then Clear the bit – JBC

- Jump if the specified bit is set.
- Then clear the bit.

✓ Compare and Jump if Not Equal – CJNE

➢ Compare the magnitude of the two operands and jump if they are not equal.

- The values are considered to be unsigned.
- The Carry flag is set / cleared appropriately.
- CJNE A, direct, rel
- CJNE Rn, #data, rel
- CJNE @Ri, #data, rel

✓ Decrement and Jump if Not Zero – DJNZ

➢ Decrement the first operand by 1 and jump to the location identified by the second operand if the resulting value is not zero.

- DJNZ Rn, rel
- DJNZ direct, rel

➢ NOP – No operation

****

Program 1: Using timers in 8051 write a program to generate square wave 100ms, 50% duty cycle. (NOV 2014, May 2016, May 2012)

```
              MOV TMOD, #01
              Here: MOV TL0, #D7
              MOV TH0, #B4
              CPL P1.3
              SETB TRO
       Again: JNB TF0, Again
              CLR TR0
              CLR TF0
              SJMP Here
```

Program 2: Write an 8051 ALP to multiply the given number 48H and 30H. (April 2017)

Mnemonics		Comments
Opcode	Operand	
MOV	A,#48	;Store data1 in accumulator
MOV	B,#30	;Store data2 in B register
MUL	AB	;Multiply both
MOV	DPTR,#4500	;Initialize memory location
MOVX	@DPTR,A	;Store lower order result
INC	DPTR	;Go to next memory location
MOV	A,B	;Store higher order result
MOVX	@DPTR,A	
L1: SJMP	L1	;Stop the program

Program 3: Write a program to add two 16 bit numbers. The numbers are 8C8D and 8D8C. Place the sum in R7 and R6. R6 should have the lower byte. (NOV 2010)

Mnemonics		Comments
Opcode	Operand	
MOV	A, #8D	;Store LSB data1 in accumulator
MOV	B, #8C	;Store LSB data2 in B register
ADD	A, B	;Add both
MOV	R6, A	;Store LSB result
MOV	A, #8C	;Store MSB data1 in accumulator
MOV	B, #8D	;Store MSB data2 in B register
ADD	A, B	;Add both
MOV	R7, A	;Store MSB result
L1: SJMP	L1	;Stop the program

UNIT V

INTERFACING MICROCONTROLLER

Programming 8051 Timers - Serial Port Programming - Interrupts Programming – LCD & Keyboard Interfacing - ADC, DAC & Sensor Interfacing - External Memory Interface- Stepper Motor and Waveform generation.

5.1: PROGRAMMING TIMERS OF 8051

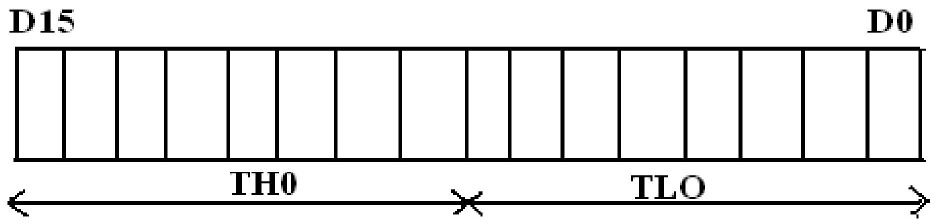
1. **Explain the different modes of operation of timers in 8051 in detail with its associated registers.**
   Describe different modes of operation of timers /counters in 8051 with its associated registers. (NOV 2009, MAY 2009. May 2007, May 2016)
   Draw and explain the functions of TCON and TMOD registers of 8051. (Dec 2008)
   Explain the on-chip timer modes of an 8051 Microcontroller. (April 2010, NOV 2016)

Timer Registers.

✓ The 8051 has two timers/counters, they can be used either as timers (used to generate a time delay) or as event counters.

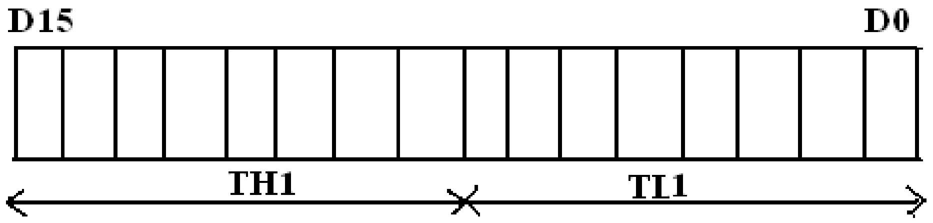
TIMER 0:

- ✓ Timer 0 is a 16-bit register and can be treated as two 8-bit registers (TL0 & TH0).
- ✓ These registers can be accessed similar to any other registers like A, B or R1 etc
- ✓ Ex : The instruction MOV TL0,#07 moves the value 07 into lower byte of Timer0.
- ✓ Similarly MOV R1, TH0 saves the contents of TH0 in the R1 register.



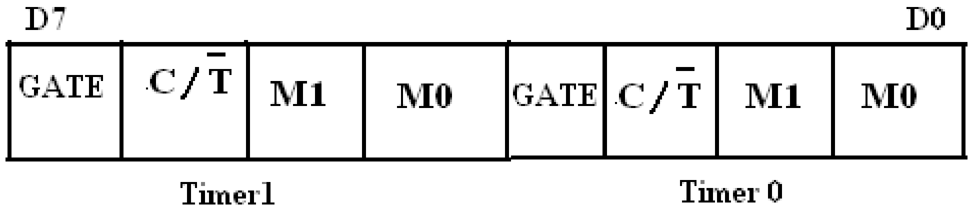
TIMER 1:

- ✓ Timer 1 is also a 16-bit register and can be treated as two 8-bit registers (TL1 & TH1).
- ✓ These registers can be accessed similar to any other registers like A, B or R1etc
- ✓ Ex : The instruction MOV TL1,#05 moves the value 05 into lower byte of Timer1.
- ✓ Similarly MOV R0,TH1 saves the contents of TH1 in the R0 register.



TMOD (Timer mode Register):

- ✓ The various operating modes of both the timers T0 and T1 are set by a TMOD register.
- ✓ TMOD is a 8-bit register.
- ✓ The lower 4 bits are for Timer 0
- ✓ The upper 4 bits are for Timer 1
- ✓ In each case,
  - The lower 2 bits are used to set the timer mode
  - The upper 2 bits to specify the operation



GATE:

- ✓ This bit is used to start or stop the timers by hardware.
- ✓ When GATE= 1, the timers can be started / stopped by the external sources.
- ✓ When GATE= 0, the timers can be started or stopped by software instructions like SETB TR_X or CLR $TR_{X.}$

C/T (Counter/Timer):

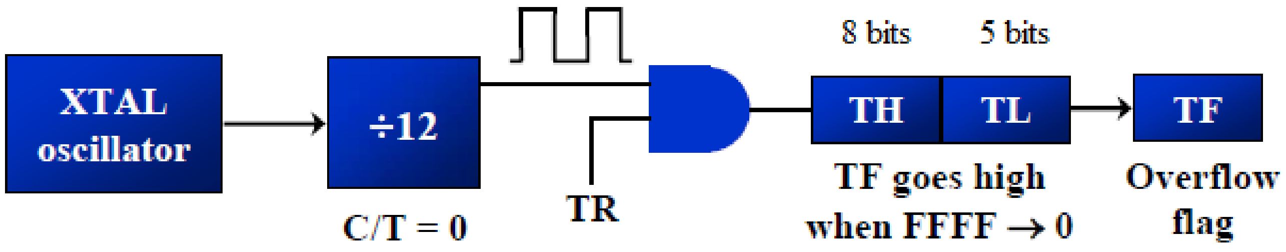
- ✓ This bit decides whether the timer is used as delay generator or event counter.
- ✓ When $C/\overline{T}$ **= 0,** timer is used as delay generator.
- ✓ When $C/\overline{T}$ **=1,** timer is used as an event counter.
- ✓ The clock source for the time delay is the crystal frequency of 8051.
- ✓ The clock source for the event counter is the external clock source.

M1, M0 (Mode):

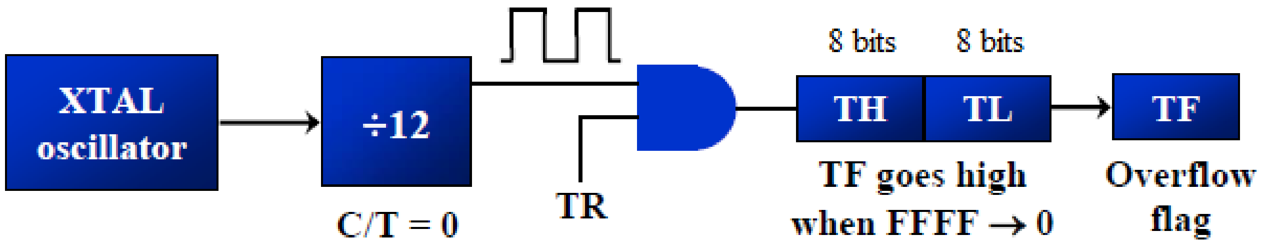
- ✓ These two bits are the timer mode bits.
- ✓ The timers of the 8051 can be configured in four modes Mode0, Mode1, Mode2 & Mode 3.
- ✓ The selection and operation of the modes is shown below.

S.No	M0	M1	Mode	Operation
1	0	0	Mode 0	**13-bit Timer mode.** 8-bit Timer/counter THx with TLx as 5-bit prescaler
2	0	1	Mode 1	**16-bit Timer mode.**16-bit timer /counter THx and TLx are cascaded. There is no presacler
3	1	0	Mode 2	**8-bit auto reload.** 8-bit auto reload timer/counter. THx holds a value which is to be reloaded TLx each time it overflows
4	1	1	Mode 3	**Split timer mode**

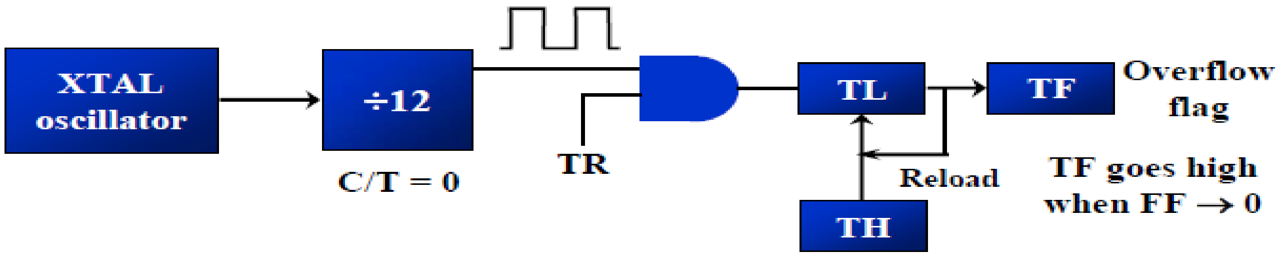
Mode 0: 13 bit Timer mode



Mode 1: 16 bit Timer mode



Mode 2: 8 bit auto reload mode



Mode 3: Split Timer mode

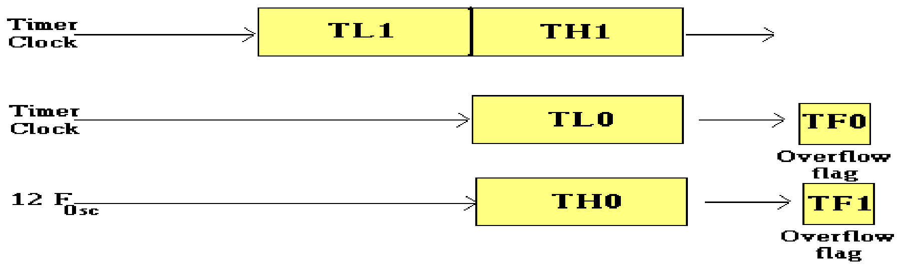


Figure: Modes of operation of Timer

TCON (Timer control register)

- ✓ TCON (timer control) register is an 8-bit register. TCON register is a bit-addressable register.

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

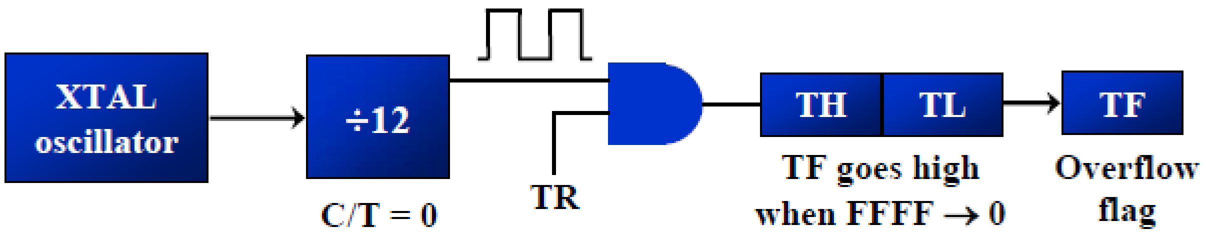
Bit Number	Bit Mnemonic	Description
7	TF1	Timer 1 overflow flag Cleared by hardware when processor vectors to interrupt routine. Set by hardware on timer/counter overflow, when the timer 1 register overflows.
6	TR1	Timer 1 run control bit Clear to turn off time/counter 1. Set to turn on timer/counter 1.
5	TF0	Timer 0 overflow flag Cleared by hardware when processor vectors to interrupt routine. Set by hardware on timer/counter overflow, when the timer 0 register overflows.
4	TR0	Timer 0 run control bit Clear to turn off time/counter 0. Set to turn on timer/counter 0.
3	IE1	External interrupt 1 edge flag. Cleared by hardware when interrupt is processed if edge-triggered. Set by hardware when external interrupt is detected on INT1 pin.
2	IT1	External interrupt 1 type control bit Clear to select low level active (level triggered) for external interrupt 1. Set to select falling edge active (edge triggered) for external interrupt 1.
1	IE0	External interrupt 0 edge flag Cleared by hardware when interrupt is processed if edge-triggered. Set by hardware when external interrupt is detected on INT0 pin.
0	IT0	External interrupt 0 type control bit Clear to select low level active (level triggered) for external interrupt 0. Set to select falling edge active (edge triggered) for external interrupt 0.

Timers of 8051 do starting and stopping by either software or hardware control

- ✓ For using software to start and stop the timer where GATE=0
- ✓ The start and stop of the timer are controlled by software using TR (timer start) bits TR_X and CLR_X
- ✓ The SETB instruction starts it, and it is stopped by the CLR instruction.
- ✓ These instructions start and stop the timers as long as GATE=0 in the TMOD register
- ✓ The hardware way of starting and stopping the timer is achieved by making GATE=1 in the TMOD register.

The following are the characteristics and operations of mode 1:

1. It is a 16-bit timer.
2. It allows value from 0000 to FFFFH.
3. Value to be loaded into the timer register TL and TH.
4. After TH and TL are loaded with a 16-bit initial value, the timer must be started
   - This is done by SETB TR0 for timer 0 and SETB TR1 for timer 1
5. After the timer is started, it starts to count up
   - It counts up until it reaches its limit of FFFFH
   - When it rolls over from FFFFH to 0000, it sets high a flag bit called TF (timer flag)
   - Each timer has its own timer flag.
   - There are TF0 for timer 0, and TF1 for timer 1.



6. Timer flag can be monitored,
   - When this timer flag is raised, to stop the timer with the CLR instructions.
   - CLR TR0 and CLR TR1, for timer 0 and timer 1 respectively.
   - After the timer reaches its limit and rolls over.
   - In order to repeat the process, TH and TL must be reloaded with the original value and TF must be reloaded to 0.

To generate a time delay

1. Load the TMOD register indicating which timer is to be used and which timer mode is selected.
2. Load registers TL and TH with initial count value.
3. Start the timer
4. Keep monitoring the timer flag (TF) with the JNB TFx , target to see if it is raised
   - Get out of the loop when TF becomes high
5. Stop the timer
6. Clear the TF flag for the next round
7. Go back to Step 2 to load TH and TL again.

Example 1:

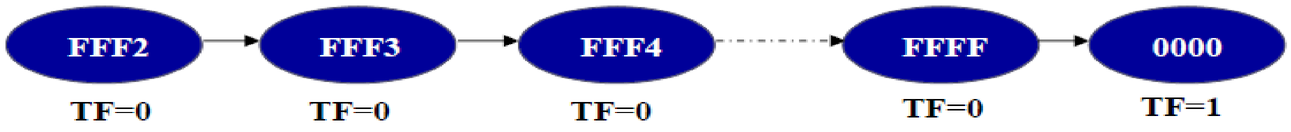
In the following program, we create a square wave of 50% duty cycle (with equal portions high and low) on the P1.5 bit. Timer 0 is used to generate the time delay. Analyze the program. (Nov 2014)

```
            MOV TMOD,#01      ;Timer 0, mode 1(16-bit mode)
HERE:       MOV TL0,#0F2H     ;TL0=F2H, the low byte
            MOV TH0,#0FFH     ;TH0=FFH, the high byte
            CPL P1.5          ;toggle P1.5
            ACALL DELAY
            SJMP HERE
DELAY:      SETB TR0          ;start the timer 0
AGAIN:      JNB TF0,AGAIN     ;monitor timer flag 0 until it rolls over
            CLR TR0           ;stop timer 0
            CLR TF0           ;clear timer 0 flag
            RET
```

In the above program notice the following steps.

1. TMOD is loaded.
2. FFF2H is loaded into TH0-TL0.
3. P1.5 is toggled for the high and low portions of the pulse.
4. The DELAY subroutine using the timer is called.
5. In the DELAY subroutine, timer 0 is started by the SETB TR0 instruction.
6. Timer 0 counts up with the passing of each clock, which is provided by the crystal oscillator.
   - As the timer counts up, it goes through the states of FFF3, FFF4, FFF5, FFF6, and so on until it reaches FFFFH.
   - One more clock rolls it to 0, raising the timer flag (TF0=1). At that point, the JNB instruction falls through.
7. Timer 0 is stopped by the instruction CLR TR0.
   - The DELAY subroutine ends and the process is repeated.

Notice that to repeat the process, we must reload the TL and TH registers, and start the process is repeated.



Example 2:

In Example 1, calculate the amount of time delay in the DELAY subroutine generated by the timer. Assume XTAL = 11.0592 MHz.

Solution:

- ✓ The timer works with a clock frequency of 1/12 of the XTAL frequency, we have 11.0592 MHz / 12 = 921.6 kHz as the timer frequency.
- ✓ As a result, each clock has a period of T =1/921.6kHz,T=1.085µs.
- ✓ In other words, Timer 0 counts up each 1.085µs resulting in delay = number of counts × 1.085µs.
- ✓ The number of counts for the roll over is FFFFH – FFF2H = 0DH (13 decimal).
- ✓ Add one to 13 because of the extra clock needed when it rolls over from FFFF to 0 and raise the TF flag.
- ✓ This gives 14 × 1.085µs = 15.19µs for half the pulse. For the entire period it is T = 2 × 15.19µs = 30.38µs as the time delay generated by the timer.

(a) In hexadecimal

(FFFF – YYXX + 1) ×1.085 µs, where YYXX are TH, TL initial values respectively. Notice that value YYXX are in hex.

(b) In decimal

Convert YYXX values of the TH, TL register to decimal to get a NNNN decimal, then (65536 - NNNN) × 1.085 µs

Example 3:

In Example 1, calculate the frequency of the square wave generated on pin P1.5.

Solution:

- ✓ In the timer delay calculation of Example 1, we did not include the overhead due to instruction in the loop.
- ✓ To get a more accurate timing, we need to add clock cycles due to these instructions in the loop.
- ✓ To do that, we use the machine cycle as shown below.

```
                                            Cycles
HERE:       MOV TL0,#0F2H                   2
            MOV TH0,#0FFH                   2
            CPL P1.5                        1
            ACALL DELAY                     2
            SJMP HERE                       2
DELAY:      SETB TR0                        1
AGAIN:      JNB TF0, AGAIN                  14
            CLR TR0                         1
            CLR TF0                         1
            RET                             2
                                Total       28
```

T = 2 × 28 × 1.085 us = 60.76 µs and F = 16458.2 Hz

Example 4:

Find the delay generated by timer 0 in the following code, using both of the Methods. Do not include the overhead due to instruction.

```
            CLR P2.3            ;Clear P2.3
            MOV TMOD,#01        ;Timer 0, 16-bitmode
HERE:       MOV TL0,#3EH        ;TL0=3Eh, the low byte
            MOV TH0,#0B8H       ;TH0=B8H, the high byte
            SETB P2.3           ;SET high timer 0
            SETB TR0            ;Start the timer 0
AGAIN:      JNB TF0,AGAIN       ;Monitor timer flag 0
            CLR TR0             ;Stop the timer 0
            CLR TF0             ;Clear TF0 for next round
            CLR P2.3
```

Solution:

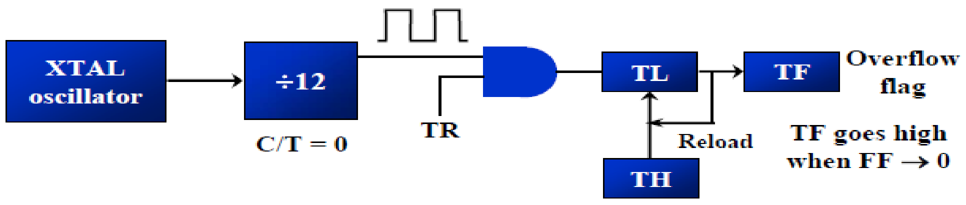
(FFFFH – B83E + 1) = 47C2H = 18370 in decimal and 18370 × 1.085 µs = 19.93145 ms

<u>The following are the characteristics and operations of mode 2:</u>

1. It is an 8-bit timer. It allows only values of 00 to FFH to be loaded into the timer register TH.
2. After TH is loaded with the 8-bit value, the 8051 copies value to TL register.
   - Then the timer must be started.
   - This is done by the instruction SETB TR0 for timer 0 and SETB TR1 for timer 1.
3. After the timer is started, it starts to count up by incrementing the TL register.
   - It counts up until it reaches its limit of FFH
   - When it rolls over from FFH to 00, it sets high the TF (timer flag)
   - When the TL register rolls from FFH to 00 and TF is set to 1.

- TL is reloaded automatically with the original value kept by the TH register.
- To repeat the process, simply clear TF.

4. This makes mode 2 an auto-reload, in contrast with mode 1 in which the programmer has to reload TH and TL



To generate a time delay

1. Load the TMOD value register indicating which timer is to be used, and the timer mode (mode 2) is selected.
2. Load the TH register with the initial count value.
3. Start timer.
4. Keep monitoring the timer flag (TF) with the JNB TFx, target, to see whether it is raised Get out of the loop when TF goes high
5. Clear the TF flag.
6. Go back to Step4, since mode 2 is auto reload.

Example 5:

Assume XTAL = 11.0592 MHz, find the frequency of the square wave generated on pin P1.0.

```
        MOV TMOD,#20H   ;T1/8-bit/auto reload
        MOV TH1,#5      ;TH1 = 5
        SETB TR1        ;start the timer 1
BACK:   JNB TF1,BACK    ;till timer rolls over
        CPL P1.0        ;P1.0 to hi, lo
        CLR TF1         ;clear Timer 1 flag
        SJMP BACK       ;mode 2 is auto-reload
```

Solution:

- ✓ In mode 2, no need to reload TH since it is auto-reload.
- ✓ Now (256 - 05) × 1.085 µs =251 × 1.085 µs = 272.33 µs is the high portion of the pulse.
- ✓ Since it is a 50% duty cycle square wave, the period T is twice.
- ✓ As a result T = 2 × 272.33 µs = 544.67 µs and the frequency = 1.83597 kHz

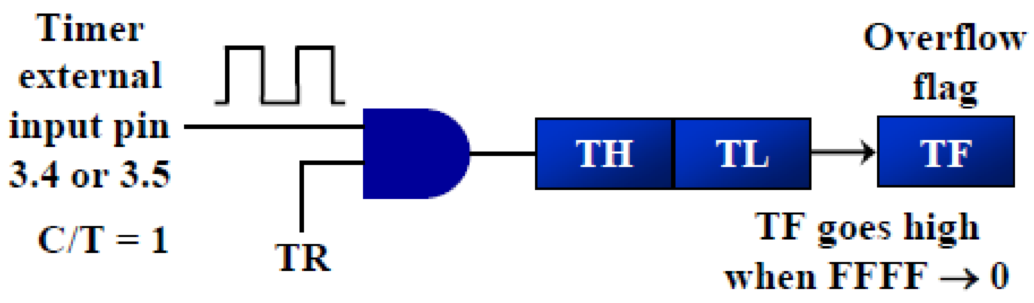
5.2: Timers as counters

Timers can also be used as counters.

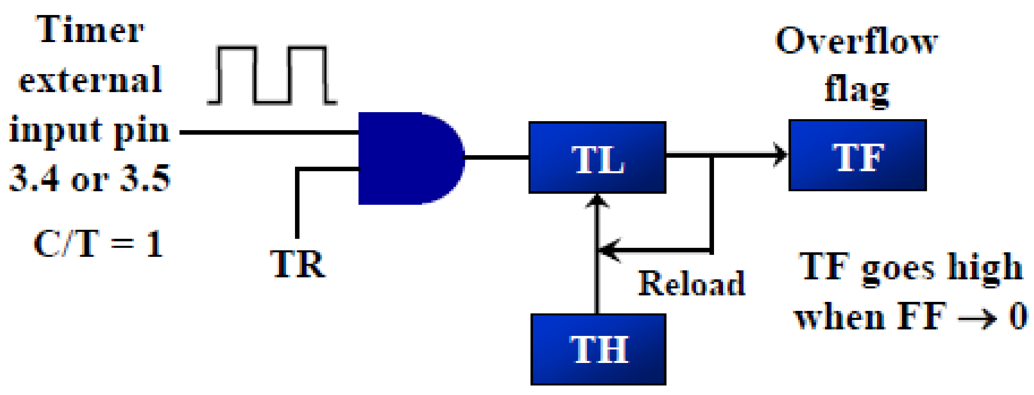
Which are used for counting events happening outside the 8051.

- When it is used as a counter, it is a pulse outside of the 8051 that increments the TH, TL register.
- TMOD and TH, TL registers are the same as in timer concept, except the source of the frequency.
- The C/T bit in the TMOD register decides the source of the clock for the timer
- When C/T = 1, the timer is used as a counter and gets its pulses from outside the 8051.
- The counter counts up as pulses are fed from pins 14 and 15.
- these pins are called T0 (timer 0 input) and T1 (timer 1 input)

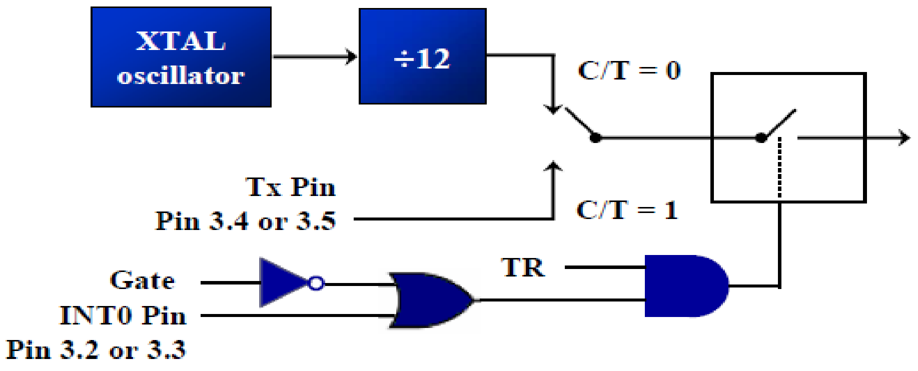
Timer with external input (Mode 1)


Timer external input pin 3.4 or 3.5
C/T = 1
TR
TH
TL
Overflow flag
TF
TF goes high when FFFF → 0

Timer with external input (Mode 2)



- ✓ If GATE = 1, the start and stop of the timer are done externally through pins P3.2 and P3.3 for timers 0 and 1, respectively
- ✓ This hardware allows starting or stopping the timer externally at any time via a simple switch



✓ The frequency for the timer is always 1/12th the frequency of the crystal attached to the 8051.

Port 3 pins used for Timers 0 and 1

Pin	Port Pin	Function	Description
14	P3.4	T0	Timer/counter 0 external input
15	P3.5	T1	Timer/counter 1 external input

Example 6:

Assuming that clock pulses are fed into pin T1, write a program for counter 1 in mode 2 to count the pulses and display the state of the TL1 count on P2, which connects to 8 LEDs.

Solution:

```
                MOV TM0D,#01100000B     ;counter 1, mode 2, C/T=1 external pulses
                MOV TH1,#0              ;clear TH1
                SETB P3.5                ;make T1 input
AGAIN:           SETB TR1                ;start the counter
BACK:           MOV A,TL1               ;get copy of TL
                MOV P2,A                 ;display it on port 2
                JNB TF1,Back            ;keep doing, if TF = 0
                CLR TR1                 ;stop the counter 1
                CLR TF1                 ;make TF=0
                SJMP AGAIN              ;keep doing it
```

✓ Notice in the above program the role of the instruction SETB P3.5.

✓ Since ports are set up for output when the 8051 is powered up.

✓ So, we make P3.5 an input port by making it high.

✓ In other words, we must configure (set high) the T1 pin (pin P3.5) to allow pulses to be fed into it.

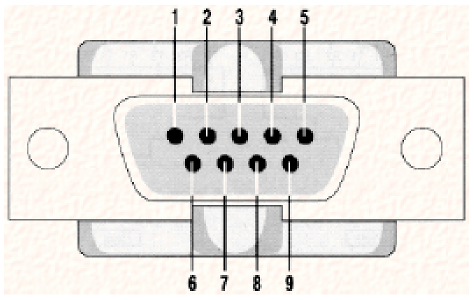
5.3: SERIAL COMMUNICATION

2. Explain the serial programming of 8051 with its associated registers. (May 2014, 2013)(Or)
Explain how to program for sending and receiving data serially using 8051 (April 2010, 2011)
Explain 8051 serial port programming with examples. (May 2016, NOV 2012)
Explain the serial modes of operation of 8051 microcontroller. (May 2007)

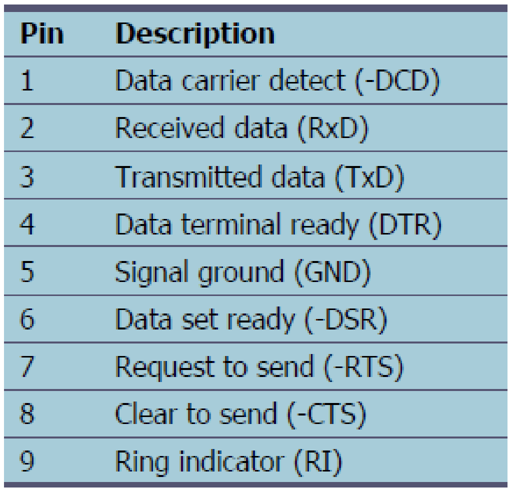
RS232

- ✓ It is an interfacing standard RS232.
- ✓ It was set by the Electronics Industries Association (EIA) in 1960.
- ✓ The standard was set long before the advent of the TTL logic family.
- ✓ Its input and output voltage levels are not TTL compatible.
- ✓ In RS232, a 0 is represented by -3 to -25 V, while a 1 bit is +3 to +25 V.
- ✓ IBM introduced the DB-9 version of the serial I/O standard.

RS232 Connector DB-9



RS232 DB-9 Pins



Pin	Description
1	Data carrier detect (-DCD)
2	Received data (RxD)
3	Transmitted data (TxD)
4	Data terminal ready (DTR)
5	Signal ground (GND)
6	Data set ready (-DSR)
7	Request to send (-RTS)
8	Clear to send (-CTS)
9	Ring indicator (RI)

<u>Handshake signals of MODEM</u>

DTR (data terminal ready)

- When DTR =1, indicate that it is ready for communication.

DSR (data set ready)

- When DSR =1, indicate that it is ready for communication.

RTS (request to send)

- It asserts RTS to signal the modem that it has a byte of data to transmit.

CTS (clear to send)

- It is to receive, it sends out signal CTS,

DCD (data carrier detect)

- The modem asserts signal DCD to inform the DTE that a valid carrier has been detected.

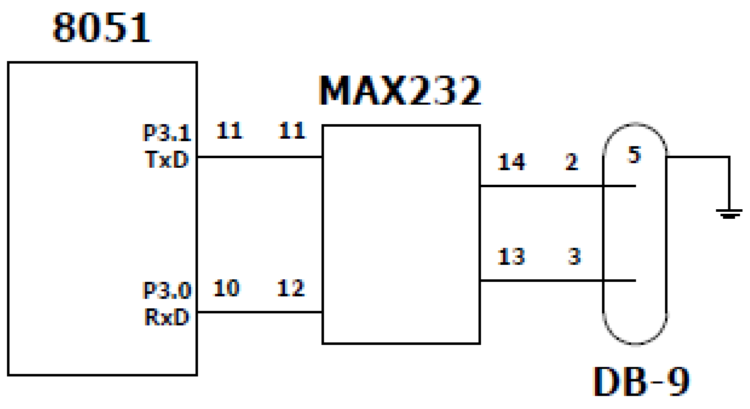
RI (ring indicator)

- An output from the modem and an input to a PC indicates that the telephone is ringing.

MAX232

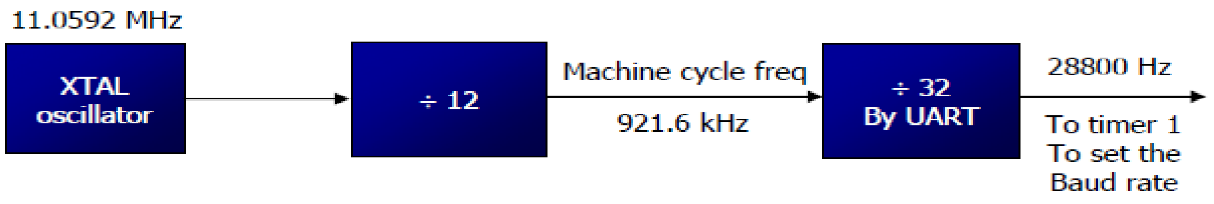
A line driver (MAX232) is required to convert RS232 voltage levels to TTL levels, and vice versa.

- 8051 has two pins that are used specifically for transferring and receiving data serially.
- These two pins are called TxD and RxD and are part of the port 3 (P3.0 and P3.1).
- These pins are TTL compatible.
- They require a line driver to make them RS232 compatible.



Baud rate:

- The baud rates in 8051 are programmable.
- 8051 divides the crystal frequency by 12 to get machine cycle frequency.
- 8051 UART circuitry divides the machine cycle frequency by 32.



- Timer 1 is used to set baud rate using TH1 register

Baud rate	TH1 (decimal)	TH1(Hex)
9600	-3	FD
4800	-6	FA
2400	-12	F4
1200	-24	E8

Explain in detail the serial communication registers of the 8051. (NOV 2009)

SBUF:

- It is an 8-bit register used for serial communication.
- For a byte data to be transferred via the TxD line:
- Byte must be placed in the SBUF register.
- Bytes are framed with the start and stop bits and transferred serially via the TxD line.
- SBUF holds the byte of data when it is received by 8051 RxD line.
- When the bits are received serially via RxD.
- 8051 de-frames byte by eliminating the stop and start bits.

SCON:

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

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

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

SM0, SM1: Serial port mode specifiers

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

In programming the 8051 to transfer character bytes serially

1. TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode 2 (8-bit auto-reload) to set baud rate.
2. The TH1 is loaded with one of the values to set baud rate for serial data transfer.
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits.
4. TR1 is set to 1 to start timer 1
5. TI is cleared by CLR TI instruction.
6. The character byte to be transferred serially is written into SBUF register.
7. The TI flag bit is monitored with the use of instruction JNB TI, xx, to see if the character has been transferred completely.
8. To transfer the next byte, go to step 5.

Write a program for the 8051 to transfer letter "A" serially at 4800 baud, continuously.

Solution:

```
            MOV   TMOD, #20H ;timer 1, mode 2 (auto reload)
            MOV   TH1, #-6   ;4800 baud rate
            MOV   SCON, #50H ;8-bit, 1 stop, REN enabled
            SETB  TR1        ;start timer 1
AGAIN:      MOV   SBUF, #"A" ;letter "A" to trtansfer
HERE:       JNB   TI, HERE   ;wait for the last bit
            CLR   TI         ;clear TI for next char
            SJMP AGAIN       ;keep sending A
```

The steps that 8051 goes through in transmitting a character via TxD

1. The byte character to be transmitted is written into the SBUF register
2. The start bit is transferred
3. The 8-bit character is transferred on bit at a time
4. The stop bit is transferred
   - It is during the transfer of the stop bit that 8051 raises the TI flag, indicating that the last character was transmitted
5. By monitoring the TI flag, we make sure that we are not overloading the SBUF

- If we write another byte into the SBUF before TI is raised, the un-transmitted portion of the previous byte will be lost.

6. After SBUF is loaded with a new byte, the TI flag bit must be forced to 0 by CLR TI in order for this new byte to be transferred

By checking the TI flag bit, we know whether or not the 8051 is ready to transfer another byte

- It must be noted that TI flag bit is raised by 8051 itself when it finishes data transfer
- It must be cleared by the programmer with instruction CLR TI
- If we write a byte into SBUF before the TI flag bit is raised, we risk the loss of a portion of the byte being transferred
- The TI bit can be checked by the instruction JNB TI,xx Using an interrupt.

Write a program for the 8051 to transfer "YES" serially at 9600 baud, 8-bit data, 1 stop bit do this continuously. (May 2006)

Solution:

```
            MOV   TMOD, #20H ;timer 1, mode 2 (auto reload)
            MOV   TH1, #-3   ;9600 baud rate
            MOV   SCON, #50H ;8-bit, 1 stop, REN enabled
            SETB  TR1        ;start timer 1
AGAIN:      MOV   A, # "Y"   ;transfer "Y"
            ACALL TRANS
            MOV   A, # "E"   ;transfer "E"
            ACALL TRANS
            MOV   A, # "S"   ;transfer "S"
            ACALL TRANS
            SJMP  AGAIN      ;keep doing it
;serial data transfer subroutine
TRANS:      MOV   SBUF, A    ;load SBUF
HERE:       JNB   TI, HERE   ;wait for the last bit
            CLR    TI        ;get ready for next byte
            RET
```

In programming the 8051 to receive character bytes serially

1. TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode (8-bit auto-reload) to set baud rate
2. TH1 is loaded to set baud rate
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits
4. TR1 is set to 1 to start timer 1
5. RI is cleared by CLR RI instruction
6. The RI flag bit is monitored with the use of instruction JNB RI, xx to see if an entire character has been received yet
7. When RI is raised, SBUF has the byte, its contents are moved into a safe place.
8. To receive the next character, go to step 5.

Write a program for the 8051 to receive bytes of data serially and put them in P1, set the baud rate at 4800, 8-bit data and 1 stop bit. (NOV 2016)

Solution:

```
            MOV   TMOD, #20H ;timer 1, mode 2 (auto reload)
            MOV   TH1, #-6   ;4800 baud rate
            MOV   SCON, #50H ;8-bit, 1 stop, REN enabled
            SETB  TR1        ;start timer 1
HERE:       JNB    RI, HERE  ;wait for char to come in
            MOV   A, SBUF    ;saving incoming byte in A
            MOV   P1, A      ;send to port 1
            CLR   RI         ;get ready to receive next byte
            SJMP  HERE       ;keep getting data
```

In receiving bit via its RxD pin, 8051 goes through the following steps.

1. It receives the start bit
   - Indicating that the next bit is the first bit of the character byte it is about to receive
2. The 8-bit character is received one bit at time
3. The stop bit is received
   - When receiving the stop bit 8051 makes RI = 1,indicating that an entire character byte has been received.

5. After the SBUF contents are copied into a safe place.

- The RI flag bit must be forced to 0 by CLR RI in order to allow the next received character byte to be placed in SBUF.
- Failure to do this causes loss of the received character.

There are two ways to increase the baud rate of data transfer

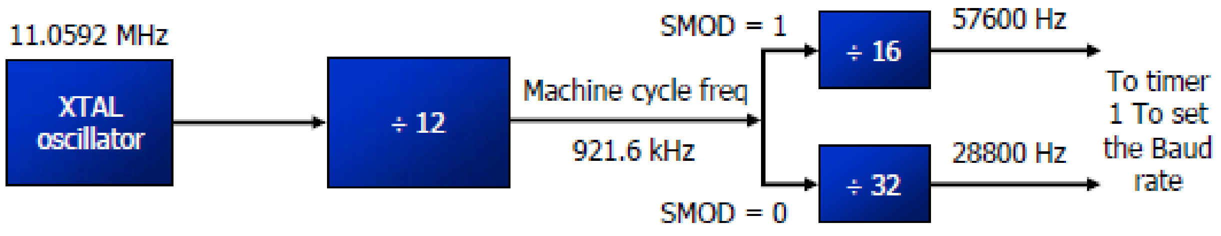
- To use a higher frequency crystal
- To change a bit in the PCON register

PCON

- PCON register is an 8-bit register
- When 8051 is powered up, SMOD is zero.
- We can set it to high by software and thereby double the baud rate.
- GF1, GF0: General flag bits
- PD: Power down mode
- IDL: Ideal mode

SMOD	--	--	--	GF1	GF0	PD	IDL

```
MOV   A,PCON      ;place a copy of PCON in ACC
SETB  ACC.7       ;make D7=1
MOV   PCON,A      ;changing any other bits
```



****

5.4: LCD Interfacing

3. Explain how LCD is used to interface with 8051. (May 2007)

How does one interface a 16 × 2 LCD Display using 8051 Microcontroller? (May2009, May 2010)

Introduction:

- Liquid Crystal displays are created by sandwiching a thin 10-12µm layer of a liquid-crystal fluid between two glass plates.
- A transparent, electrically conductive film or backplane is put on the rear glass sheet.
- Transparent sections of conductive film in the shape of the desired characters are coated on the front glass plate.
- When a voltage is applied between a segment and the backplane, an electric field is created in the region under the segment.
- This electric field changes the transmission of light through the region under the segment film.
- Most LCD's require a voltage of 2 or 3 V between the backplane and a segment to turn on the segment.

There are two types available of LCD

- **Dynamic scattering and field effect.**

Dynamic scattering types of LCD:

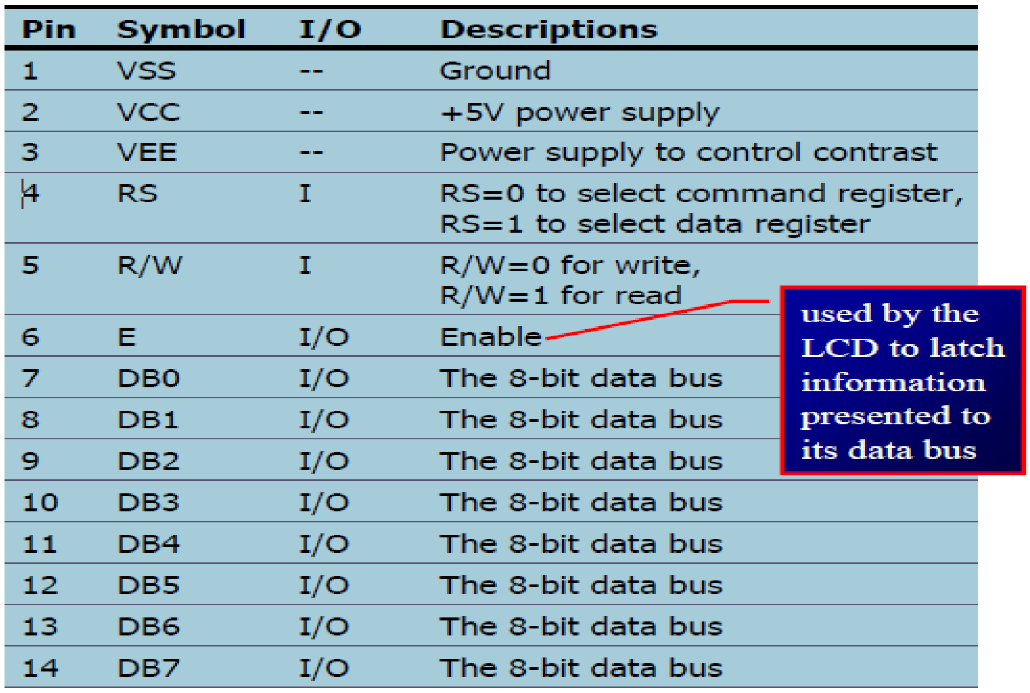
- It scrambles the molecules where the field is present.
- This produces an etched-glass-looking light character on a dark background.

Field-effect types:

- Use polarization to absorb light where the electric field is present.
- This produces dark characters on a silver- gray background.

Advantages of LCD

- LCD is finding widespread use replacing LEDs
- The declining prices of LCD
- The ability to display numbers, characters, and graphics
- Ease of programming for characters and Graphics



Pin	Symbol	I/O	Descriptions
1	VSS	--	Ground
2	VCC	--	+5V power supply
3	VEE	--	Power supply to control contrast
4	RS	I	RS=0 to select command register, RS=1 to select data register
5	R/W	I	R/W=0 for write, R/W=1 for read
6	E	I/O	Enable
7	DB0	I/O	The 8-bit data bus
8	DB1	I/O	The 8-bit data bus
9	DB2	I/O	The 8-bit data bus
10	DB3	I/O	The 8-bit data bus
11	DB4	I/O	The 8-bit data bus
12	DB5	I/O	The 8-bit data bus
13	DB6	I/O	The 8-bit data bus
14	DB7	I/O	The 8-bit data bus

used by the LCD to latch information presented to its data bus

Fig:Pin details of LCD module

Interfacing LCD module

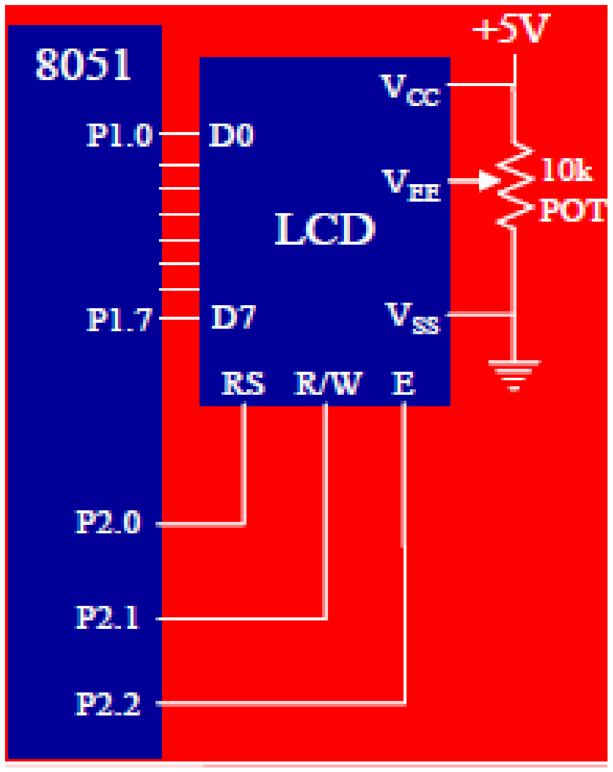
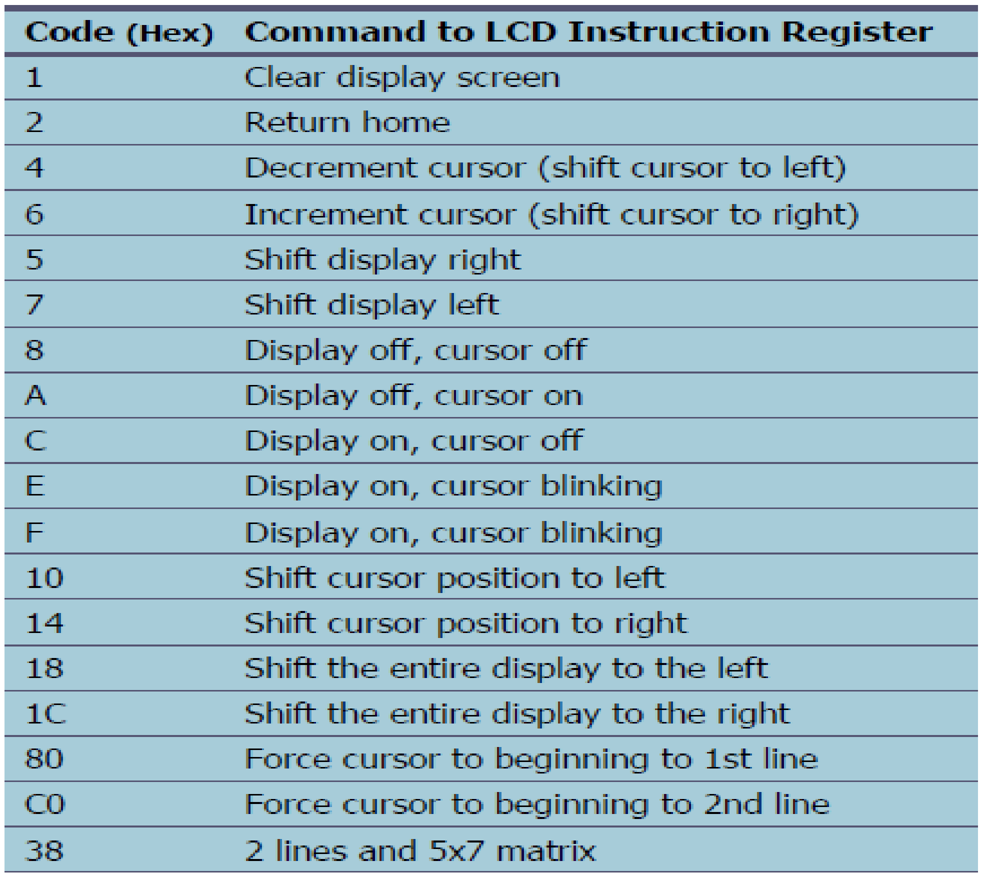


Fig: Interfacing LCD module with controller

LCD Command Codes



Code (Hex)	Command to LCD Instruction Register
1	Clear display screen
2	Return home
4	Decrement cursor (shift cursor to left)
6	Increment cursor (shift cursor to right)
5	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on
C	Display on, cursor off
E	Display on, cursor blinking
F	Display on, cursor blinking
10	Shift cursor position to left
14	Shift cursor position to right
18	Shift the entire display to the left
1C	Shift the entire display to the right
80	Force cursor to beginning to 1st line
C0	Force cursor to beginning to 2nd line
38	2 lines and 5x7 matrix

- To send any of the commands to the LCD, make pin RS=0. For data, make RS=1.
- Then send a high-to-low pulse to the E pin to enable the internal latch of the LCD.
- This is shown in the code below.

;calls a time delay before sending next data/command

```
;P1.0-P1.7 are connected to LCD data pins D0-D7
;P2.0 is connected to RS pin of LCD
;P2.1 is connected to R/W pin of LCD
;P2.2 is connected to E pin of LCD
ORG 0H
MOV A,#38H ;INIT. LCD 2 LINES, 5X7 MATRIX
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time
MOV A,#0EH ;display on, cursor on
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time
MOV A,#01 ;clear LCD
```

```
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time
MOV A,#06H ;shift cursor right
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time
MOV A,#84H ;cursor at line 1, pos. 4
ACALL COMNWRT ;call command subroutine
ACALL DELAY ;give LCD some time
MOV A,#'N' ;display letter N
ACALL DATAWRT ;call display subroutine
ACALL DELAY ;give LCD some time
MOV A,#'O' ;display letter O
ACALL DATAWRT ;call display subroutine
AGAIN: SJMP AGAIN ;stay here
COMNWRT: ;send command to LCD
MOV P1,A ;copy reg A to port 1
CLR P2.0 ;RS=0 for command
CLR P2.1 ;R/W=0 for write
SETB P2.2 ;E=1 for high pulse
ACALL DELAY ;give LCD some time
CLR P2.2 ;E=0 for H-to-L pulse
RET
DATAWRT: ;write data to LCD
MOV P1,A ;copy reg A to port 1
SETB P2.0 ;RS=1 for data
CLR P2.1 ;R/W=0 for write
SETB P2.2 ;E=1 for high pulse
ACALL DELAY ;give LCD some time
CLR P2.2 ;E=0 for H-to-L pulse
RET
DELAY: MOV R3,#50 ;50 or higher for fast CPUs
HERE2: MOV R4,#255 ;R4 = 255
HERE: DJNZ R4,HERE ;stay until R4 becomes 0
DJNZ R3,HERE2
RET
END
```

```
;Check busy flag before sending data, command to LCD
;p1=data pin
;P2.0 connected to RS pin
;P2.1 connected to R/W pin
;P2.2 connected to E pin

ORG 0H
MOV A,#38H ;init. LCD 2 lines ,5x7 matrix
ACALL COMMAND ;issue command
MOV A,#0EH ;LCD on, cursor on
ACALL COMMAND ;issue command
MOV A,#01H ;clear LCD command
ACALL COMMAND ;issue command
MOV A,#06H ;shift cursor right
ACALL COMMAND ;issue command
MOV A,#86H ;cursor: line 1, pos. 6
ACALL COMMAND ;command subroutine
MOV A,#'N' ;display letter N
ACALL DATA_DISPLAY
MOV A,#'O' ;display letter O
ACALL DATA_DISPLAY
HERE:SJMP HERE ;STAY HERE
**********************************************************************************
```

5.5: KEYBOARD INTERFACING

4. With neat circuit diagram explain how a 4 x 4 keypad is interfaced with 8051 microcontroller and write 8051 ALP for keypad scanning. (May 2013, Nov 2015, NOV 2007)

- Keyboards are organized in a matrix of rows and columns
- The CPU accesses both rows and columns through ports.
- Therefore, with two 8-bit ports, an 8 x 8 matrix of keys can be connected to a microprocessor.
- When a key is pressed, a row and a column make a contact.
- Otherwise, there is no connection between rows and columns.
- A 4x4 matrix is connected to two ports.
- The rows are connected to an output port and the columns are connected to an input port

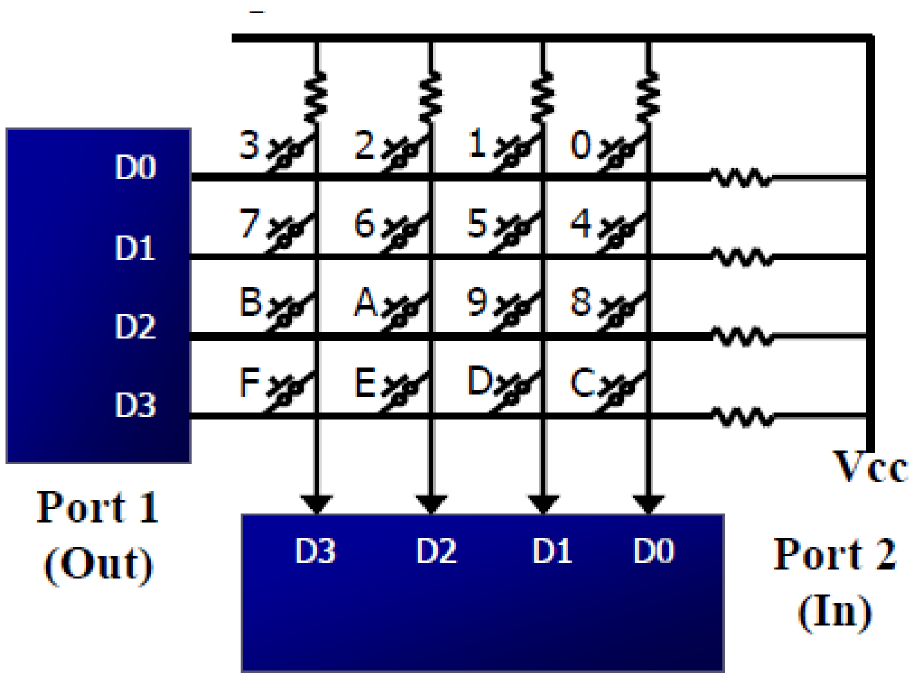
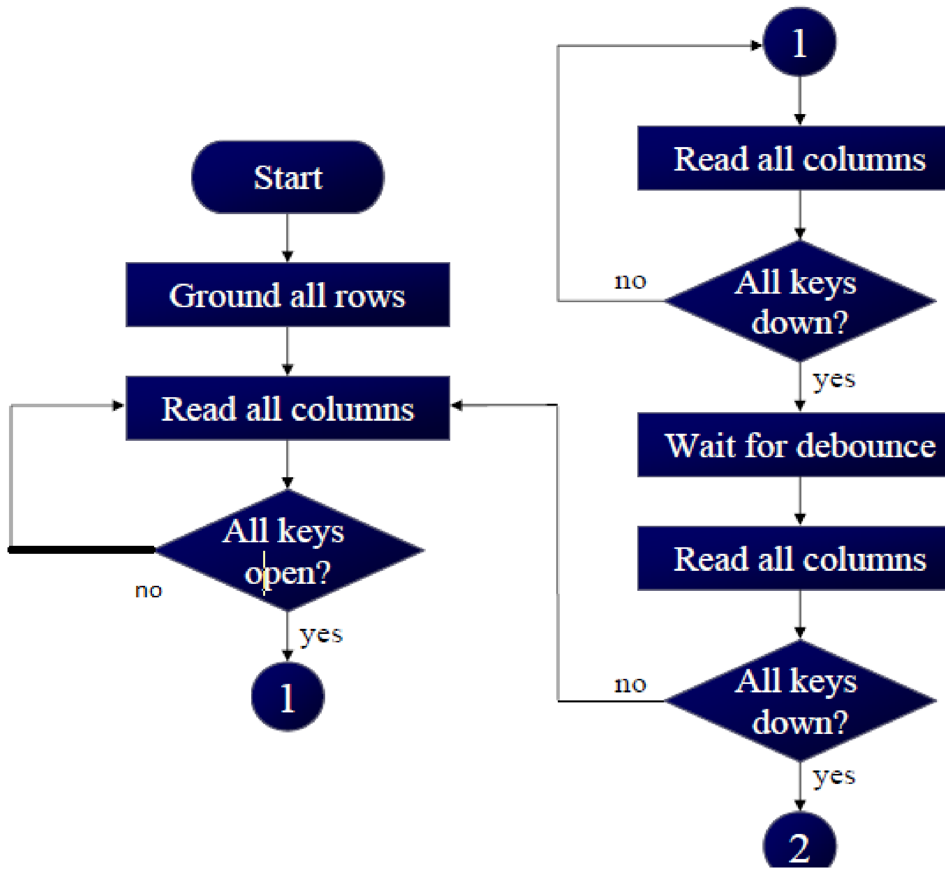
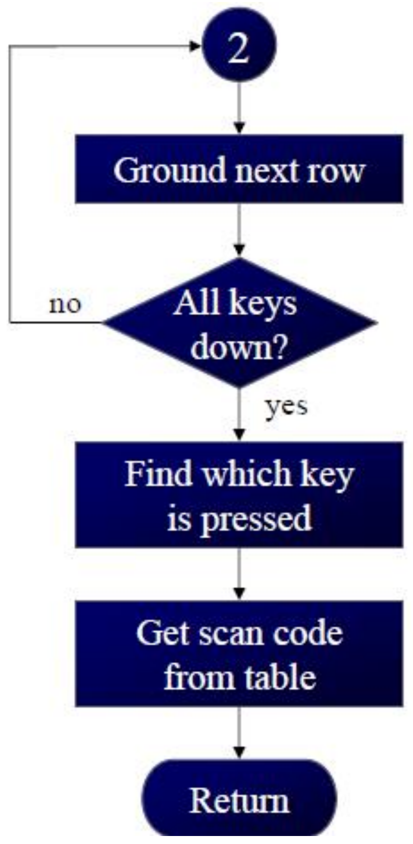


Fig: Model of 4x4 matrix KEYBOARD

- It is the function of the microcontroller to scan the keyboard continuously to detect and identify the key pressed

  <u>To detect a pressed key:</u>
- The microcontroller grounds all rows by providing 0, then it reads the columns
- Data read from columns is D3 – D0 = 1111. No key has been pressed and the process continues till key press is detected
- If one of the column bits has a zero, a key press has occurred.
- For example, if D3 – D0 = 1101, a key in the D1 column has been pressed.
- After detecting a key press, microcontroller will go through the process of identifying the key.
- Starting with the top row, the microcontroller grounds it by providing a low to row D0 only
- It reads the columns, if the data read is all 1s, no key in that row is activated.
- The process is moved to the next row
- It grounds the next row, reads the columns, and checks for any zero
- This process continues until the row is identified.
- After identification of the row in which the key has been pressed
- Find out which column the pressed key belongs to corresponding key is displayed.





Flowchart: Scan the keyboard to detect and identify the key.

5.6: Interfacing 8051 to ADC

5. Explain how to interface an 8-bit ADC with 8051 Microcontroller.(April 2010, May 2008, Nov 2014)

ADCs (analog-to-digital converters) are widely used devices for data acquisition. A physical quantity (temperature, pressure, humidity, and velocity, etc.,) is converted to electrical (voltage, current) signals using a device called a transducer, or sensor.

- We need an analog-to-digital converter to translate the analog signals to digital numbers, so microcontroller can read them.

ADC804 IC is an analog-to-digital converter

- It works with +5 volts and has a resolution of 8 bits.
- Conversion time is defined as the time it takes the ADC to convert the analog input to a digital(binary) number.
- In ADC804 conversion time varies depending on the clocking signals applied to CLK R and CLK IN pins, but it cannot be faster than 110 µs.

CLK IN and CLK R

- CLK IN is an input pin connected to an external clock source
- To use the internal clock generator (also called self-clocking), CLK IN and CLK R pins are connected to a capacitor and a resistor, and the clock frequency is determined by

f = 1/1.1 RC

- Typical values are R = 10K ohms and C = 150 Pf. We get f = 606 kHz and the conversion time is 110 µs

D0-D7

- The digital data output pins
- These are tri-state buffered
- The converted data is accessed only when CS = 0 and RD is forced low
- To calculate the output voltage, use the following formula

$$D_{out} = \frac{V_{in}}{step\ size}$$

Dout = digital data output (in decimal),

Vin = analog voltage and step size (resolution) is the smallest change

Vref/2

- It is used for the reference voltage
- If this pin is open (not connected), the analog input voltage is in the range of 0 to 5 volts (the same as the Vcc pin)
- If the analog input range needs to be 0 to 4 volts, Vref/2 is connected to 2 volts.

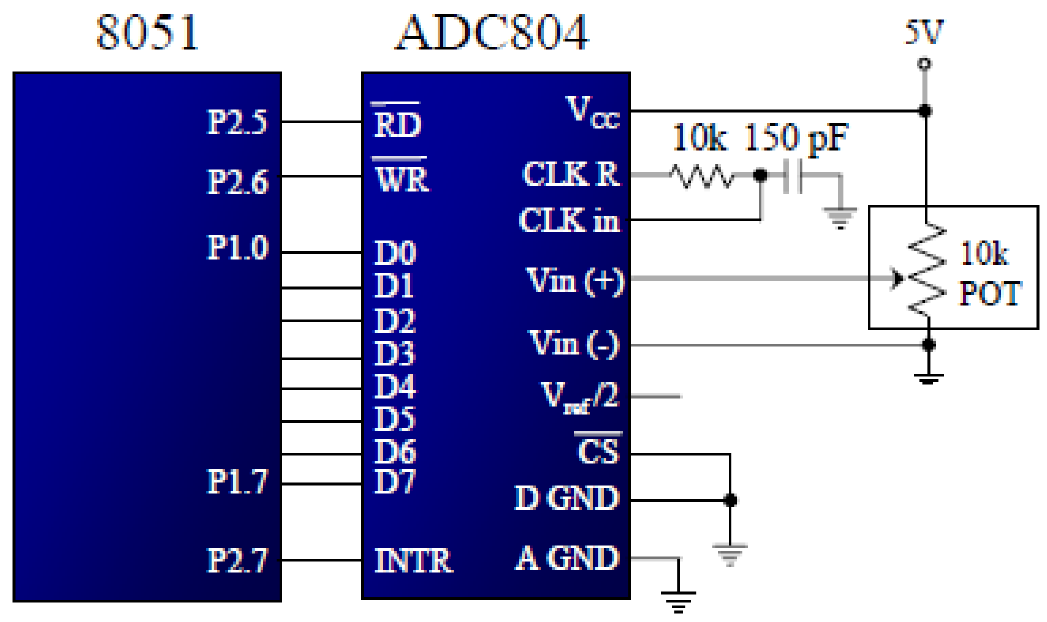
Analog ground and digital ground

- Analog ground is connected to the ground of the analog Vin
- Digital ground is connected to the ground of the Vcc pin
- To isolate the analog V_{in} signal from transient voltages caused by digital switching of the output D0 – D7. This contributes to the accuracy of the digital data output.

The following steps must be followed for data conversion by the ADC804 chip

- Make CS = 0 and send a low-to-high pulse to pin WR to start conversion
- Keep monitoring the INTR pin
- If INTR is low, the conversion is finished
- If the INTR is high, keep polling until it goes low
- After the INTR has become low, we make CS = 0 and send a high-to-low pulse to the RD pin to get the data out of the ADC804.

8051 Connection to ADC804 with Self-Clocking



Examine the ADC804 connection to the 8051 in Figure. Write a program to monitor the INTR pin and bring an analog input into register A. Then call a hex-to ACSII conversion and data display subroutines. Do this continuously. (NOV 2007)

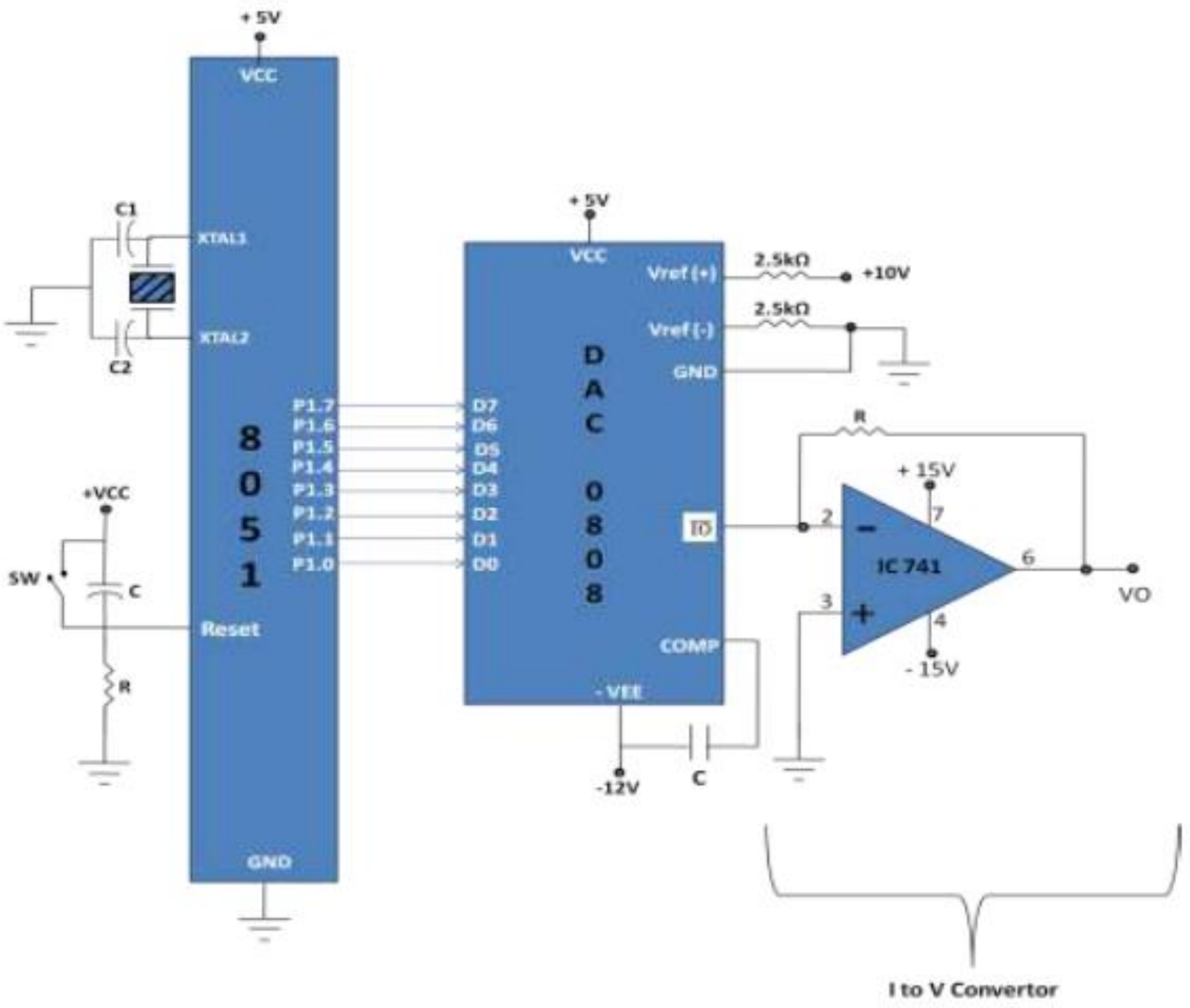
```
;p2.6=WR (start conversion needs to L-to-H pulse)
;p2.7 When low, end-of-conversion)
;p2.5=RD (a H-to-L will read the data from ADC chip)
;p1.0 – P1.7= D0 - D7 of the ADC804
        MOV P1,#0FFH            ;make P1 = input
BACK:   CLR P2.6                ;WR = 0
        SETB P2.6               ;WR = 1 L-to-H to start conversion
HERE:   JB P2.7,HERE            ;wait for end of conversion
        CLR P2.5                ;conversion finished, enable RD
        MOV A,P1                ;read the data
        ACALL CONVERSION        ;hex-to-ASCII conversion
        ACALL DATA_DISPLAY      ;display the data
        SETB p2.5               ;make RD=1 for next round
        SJMP BACK
```

5.7: DAC Interfacing with 8051

6. Explain the DAC interface with 8051. (May 2008)

Develop 8051 based system having 8Kbyte RAM to generate the triangular wave using DAC. (April 2017)

- Microcontroller is used in wide variety of applications like for measuring and control of physical quantity like temperature, pressure, speed, distance, etc.
- Microcontroller generates output which is in digital form.
- But the controlling system requires analog signal, so use DAC which converts digital data into equivalent analog voltage.
- In the figure shown, we use 8-bit DAC 0808. This IC converts digital data into equivalent analog Current.
- Hence we require an I to V converter to convert this current into equivalent voltage.



- According to theory of DAC Equivalent analog output is given as:

Ex:

1. IF data =00H [00000000], Vref= 10v

$$V0= 10\left[\frac{0}{2}+\frac{0}{4}+\frac{0}{8}+\frac{0}{16}+\frac{0}{32}+\frac{0}{64}+\frac{0}{128}+\frac{0}{256}\right]$$

Therefore, V0= 0 Volts.

2. If data is 80H [10000000], Vref= 10VTherefore, V0= 5 Volts.

$$V0=10\left[\frac{1}{2}+\frac{0}{4}+\frac{0}{8}+\frac{0}{16}+\frac{0}{32}+\frac{0}{64}+\frac{0}{128}+\frac{0}{256}\right]$$

Different Analog output voltages for different Digital signals are given as:

Data	Output Voltage
00H	0V
80H	5V
FFH	10V

Program to generate square wave:

Label	Mnemonics		Comments
	Opcode	Operand	
LOOP2 :	MOV	A,# 00	; Set Logic 0 level
	MOV	P1, AL	
	ACALL	Delay	;Generate timing delay
	MOV	A,#FF	;Set logic 1 level
	MOV	P1, AL	
	ACALL	Delay	; Generate timing delay
	SJMP	LOOP2	:Repeat to generates Square Wave
Delay:	MOV	R_O, #F0	:Delay Program
LOOP3:	DJNZ	$R_{O,}$ LOOP3	
	RET		

Program to generate Triangular wave:

Label	Mnemonics		Comments
	Opcode	Operand	
LOOP3 :	MOV	B,# 00	;Set logic 0
LOOP1:	MOV	A, B	;copy logic 0
	MOV	P1, A	
	INC	B	; Increment
	JNZ	LOOP1	; If ZF=0, jump to next
	MOV	B, #FF	Set logic 1
LOOP2:	MOV	A, B	;copy logic 1
	MOV	P1, A	
	DJNZ	B, LOOP2	; Decrement &jump to next
	SJMP	LOOP3	;Repeat

5.8: Temperature Sensors

7. Explain the interfacing of temperature sensor with 8051

- A thermistor responds to temperature change by changing resistance, but its response is not linear.

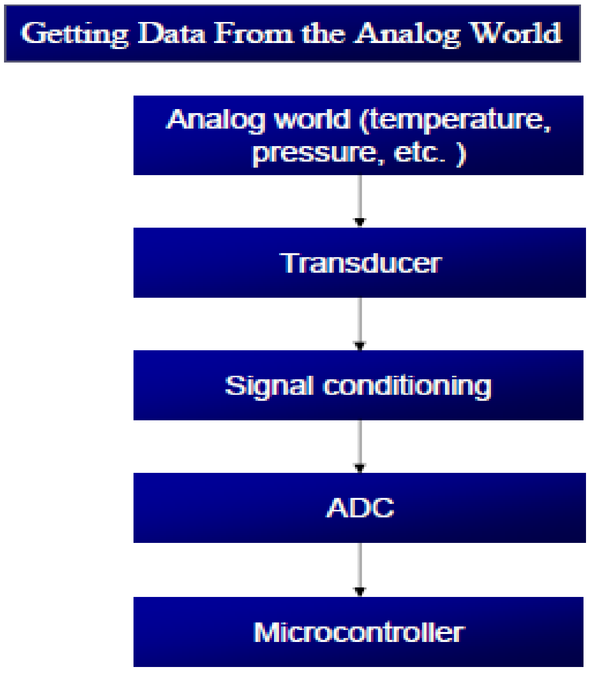
Temperature (C)	Tf (K ohms)
0	29.49
25	10
50	3.893
75	1.7
100	0.817

Signal conditioning is a widely used term in the world of data acquisition

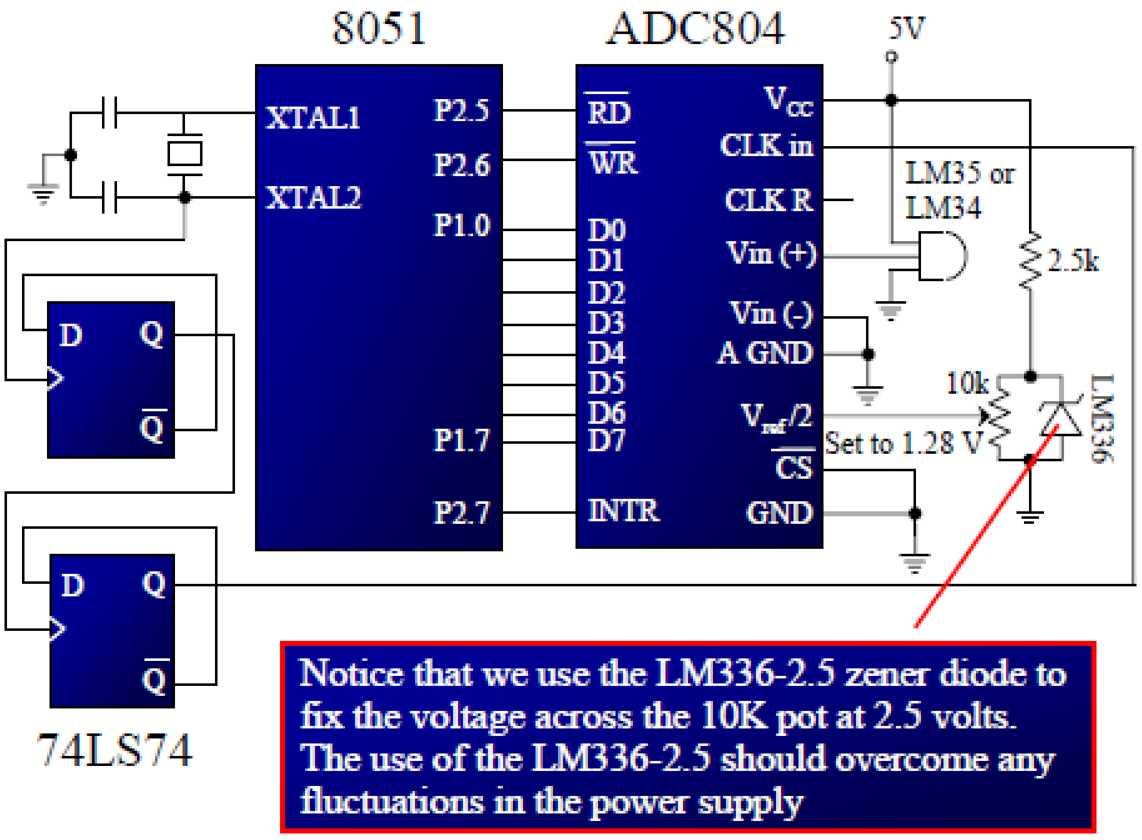
- It is the conversion of the signals (voltage, current, charge, capacitance, and resistance) produced by transducers to voltage, which is sent to the input of an A to-D converter.

Signal conditioning can be a current-to voltage conversion or a signal amplification

- The thermistor changes resistance with temperature, while the change of resistance must be translated into voltage in order to be of any use to an ADC.
- The sensors of the LM34/LM35 series are precision integrated-circuit temperature sensors whose output voltage is linearly proportional to the Fahrenheit/Celsius temperature.
- The LM34/LM35 requires no external calibration since it is inherently calibrated.
- It outputs 10 mV for each degree of Fahrenheit/Celsius temperature.
- The LM336 is used to overcome any power fluctuation in the power supply.


Getting Data From the Analog World
Analog world (temperature, pressure, etc.)
Transducer
Signal conditioning
ADC
Microcontroller

8051 Connection to ADC804 and Temperature Sensor



Notice that we use the LM336-2.5 zener diode to fix the voltage across the 10K pot at 2.5 volts. The use of the LM336-2.5 should overcome any fluctuations in the power supply

5.9: Interfacing to external memory

8. Describe the external memory and its interfacing with 8051. (NOV 2009, May 2008)

Write short notes on memory addressing. (Nov /Dec 2007)

How a program memory and a data memory are interfaced with 8051. (NOV 2007)

5.9.1: Connection to External Program ROM:

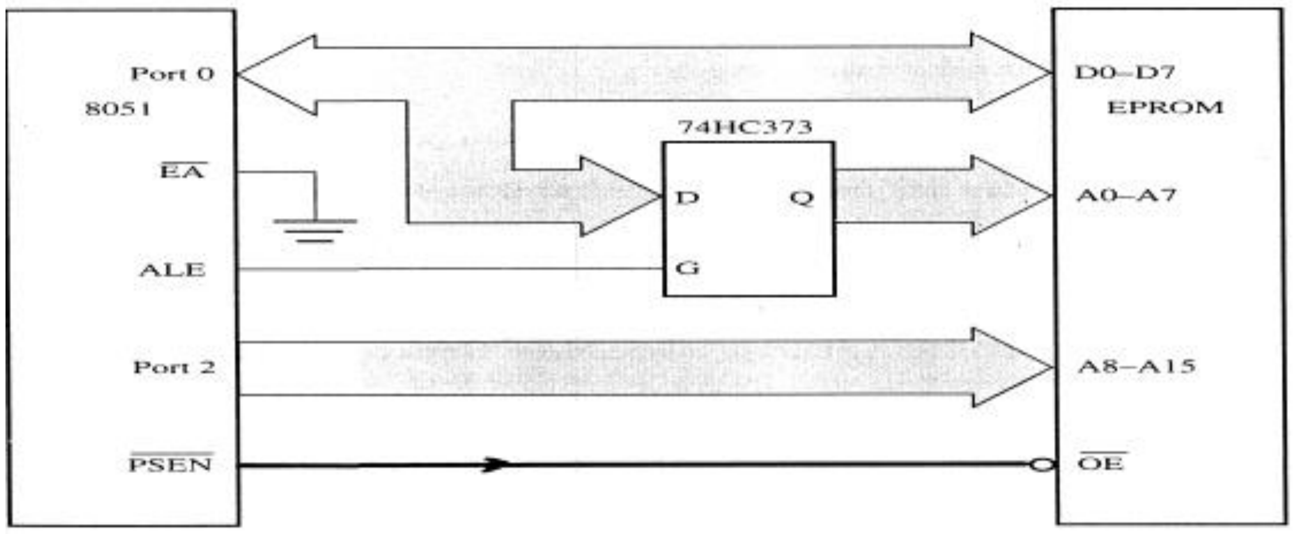
Explain how to interface ROM with the 8051. (NOV 2009)

We use RD to connect the 8051 to external ROM containing data.

For the ROM containing the program code, PSEN is used to fetch the code.

- 64K bytes are set aside for program code
- Program space is accessed using the program counter (PC) to locate and fetch instructions
- In some example we placed data in the code space and use the instruction MOVC A,@A+DPTR to get data, where C stands for code

- The other 64K bytes are set aside for data.
- The data memory space is accessed using the DPTR register and an instruction called MOVX , where X stands for external – The data memory space must be implemented externally.



- In the 8031/51, port 0 and port 2 provide the 16-bit address to access external memory.
- P0 provides the lower 8 bit address A0 – A7, and P2 provides the upper 8 bit address A8 – A15
- P0 is also used to provide the 8-bit data bus D0 – D7.
- P0.0 – P0.7 are used for both the address and data paths using address/data multiplexing.

EA (External access)

- Connect the **EA** pin to **Vcc** to indicate that the program code is stored in the microcontroller’s **on-chip ROM**.
- To indicate that the program code is stored in **external ROM**, this pin must be connected to **GND.**

ALE (**address latch enable)** pin is an output pin for 8051

- ALE = 0, P0 is used for data path.
- ALE = 1, P0 is used for address path.
- To extract the address from the P0 pins we connect P0 to a 74LS373 and use the ALE pin to latch the address.

PSEN (program store enable) signal is an output signal for the 8051 microcontroller and must be connected to the OE pin of a ROM containing the program code.

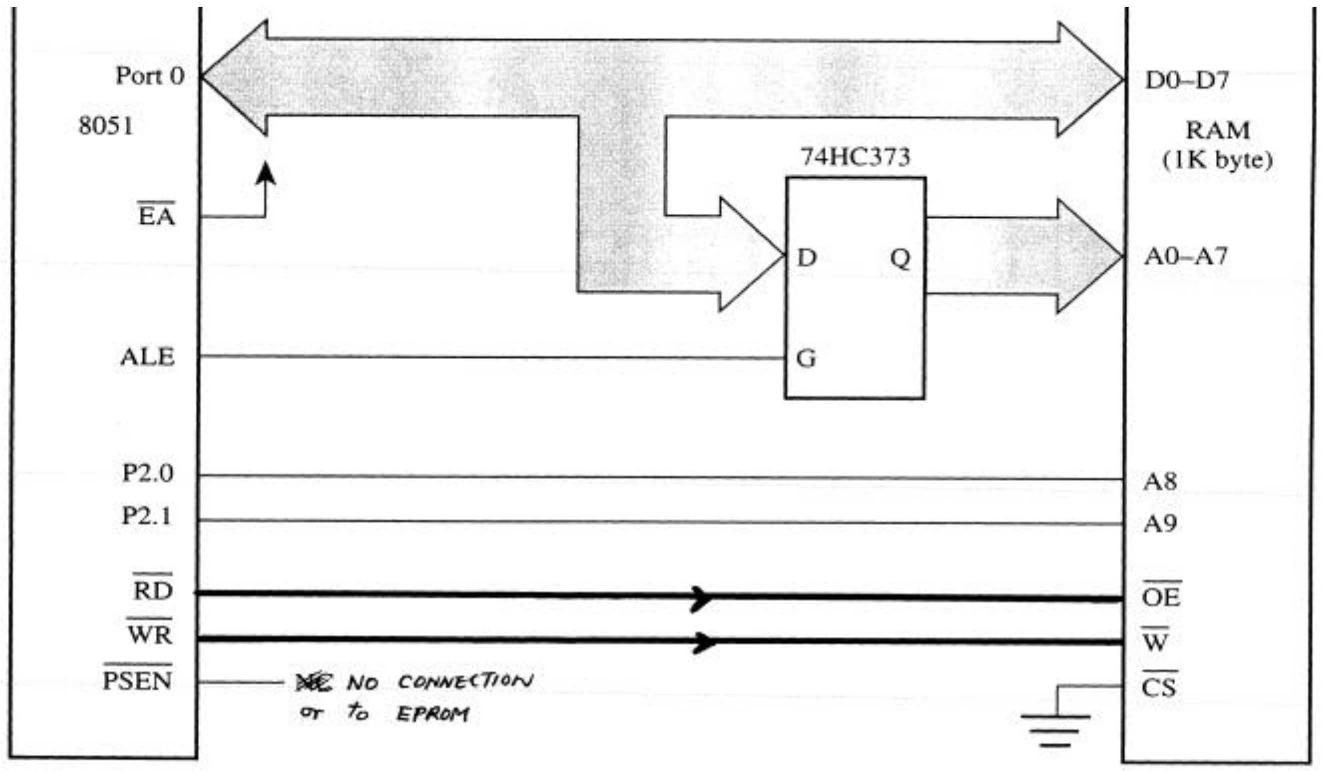
5.9.2: Connection to External Data RAM

Interfacing external RAM with 8051

Write a brief note on external data move operations in 8051. (May 2016)

MOVX is a widely used instruction allowing access to external data memory space

- To bring externally stored data into the CPU, we use the instruction MOVX A,@DPTR
- To connect the 8051 to an external SRAM, we must use both RD (P3.7) and WR (P3.6)
- In writing data to external data RAM, we use the instruction MOVX @DPTR,A



- In the 8031/51, port 0 and port 2 provide the 16-bit address to access external memory.
- P0 provides the lower 8 bit address A0 – A7, and P2 provides the upper 8 bit address A8 – A15
- P0 is also used to provide the 8-bit data bus D0 – D7.
- P0.0 – P0.7 are used for both the address and data paths using address/data multiplexing.
- Data space is accessed using the program counter (PC) to locate, read and write data.

ALE (**address latch enable)** pin is an output pin for 8051

- ALE = 0, P0 is used for data path.
- ALE = 1, P0 is used for address path.
- To extract the address from the P0 pins we connect P0 to a 74LS373 and use the ALE pin to latch the address.
- EA & PSEN are not connected

****

5.10: Interfacing stepper motor with 8051

9. Demonstrate the interfacing of the stepper motor with 8051 and explain its interfacing diagram and develop to rotate the motor in clock wise direction (April 2017, NOV 2016, May 2016, May 2010, May 2009, May 2008, May 2007, Nov 2015, 2014, 2013, 2011, 2010 & May 2013)
Describe in detail the microcontroller based system design with an example. (NOV 2102)

STEPPER MOTOR

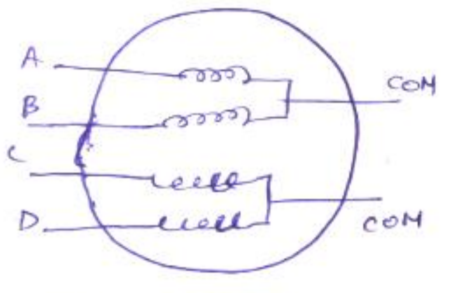
- A stepper motor is a brushless, synchronous electric motor that converts digital pulses into mechanical shaft rotation.
- Every revolution of the stepper motor is divided into a discrete number of steps, and the motor must be sent a separate pulse for each step.

Applications:

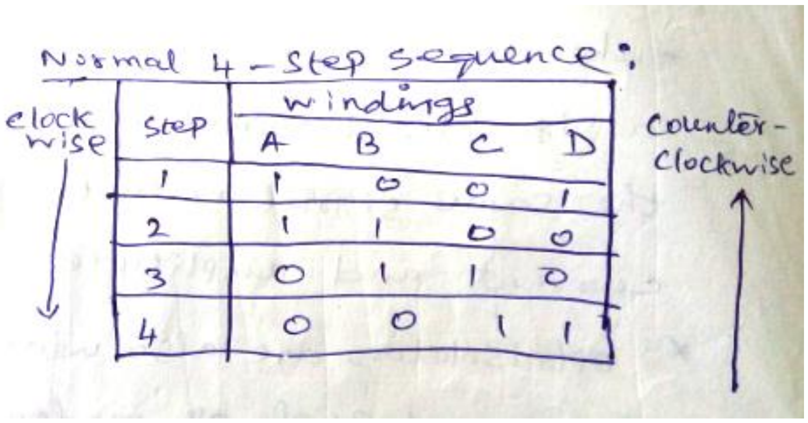
Stepper motors can be used for position control in disk drive, dot matrix printers, robotics etc.

Construction:

- The stepper motor has four stator windings that are paired with a centre tapped common.
- This type of a stepper motor is commonly referred to as a four phase unipolar stepper motor.
- The center tap allows a change of current direction in each of two coils when a winding is grounded, there by resulting in a polarity change of the stator.



Stator windings configuration


clock wise
Counter-Clockwise

Normal 4-Step sequence:

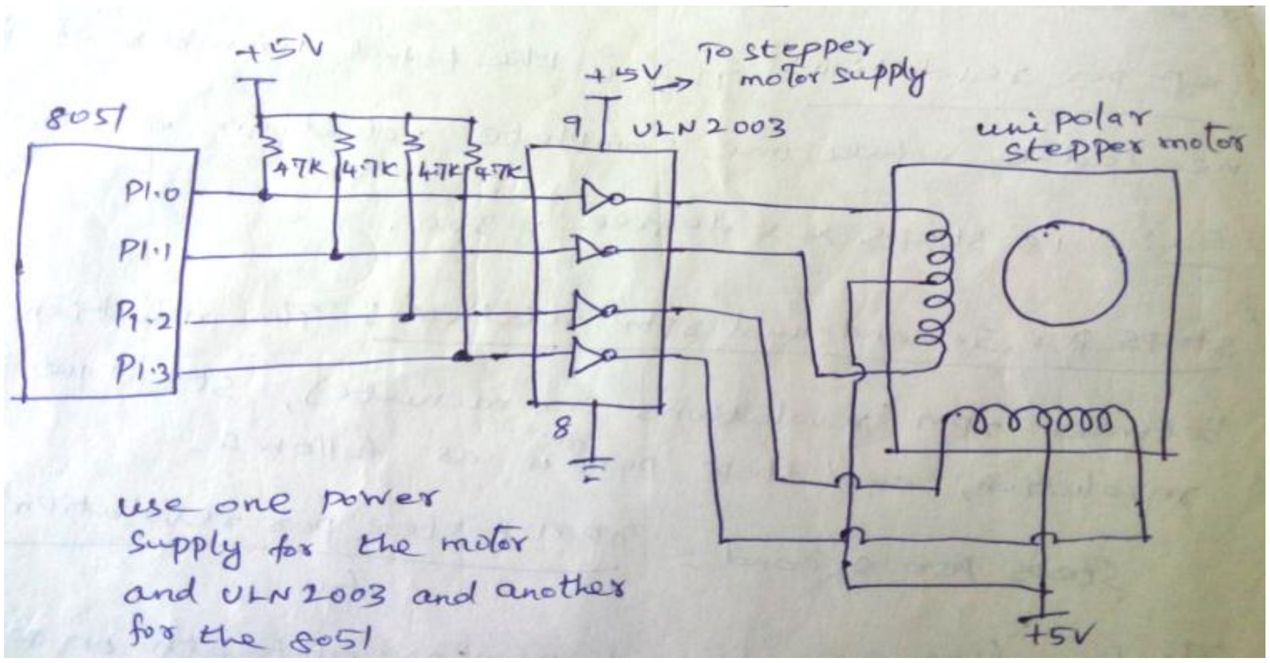
Step	A	B	C	D
	windings			
1	1	0	0	1
2	1	1	0	0
3	0	1	1	0
4	0	0	1	1

- It has total of 6 leads: 4 leads representing the four stator windings and 2 commons for the center tapped leads.
- The stepper motor shaft moves in a fixed repeatable increment, which allows one to move it to a accurate position.
- As the direction of the current is changed, the polarity is also changed causing the reverse motion of the rotor.
- As the sequence of power is applied to each stator winding, the rotor will rotate.

- We can start with any of the sequences, once started, and must continue in the proper order.
- Step angle of the stepper motor is defined as the minimum degree of rotation associated with a single step.
- To calculate step angle, simply divide 360 by number of steps a motor takes to complete one revolution.
- Motor rotating in full mode takes 4 steps to complete a revolution ,so step angle can be calculated as step angle **ø = 360° / 4 =90.**
- By knowing the stepper motor step angle helps to move the motor in correct angular position**.**

8051 Microcontroller connection to Stepper motor:

- The stepper motor is connected with Microcontroller output port pins through a ULN2003 driver.
- The Microcontroller's pin can provide a maximum of 1-2mA current.
- Place a driver, such as the ULN 2003/ULN2803or a power transistor between the Microcontroller and the coil to energize the stator.



(Connection diagram between 8051 and Stepper motor)

- So when the microcontroller is giving pulses with particular frequency, the motor is rotated in clockwise or anticlockwise.

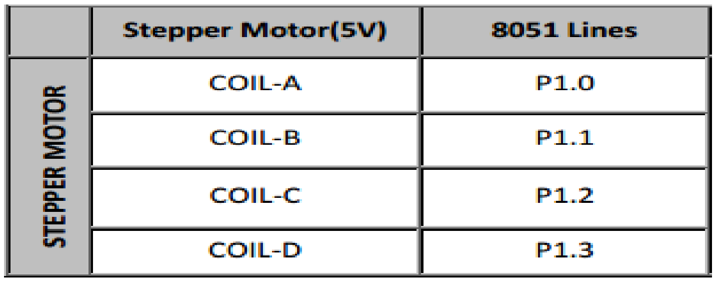
Program to interface Stepper motor with 8051:

The following steps show the 8051 connection to the stepper motor and its programming.

1. The common wires are connected to the positive side of the motor's power supply (+5V is sufficient).

2. Four stator windings are controlled by four bits of the 8051 port (P1.0 to P1.3). Driver ULN2003 used to energize the stator.

Pin assignment with 8051:



	Stepper Motor(5V)	8051 Lines
STEPPER MOTOR	COIL-A	P1.0
	COIL-B	P1.1
	COIL-C	P1.2
	COIL-D	P1.3

By giving the excitation as indicated above through port 1 we can rotate stepper motor in clockwise or anti clock wise direction.

Assembly Language Program:

To rotate motor in the clock wise direction (Forward direction):

```
            MOV A,#66H              ; Copy step value to Accumulator
BACK:       MOV P1,A                ; Copy step value from Accumulator to Port 1
            RR A                    ; Rotate right to get consecutive step values
            ACALL DELAY             ; Call delay program
            SJMP BACK               ; repeat for next sequence.
DELAY:      MOV R2,#F0
DELAY2:     MOV R3,#FF
DELAY1:     DJNZ R3,# DELAY1
            DJNZ R2,# DELAY2
            RET                     ; Return to main program
```

To rotate motor in the counter (Anti) clock wise direction (Reverse direction):

```
            MOV A,#66H              ; Copy step value to Accumulator
BACK:       MOV P1,A                ; Copy step value from Accumulator to Port 1
            RL A                    ; Rotate left to get consecutive step values
            ACALL DELAY             ; Call delay program
            SJMP BACK               ; repeat for next sequence.
DELAY:      MOV R2,#F0
DELAY2:     MOV R3,#FF
DELAY1:     DJNZ R3,# DELAY1
            DJNZ R2,# DELAY2
            RET                     ; Return to main program
```

To rotate motor in both forward and reverse direction:

```
          MOV A,#66H          ; Copy step value to Accumulator
FORWARD:  MOV P1,A            ; Copy step value from Accumulator to Port 1
          RR A                ; Rotate right to get consecutive step values
          ACALL DELAY         ; Call delay program
          MOV R0,#04
          DJNZ R0,FORWARD

REVERSE:  MOV P1,A            ; Copy step value from Accumulator to Port 1
          RL A                ; Rotate left to get consecutive step values
          ACALL DELAY         ; Call delay program
          MOV R0,#04
          DJNZ R0,REVERSE
          SJMP FORWARD        ; repeat for next sequence.

DELAY:    MOV R2,#F0          :Delay program
DELAY2:   MOV R3,#FF
DELAY1:   DJNZ R3,# DELAY1
          DJNZ R2,# DELAY2
          RET                 ; Return to main program
```

The PIC microcontroller was developed by General Instruments in 1975. PIC was developed when Microelectronics Division of General Instruments was testing its 16-bit CPU CP1600.
Although the CP1600 was a good CPU but it had low I/O performance. The PIC controller was used to offload the I/O the tasks from CPU to improve the overall performance of the system.

In 1985, General Instruments converted their Microelectronics Division to Microchip Technology.
PIC stands for Peripheral Interface Controller. The General Instruments used the acronyms Programmable Interface Controller and Programmable Intelligent Computer for the initial PICs (PIC1640 and PIC1650).

In 1993, Microchip Technology launched the 8-bit PIC16C84 with EEPROM which could be programmed using serial programming method.
The improved version of PIC16C84 with flash memory (PIC18F84 and PIC18F84A) hit the market in 1998.
Development:
Since 1998, Microchip Technology continuously developed new high performance microcontrollers with new complex architecture and enhanced in-built peripherals. PIC microcontroller is based on Harvard architecture. At present PIC microcontrollers are widely used for industrial purpose due to its high performance ability at low power consumption. It is also very famous among hobbyists due to moderate

cost and easy availability of its supporting software and hardware tools like compilers, simulators, debuggers etc. The 8-bit PIC microcontroller is divided into following four categories on the basis of internal architecture:

1. Base Line PIC
2. Mid-Range PIC
3. Enhanced Mid-Range PIC
4. PIC18

1. Base Line PIC

Base Line PICs are the least complex PIC microcontrollers. These microcontrollers work on 12-bit instruction architecture which means that the word size of instruction sets are of 12 bits for these controllers. These are smallest and cheapest PICs, available with 6 to 40 pin packaging. The small size and low cost of Base Line PIC replaced the traditional ICs like 555, logic gates etc. in industries.

2. Mid-Range PIC

Mid-Range PICs are based on 14-bit instruction architecture and are able to work up to 20 MHz speed. These controllers are available with 8 to 64 pin packaging. These microcontrollers are available with different peripherals like ADC, PWM, Op-Amps and different communication protocols like USART, SPI, I2C (TWI), etc. which make them widely usable microcontrollers not only for industry but for hobbyists as well.

3. Enhanced Mid-Range PIC

These controllers are enhanced version of Mid-Range core. This range of controllers provides additional performance, greater flash memory and high speed at very low power consumption. This range of PIC also includes multiple peripherals and supports protocols like USART, SPI, I2C and so on.

4. PIC18

PIC18 range is based on 16-bit instruction architecture incorporating advanced RISC architecture which makes it highest performer among the all 8-bit PIC families. The PIC18 range is integrated with new age communication protocols like USB, CAN, LIN, Ethernet (TCP/IP protocol) to communicate with local and/or internet based networks. This range also supports the connectivity of Human Interface Devices like touch panels etc.
MIPS stand for Millions of Instructions per Second

Besides 8-bit microcontrollers, Microchip also manufactures 16-bit and 32-bit microcontrollers. Recently Microchip developed XLP (Extreme Low Power) series microcontrollers which are based on NanoWatt technology. These controllers draw current in order of nanoamperes(nA).
PIC microcontrollers are also available with extended voltage ranges which reduce the frequency range. The operating voltage range of these PICs is 2.0-6.0 volts. The letter 'L' is included in controller's name to denote extended voltage range controllers. For example, PIC16LFxxx (Operating voltage 2.0-6.0 volts).

The following section covers the PIC architecture in further detail. PIC18 series has been selected for the study because it is enhanced series of 8-bit PIC microcontroller. In this series, PIC18F4550 has been chosen to describe the architecture and other features due its moderate complexity.

Architecture:

PIC microcontrollers are based on advanced RISC architecture. RISC stands for Reduced Instruction Set Computing. In this architecture, the instruction set of hardware gets reduced which increases the execution rate (speed) of system.

PIC microcontrollers follow Harvard architecture for internal data transfer. In Harvard architecture there are two separate memories for program and data. These two memories are accessed through different buses for data communication between memories and CPU core. This architecture improves the speed of system over Von Neumann architecture in which program and data are fetched from the same memory using the same bus. PIC18 series controllers are based on 16-bit instruction set.

The question may arise that if PIC18 are called 8-bit microcontrollers, then what about them being based on 16-bit instructions set. 'PIC18 is an 8-bit microcontroller' this statement means that the CPU core can receive/transmit or process a maximum of 8-bit data at a time. On the other hand the statement 'PIC18 microcontrollers are based on 16-bit instruction set' means that the assembly instruction sets are of 16-bit. The data memory is interfaced with 8-bit bus and program memory is interfaced with 16-bit bus as depicted in the following figure

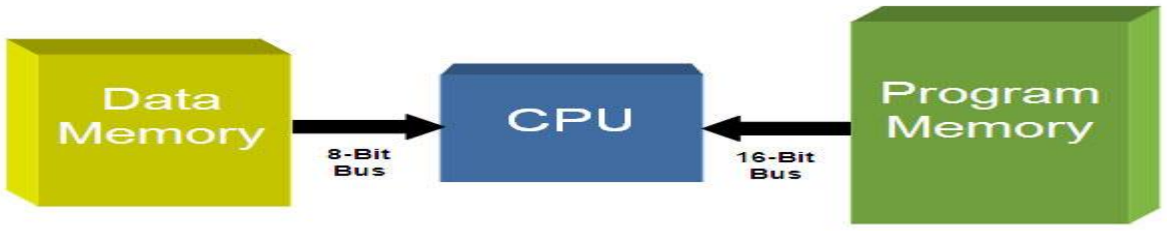
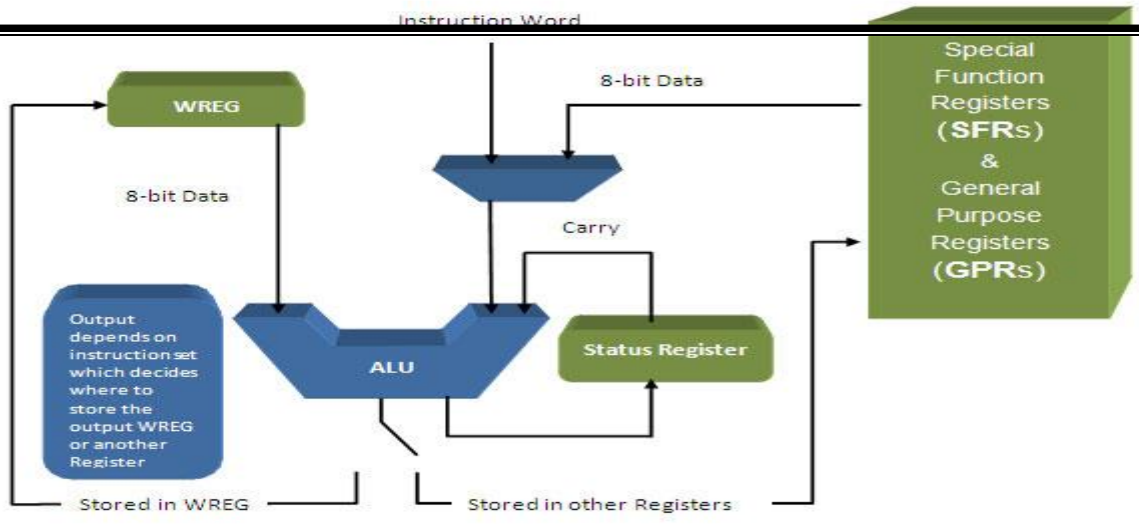


Fig. 4: Simple Block Diagram Of CPU Interfacing With Data And Program Memory In PIC

PIC18 Harvard Architecture

PIC microcontroller contains an 8-bit ALU (Arithmetic Logic Unit) and an 8-bit Working Register (Accumulator). There are different GPRs (General Purpose Registers) and SFRs (Special Function Registers) in a PIC microcontroller. The overall system performs 8-bit arithmetic and logic functions. These functions usually need one or two operands. One of the operands is stored in WREG (Accumulator) and the other one is stored in GPR/SFR. The two data is processed by ALU and stored in WREG or other registers.



Instruction Word
8-bit Data
Special Function Registers (SFRs) & General Purpose Registers (GPRs)
WREG
8-bit Data
Carry
Output depends on instruction set which decides where to store the output WREG or another Register
ALU
Status Register
Stored in WREG
Stored in other Registers

Fig. 5: Simple Block Diagram of Data Processing in PIC18 Harvard

The above process occurs in a single machine cycle. In PIC microcontroller, a single machine cycle consists of 4 oscillation periods. Thus an instruction needs 4 clock periods to be executed. This makes it faster than other 8051 microcontrollers.

Pipelining:
Early processors and controllers could fetch or execute a single instruction in a unit of time. The PIC microcontrollers are able to fetch and execute the instructions in the same unit of time thus increasing their instruction throughput. This technique is known as instruction pipelining where the processing of instructions is split into a number of independent steps.

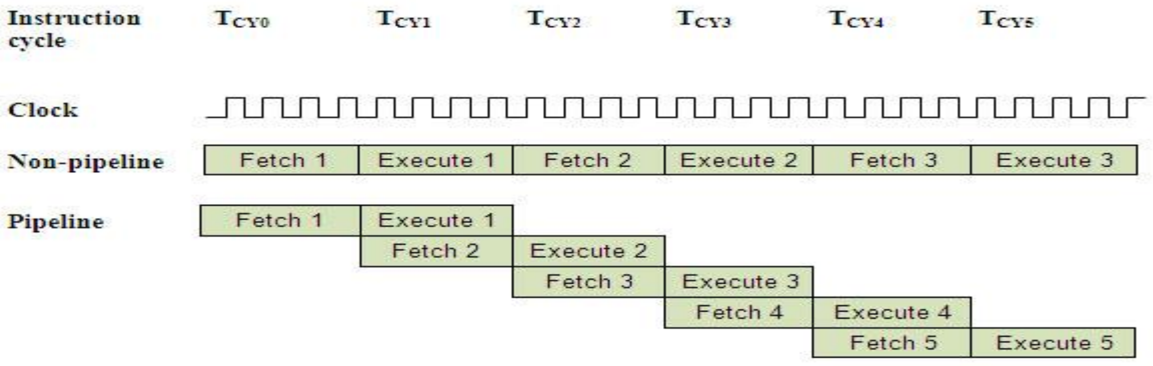


Fig. 6: Diagram Showing Instruction Pipelining Technique In PIC

Features and Peripherals

The PIC18F consists of the following features and peripherals.

Features:
- C Compiler Optimized Architecture with Optional Extended Instruction Set
- 100,000 Erase/Write Cycle Enhanced Flash
- Program Memory Typical
- 1,000,000 Erase/Write Cycle Data EEPROM Memory Typical
- Flexible oscillator option

  Four Crystal modes, including High-Precision PLL for USB

- o Two External Clock modes, Up to 48 MHz
- o Internal Oscillator: 8 user-selectable frequencies, from 31 kHz to 8 MHz
- o Dual Oscillator Options allow Microcontroller and USB module to Run at different Clock Speeds

Peripherals:
The PIC18F4550 microcontroller consists of following peripherals:

· I/O Ports: PIC18F4550 have 5 (PORTA, PORTB, PORTC, PORTD and PORTE) 8-bit input-output ports. PortB & PortD have 8 I/O pins each. Although other three ports are 8-bit ports but they do not have eight I/O pins. Although the 8-bit input and output are given to these ports, but the pins which do not exist, are masked internally.

Memory: PIC18F4550 consists of three different memory sections:

1. **Flash Memory:** Flash memory is used to store the program downloaded by a user on to the microcontroller. Flash memory is non-volatile, i.e., it retains the program even after the power is cut-off. PIC18F4550 has 32KB of Flash Memory.

2. **EEPROM:** This is also a nonvolatile memory which is used to store data like values of certain variables. PIC18F4550 has 256 Bytes of EEPROM.

3. **SRAM:** Static Random Access Memory is the volatile memory of the microcontroller, i.e., it loses its data as soon as the power is cut off. PIC18F4550 is equipped with 2 KB of internal SRAM.

· Oscillator: The PIC18F series has flexible clock options. An external clock of up to 48 MHz can be applied to this series. These controllers also consist of an internal oscillator which provides eight selectable frequency options varying from 31 KHz to 8 MHz.

· 8x8 Multiplier: The PIC18F4550 includes an 8 x 8 multiplier hardware. This hardware performs the multiplications in single machine cycle. This gives higher computational throughput and reduces operation cycle & code length.

· ADC Interface: PIC18F4550 is equipped with 13 ADC (Analog to Digital Converter) channels of 10-bits resolution. ADC reads the analog input, for example, a sensor input and converts it into digital value that can be understood by the microcontroller.

· Timers/Counters: PIC18F4550 has four timer/counters. There is one 8-bit timer and the remaining timers have option to select 8 or 16 bit mode. Timers are useful for generating precision actions, for example, creating precise time delays between two operations.

· Interrupts: PIC18F4550 consists of three external interrupts sources. There are 20 internal interrupts which are associated with different peripherals like USART, ADC, Timers, and so on.

· EUSART: Enhanced USART (Universal Synchronous and Asynchronous Serial Receiver and Transmitter) module is full-duplex asynchronous system. It can also be configured as half-duplex synchronous system. The Enhanced USART has the feature for automatic baud rate detection and calibration, automatic wake-up on Sync Break reception and 12-bit Break character transmit. These features make it ideally suited for use in Local Interconnect Network bus (LIN bus) systems.

· ICSP and ICD: PIC18F series controllers have In Circuit Serial Programming facility to program the Flash Memory which can be programmed without removing the IC from the circuit. ICD (In Circuit Debugger) allows for hardware debugging of the controller while it is in the application circuit.

· SPI: PIC18F supports 3-wire SPI communication between two devices on a common clock source. The data rate of SPI is more than that of USART.

· I2C: PIC18F supports Two Wire Interface (TWI) or I2C communication between two devices. It can work as both Master and Slave device.

ARM Processors

ARM, previously Advanced RISC Machine, originally Acorn RISC Machine, is a family of reduced instruction set computing (RISC) architectures for computer processors, configured for various environments. Arm Holdings develops the architecture and licenses it to other companies, who design their own products that implement one of those architectures—including systems-on-chips (SoC) and systems-on-modules (SoM) that incorporate memory, interfaces, radios, etc. It also designs cores that implement this instruction set and licenses these designs to a number of companies that incorporate those core designs into their own products.

Processors that have a RISC architecture typically require fewer transistors than those with a complex instruction set computing (CISC) architecture (such as the x86 processors found in most personal computers), which improves cost, power consumption, and heat dissipation. These characteristics are desirable for light, portable, battery-powered devices—including smartphones, laptops and tablet computers, and other embedded systems. For supercomputers, which consume large amounts of electricity, ARM could also be a power-efficient solution.

Arm Holdings periodically releases updates to the architecture. Architecture versions ARMv3 to ARMv7 support 32-bit address space (pre-ARMv3 chips, made before Arm Holdings was formed, as used in the Acorn Archimedes, had 26-bit address space) and 32-bit arithmetic; most architectures have 32-bit fixed-length instructions. The Thumb version supports a variable-length instruction set that provides both 32- and 16-bit instructions for improved code density. Some older cores can also provide hardware execution of Java bytecodes. Released in 2011, the ARMv8-A architecture added support for a 64-bit address space and 64-bit arithmetic with its new 32-bit fixed-length instruction set.

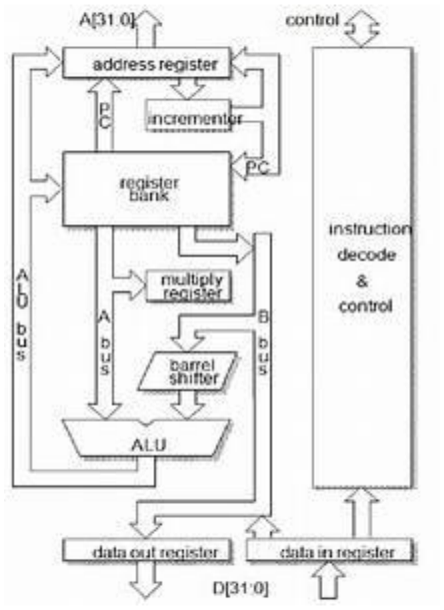
With over 100 billion ARM processors produced as of 2017, ARM is the most widely used instruction set architecture and the instruction set architecture produced in the largest quantity. Currently, the widely used Cortex cores, older "classic" cores, and specialized SecurCore cores variants are available for each of these to include or exclude optional capabilities.

The British computer manufacturer Acorn Computers first developed the Acorn RISC Machine architecture (ARM) in the 1980s to use in its personal computers. Its first ARM-based products were coprocessor modules for the BBC Micro series of computers. After the successful BBC Micro computer, Acorn Computers considered how to move on from the relatively simple MOS Technology 6502 processor to address business markets like the one that was soon dominated by the IBM PC, launched in 1981.

The Acorn Business Computer (ABC) plan required that a number of second processors be made to work with the BBC Micro platform, but processors such as the Motorola 68000 and National Semiconductor 32016 were considered unsuitable, and the 6502 was not powerful enough for a graphics-based user interface

According to Sophie Wilson, all the processors tested at that time performed about the same, with about a 4 Mbit/second bandwidth.

After testing all available processors and finding them lacking, Acorn decided it needed a new architecture. Inspired by papers from the Berkeley RISC project, Acorn considered designing its own processor. A visit to the Western Design Center in Phoenix, where the 6502 was being updated by what was effectively a single-person company, showed Acorn engineers Steve Furber and Sophie Wilson they did not need massive resources and state-of-the-art research and development facilities.



Wilson developed the instruction set, writing a simulation of the processor in BBC BASIC that ran on a BBC Micro with a 6502 second processor. This convinced Acorn engineers they were on the right track. Wilson approached Acorn's CEO, Hermann Hauser, and requested more resources. Hauser gave his approval and assembled a small team to implement Wilson's model in hardware