# Instruction Sets of 8086:

8086 $\mu P$ has got following sets of instructions:

1. Data Transfer Instruction
2. Arithmetic Instruction
3. Logic / Bit Manipulation Instruction
4. Program Execution Transfer Instruction
5. String Instruction
6. Process Control Instruction

## 1. Data Transfer Instruction

- Used to transfer data from source to destination
- Operand can be constant, memory location register or I/O port address.

It consists of the following:

@ MOV Des, Src:

eg: MOV CX, 0025H
MOV AL, BL
MOV BX, [0205H]

ⓑ PUSH Operand:

It pushes the operands onto the top of stack.

Eg: PUSH BX.

ⓒ POP Des:

It pops the operand from the top of the ~~state~~ stack to Des.

- Des can be general purpose registers, segment registers (except CS). or memory location.

Eg: POP AX.

(d) XCHG Des, Src :
→ This instruction exchanges src with Des.
→ It can't exchange, has memory locations directly.
Eg: XCHG, DX, AX.

(e) IN Accumulator, Port Address :
→ It transfers the operand from specified port to the Accumulator.
Eg: IN AX, 0028H.

(f) OUT port address, Accumulator :
→ It transfers operand from Accumulator to the specified port.
Eg: OUT 0028H, AX.

(g) LEA Register, Src :
→ It loads the 16-bit register with the offset address of the data specified by the src.
Eg: LEA BX, [DI].

(h) LDS Des, Src :
→ It loads the 32-bit pointer from memory source to destination register & DS (Data Segment)
→ The offset is placed in the destination register and the segment is placed in DS.
Eg: LDS BS, [0301H].

(i) LES Des, Src :
→ It loads the 32-bit pointer from memory source to destination register and ES.
→ The offset is placed to des register and the segment is placed in ES. Eg: LES BX, [0301H]

(j) LAHF
→ It copies the lower byte of flag register to AH.

(k) SAHF
→ It copies the content of AH to lower byte of flag.

(l) PUSHF
→ It pushes flag register onto the top of stack.

(m) POPF
→ It pops the top of stack into the flag register.

2. Arithmetic Instructions

(a) ADD Des, Src
→ It adds a byte to byte or word to word.
→ It affects AF, CF, OF, PF, SF, ZF flags.
Eg:
ADD AL, 72H — AL ← AL+72H
ADD DX, AX
ADD AX, [BX] — AX ← AX + (BX)
↳ Memory bhitrako data

(b) ADC Des, Src
→ It adds two operands with CF.
→ It affects AF, CF, OF, PF, SF, ZF flags.
Eg: ADC AL, 72H
ADC DX, AX
ADD AX, [BX]

(c) SUB Des, Src

→ It subtracts a byte from byte or word from word.

→ It affects AF, CF, OF, PF, SF, ZF flags.

→ CF acts as a borrow flag.

Eg:

```
SUB AL, 72H
SUB DX, AX
SUB AX, [BX]
```

(d) SBB Des, Srz

→ It subtracts the two operand and also borrow from the result.

→ It affects AF, CF, OF, PF, SF, ZF Flag.

Eg:

```
SBB AL, 72H
SBB DX, AX
SBB AX, [BX]
```

(e) INC Src

→ It increments the byte or word by 1.

→ The operand can be register or memory.

→ It affects AF, OF, PF, SF, ZF flags.

→ CF is not affected.

Eg: INC bl

(f) Dec Src

→ It decrements the byte / word by 1.

Eg: Dec bX

(g) AAA (ASCII Adjust After Addition).

→ The data entered from the terminal is in ASCII format.

→ In ASCII 0-9 are represented by 30H - 39H.

→ This instruction allows us to add the ASCII codes.

(h) AAS

(i) AAM

(j) AAD

(k) NEG Src

→ It creates 2's complement of the given number / src.

(l) MUL Src

→ It is an unsigned multiplication instruction.

→ It multiplies two bytes to produce a word or two word to produce a double word.

AX = AL * Src.

AX = AX * Src.

→ This instruction assumes one of the operands in AL or AX.

→ Src can be register or memory.

(m) DIV Src

→ It is unsigned division.

→ The operand is stored in AX, Divisor is Src and the result is stored as:

AH = remainder,

AL = quotient

3 Logic / BIT Manipulation Instruction

(a) NOT Src

→ It complements each bit of src to produce 1's complement of specified operand.

→ The operand can be register or memory.

(b) AND Des, Src

→ AND operator of Src and Des

→ Src can be immediate number, register or memory.

→ Des can be register or memory.

→ Both operands can't be memory simultaneously.

Eg: AND AL, BL.

(c) OR Des, Src

(d) XOR Des, Src

(e) SHL Des, Count

→ It shifts bits of byte/word left by count.

→ It puts zeros in LSB.

→ MSB is shifted into Carry Flag.

(f) SHR Des, Count

→ It shifts bits of byte/word right by count

→ It puts zeros in MSB.

→ LSB is shifted into CF.

(g) ROL Des, Count

→ It rotates bits of byte/word left by count.

→ MSB is transferred to LSB and CF.

(h) ROR Des, Count
→ Rotate right by count
→ LSB is transferred to MSB and CF

(i) CMP Des, Src
→ It compares two specified bytes/words
→ Src, Des can be constant, register or memory.
→ Flags are modified according to the result

⇒ See 8085 (For Flags Condn)

4. Program Execution Transfer Instruction
(a) CALL Des:
→ This instruction is used to call a subroutine or procedure.
→ The address of next instruction after CALL is saved onto stack.

(b) RET
→ It returns the control from procedure to CALLing program.

(c) JMP Des
→ Unconditional jump from one place to another.

(d) JXX Des
→ Conditional Jump Instructions (that effects the Flags)
→ The following table shows the list of instruction:

| Conditional Jump Instructions | Condition |
|---|---|
| JA → Jump if above | CF → 0, ZF → 0 |
| JAE → Jump if above or equal | CF → 0 |
| JB → Jump if below | CF → 1 |
| JBE → Jump if below or equal | CF → 1, ZF → 1 |
| JC → Jump if carry | CF → 1 |
| JNC → Jump if no carry | CF → 0 |
| JNZ → Jump if zero | ZF → 1 |
| JPE NZ → Jump if no zero | ZF → 0 |
| JPE → Jump if parity even | PF → 1 |
| JPO → Jump if parity odd. | PF → 0 |

6. LOOP Des

→ This is looping instruction and the number of loop/iteration is placed in cx register

→ With each iteration, the content of cx are decremented.

5. String Instructions

→ String in assembly language is just a sequential stored bytes/words.

→ By using the instructions, the size of the program is considerably reduced.

→ It consists of the following instructions:

@ CMP Des, Src:

It ~~consist~~ compares the string bytes or words.

ⓑ SCAS String:

It scans a string. It compares the string with byte in AL or with word in AX.

⑤ MOVS / MOVB / MOVSW :

→ It causes moving of bytes or word from one string to another. In this instruction the source is in data segment and the destination is in extra segment.

→ SI and DI store the offset values for the source & destination string.

④ Rep (Repeat) :

→ This is an instruction prefix.

→ It causes the repetition of instruction until cx becomes zero.

⑥ Processor Control Instruction :

These instructions controls the processor itself. Some of these instructions are.

I. STC : It sets the CF to 1.

II. CLC : It clears the CF to 0.

III. CMC : It complements the CF.

IV. STD : It sets the direction Flag (DF) to 1.

V. CLD : It clears the DF to 0.

Very Imp

# Assembly Language Programming of 8086

* Sample Program (To display a string "Hello World")

Assembler Directive
- .Model small
- .Stack 100h → Stack segment ko size 100
- .Data
  String db "Hello World!$"
- .Code

main proc
- START UP
  - mov ax, @DATA ; @DATA is predefined symbol for initial address of DS
  - mov ds, ax ; initialize data segment.
  - mov dx, offset string ; load the offset address into dx
  - OR
  - LEA dx, string
  - mov ah, 09h ; ax = 09h for string display until $.
  - Int 21h ; DOS interrupt function.
- EXIT
  - mov ax, 4c00h ; End request with AX = 4c00h
  - Int 21h ; return control to OS

main endp ; end procedure
end main ; end program

# ALP Development Tools:

1. Editor • asm
2. Assembler → One pass Assembler
   → Two pass Assembler (.obj)
3. Linker (.obj)
4. Debugger → .exe
5. Emulator (combⁿ of ~~hlw one~~ hardware and software)

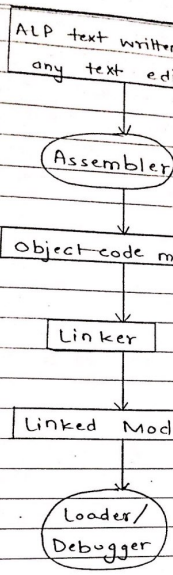



Fig: Flowchart for the development of the ALP tools.

Qn: Differentiate one pass assembler and two pass assembler.

Imp # Assembler Directives

An ALP contains two types of statements: instruction and assembler directives. The instructions are translated into machine codes whereas directives are not converted into machine codes. The directives are statements to give directions to the assembler to

perform the task of assembly process. They indicate how an operand or a section of a program is to be processed by the assembler. The assembler supports directives for data definition, segment organizations, procedure, macro definitions etc.

↳ Some imp directives are;

① .DATA : It provides shortcut in definition of the data segment.

② DB (Define Byte) : The directive DB defines a byte type variable. It directs the assembler to reserve one byte of memory and initialize the byte with the specified value.

→ It can define single or multiple variables.

Example :

```
TEMP    DB  5
TEMP    DB  ?
TEMP    DB  11,22,33,44.
```

③ DW (Define Word) → 2 bytes

④ DD (Define double word) → 4 bytes

⑤ DQ (Define Quad word) → 8 bytes

⑥ .CODE : This provide shortcut in definition of the code segment. It is basically specified to distinguish different code segment when there are multiple CS in the program.

Eg: .CODE [name]
optional

(7) PROC : It tells the start of the procedure or subroutine.

(8) SEGMENT :

It indicates the beginning of a logical segment. The name of the segment is written before the directive SEGMENT.

Eg: Program SEGMENT
Segment name

(9) ENDS :

It informs the assembler, the end of the segment. It is used with the segment directive.

(10) ENDP :

It informs the assembler the end of the procedure. This directive together with PROC is used to enclose a procedure.

(11) END

This directive is used after the last statement of the program to inform the assembler that this is the end of a program module.

(12) ASSUME

It tells the name of a logical segment which is to be used for specified segment

(13) STACK

It indicates the start of the stack segment. The default size of stack is 1024 bytes.

(14) • Model:

It provides the info of the segment size that is to be used in a program module. The size of the model may be tiny, small, large, etc.

(15) • Startup:

It indicates the start of the program when using program.

(16) • EXIT:

It indicates the end of the program when using program.

# DOS Interrupt Function of 8086:

→ The list of interrupts are:

* INT 10H function:

① Int 10h/AH = 00H → Set video mode input
② Int 10h/ah = 01H → Set text mode cursor shape
③ Int 10h/ah = 02H → Set cursor position
④ Int 10h/ah = 06H → Scroll window up
⑤ Int 10h/ah = 08H → read character and attribute at cursor position.
⑥ Int 10h/ah = 0AH → write character only at cursor position.

* INT 21h functions:

① Int 21h/ah = 01h → read character from standard input, with echo, result is stored in AL
② Int 21h/ah = 02h → write character to standard output. eg: mov ah, 02h (the character must be in dl)
mov dl, 'a'
Int 21h.

③ Int 21h/ah = 05h → o/p character to printer.

④ Int 21h/ah = 09h → output of a string at DS:DX
String must be terminated by $.

⑤ Int 21h/ah = 0Ah → Input of a string to DS:DX
first byte is buffer size, second byte is number of characters actually used.
→ $ not required to terminate the string.

⑥ Int 21h/ah = 4Ch → Return control to operating system (Stop program).

# WAP in 8086 to add 8-bit number

```
.model small
.stack 100h
.Data                          dB → Define Byte
    Val1 dB 25h
    Val2 dB 35h
.Code
    main proc
.startup { mov ax, @DATA
           mov ds, ax
           mov bl, val1
           add bl, val2
           mov result, bl
.exit    { mov ax, 4c00h   or   mov ah, 4ch
           Int 21h
main endp
end main
```

# WAP to display alphabets from a to z.

```
.model small
.stack 100h
.data
    alpha dB 'a'
.code
    main proc
      .startup
          mov cx, 26
          mov bl, alpha
          mov dl, bl
again :   mov ah, 02h      } a display.....
          Int 21h
          Inc dl
          loop again
      .exit
    main endp
    end main
```

# WAP to display numbers from 0 to 9.

```
.model small
Space macro
.stack 100h
mov dl, ' '
.data mov ah, 02h
      int 21h
    endm
.stack 100h
.data
    digit dB '0'
    .code
        display proc
        .startup
          mov cx, 10
          mov bl, digit
again :   mov dl, bl
          mov ah, 02h
          int 21h
          space
          inc bl
          loop again
        .exit
        display endp
        end display
```

Imp

# MACRO :

It is a group of instructions. The macro assembler generates the code in the program each time where the macro is called. Macro's can be defined by MACRO and ENDM directives.

Example :

```
space  macro ; macro definition
(name)
mov  dl, c
mov  ah, 02h   ; body
int  21h
endm ; end macro
```

# Macro Vs Procedure

| | Procedure | Macro |
|---|---|---|
| 1 | Accessed by CALL and RET instruction during program execution. | Accessed during Assembly with name given to macro. |
| 2 | Machine code is generated only once in memory. | Machine code is generated each time when it is called. |
| 3 | Less memory (memory efficient) | More memory is required. |
| 4 | Parameters can be passed in register, memory location, stack. | Parameters passed as a part of the statement which calls macro. |
| 5 | long codes | Short codes |
| 6 | eg: main proc<br>{<br>main endp<br>end main | eg: space macro<br>{<br>endm |

# WAP to display alphabets 'a to z' with space (Use macro)

→

```
.model small
space macro
mov dl, ' '
mov ah, 02h
Int 21h
endm
.stack 100h
.data
    alpha dB 'a'
.code display proc
    .startup
    mov cx, 26
    mov bl, alpha
again: mov dl, bl
    mov ah, 02h
    Int 21h
    space
    Inc bl
    loop again
    .exit
    display endp
    end display
```

# WAP to display a string 'microprocessor' characterwise in console.

→
```
.model small
.stack 100h
.data
    string dB 'microprocessor"
.code
main proc
    .startup
    mov cx, 14
    mov si, offset string
again: mov dl, [si]
    mov dl, bl
    mov ah, 02h
    Int 21h
    Inc Si
    loop again
    .exit
main endp
end main
```

# To display a string 'microprocessor' in reverse order characterwise.

→
```
.model small
.stack 100h
.data
    string dB 'microprocessor"
.code
main proc
    .startup
    mov cx, 14
    mov si, offset string
```

```
add si,13
again: mov bl,[si]
       mov dl,bl
       mov ah,02h
       int 21h
       Dec si
       loop again
  .exit
  main endp
  end main
```

# WAP to transfer a block of data from memory location list1 to list2. Consider the block of six bytes.

→ Title copy of block of data.

```
Dosseg
.model small
.stack 100h
.data
    list1 dB 10,20,30,40,50,60
    list2 dB 6dup(?)
.code
  main proc
    .startup
      mov si,offset list1
      mov di,offset list2
      mov cx,06
  back: mov al,[si]
      mov [di],al
      inc si
      inc di
      loop back
      .exit
      main endp
      end main
```

# You are given two strings "assembly program" and "language". Now write an assembly language program to display the string 'assembly language program'.

→

```
.model small
  display macro
    mov dx, offset string2
    mov ah, 09h
    int 21h
    end m
.stack 100h
.data
    String1 dB "Assembly Program"
    String2 dB "Language"
.code
  main proc
   .startup
      mov  cx, 09
      mov  Si, offset String1
back: mov  dl, [Si]
      mov  ah, 02h
      Int  21h
      Inc  Si
      loop back
      display
      mov di, offset string1
      add dl, 10
again : mov dl, [di]
      mov ah, 02h
      Int 21h
      Inc di
      loop again
main .exit
     end
```

# Chapter 5 - Interrupts

→ An interrupt is a signal that a peripheral board sends to the central processor in order to request attention

→ In response to the interrupt, the processor stops what it is currently doing and executes the Interrupt Service Routine (ISR) in order to handle the interrupt.

→ When the execution of service routine is terminated, the original process may resume its previous operation.

→ The interrupt is initiated by an external device and is asynchronous i.e it can be initiated at any time without reference to the system clock. However, the response to an interrupt request is directed or controlled by the microprocessor.

→ Interrupts are primarily issued on:
- Initiation of I/O operation
- Completion of I/O operation.
- Occurance of Hardware / Software Errors.

## Sources of Interrupts

1. Processor (Internal) Interrupts,
2. Hardware (External) Interrupts, and
3. Software Interrupts.

### 1. Processor (Internal) Interrupts

→ This interrupt is caused by some error conditions produced in the processor internally during the execution of instructions.

→ For example, the various conditions like - divide by zero, register overflow, stack overflow, etc. generate processor interrupts.

2. Hardware (External) Interrupts :

→ This interrupt is caused by external signal applied to the Interrupt pins.

→ For example, in 8086 microprocessors, the external requests are caused through NMI (Non-Maskable Interrupts) and INTR (Interrupt Request) Pins.

3. Software Interrupts

→ This interrupt is caused by the available interrupt instructions.

→ For example, in 8086 microprocessor. INT instructions eg: INT 21H instruction causes DOS interrupts to execute. Similarly in 8085 RST0 - RST7 instructions cause software interrupts.

# Classification of Interrupts

→ Interrupts can be broadly classified into :

1. Maskable and non-maskable Interrupts,
2. Vectored and non-vectored Interrupts

1. Maskable and non-maskable Interrupts

i) Maskable Interrupts

→ These interrupts can be enabled or disabled by executing instructions such as EI (Enable Interrupts) DI (Disable Interrupts) and SIM (Set Interrupt Masked Instructions.

→ Thus, those interrupts which can be disabled or delayed are called maskable interrupts.

→ In 8085 microprocessor, the interrupt requests occured through INTR, RST 7.5, RST 6.5 and RST 5.5 are maskable interrupts. Similarly, in 8086 microprocessor,

INTR is the maskable interrupt pin.

ii) Non-maskable Interrupts:

→ These interrupts cannot be enabled or disabled using instructions.

→ Thus, those interrupts which cannot be disabled or delayed or blocked by the microprocessor are called the maskable interrupts.

→ For example, in 8085 microprocessor, TRAP is the only non-maskable interrupt pin and in 8086 microprocessor NMI is non-maskable interrupt pin.

→ Normally, the emergency circuits like-power protection circuits are connected to non-maskable interrupt pins.

2 Vectored and Non-Vectored Interrupts

i) Vectored Interrupts.

→ Those interrupts whose ISR address is already known to the microprocessor are called vectored interrupts.

→ For example, TRAP, RST 7.5, RST 6.5 and RST 5.5 are vectored interrupts in 8085.

| 8085 Interrupts | Call location (or Interrupt Vector) |
|---|---|
| TRAP | 0024H |
| RST 7.5 | 003CH |
| RST 6.5 | 0034H |
| RST 5.5 | 002CH |

ii) Non-vectored Interrupts.

→ In this interrupts, the address of the Interrupt Service Routine (ISR) is not already known to the microprocessor and needs to be supplied externally.

→ Normally, the interrupting device or an external hardware needs to supply the interrupt vector (i.e starting address of ISR) to the microprocessor.

→ For example → in 8085 and 8086 microprocessor INTR (Interrupt Request) is the non-vectored interrupt pin.

# 8085 Interrupts

→ 8085 microprocessor has 5 interrupts pins. They are

1. INTR
2. RST 7.5
3. RST 6.5
4. RST 5.5
5. TRAP

1. INTR (Interrupt Requests)

→ It is the maskable and non-vectored pin in 8085 microprocessor.

→ The interrupt request in IMR can be enabled or disabled using EI (Enable Interrupt) and DI (Disable Interrupt) instructions.

→ The request in INTR pin is acknowledged by microprocessor using INTA (Interrupt Acknowledge) pin.

2. RST 7.5 :-

→ It is the hardware reset pin in 8085 microprocessor.
→ It is highest priority reset pin among all reset pins.
→ It is maskable and vectored interrupt, that can be enabled or disabled using SIM (Set Interrupt Mask) instruction.

3. RST 6.5 :-

→ It is the reset pin in 8085 microprocessor with second highest priority.
→ It is also maskable and vectored interrupt and also can be enabled or disabled using SIM instruction.

4 RST 5.5 :-

→ It is also the reset pin in 8085 microprocessor with lowest priority.
→ It is also maskable and vectored interrupt and can be enabled or disable using SIM instruction.

5. TRAP

→ It is the only non-maskable interrupts in 8085 ~~Pin~~ MP.
→ It has highest priority among all interrupt pins.
→ It is the vectored interrupt and is generally used for critical events such as power failure, emergency shut off, etc.

# Priority of 8085 Interrupts

→ In 8085, TRAP pin has the highest and INTR pin has the lowest priority. The priority of all interrupt pins is as follows:

1. TRAP
2. RST 7.5
3. RST 6.5
4. RST 5.5
5. INTR

Priority decreases.

# 8086 Interrupts and Interrupt Vector Table (IVT):

imp Interrupt Vector Table (IVT)

- In 8086, the first 1KB of memory from 00000H to 003FFH is set aside as a table for storing the starting address of the Interrupt Service Routine (ISR) or Interrupt Service Procedures. This table is called Interrupt Vector Table (IVT)
- → Since 4 bytes are required to store the CS and IP values for each interrupt service procedures, the table can hold the starting addresses for upto 256 interrupt procedures.
- → The starting address of an interrupt-service procedure is often called the interrupt vector or the interrupt pointer, so the table is referred to as interrupt-vector table (IVT) or interrupt-pointer table (IPT).
- Figure shows the IVT indicating the arrangement of 256 interrupt vectors.

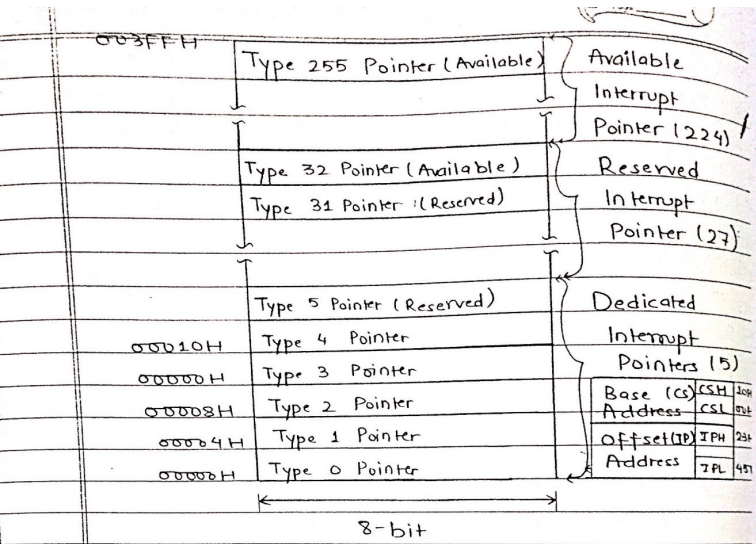



Fig: 8086 Interrupt Vector Table

## 8086 Interrupt Types

1. Software Interrupts

→ These interrupts are initiated by executing an interrupt instruction. For example, INT 21H is the software interrupt in 8086 that is the DOS interrupt and provides more than 80 different services.

2. Hardware Interrupts.

→ These interrupts are initiated by applying an electrical signal to the interrupt pins of the processor.

→ For example, INTR and NMI pin of 8086 microprocessor for providing external hardware interrupts to 8086 microprocessor.

3. Pre-defined or Dedicated Interrupts:

→ These include first five interrupts in 8086 Interrupt Vector Table and are internal interrupts.

(a) Type 0: Divide-Error Interrupt

→ This interrupt is issued by INT 00H instruction.

→ Type 0 interrupt occurs whenver the result of the division overflows or whenever an attempt is made by divide by zero.

(b) Type 1: Single-step Interrupt

→ In this interrupt, microprocessor executes in single-step mode, i.e goes for debugging after execution of single step of instruction.

→ For this type, Trap Flag (TF) should be set.

→ Thus, this type of interrupt is useful for monitoring & debugging programs.

(c) Type 2: Non mask-able Interrupt:

→ This interrupt is initiated when NMI pin of MPU receives a low to high transition. This interrupt is normally used for catastrophic conditions such as power failure.

(d) Type 3: Break-point Interrupt:

→ In this interrupt, a break-point is set and is non-maskable.

→ When we insert a breakpoint and then goes to the desired breakpoint subroutine for debugging.

(e) Type 4: Overflow Interrupt

→ Overflow interrupt can be generated using INT 4 or INTO (Interrupt overflow) instructions.

→ Overflow flag should be set in order to execute INTO instructions.

# 8086 Interrupts Priority:

→ The priority of various 8086 interrupts are as follow

1. Divide Error, INTO
2. NMI
3. INTR
4. Single-Step
5. Internal Interrupts (Except Single Step)

Priority decreases ↓

# Servicing Multiple Interrupts (Prioritising Multiple Interrupts)

→ There are mainly two ways of servicing multiple interrupts. They are:

1. Polled Interrupts (Equal Priority Devices)
2. Chained (Vectored) Interrupts

1. Polled Interrupts (Equal Priority Devices)

→ In this method, there is one-common branch address for all interrupts. The program that takes care of interrupts begins at the branch address and polls the interrupt sources in sequences.

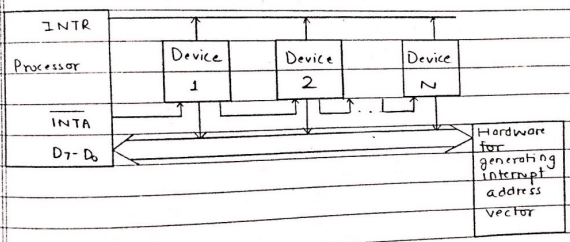
→ The order in which they are tested determines the priority of each interrupt.

→ The highest priority is tested first, and if its interrupt signal is on, control branches to the service routine for this source. Otherwise, the next lower priority source is tested and so on.

→ Polled interrupts are very simple but for large number of devices, polled interrupts are slower. Moreover, as these interrupts are handled using software, they are slower than vectored interrupts.

2. Chained (Vectored) Interrupts

→ This is the hardware concept of handling multiple interrupts. In this technique, the devices are connected in a chain function as shown in figure below for setting up the priority.




→ Here, the device with highest priority is placed in the first position, followed by the lowest priority devices.

→ Suppose that one or more devices interrupt the processor at a time. In response, the processor saves its current status and then generates an interrupt acknowledge ($\overline{INTA}$) signal to the highest priority device, which is device 1 in above fig. If this device has generated interrupt it will

accept the $\overline{INTA}$ from the processor, otherwise it passes $\overline{INTA}$ on to the next devices until the $\overline{INTA}$ is accepted by the interrupting device. Once accepted, the device provides a means to the processor for finding the interrupt address vector using external hardware.

→ It is faster than polled interrupt.

# Chapter - 6 : Input - Output Interfaces

1. Serial I/O Standards : 8251A USART
2. 8259A PIC
3. 8255A PPI
4. 8254 PIT
5. DMA

## 1. Serial I/O Standards : 8251A USART (Universal Synchronous Asynchronous Receiver Transmitter)

→ The 8251A is a programmable chip designed for synchronous and asynchronous serial data communication, packaged in a 28-pin DIP.

The USART accepts data characters from CPU in parallel format and converts them into a continuous serial data stream for transmission.

Simultaneously, it can receive serial data streams and converts them into parallel data character for CPU.

• Serially I/O :

→ Specially, there are two basic approaches for serial data communication. They are:

(A) Software-controlled I/O (eg: using SID, SOD pins, RIM & SIM instructions in 8085)

(B) Hardware-controlled I/O (using 8250 UART, 8251 USART)

Serial Data Standards (protocols)

1. RS232C
2. BISYNC

Qn. Explain RS232C standard with necessary conditions.

# Block Diagram of 8251A USART:

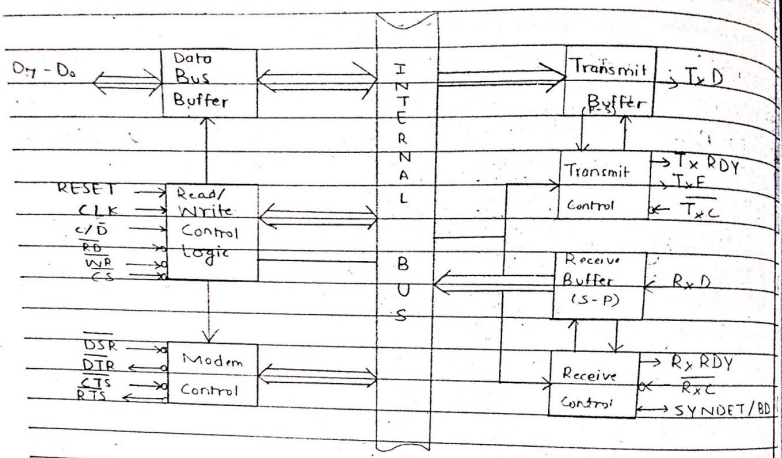



Fig: Functional Block Diagram of 8251A USART

Explanation:

Above functional block diagram of 8251A USART includes 5 selections which are as follows:

1. Read / Write Control Logic and Registers.
2. Transmitter Section
3. Receiver Section
4. Data Bus Buffer Section
5. Modem Control Section.

Read/Write Control Logic and Registers:

This control logic interfaces the chip with the MPU, determines the functions of the chip according to the control word in its register, and monitors the data flow. This section includes R/W control logic, six input signals, control logic and three buffer registers: data registers, control register and status register.

The various signals to control logic are as follows:

① $\overline{CS}$ (Chip Select)

- When this signal goes low, the 8251A is selected for by the MPU for communication.

② C/$\overline{D}$ (Control/Data)

- When this signal is high, the control register or the status register is addressed, when it is low the data buffer is addressed.

③ $\overline{WR}$ (Write)

- When this signal goes low, the MPU either writes in the control register or sends output to the data buffer.

④ $\overline{RD}$ (Read)

- When this signal goes low, the MPU either reads a status from the status register or accepts data from the data buffer.

| CS | C/D | RD | WR | Function |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | MPU writes instructions in the ctrl |
| 0 | 1 | 0 | 1 | MPU reads status from the stat |
| 0 | 0 | 1 | 0 | MPU outputs (works) data to the data |
| 0 | 0 | 0 | 1 | MPU accepts (read) data from data |
| 1 | x | x | x | 8251A USART is not selec |

(e) RESET (Reset)

- A high on this input resets the 8251A and forces it into the idle mode.

(f) CLK (Clock):

- This is the clock input, usually connected to the system clock.
- This clock doesnot control either the transmission or the reception rate.

* Control Register

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

- Status Register

This input register checks the ready status of the peripheral. This register is addressed as an input port when C/$\bar{D}$ pin is high.

② Transmitter Section :

- This section accepts parallel data from the mpu and converts them into serial data.
- It has two registers: a buffer registers to hold eight bits and an output register to convert eight bits into a stream of serial bits.
- This section transmits data on the $T_xD$ pin with the appropriate framing bits (start and stop bit).
- Different signals associated with Transmitter Section are as follows :

ⓐ $T_xD$ (Transmit Data) :-

- Serial bits are transmitted on this line.

ⓑ $T_xC$ (Transmitter Clock) :-

- This signal controls the rate at which bits are transmitted by the 8251A USART.
- The clock frequency can be 1, 16 or 64 times the baud.

ⓒ $T_xRDY$ (Transmitter Ready) :-

- When this output signal is high, it indicates that the buffer register is empty and the USART is ready to accept the byte.

bit rate → bits per second
baud rate → symbols per second

(d) TxE (Transmitter Empty)
- When this output signal is high, it indicates that the output register is empty.

(3) Receiver Section:
- This section accepts the serial data on the RxD line from a peripheral and converts them into parallel data.
- This section has two registers: the receiver input register and the buffer register.
- When the RxD line goes low, the control logic assumes it as start bit, waits for half time, and samples the line again.
- If the line is still low, the input register accepts the following bits, forms a character, and loads it into the buffer register.
- The various signals on receiver section are:
  @ RxD (Receive Data):
    → Bits are received serially on this line and converted into a parallel byte in the receiver input register.

  (b) Rx C (Receive Clock):
    → This is a clock signal that contains the rate at which bits are received by the USART.
    → In the asynchronous mode, the clock can be set to 1, 16 or 64 times the baud.

7) RxRDY (Receiver Ready):
→ This output signal goes high when the USART has a character in the buffer register and is ready to transfer it to the mPU.

Initializing the 8251A:
→ To implement the serial communication, the mPU must

④ Data Bus Buffer Section:
→ This section holds 8-bit data ($D_7$-$D_0$) before data is written to transmitter section or after data is read from receiver section.
→ It transmits data to and from mPU and 8251A USART.

⑤ Modem Control Section: and
→ This section checks for the handshake signals, controls the communication by checking the readiness, termination of the data transfer.

Initializing the 8251A:
→ To implement the serial communication, the mPU must inform the 8251A of all details, such as mode, baud, stop bits, parity, etc.
→ Items set by mode instructions are;
→ Synchronous / Asynchronous mode,
→ Stop bit length (asynchronous mode)
→ Character length,
→ parity bit
→ Baud rate factor (asynchronous mode), etc.

- Similarly items set by common word are:
  - → Transmit Enable / Disable,
  - → Receive Enable / Disable,
  - → DTR, RTS output of Data,
  - → Resetting of error flag, etc.
- Thus, before data transfer, a set of control words must be loaded into 16-bit control register of 8251A.
- The control words are divided into two formats:
  1. Mode word,
  2. Common word

Stop-bit length

| | | | |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| Inhibit | 1 bit | 1.5 bits | 2 bits |

1. Mode word

| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
|---|---|---|---|---|---|---|---|
| $S_2$ | $S_1$ | EP | PEN | $L_2$ | $L_1$ | $B_2$ | $B_1$ |

Baud rate factor

| | | | |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| SYNC Mode | 1X | 16X | 64X |

Character length

| | | | |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 5 bits | 6 bits | 7 bits | 8 bits |

Parity Check

| | | | |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| Disable | Odd Parity | Disable | Even Parity |

Fig: Bit configuration of Mode Word

Date
Page 3

2 Command word

| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
|---|---|---|---|---|---|---|---|
| EH | IR | RTS | ER | SBRK | RXE | DTR | TXEN |

1 = Hunt Mode
0 = Normal operation

1 = Internal Reset
0 = Normal operation

$\overline{RTS}$ (Request To Send)
1 → $\overline{RTS}$ = 0
0 → $\overline{RTS}$ = 1

1 = Reset Error Flag
0 = Normal Operation

1 = Sent Break Character
0 = Normal operation

1 = Receiver Enable
0 = Disable

$\overline{DTR}$
1 → $\overline{DTR}$ = 0
0 → $\overline{DTR}$ = 1

1 = Transmit Enable
0 = Disable

Fig : Bit configuration of Command Word.

# Status Word

It is possible to see the internal status of 8251A by reading a status word.

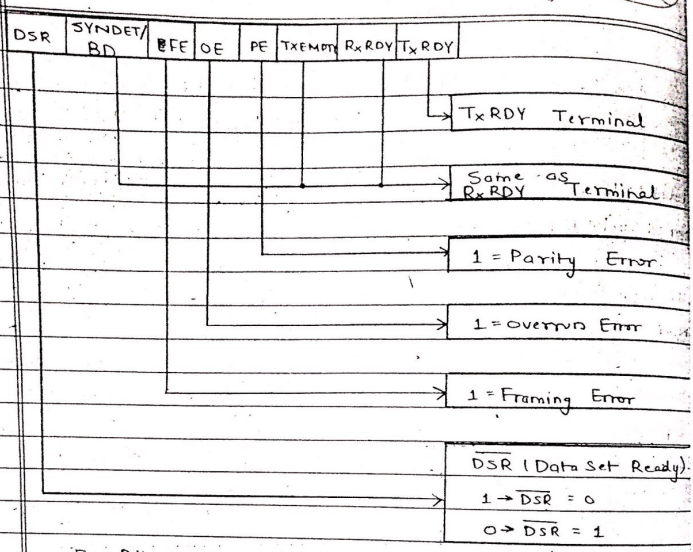



Fig: Bit configuration of Status Word

2.

V.Imp # 8259A PIC (Programmable / Priority Interrupt Controller)

- The 8259A is a programmable / priority interrupt controller that is used to manage and resolve the interrupts issued from a number of input, output devices at the same time.
- It is a 28-pin IC that is designed to work with Intel 8085, 8086 and 8088 microprocessors.

Normally, most of the microprocessors have limited number of pins for handling the interrupt request from peripheral devices. For example, if we are working with 8086, we have only two interrupt input pins: NMI and INTR. If we save NMI for power failure interrupt, this leaves only one interrupt pin for all other application which is difficult to manage.

8259A is a most common PIC that accepts interrupt requests from upto 8 I/O devices (in single mode) and upto 64 devices in cascaded mode.

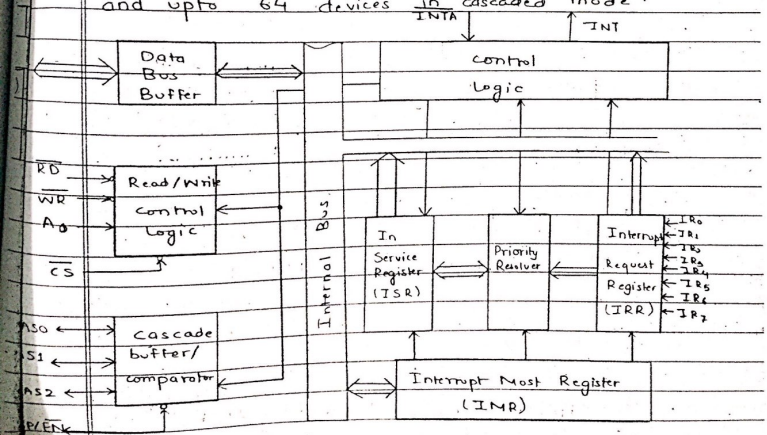



Fig: Block diagram of 8259A PIC

# 8259A Block Diagram explanation :

→ The functional block diagram of 8259A consists of following four sections :

1. Interrupt and control logic Section,
2. Data Bus Buffer Section,
3. Read/Write control logic Section, and
4. Cascade Buffer / Comparator Section

1. Interrupt and control logic Section
- This section consists of 5 sub-sections which are as follows :

@ Interrupt Request Register (IRR) :
→ This 8 bit register is used to store the status of all 8 interrupt input lines ($IR_0$-$IR_7$) which are requesting service. The 8 interrupt inputs are set corresponding bits of interrupt request registers.

ⓑ In-Service Register (ISR) :
→ This register is used to store the information about the interrupt levels that are currently being serviced.

ⓒ Interrupt Mask Register (IMR) :
→ This register stores the masking bits of interrupt lines.
→ Masking of higher priority input will not affect the request lines of lower priority.
→ This register can be programmed by OCW1 (Operation Command word)

(d) Priority Resolver:

→ This logic block determines the priorities of the bit set in the IRR.

→ The bit corresponding to the highest priority interrupt is selected and stored into corresponding bit of ISR during $\overline{INTA}$ pulse.

(e) Control Logic Block:

→ This block has two pins: INT (Interrupt) as an output and $\overline{INTA}$ (Interrupt Acknowledgement) as an input.

→ The INT pin is connected to Interrupt Request pin (eg: INTR in 8085) and $\overline{INTA}$ pin is connected to Interrupt Acknowledgement pin (eg: $\overline{INTA}$ in 8085) of MPU.

2. Data Bus Buffer Section:

→ The 8-bit tri-state, bidirectional 8-bit buffer is used to interface the 8259A to the system data bus of MPU.

→ Control words and status informations are transferred through Data Bus Buffer.

3. Read/Write control Logic:

→ The function of this block is to accept output commands to from the CPU.

→ It contains the Initialization Command Word (ICW) and Operation command Word (OCW) registers which store the various control formats for the device operation and setup 8259A to operate in various modes.

→ The different input lines to Read/Write control Logic are as follows:

① $\overline{CS}$ (Chip Select):
— A low on this input enables the 8259A. No reading or writing will occur unless the device is selected

② $\overline{WR}$ (Write):
— A low on this input enables the CPU to write control words (ICWs and OCWs) to the 8259A.

③ $\overline{RD}$ (Read):
— A low on this input enables the 8259A to send the status of IRR, ISR, IMR or the ~~int~~ interrupt level on the data bus.

(4) $A_0$

→ This signal is directly tied to one of the address lines of MPU.

→ This signal is used in conjunction with $\overline{WR}$ and $\overline{RD}$ signals to write commands into various command registers, as well as reading the various status registers of the chip.

4) Cascade Buffer/Comparator Section:

→ This section is used to operate 8259A in cascaded mode, in which it can handle interrupt requests from maximum upto 64 I/O devices.

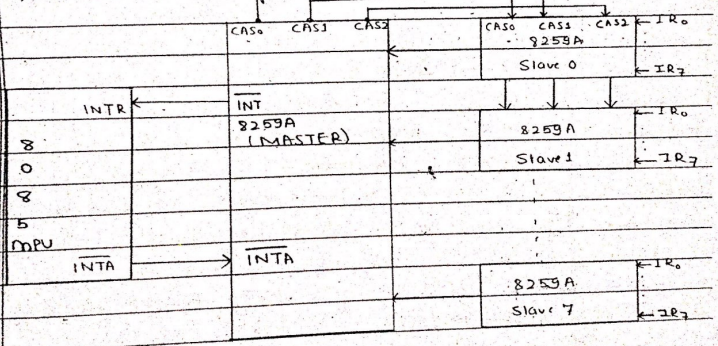



Fig: Cascading 8259As

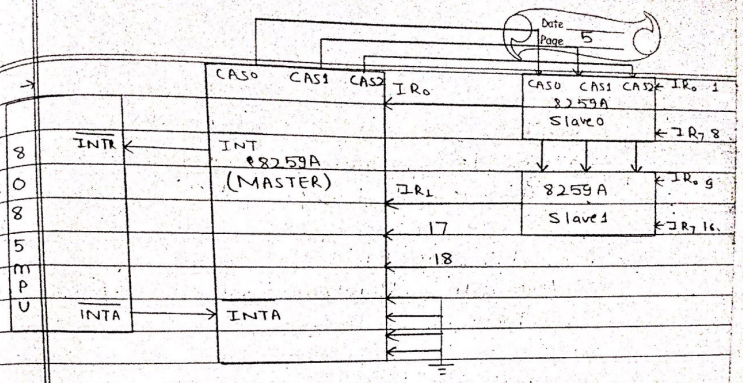
As shown in above fig to operate in cascade mode, multiple 8259As chip are configured in master-slave pattern.

The associated three pins: CAS2 - CAS0 are outputs when the 8259A is used to master and are inputs when 8259A is used as slave.

The master puts out an identification code of 3 bits (8 devices) on the CAS0 to CAS2 lines and as per the code the corresponding slave is selected. Eg: If CAS2 - CAS0 have value 000 resp then 8259A slave 0 is selected.

* $\overline{SP}/\overline{EN}$ (Slave Program / Enable Buffer):
  - This is a dual function pin.
  - When in the buffered mode, it can be used as an output to control buffer transceivers ($\overline{EN}$).
  - When not in buffered mode, it can be used as an input to designate a master. ($SP = 1/SP = 0$)
  - Buffered and non-buffered mode of operation of 8259A can be specified during initialization.

Qn Important: How can you handle 18 interrupt requests using 8259A PICs?




Exp # Modes of operation of 8259A

There are various operating modes of 8259A which are as follows:

1. Fully nested Mode (FNM),
2. Specially Fully Nested Mode (SFNM),
3. Rotation Priority Mode
   - (a) Automatic Rotation
   - (b) Specific Rotation
4. Special Mask Mode.
5. Polled Mode.

1. Fully Nested Mode

- It is the default mode set after initialization i.e this mode is entered after initialization, unless another mode is programmed or changed through OCW.
- The interrupt requests are ordered in priority from 0 to 7 (0 highest and 7 lowest priorities).

2 Specially Fully Nested Mode (SFNM):
- used in cascade connection,
- allows further interrupt requests from slave
- SFNM is set up by ICW4 during initia

3 Rotation Priority Mode:
@ Automatic Rotation (Equal Priority Device Decode
- In this mode, a device after being serviced receives the lowest priority, so a device requesti an interrupt will have to wait, in the worst ca until each of 7 other devices are serviced al most once.
Example, if before rotation IR4 has the high priority & currently serviced.

| IS7 | IS6 | IS5 | IS4 | IS3 | IS2 | IS1 | IS0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

Now, After rotation (IR4 was serviced, all other priorities rotated correspondingly) as,

| IR7 | IR6 | IR5 | IR4 | IR3 | IR2 | IR1 | IR0 |
|---|---|---|---|---|---|---|---|
| 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 |

(b) Specific Rotation (Specific Priority)
- The programmer can change priorities by programming the bottom priority and thus fixing all other priorities i.e if IR5 is programmed as the bottom priority device, then IR6 will have the highest one.

- The set priority command is issued in OCW2, where R=1, S=1, L0-L2 is the binary priority level code of the bottom priority device.

3. Special Mask Mode:
   - In this mode, when mask bit is set in OCW1, it inhibits further interrupts at that level and enables interrupts from all other levels (lower as well as higher) that are not masked.
   - Thus, any interrupts may be selectively enabled or masked by loading the mask register.
   - The special mask mode is set by OCW3.

4. Polled Mask:
   - The poll command is issued by setting P="1" in OCW3.

| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
|---|---|---|---|---|---|---|---|
| I=1 | – | – | – | – | W2 | W1 | W0 |

Here, W0-W2 : Binary code of the highest priority level requesting service.
I=1, sets in Polled mode, if there is interrupt.

(PPI)

Date
Page

# 3. 8255A Programmable Peripheral Interface

- The 8255A is a widely used, programmable, parallel I/O device that provides three 8-bit input/output ports in one 40-pin IC-package.
- It can be programmed to transfer data under various conditions, from simple I/O to interrupt
- It is an important general-purpose I/O device th can be used with almost any microprocessor
- It has 24 I/O pins that can be grouped pri into two 8-bit parallel ports A and B with the remaining 8-bits as port C.
- The eight bits of port C can be used as an individual bits or can be grouped into two 4-bit ports: Port ~~C~~ $C_{UPPER}$ ($C_U$) & Port $C_{LOWER}$ ($C_L$).
- Basically the functions of 8255A can be classi into two modes:
  1. Bit Set/Reset (BSR) Mode; and
  2. I/O Mode
- The BSR mode is used to set or reset the bits in port C.
- The I/O mode is further divided into three modes:
  ① Mode 0, ② Mode 1, and ③ Mode 2
- In Mode 0, all port functions as simple I/O ports.
- In Mode 1, ports A and B use some bits from po as handshake signals. In the handshake mode, tw types of I/O data transfer can be implemented: status check and interrupt.

In Mode 2, port A can be setup for bidirectional data transfer using handshake signals, from port C and port B can be setup either in mode 0 or mode 1.

# Various Applications:
1. Printer Interface,
2. Keyboard and display interface,
3. A/D Converter and D/A Converter interface,
4. Floppy disk interface, etc.

# Why 8255A is used?
- to expand the I/O ports.
- no use of external combinational logic circuit.

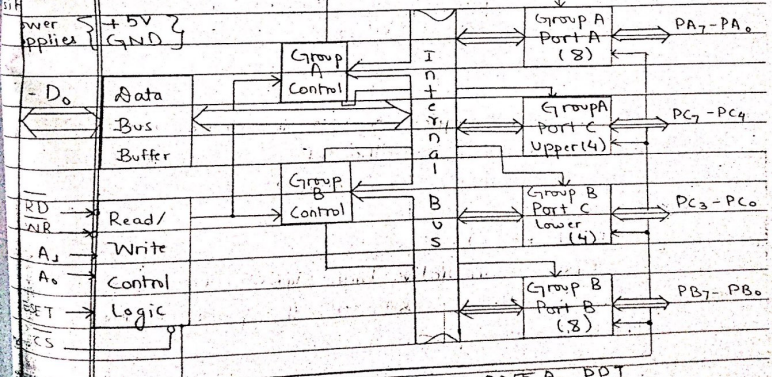



Fig: Functional Block Diagram of 8255A PPI

Internal Block Diagram of 8255A PPI

# Description

The different blocks included in the functional block diagram of 8255A are as follows:

1. DATA BUS BUFFER:

→ This is 8-bit bidirectional buffer used to interface the internal data bus of 8255A with system data bus of MPU.

2. READ/WRITE CONTROL LOGIC:

→ It accepts address and control signals from the microprocessor.

→ The control section has 6 lines which are follo

- $\overline{RD}$ (Read):

The control signal enables the Read operation. When this signal is low, the MPU reads data from a selected I/O port of 8255A.

- $\overline{WR}$ (Write):

The control signal enables the write operation. When this signal goes low, the MPU writes into a selected I/O Port or Control Register.

- RESET (Reset):

This is an active high signal that clears the control register and sets all the ports in the Input mode.

- $\overline{CS}$, $A_0$ and $A_1$ : Select

These are the device signals. $\overline{CS}$ is connected to decoded address and should be low to select 8255A chip and $A_0$ and $A_1$ lines are address lines generally connected to mpu address lines $A_0$ and $A_1$ respectively.

| $\overline{CS}$ | $A_1$ | $A_0$ | Selection |
|---|---|---|---|
| 0 | 0 | 0 | Port A |
| 0 | 0 | 1 | Port B |
| 0 | 1 | 0 | Port C |
| 0 | 1 | 1 | Control Register |
| 1 | x | x | 8255A is not selected |

3. Group A Control:
- This control block ~~contains~~ controls port A and port C upper
- It accepts control signals from the control word and forwards them to respective ports.

4. Group B Control:
- This control block controls port B and port C lower.
- It accepts control signals from the control word and forwards them to the respective ports.

5. Port A, Port B and Port C :
- These are 8-bit bidirectional ports
- They can be programmed to operate in various modes as follows:

| Port | Mode 0 | Mode 1 | Mode 2 |
|---|---|---|---|
| Port A | Yes | Yes | Yes |
| Port B | Yes | Yes | No (Mode 0 or |
| Port C | Yes | No (Handshake Signals) | No (Handshake Signals) |

# Control Word of 8255A PPI :

- The content of control register is called control word.
- It specifies I/O functions for each port.
- The control register can be accessed to write a control word when $A_1 A_0 = 11$. This register is not accessible for Read Operation.

# Control Word for I/O mode :-

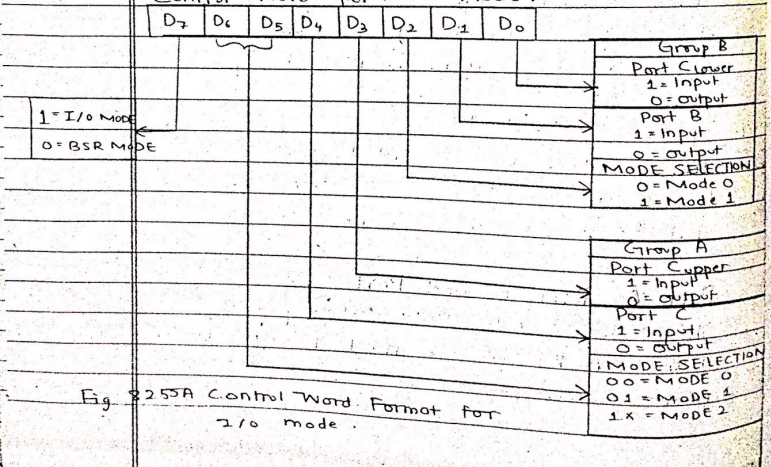



Fig. 8255A Control Word Format for I/O mode.

Date
Page 7

The various I/O modes of 8255A are as follows:

1. Mode 0 : Simple Input or output:

- In this mode, ports A and B are used as two simple I/O ports and port C as two 4-bit ports.
- Each port (or half port) in port C can be programmed to function as input or output port.
- The input/output features in mode 0 are as follows:
  1. Outputs are latched.
  2. Inputs are not latched.
  3. Ports donot have interrupt or handshake capability.
- A common example is the keyboard used for data entry.

2. Mode 1 : Input or output with Handshake:

- In mode 1, handshake signals are exchanged betn mpu and peripheral prior to data transfer.
- The features of this mode include the following:
  (a) Two ports (A and B) function as 8-bit I/O ports. They can be configured either as input port or output port.
  (b) Each port uses three lines from port C as handshake signals. The remaining two lines of port C can be used for simple I/O functions.
  (c) Input and output data are latched.
  (d) Interrupt logic is supported.
- If Port B is initialized in mode 1 for either input or output pins $PC_0$, $PC_1$ and $PC_2$ function as handshake lines.

- If Port A is initialized in mode 1 for either input o output, pins $PC_3$, $PC_4$ and $PC_5$ function as handsha
- Remaining lines $PC_6$ and $PC_7$ are available for in or output lines.

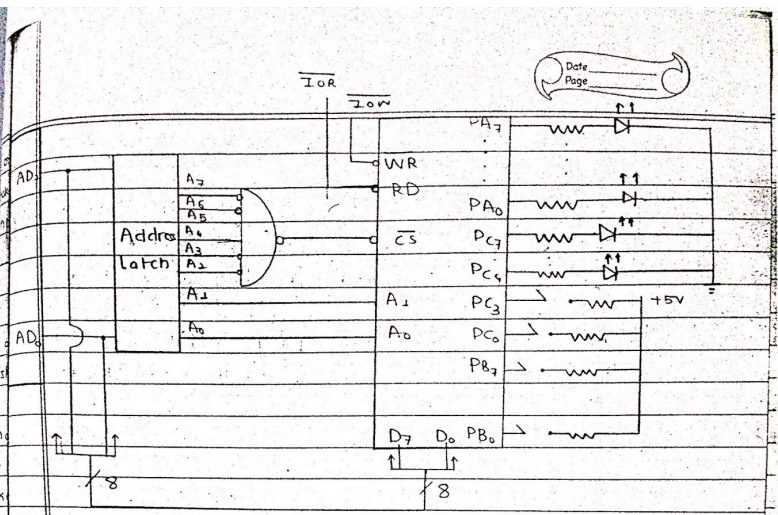
3 Mode 2: Bidirectional data transfer

- This mode is used primarily in applications such data transfer between two computers or floppy dis controller interface.
- In mode 2, port A can be configured as bidirection port and port B either in mode 0 or mode 1.
- Port A uses 5 signals from Port C for handshak
- The remaining three signals from Port C can be us for simple I/O or handshake for Port B.
- If port B is in mode 0, then $PC_0$, $PC_1$ and $PC_2$ a used for I/O, and if port B is in mode 1, then $PC_0$, $PC_1$ and $PC_2$ are used as handsha lines.

# 8255 Program:

Q.1 Write a program for 8255A PPI to read the DIP switches and display the reading from Port B at Port A and from Port C lower at Port C upper using the given circuit.

(Assume: Port Address: Port A = 30H)




Solution :→

Basic Addressing :

here, Port B and Port C lower = Input

Port A and Port C upper = output

Now, Control word for this condition is,

| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ | |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | = 83H |

We have Address calculation from above circuit can be done as :

| $A_7$ | $A_6$ | $A_5$ | $A_4$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ | Port Selection | Port Address |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | Port A | 30 H |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | Port B | 31 H |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | Port C | 32 H |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | Control Register | 33 H |

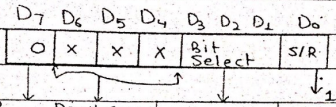
Now, Required program is as follows:

```
MVI A, 83H ; Load control word in Accumulator
OUT 33H ; Store control word in control Register
IN 31H ; Read switches at Port B.
OUT 30H ; Display the reading at Port A.
IN 32H ; Read switches at Port C.
ANI 0FH ; Mask the upper 4-bits of Port C.
RLC ; Rotate and place data in upper half of
      Accumulator
RLC
RLC
RLC
OUT 32H ; Display data at Port C upper.
HLT ; Terminate the program.
```

4 BSR (Bit Set/Reset) mode:

- Any bit of port-C can be set or reset using this mode.
- BSR mode doesn't alter any previously transmitted control word with bit $D_7 = 1$.
- In BSR mode, individual bits of port C can be used for applications such as on/off switch.

# BSR Control Word:




| | |
|---|---|
| | 000 = Bit 0 |
| | 001 = Bit 1 |
| | 010 = Bit 2 |
| | 011 = Bit 3 |
| | 100 = Bit 4 |
| | 101 = Bit 5 |
| | 110 = Bit 6 |
| | 111 = Bit 7 |

Fig: 8255A Control Word for BSR mode

Qn: Write BSR Control Word sub-routine to set bits $PC_7$ and $PC_3$ and reset them after 10 ms. (Assume, control register address = 83H and delay subroutine is available)

Solution: Here,

BSR Control Words:

| | $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ | |
|---|---|---|---|---|---|---|---|---|---|
| To set bit $PC_7$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | = 0FH |
| To reset bit $PC_7$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | = 0EH |
| To set bit $PC_3$ | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | = 07H |
| To reset bit $PC_3$ | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | = 06H |

Now,

Port address of Control Register = 83H

Required Subroutine is as follows.

```
BSR: MVI A, OFH ; Load byte in Accumulator to set PC
     OUT 83H ; Store control word to set PC7 on control
     MVI A, 07H ; Load byte in accumulator to set PC
     OUT 83H ; Store control word to set PC3 on control re
     CALL Delay ; Call to available subroutine 'Delay' for
                                              1oms delay.
     MVI A, OEH ; Load byte in accumulator to reset PC7.
     OUT 83H ; Store control word to reset PC7 on control
     MVI A, 06H ; Load byte in accumulator to reset PC3.
     OUT 83H ; Store control word to reset PC3 on control r
     RET ; Return to main program.
```

## 8237 DMA Controller:

Direct Memory Access (DMA):

- DMA is an I/O technique commonly used for high speed for data transfer. For example, data transfer b system memory and a floppy disk.
- In status check I/O and interrupt I/O, data transf is relatively slow because each instruction needs to be fetched and executed.
- In DMA, mPU releases the control of the buses to device called DMA controller.
- The DMA Controller manages data transfer betn memory and a peripheral under its control, thus bypassing the mPU.

DMA Transfer uses two signals: HOLD and HLDA in 8085 microprocessor.

In Normal operation, DMA controller acts as a slave and MPU treats it as normal peripheral device but during DMA Operation, DMA Controller acts as a master bypassing CPU.

## Concept of DMA:-

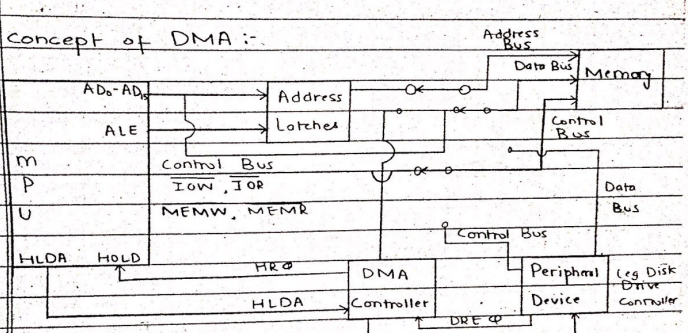



Fig: Block Diagram showing DMA Transfer.

Explanation.

The sequence of DMA Transfer as shown in above block diagram are as explained below:

1. Originally, microprocessor is connected to memory as shown in figure above with switches closed for address, data and control buses to MPU. When pheripheral wants to transfer data using DMA Transfer, it sends DMA Request, DREQ signal to the DMA Controller.

2 If the input (channel) of DMA Controller is unmasked, the DMA Controller will send a hold-request (HRQ) signal to the microprocessor's HOLD input.

3 The MPU finishes the current machine cycle and floats its buses, sending out hold-acknowledge signal, HLDA to the DMA Controller.

4 When DMA Controller receives HLDA signal, it will send a control signal which connects the three bus switches to DMA Controller and the peripheral. This disconnects the processor from buses and connects DMA Controller to ~~have~~ the buses. Now, DMA Controller sends out the address of the byte to be transferred and sends out DMA-Acknowledge (DACK) signal to the peripheral device.

5 Then the DMA transfer begins and finally when the data transfer is complete, the DMA Controller unasserts its HOLD-Request signal i.e HRQ=0 and releases the buses to MPU.

# The 8237 DMA Controller:

- The 8237 is a programmable DMA Controller packaged in 40 pin IC package.
- It has 4-independent channels with each channels capable of transferring 64KB.
- It must interface with two types of devices: the MP and the peripherals such as floppy disks and be able to operate in two modes: master mode (DMA Transfer) and a slave mode (during normal op
- Many of the signals that are input in the I/O mode becomes output in the processor mode.

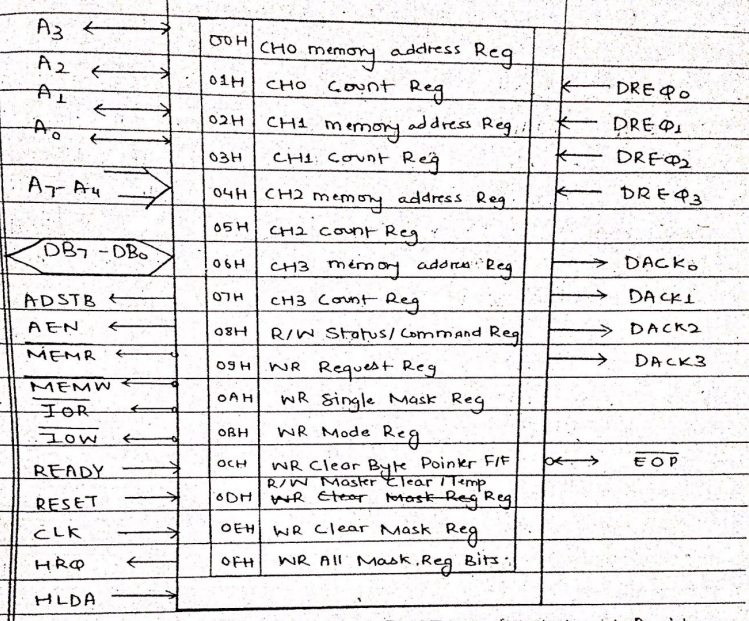



Fig: Block Diagram of 8237A with Internal Registers.

# DMA CHANNELS AND INTERACTIONS:

- The 8237 has 4 independent channels: CH0 to CH3
- Internally, two 16-bit registers are associated with each channel:

@Count Register:

- It is used to load a count of the number of bytes to be copied.

(6) Memory Address Register (MAR):

- It is used to load the starting address of the byte to be copied.
- The address of these registers are determined by address lines : $A_3$ to $A_0$ and the chip select ($\overline{CS}$) signal.
- For example, Address 0000 on Address lines $A_3$-$A_0$ selects CH0 memory Address Register (MAR) and address 0001 on Address lines $A_3$-$A_0$ selects CH0 Count Register.
- Similarly, all the remaining registers are selected in sequential order.
- The address of internal registers range from 00H to 0FH.

| | $A_7$ | $A_6$ | $A_5$ | $A_4$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ | |
|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | = 00H (CH0 |
| | | | | | ↓ | . . . | | ↓ | |
| Chip Select ($\overline{CS}$) | | | | | 1 | 1 | 1 | 1 | = 0FH (m |

# DMA SIGNALS:

- The different signals request to understand DMA Transfer are as explained below:

DREQ0 - DREQ3 (DMA Request)

→ These are 4 independent asynchronous signal inputs to the DMA Channels from peripherals such as floppy disks, hard disks, etc.
→ To obtain DMA service, a request is generated by activating the DREQ lines of the channel.

Date
Page

DACK0 - DACK3 (DMA Acknowledge)

→ These are output lines to inform the individual peripherals that a DMA is granted.

→ DREQ and DACK are equivalent to handshake signals in I/O device.

AEN and ADSTB (Address Enable and Address Strobe):

→ These are active high output signals that are used to latch higher-order address byte to generate 16-bit address.

$\overline{MEMW}$ and $\overline{MEMR}$ (Memory Write and Memory Read)

→ These are output signals used during DMA cycle to write and read data from memory.

$\overline{IOW}$ and $\overline{IOR}$ (Input output Write and I/O Read)

→ These are output signals used during DMA cycle to write & read data to and from I/O device.

$A_3 - A_0$ and $A_7 - A_4$ (Address lines):

→ $A_3 - A_0$ are d bidirectional Address Lines. They are used as inputs to access control registers.

→ During DMA cycles, these lines are used at output lines to generate a low order address that is combined with the remaining address lines $A_7 - A_4$.

HRQ and HLDA (Hold Request and Hold Acknowled

→ HRQ is an output signal used to request the mpu control of the system bus.

→ After receiving the HRQ, the mpu completes the bus cycle in process and issues the HLDA signal.

# Modes of Operation of 8237:

→ The 8237 chip operates in two major cycles 1. Idle Cycle and 2. Active Cycle.

1. Idle Cycle.

→ When no channel is requesting service, the 8237 will enter in idle cycle. In this cycle, the 8237 sample DREQ lines every clock cycle to dekm if any channel is requesting service.

2. Active Cycle.

→ When 8237 is in idle cycle and channel requests DMA service, the device will output an HRQ to th microprocessor and enter the active cycle. In active cycle, the DMA transfer occurs in one of the 4 modes.

1. Single Transfer Mode (or Byte Transfer Mode
2. Block Transfer Mode.
3. Demond Transfer Mode, and
4. Cascade Mode.

1 Single (Byte) Transfer Mode :

→ In this mode, the device is programmed to make only one transfer ie only one byte is transferred. The word count will be decremented by one and the address is also incremented or decremented following each transfer.

2 Block Transfer Mode :

→ In this mode, the device is activated by DREQ to continue making transfer the multiple bytes, until Terminal Count (TC) becomes zero or an external End of Process (EOP) is encountered.

3 Demand Transfer Mode :

→ In this mode, the device is programmed to continue making transfer until a TC becomes zero or EOP = 0 or until DREQ lines goes inactive. Thus, transfer may continue until the I/O device has exhausted its data capacity.

4 Cascade Mode :

→ This mode is used to cascade more than one 8237 together for simple system expansion.

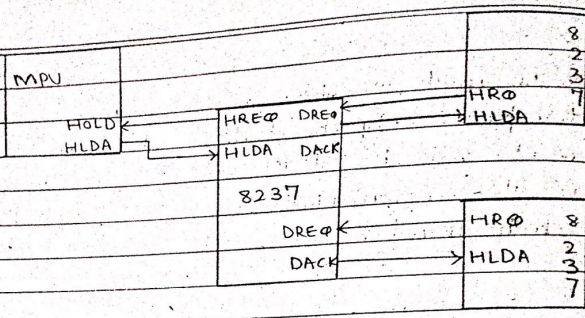



Fig: Cascading multiple 8237

5. # 8254 Programmable Interval Times (PIT):

→ The Intel 8254 is a counter/timer device designed to solve the common timing & counter problems in microprocessor system design.

→ It provides three independent 16-bit counters, each capable of handling clock inputs upto 10MHz.

→ Instead of setting up timing loops in software the programmer configures the 8254 to match his/her requirements and programs one of the counters for the desired delay. After the desired delay, the 8254 will interrupt the CPU.

→ It operates in +5V regulated power supply and 24 signal pins.

→ The 8254 is the advanced version of 8253, and includes the feature read back command that is not available in 8253.

# Applications :

- Real time clock ,
- To generate accurate time delays ,
- As an event - counter ,
- Square wave generator ,
- Programmable Rate Generator ,
- Complex waveform Generator .

# Block Diagram of 8254, PIN

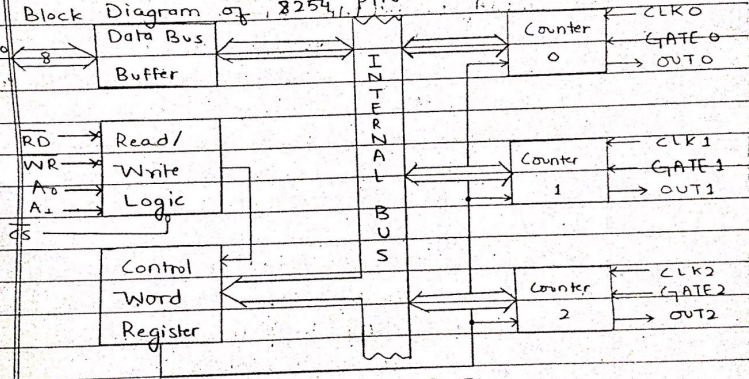



Figure : Block Diagram of 8254

# Description ;

1. Data Bus Buffer

→ This 3-state, bidirectional 8-bit buffer is used to interface the 8254 to the system bus of MPU.

2 Read / Write Logic.

→ The Read/Write Logic accepts iputs from the system bus and generates control signals for the other functional blocks of the 8254.

→ This section has five signals, : $\overline{RD}$ (Read), $\overline{WR}$ (Write), $\overline{CS}$ (Chip Select) and address lines $A_0$ and $A_1$.

→ $\overline{RD}$ (Read) and $\overline{WR}$ (Write) are connected to $\overline{IOR}$ (Input output Read) and $\overline{WR}$ (Write) are connected to $\overline{MEMR}$ (Memory Read) and $\overline{MEMW}$ (Memory Write) respectively in Memory mapped IO.

→ Address lines $A_{0_1}$ and $A_0$ are connected to the address lines $A_1$ and $A_0$ of MPU and $\overline{CS}$ is used to select the device connected to the decoded address.

→ $\overline{CS}$ is used to select the device. If $\overline{CS} = 0$, the 8254 chip is selected for operation.

→ The counters and control word register are selected according to signals on the lines $A_1$ and $A_0$ as shown in table below:

| $A_1$ | $A_0$ | Selection |
|---|---|---|
| 0 | 0 | Counter 0 |
| 0 | 1 | Counter 1 |
| 1 | 0 | Counter 2 |
| 1 | 1 | Control Register |

3 Control Word Register

→ The Control Word Register is selected by the Read/Write Logic when $A_1 A_0 = 11$.

→ If the MPU does the 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.

4 Counters

→ 8254 includes three identical 16-bit counters: Counter 0, Counter 1 and Counter 2 that can operate independently in any six modes of operation.

→ To operate as a counter, a 16-bit count is loaded into its register.

→ The counter can count either in BCD or Binary.

# Control Word Format of 8254:

→ The 8-bit control word Format in Control Register of 8254 is as shown below:

| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
|---|---|---|---|---|---|---|---|
| SC1 | SC0 | RW1 | RW0 | $M_2$ | $M_1$ | $M_0$ | BCD |

where, SC → Select Counter

| SC1 | SC2 | Selection |
|---|---|---|
| 0 | 0 | Counter 0 |
| 0 | 1 | Counter 1 |
| 1 | 0 | Counter 2 |
| 1 | 1 | Read-Back command |

RW → Read / Write.

| RW1 | RW2 | |
|---|---|---|
| 0 | 0 | Counter Latch Command |
| 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. |

M → Mode

| $M_2$ | $M_1$ | $M_0$ | |
|---|---|---|---|
| 0 | 0 | 0 | Mode 0 |
| 0 | 0 | 1 | Mode 1 |
| X | 1 | 0 | Mode 2 |
| X | 1 | 1 | Mode 3 |
| 1 | 0 | 0 | Mode 4 |
| 1 | 0 | 1 | Mode 5 |

BCD (X=0)

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

Fig: Control Word Format of 8254 PIT

Q. Generate an 8254 control word to operate it as a BCD counter in mode 2 using counter 0 with 16-bit value of count.

Soln:

| $D_7$ | $D_6$ | $D_5$ | $D_4$ | $D_3$ | $D_2$ | $D_1$ | $D_0$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |

∴ Control Word = 35H

# Modes of Operation of 8254:

→ 8254 can operate in 6 different values of operation.

**Mode 0: Interrupt on Terminal Count**

- Mode 0 is typically used for event counting.
- 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 in the counter.
- GATE = 1 enables counting and GATE = 0 disables counting. GATE has no effect on OUT.

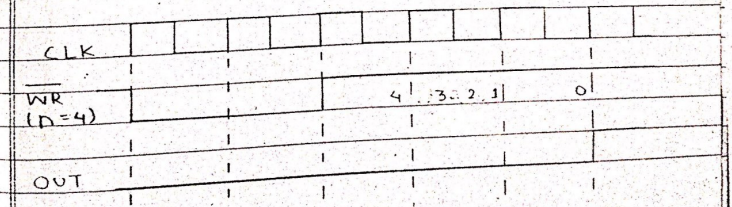



Fig. Mode 0 Timing Diagram

## Mode 1 : Hardware re-triggable one-Shot :

- OUT is initially high.
- OUT will go low on the CLK pulse following a trigger to begin the one-shot pulse, and will remain low until counter reaches zero.
- OUT will then go high and remain high until the CLK pulse after the next trigger.
- In this mode, GATE signal is used to trigger.
- The one-shot is re-triggerable, hence OUT will remain low for N CLK pulses after any trigger.

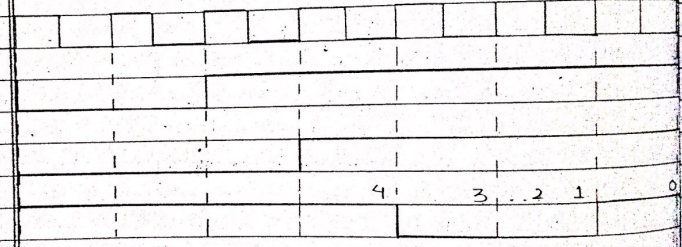



Fig: Mode 1 Timing Diagram

## Mode 2 : Rate Generator

→ This mode 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 ~~high~~ low for one clock pulse.

→ OUT then goes high again, the counter re-loads the initial count and the process is repeated.

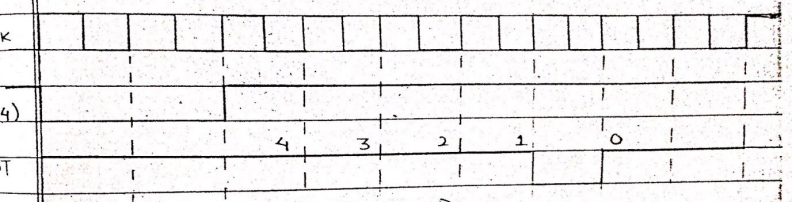



Fig: Mode 2 Timing Diagram.

## Mode 3 : Square Wave Generator.

→ Mode 3 is typically used for Baud Rate Generation.

→ In this mode, when a count is loaded, the OUT goes high.

→ The count is decremented by every two clock cycle and when it reaches zero OUT goes low and the count is released again.

→ This is repeated continuously, thus a square wave with period equal to the period of count is generated.

i.e frequency of square wave = $\frac{\text{(Frequency of clock)}}{\text{count}}$

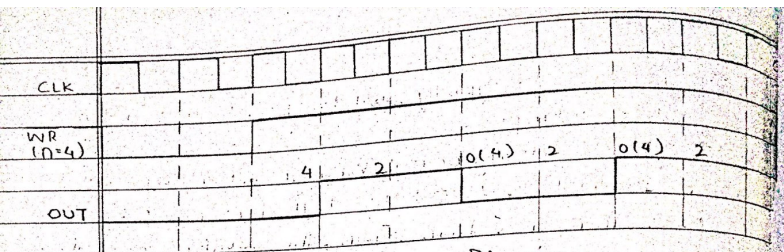



Fig Mode 3 : Timing Diagram

Mode 4 : Software Triggered Strobe.

→ In this mode, OUT is initially high.

→ OUT goes low for one clock period at the end of the count.

→ The count must be recorded for subsequent outputs.

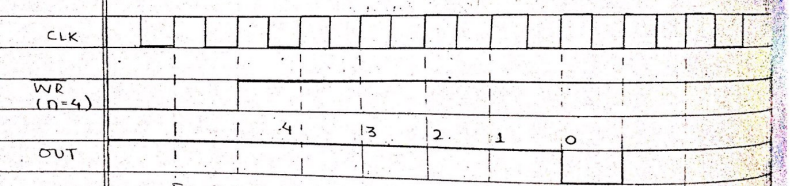



Fig Mode 4 : Timing Diagram

Mode 5 : Hardware Triggered Strobe.

→ This mode is similar to mode 4 except that it is triggered by the rising pulse at the GATE

→ Initially, OUT is high and when GATE pulse is triggered the count begins.

→ At the end of the count, the OUT goes low for one clock cycle.

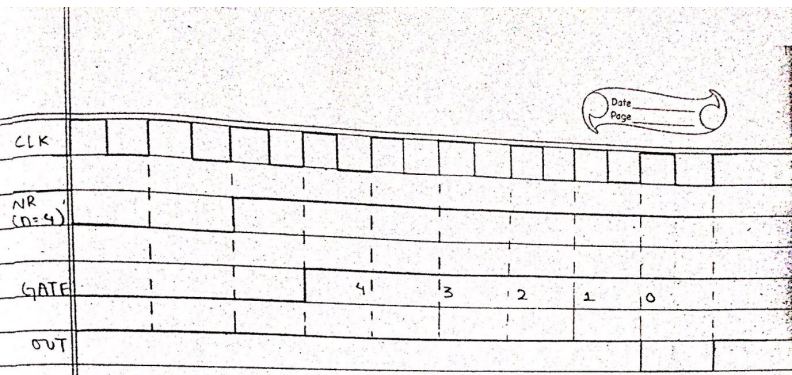



Fig: Mode 5 Timing Diagram