```
                                ; (stack) for destination series
NEXT: POP AX                    ; Get the first word from series
      MOV DX, SP                ; Save source stack pointer
      MOV SP, BX                ; Get destination stack pointer
      PUSH AX                   ; Save AX to stack
      MOV BX, SP                ; Save destination stack pointer
      MOV SP, DX                ; Get source stack pointer for
                                ; the next number
      DCR CL                    ; Decrement count
      JNZ NEXT                  ; If count is not zero, go to the next num
      MOV AH, 4CH               ; Else, return to DOS
      INT 21H                   ; prompt
CODE ENDS
      END   START
```

Program 4.2 *Listing*

The above program may also be written using just simple data transfer instructions. Here we have used the stack to change the order of data words.

It is a common practice to push all the registers to the stack at the start of a subroutine and pop it at the end of the subroutine so that the original contents are retrieved. Thus during the subroutine execution, all the registers except SP and SS are free for use.

The stack mechanism is also used in case of interrupt service routines to store the instruction pointer and code segment of the return address. The maximum size of stack segment like a general segment is 64 K. The stack size for a particular program may be set using DB or DW directive as per the requirements, as shown in the above program.

4.3 INTERRUPTS AND INTERRUPT SERVICE ROUTINES

The dictionary meaning of the word 'interrupt' is to break the sequence of operation. While the CPU is executing a program, an 'interrupt' breaks the normal sequence of execution of instructions, diverts its execution to some other program called *Interrupt Service Routine* (ISR). After executing ISR, the control is transferred back again to the main program which was being executed at the time of interruption.

Suppose you are reading a novel and have completed up to page 100. At this instant, your younger brother distracts you. You will somehow mark the line and the page you are reading, so that you may be able to continue after you attend to him. Say you have marked page number 101. You will now go to his room to solve his problem. While you are helping him a friend of yours comes and asks you for a textbook. Now, there are two options in front of you. The first is to make the friend wait till you complete serving your brother, and thereafter you serve his request. In this, you are giving less priority to your friend. The second option is to ask your brother to wait; remember the solution of his problem at the intermediate state; serve the friend; and after the friend is served, continue with the solution that was in the intermediate state. In this case, it may be said that you have given higher priority to your friend. After serving both of them, again you may continue reading from page 101 of the novel. Here, first you are interrupted by your brother. While you are serving your brother, you are again interrupted by a friend. This type of sequence of appearance of interrupts is called nested interrupt, i.e. interrupt within interrupt.

Scanned by CamScanner

Whenever a number of devices interrupt a CPU at a time, and if the processor is able to handle them properly, it is said to have *multiple interrupt processing capability*. For example, 8085 has five hardware interrupt pins and it is able to handle the interrupts simultaneously under the control of software. In case of 8086, there are two interrupt pins, viz. NMI and INTR. The NMI is a *nonmaskable* interrupt input pin which means that any interrupt request at NMI input cannot be masked or disabled by any means.The INTR interrupt, however, may be masked using the Interrupt Flag (IF). The INTR, further, is of 256 types. The INTR types may be from 00 to FFH (or 00 to 255). If more than one type of INTR interrupt occurs at a time, then an external chip called programmable interrupt controller is required to handle them. The same is the case for INTR interrupt input of 8085. Interrupt Service Routines (ISRs) are the programs to be executed by interrupting the main program execution of the CPU, after an interrupt request appears. After the execution of ISR, the main program continues its execution further from the point at which it was interrupted.

4.4 INTERRUPT CYCLE OF 8086/8088

Broadly, there are two types of interrupts. The first out of them is *external interrupt* and the second is *internal interrupt*. In external interrupt, an external device or a signal interrupts the processor from outside or, in other words, the interrupt is generated outside the processor, for example, a keyboard interrupt. The internal interrupt, on the other hand, is generated internally by the processor circuit, or by the execution of an interrupt instruction. The examples of this type are divide by zero interrupt, overflow interrupt, interrupts due to INT instructions, etc.

Suppose an external device interrupts the CPU at the interrupt pin, either NMI or INTR of the 8086, while the CPU is executing an instruction of a program. The CPU first completes the execution of the current instruction. The IP is then incremented to point to the next instruction. The CPU then acknowledges the requesting device on its $\overline{\text{INTA}}$ pin immediately if it is a NMI, TRAP or Divide by Zero interrupt. If it is an INT request, the CPU checks the IF flag. If the IF is set , the interrupt request is acknowledged using the $\overline{\text{INTA}}$ pin. If the IF is not set, the interrupt requests are ignored. Note that the responses to the NMI, TRAP and Divide by Zero interrupt requests are independent of the IF flag. After an interrupt is acknowledged , the CPU

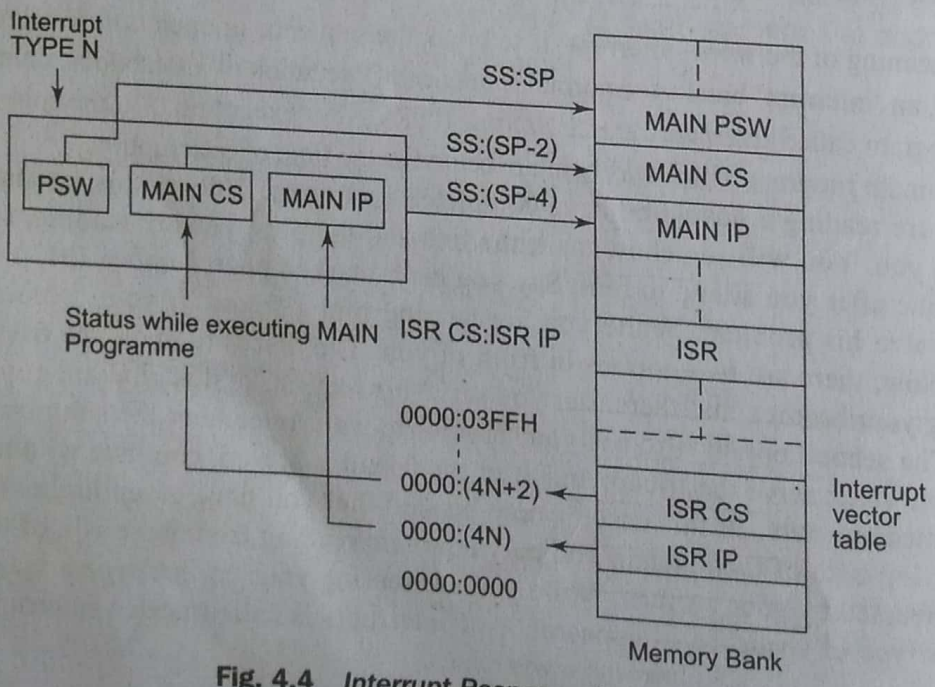


Fig. 4.4 *Interrupt Response Sequence*

Scanned by CamScanner

computes the vector address from the type of the interrupt that may be passed to the interrupt structure of the CPU internally (in case of software interrupts, NMI, TRAP and Divide by Zero interrupts) or externally, i.e. from an interrupt controller in case of external interrupts. (The contents of IP and CS are next pushed to the stack. The contents of IP and CS now point to the address of the next instruction of the main program from which the execution is to be continued after executing the ISR. The PSW is also pushed to the stack.) The Interrupt Flag (IF) is cleared. The TF is also cleared, after every response to the single step interrupt. The control is then transferred to the interrupt service routine for serving the interrupting device. The new address of ISR is found out from the interrupt vector table. The execution of the ISR starts. If further interrupts are to be responded to during the time the first interrupt is being serviced, the IF should again be set to 1 by the ISR of the first interrupt. If the interrupt flag is not set, the subsequent interrupt signals will not be acknowledged by the processor, till the current one is completed. The programmable interrupt controller is used for managing such multiple interrupts based on their priorities. At the end of ISR the last instruction should be IRET. When the CPU executes IRET , the contents of flags, IP and CS which were saved at the start by the CALL instruction are now retrieved to the respective registers. The execution continues onwards from this address, received by IP and CS.

We now discuss how the 8086/88 finds out the address of an ISR. Every external and internal interrupt is assigned with a type (N), that is either implicit (in case of NMI, TRAP and divide by zero) or specified in the instruction INT N (in case of internal interrupts). In case of external interrupts, the type is passed to the processor by an external hardware like programmable interrupt controller. In the zeroth segment of physical

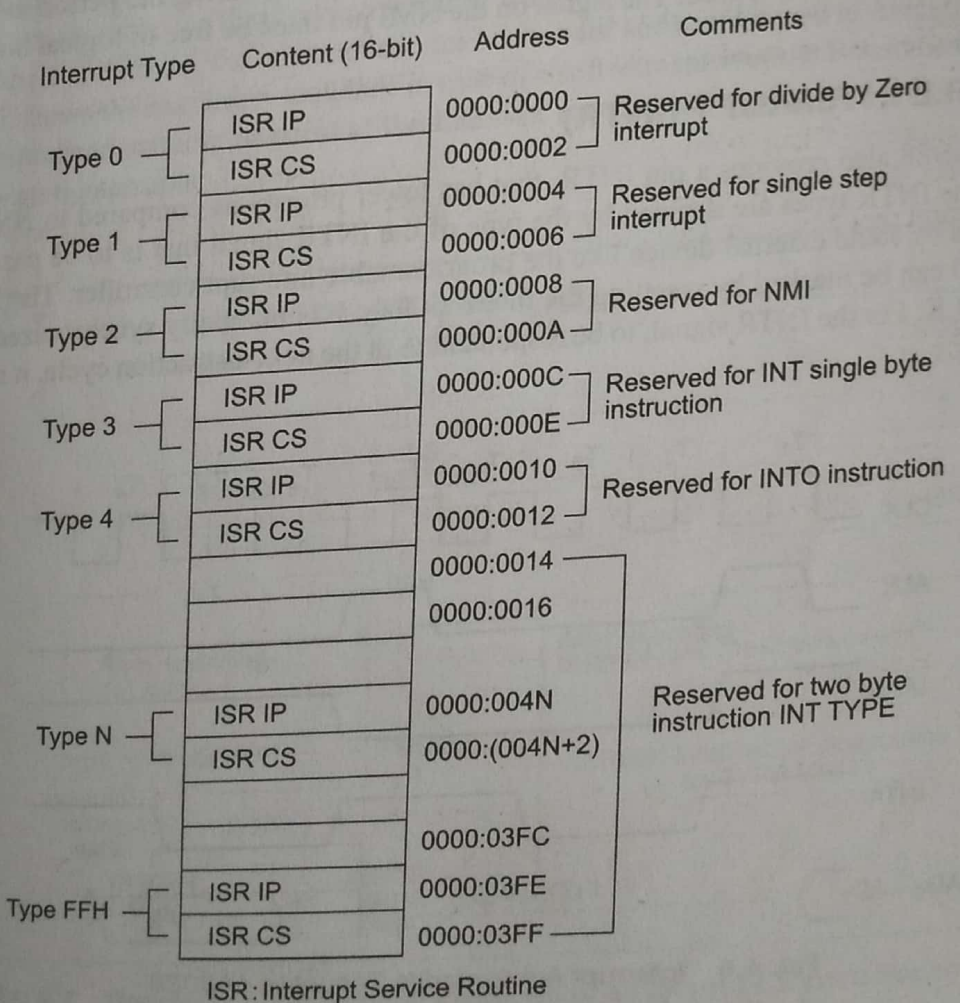


Fig. 4.5 *Structure of Interrupt Vector Table of 8086/88*

Scanned by CamScanner

address space, i.e. CS = 0000, Intel has reserved 1,024 locations for storing the interrupt vector table. The 8086 supports a total of 256 types of the interrupts, i.e. from 00 to FFH. Each interrupt requires 4 bytes, i.e. two bytes each for IP and CS of its ISR. Thus a total of 1,024 bytes are required for 256 interrupt types, hence the interrupt vector table starts at location 0000:0000 and ends at 0000:03FFH. The interrupt vector table contains the IP and CS of all the interrupt types stored sequentially from address 0000:0000 to 0000:03FF H. The interrupt type N is multiplied by 4 and the hexadecimal multiplication obtained gives the offset address in the zeroeth code segment at which the IP and CS addresses of the interrupt service routine (ISR) are stored. The execution automatically starts from the new CS:IP.

Figure 4.4 shows the interrupt sequence of 8086/88 and Fig. 4.5 shows the structure of interrupt vector table.

4.5 NON MASKABLE INTERRUPT

The processor 8086/88 has a Non-Masksable Interrupt input pin (NMI), that has the highest priority among the external interrupts. TRAP(Single Step-Type 1) is an internal interrupt having the highest priority amongst all the interrupts except the Divide By Zero (Type0) exception. The NMI is activated on a positive transition (low to high voltage). The assertion of the NMI interrupt is equivalent to an execution of instruction INT 02, i.e. Type 2 INTR interrupt.

The NMI pin should remain high for at least two clock cycles and need not synchronized with the clock for being sensed. When the NMI is activated, the current instruction being executed is completed and then the NMI is served. In case of string type instructions, this interrupt will be served only after the complete string has been manipulated. Another high going edge on the NMI pin of 8086, during the period in which the first NMI is served, triggers another response. The signal on the NMI pin must be free of logical bounces to avoid erratic NMI responses.

4.6 MASKABLE INTERRUPT (INTR)

The processor 8086/88 also provides a pin INTR, that has lower priority as compared to NMI. Further the priorities within the INTR types are decided by the type of the INTR signal that is to be passed to the processor via data bus by some external device like the programmable interrupt controller. The INTR signal is level triggered and can be masked by resetting the interrupt flag. It is internally synchronized with the high transition of the CLK. For the INTR signal, to be responded to in the next instruction cycle, it must go high in

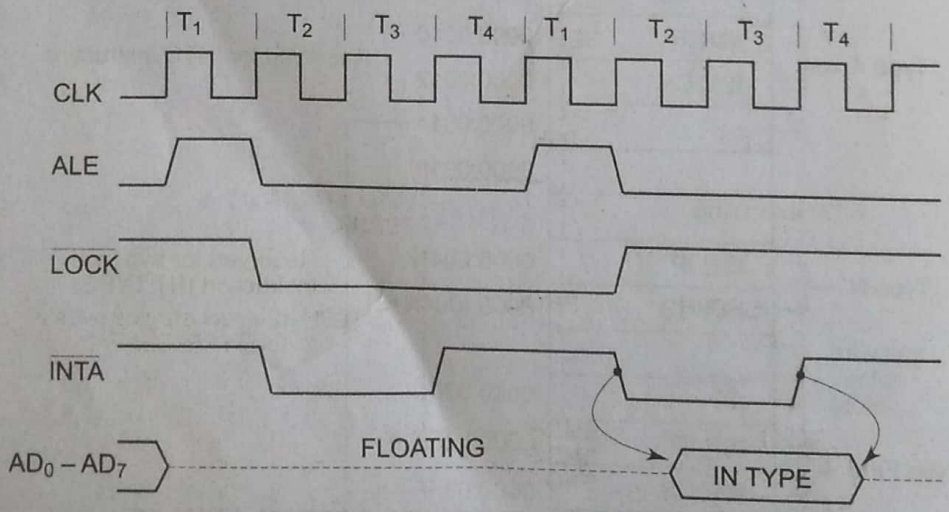


Fig. 4.6 *Interrupt Acknowledge Sequence of 8086*

Scanned by CamScanner

the last clock cycle of the current instruction or before that. The INTR requests appearing after the last clock cycle of the current instruction will be responded to after the execution of the next instruction. The status of the pending interrupts is checked at the end of each instruction cycle.

If the IF is set, the processor is ready to respond to any INTR interrupt if the IF is reset, the processor will not serve any interrupt appearing at this pin. However, once the processor responds to an INTR signal, the IF is automatically reset. If one wants the processor to further respond to any type of INTR signal, the IF should again be set. The interrupt acknowledge sequence is as shown in Fig. 4.6.

Suppose an external signal interrupts the processor and the pin $\overline{LOCK}$ goes low at the trailing edge of the first ALE pulse that appears after the interrupt signal preventing the use of bus for any other purpose.

The pin $\overline{LOCK}$ remains low till the start of the next machine cycle.

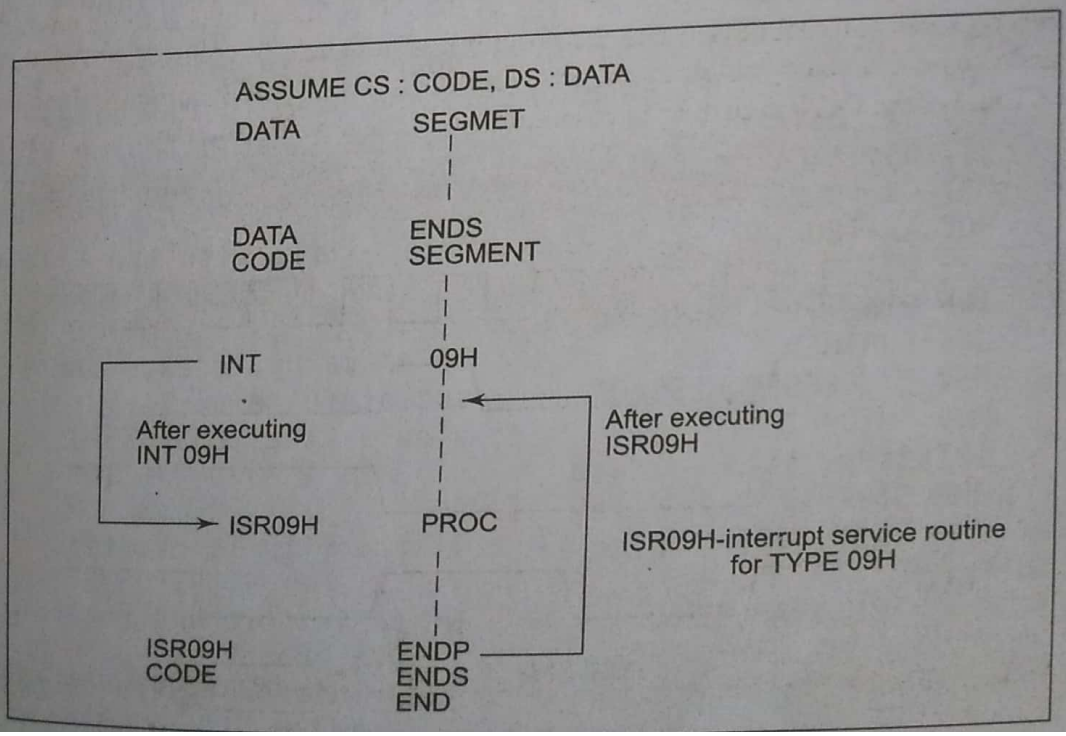
With the trailing edge of $\overline{LOCK}$, the $\overline{INTA}$ goes low and remains low for two clock states before returning back to the high state.

It remains high till the start of the next machine cycle, i.e. next trailing edge of ALE.

Then $\overline{INTA}$ again goes low, remains low for two states before returning to the high state.The first trailing edge of ALE floats the bus AD_0-AD_7 , while the second trailing edge prepares the bus to accept the type of the interrupt. The type of the interrupt remains on the bus for a period of two cycles.

4.7 INTERRUPT PROGRAMMING

While programming for any type of interrupt, the programmer must, either externally or through the program, set the interrupt vector table for that type preferrably with the CS and IP addresses of the interrupt service routine. The method of defining the interrupt service routine for software as well as hardware interrupt is the same. Figure 4.7 shows the execution sequence in case of a software interrupt. It is assumed that the interrupt vector table is initialised suitably to point to the interrupt service routine. Figure 4.8 shows the transfer of control for the nested interrupts.



Scanned by CamScanner

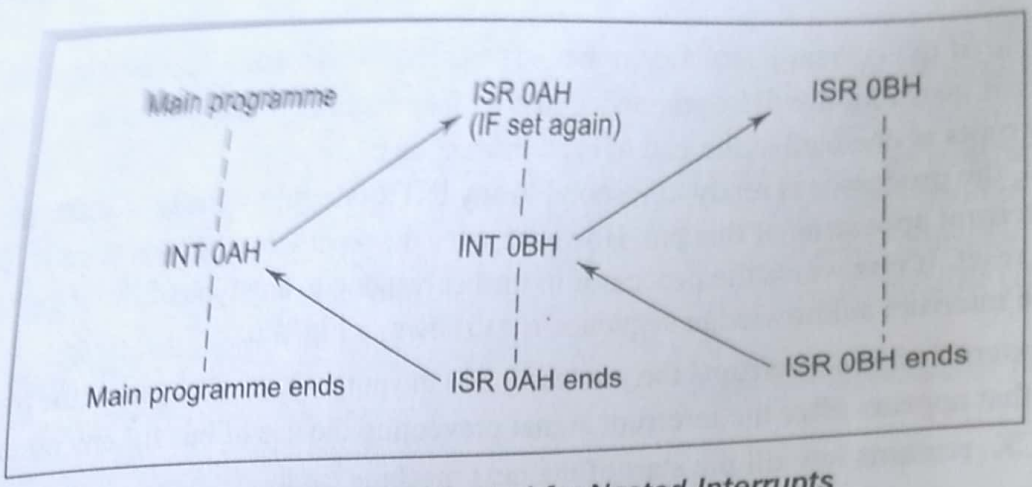


Fig. 4.8 *Transfer of Control for Nested Interrupts*

Program 4.3

Write a program to create a file RESULT and store in it 500H bytes from the memory block starting at 1000:1000, if either an interrupt appears at INTR pin with Type 0AH or an instruction equivalent to the above interrupt is executed.

Note: Pin IRQ_2 available at IO channel of PC is equivalent to Type 0AH interrupt.

```
ASSSUME CS : CODE, DS : DATA
DATA        SEGMENT
    FILENAME     DB "RESULT", "$"
    MESSAGE DB "FILE WASN'T CREATED SUCCESSFULLY",0AH,0DH,"$"
DATA        ENDS
CODE        SEGMENT
START:      MOV AX, CODE
            MOV DS, AX                  ; Set DS at CODE for setting IVT.
            MOV DX, OFFSET ISR0A        ; Set DX at the offset of ISR0A.
            MOV AX,250AH                ; Set IVT using function value 250AH
                                                  in AX
            INT 21H                     ; under INT21H.
            MOV DX, OFFSET FILENAME     ; Set pointer to Filename.
            MOV AX, DATA                ; Set the DS at DATA for Filename
            MOV DS, AX
            MOV CX, 00H
            MOV AH, 3CH                 ; Create file with the File name
                                                  'RESULT'.
            INT 21H
            JNC FURTHER                 ; If no carry, create operation is
            MOV DX,OFFSET MESSAGE       ; successful else
            MOV AH,09H                  ; display the MESSAGE.
            INT 21H
            JMP STOP
FURTHER :       INT 0AH                 ; If the file is created
successfully,
STOP :          MOV AH,4CH              ; write into it and return
                INT 21H                 ; to DOS prompt.
                                        ; This interrupt service routine
                                              writes 500 bytes into the
```

Scanned by CamScanner

Prepare the EXE file of the above program as usual. Execute it at DOS prompt that will hang the system. Now apply a pulse to IRQ_2 pin. The message 'IRT_2 is OK' is displayed on the screen. Then apply a pulse to IRQ_3 pin of IO channel. The message 'IRT3 is OK' is displayed on screen.

4.8 PASSING PARAMETERS TO PROCEDURES

Procedures or subroutines may require input data or constants for their execution. Their data or constants may be passed to the subroutine by the main program (host or calling program) or some subroutine may access readily available data of constants available in memory.

Generally, the following techinques are used to pass input data/parameter to procedures in assembly language programs.

(i) Using global declared variable
(ii) Using registers of CPU architecutre
(iii) Using memory locations (reserved)
(iv) Using stack
(v) Using PUBLIC & EXTRN.

Besides these methods if a procedure is interactive it may directly accept inputs from input devices.

As discussed in Chapter 2, a variable or a parameter label may be declared global in the main program and the same variable or parameter label can be used by all the routines or procedures of the application. Examples of passing parameters.

Example 4.1

```
ASSUME CS:CODE1,DS:DATA
DATA SEGMENT
NUMBER EQU 77H GLOBAL
DATA ENDS
CODE1 SEGMENT
    START :          MOV AX,DATA
```

Scanned by CamScanner

```
                MOV DS,AX
                  •
                  •
                MOV AX,NUMBER
                  •
                CODE1 ENDS
ASSUME CS:CODE2
CODE2 SEGMENT
        MOV AX,DATA
        MOV DS,AX
        MOV BX,NUMBER

  CODE2 ENDS
        END START
```

The CPU general purpose registers may be used to pass parameters to the procedures. The main program may store the parameters to be passed to the procedure in the available CPU registers and the procedure may use the same register contents for execution. The original contents of the used CPU register may change during execution of the procedure. This may be avoided by pushing all the register content to be used to the stack sequentially at the start of the procedure and by poping all the register contents at the end of the procedure in opposite sequence.

Example 4.2

```
ASSUME CS:CODE
CODE SEGMENT
  START :       MOV AX,5555H
                MOV BX,7272H
                  •
                  •
                CALL PROCEDURE1
                  •
                  •
                  •
  PROCEDURE     PROCEDURE1 NEAR
                  •
                  •
                ADD AX,BX
                  •
                  •
                RET
  PROCEDURE1    ENDP
  CODE ENDS
                END START
```

Scanned by CamScanner

Example 4.3

```
ASSUME CS:CODE, DS:DATA
DATA SEGMENT
NUM DB (55H)
COUNT EQU 10H
DATA ENDS
CODE SEGMENT
    START :     MOV AX,DATA
                MOV DS,AX
                    •
                    •
                CALL ROUTINE
                    •
                    •
                    •
    PROCEDURE   ROUTINE NEAR
                MOV BX,NUM
                MOV CX,COUTN
                    •
    ROUTINE     ENDP
    CODE        ENDS
                END START
```

Stack memory can also be used to pass parameters to a procedure. A main program may store the parameters to be passed to a procedure in its CPU registers. The registers will further be pushed on to the stack. The procedure during its execution pops back the appropriate parameters as and when required. This procedure of poping back the parameters must be implemented carefully because besides the parameters to be passed to the procedure the stack contains other important information like contents of other pushed registers, return addresses from the current procedure and other procedure or interrupt service routines.

Example 4.4

```
ASSUME CS:CODE, SS:STACK
CODE SEGMENT
    START :     MOV AX,STACK
                MOV SS,AX
                MOV AX,5577H
                MOV BX,2929H
                    •
                PUSH AX
                PUSH BX
                CALL ROUTINE ; Decrements SP by 2 (by 4 far routine)
                    •
                    •
```

Scanned by CamScanner

```
PROCEDURE    ROUTINE NEAR
                  •
        •
        MOV DX,SP
        ADD SP,02    ;      Leave initial two stack bytes of
                            return offset and
                            segment address after executing
                            subroutine
                            POP BX              ;      The data is
                            POP AX              ;      Passes in BX,AX
                            MOV SP,DX
                                •
                                •
                                •
STACK SEGMENT
STACKDATA DB 200H DUP (?)
STACK ENDS
```

For passing the parameters to procedures using the PUBLIC & EXTRN directives, must be declared PUBLIC (for all routines) in the main routine and the same should be declared EXT RN in the procedure. Thus the main program can pass the PUBLIC parameter to a procedure in which it is declared EXTRN (external)

Example 4.5

```
ASSUME CS:CODE, DS:DATA
DATA SEGMENT
PUBLIC NUMBER EQU 200H
        DATA ENDS
        CODE SEGMENT
    START :   MOV AX,DATA
              MOV DS,AX
                •
                •
                •
              CALL ROUTINE
                •
                •
                •
              PROCEDURE ROUTINE NEAR
          EXTRN NUMBER
              MOV AX,NUMBER
                •
                •
                •
          ROUTINE   ENDP
```

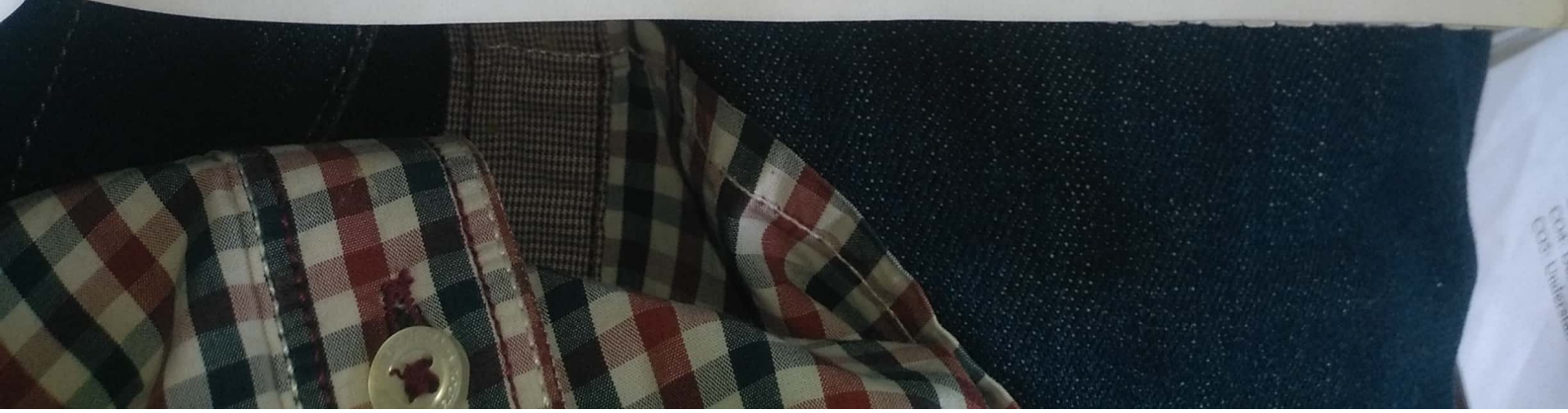
Scanned by CamScanner

```
                    AX,DATA
                    MOV DS,AX
                      •
                      •
                    CALL FAR_PTR ROUTINE1
                      •
                      •
                    CALL FAR_PTR ROUTINE2
                      •
                      •
CODE1               ENDS
PROCEDURE           ROUNTINE1 FAR
                      •
                      •
                      •
ROUTINE1            ENDP
PROCEDURE           ROUTINE2 FAR
                      •
                      •
ROUTINE2            ENDP
    END             START
```

4.10 MACROS

Till now, we have studied the stack, subroutines, interrupts and interrupt service routines. It is a notable point that the control is transferred to a subroutine or an interrupt service routine whenever it is called or an interrupt signal appears at the interrupt pin of the processor. After executing these routines the control is again transferred back to the main calling program. Hence rather than writing a complete routine again and again, one may call it as many times as required. This imparts flexibility in programming as well as ease of troubleshooting. The concept of subroutine as well as interrupt service routine can be compared with an office where the main (calling) program acts as a head while the subroutines and interrupt service routines act as subordinates. The head may ask his subordinates to work out a particular task and be ready with the results. Here the main program calls subroutines and interrupt service routines and may refer the results of their execution for further processing. The subroutines and interrupt service routines are assigned labels for references.

The macro is also a similar concept. Suppose, a number of instructions are repeating through in the main program, the listings becomes lengthy. So a macro definition, i.e. a label, is assigned with the repeatedly appearing string of instructions. The process of assigning a label or macroname to the string is called defining a macro. A macro within a macro is called a nested macro. The macroname or macro definition is then used throughout the main program to refer to that string of instructions.



Scanned by CamScanner

The difference between a macro and a subroutine is that in the macro the complete code of the instructions string is inserted at each place where the macro-name appears. Hence the EXE file becomes lengthy. Macro does not utilise the service of stack. There is no question of transfer of control as the program using the macro inserts the complete code of the macro at every reference of the macroname. On the other hand, subroutine is called whenever necessary, i.e. the control of execution is transferred to the subroutine, every time it is called. The executable code in case of the subroutines becomes smaller as the subroutine appears only once in the complete code. Thus, the EXE file is smaller as compared to the program using macro. The control is transferred to a subroutine whenever it is called, and this utilizes the stack service. The program using subroutine requires less memory space for execution than that using macro. Macro requires less time for execution, as it does not contain CALL and RET instructions as the subroutines do.

4.10.1 Defining a MACRO

A MACRO can be defined anywhere in a program using the directives MACRO and ENDM. The label prior to MACRO is the macro name which should be used in the actual program. The ENDM directive marks the end of the instructions or statements sequence assigned with the macro name. The following macro DISPLAY displays the message MSG on the CRT. The syntax is as given:

```
DISPLAY MACRO
        MOV AX, SEG MSG
        MOV DS, AX
        MOV DX, OFFSET MSG
        MOV AH, 09 H
        INT 21 H
ENDM
```

The above definition of a macro assigns the name DISPLAY to the instruction sequence between the directives MACRO and ENDM. While assembling, the above sequence of instructions will replace the label 'DISPLAY', whenever it appears in the program.

A macro may also be used in a data segment. In other words, a macro may also be used to represent statements and directives. The concept of macro remains the same independent of its contents. The following example shows a macro containing statements. The macro defines the strings to be displayed.

```
STRINGS MACRO
      MSG1 DB 0AH,0DH, "Program terminated normally",0AH,0DH, "$"
      MSG2 DB 0AH,0DH, "Retry , Abort, Fail",0AH,0DH, "$"
ENDM
```

A macro may be called by quoting its name, along with any values to be passed to the macro. Calling a macro means inserting the statements and instructions represented by the macro directly at the place of the macroname in the program.

4.10.2 Passing Parameters to a MACRO

Using parameters in a definition, the programmer specifies the parameters of the macro those are likely to be changed each time the macro is called. For example, the DISPLAY macro written in Section 4.10.1 can be made to display two different messages MSG1 and MSG2, as shown.

```
DISPLAY MACRO MSG
        MOV AX, SEG MSG
        MOV DS, AX
```

Scanned by CamScanner

```
                MOV DX, OFFSET MSG
                MOV AH, 09 H
                INT 21 H
        ENDM
```

This parameter MSG can be replaced by MSG1 or MSG2 while calling the macro as shown.

```
        .
        .
        .
DISPLAY MSG1
        .
        .
        .
DISPLAY MSG2
        .
        .
        .
MSG1 DB OAH,ODH, "Program Terminated Normally",OAH,ODH, "$"
MSG2 DB OAH,ODH, "Retry, Abort, Fail",OAH,ODH, "$"
```

There may be more than one parameter appearing in the macro definition, meaning thereby that there may be more than one parameters to be passed to the macro, and each of them is liable to be changed. All the parameters are specified in the definition sequentially and also in the call with the same sequence.

A macro may be defined in another macro or in other words a macro may be called from inside a macro. This type of macro is called a nested macro. All the directives available in MASM can also be used in a macro and carry the same significance.

4.11 TIMINGS AND DELAYS

It is obvious from the studies of the timing diagrams that every instruction requires a definite number of clock cycles for its execution. Thus every instruction requires a fixed amount of time, i.e. multiplication of the number of clock cycles required for the execution of the instruction and the period of the clock at which the microprocessor is running. The duration required for the execution of an instruction can be used to derive the required delays. A sequence of instructions, if executed by a microprocessor, will require a time duration that is the sum of all the individual time durations required for execution of each instruction. Note that in a loop program, the number of instructions in the program may be less but the number of instructions actually executed by the microprocessor depend on the loop count. Also in case of subroutines and interrupt service routines the actual number of instructions executed by the microprocessor depends on the procedure or interrupt service routine length along with the main calling program. The required number of clock states for execution of each instruction of 8086/88 are given in Appendix A.

The procedure of generating delays using a microprocessor based system can be stepwise described as follows.

1. Determine the exact required delay.
2. Select the instructions for delay loop. While selecting the instructions, care should be taken that the execution of these instructions does not interfere with the main program execution. In other words, any memory location or register used by the main program must not be modified by the delay routine. The instructions executed for the delay loop are dummy instructions in the sense that the result of those instructions is useless but the time required for their execution is an elemental part of the required delay.

Scanned by CamScanner

Scanned by CamScanner

For interfacing a 16-bit output port with 8086, we may use two 8-bit output ports to form a 16-bit port, with a single address, as shown in Fig. 5.16. Both the 8-bit ports are in this case addressed as a single 16-bit port with a single address.

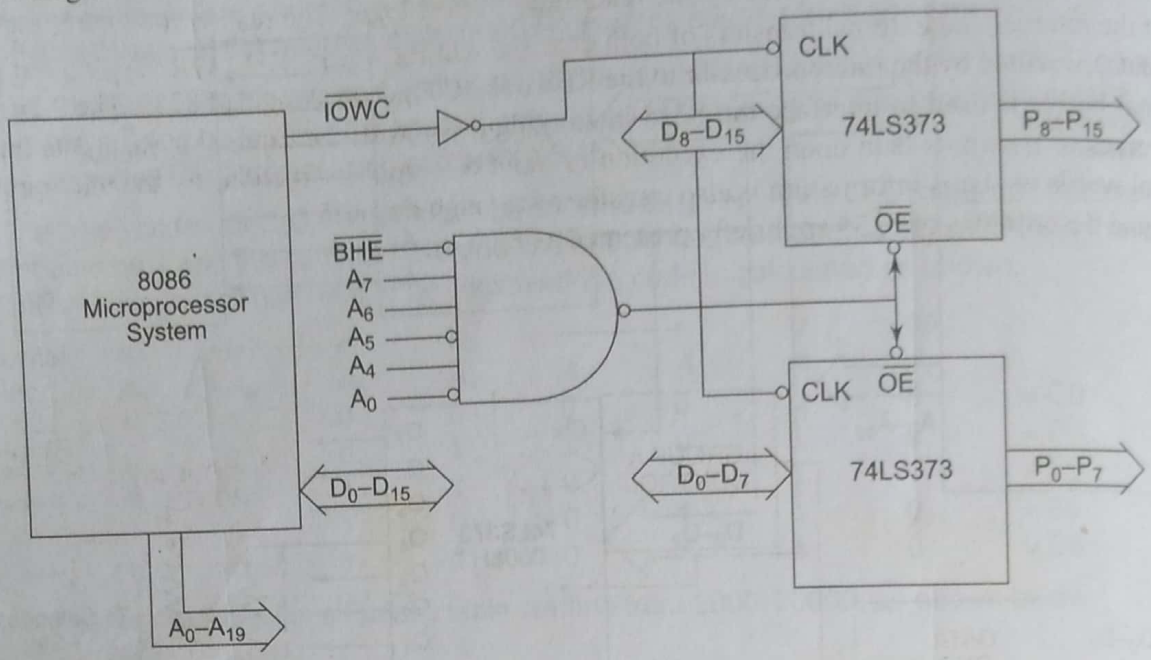


Fig. 5.16 *Interfacing a Circuit of 16 bit output port*

The OUT instruction of 8086 is able to output 16-bit data directly in a single cycle, and the programming technique is identical to that of the 8-bit port. The instruction uses 16-bit register AX as source operand as shown:

```
OUT Port_Add, AX
OUT [DX], AX
```

A 16-bit input port may also be interfaced similarly. Note the use of A0 and BHE signals in interfacing the 16-bit ports. One may find out address of the output port in Fig. 5.16.

Till now we have studied some common interfacing methods for I/O ports and have also solved some problems based on I/O ports. A few more problems will be discussed while studying the Programmable Peripheral Input/Output port chip 8255. The ports in this section were either input or output, but in 8255 the same port may be programmed either to work as input port or as output port. Hence the chip is called Programmable Input-Output (PIO) device.

5.4 PIO 8255 [PROGRAMMABLE INPUT-OUTPUT PORT]

The parallel input-output port chip 8255 is also known as programmable peripheral input-output port. The Intel's 8255 is designed for use with Intel's 8-bit, 16-bit and higher capability microprocessors. It has 24 input/output lines which may be individually programmed in two groups of twelve lines each, or three groups of eight lines. The two groups of I/O pins are named as Group A and Group B. Each of these two groups contain a subgroup of eight I/O lines called as 8-bit port and another subgroup of four I/O lines or a 4-bit port. Thus Group A contains an 8-bit port A along with a 4-bit port, C upper. The port A lines are identified by symbols PA_0–PA_7 while the port C lines are identified as PC_4–PC_7. Similarly, Group B contains an 8-bit port B, containing lines PB_0 – PB_7 and a 4-bit port C with lower bits PC_0–PC_3. The port C upper and port C lower can be used in combination as an 8-bit port C. Both the port Cs are assigned the same address. Thus one may have either three 8-bit I/O ports or two 8-bit and two 4-bit I/O ports

Scanned by CamScanner

from 8255. All of these ports can function independently either as input or as output ports. This can be achieved by programming the bits of an internal register of 8255 called as Control Word Register (CWR). The internal block diagram and the pin configuration of 8255 are shown in Figs 5.17 (a) and (b).

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. $\overline{RD}$, $\overline{WR}$, 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 upon the execution of input or output instructions by the microprocessor. The control words or status information is also transferred through the buffer.

The signal descriptions of 8255 are briefly presented as follows:

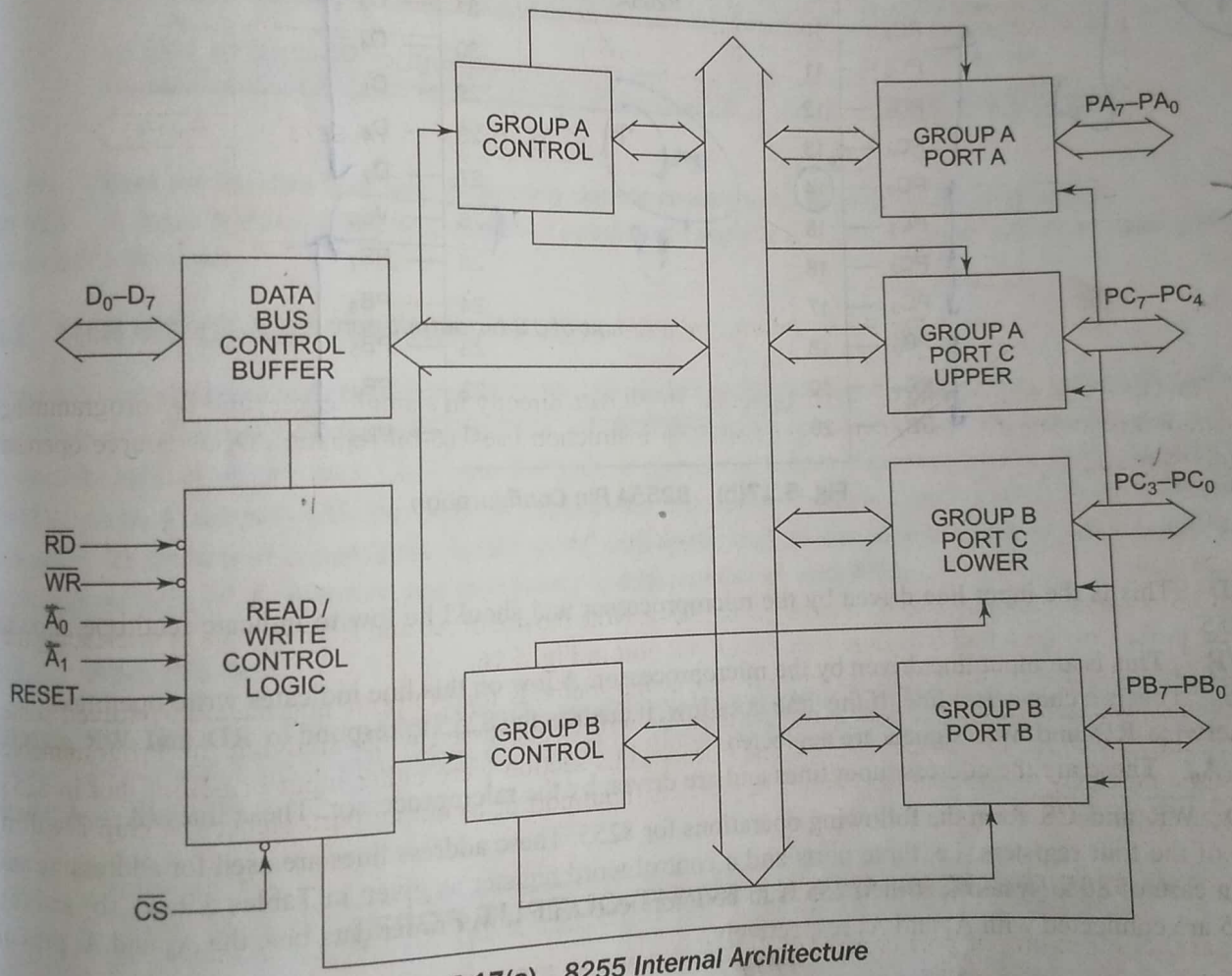


Fig. 5.17(a) *8255 Internal Architecture*

PA_7–PA_0 These are eight port A lines that act 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 mode1 or mode 2.

PC_3–PC_0 These are the lower port C lines, other details are the same as PC_7–PC_4 lines.

PB_0–PB_7 These are the eight port B lines which are used as latched output lines or buffered input lines in the same way as port A.

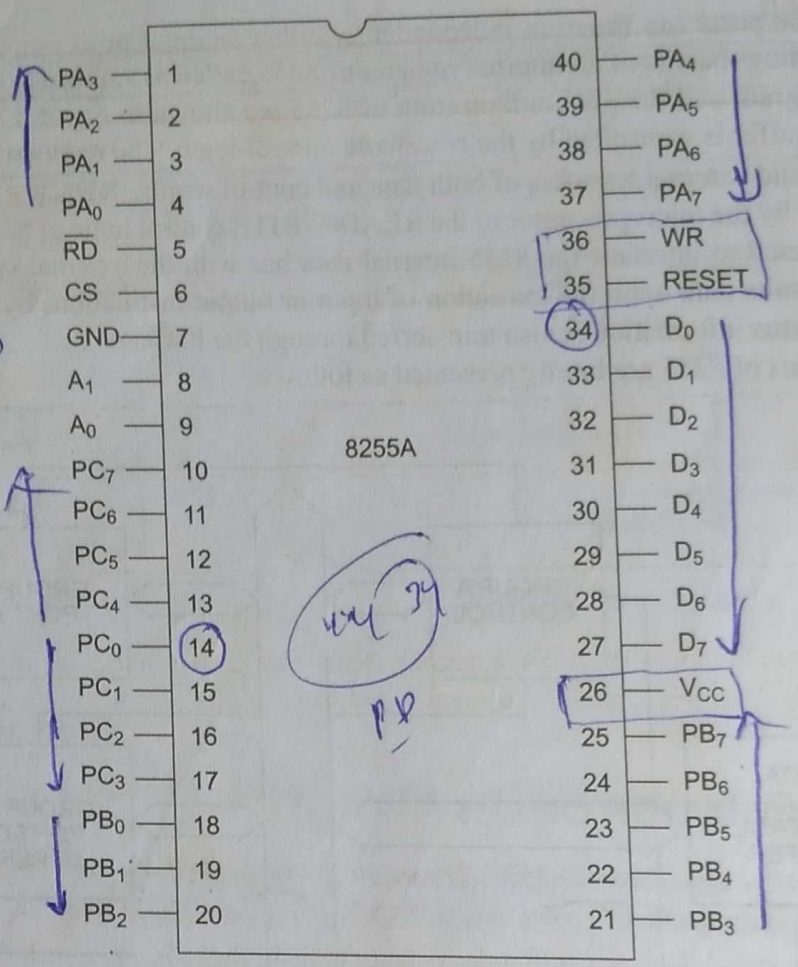


Fig. 5.17(b) *8255A Pin Configuration*

$\overline{RD}$ This is the input line driven by the microprocessor and should be low to indicate read operation to 8255.

$\overline{WR}$ This is an input line driven by the microprocessor. A low on this line indicates write operation.

$\overline{CS}$ This is a chip select line. If this line goes low, it enables the 8255 to respond to $\overline{RD}$ and $\overline{WR}$ signals, otherwise $\overline{RD}$ and $\overline{WR}$ signals are neglected.

A_1–A_0 These are the address input lines and are driven by the microprocessor. These lines ($A_1 - A_0$) with $\overline{RD}$, $\overline{WR}$ and $\overline{CS}$ form the following operations for 8255. These address lines are used for addressing any one of the four registers, i.e. three ports and a control word register as given in Tables 5.9 (a), (b) and (c).

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 respectively.

Table 5.9(a)

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	*Input (Read) cycle*
0	1	0	0	0	Port A to data bus
0	1	0	0	1	Port B to data bus
0	1	0	1	0	Port C to data bus
0	1	0	1	1	CWR to data bus

Scanned by CamScanner

Table 5.9 (b)

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	*Output (Write) cycle*
1	0	0	0	0	Data bus to Port A
1	0	0	0	1	Data bus to Port B
1	0	0	1	0	Data bus to Port C
1	0	0	1	1	Data bus to CWR

Table 5.9 (c)

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	*Function*
X	X	1	X	X	Data bus tristated
1	1	0	X	X	Data bus tristated

D_0–D_7 These are the data bus lines those carry data or control word to/from the micro-processor.
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.

5.5 MODES OF OPERATION OF 8255

There are two basic modes of operation of 8255–I/O mode and Bit Set-Reset mode (BSR). In the I/O mode, the 8255 ports work as programmable I/O ports, while in BSR mode only port C(PC_0–PC_7) can be used to set or reset its individual port bits. Under the IO mode of operation, further there are three modes of operation of 8255, so as to support different types of applications, viz. *mode 0* , *mode 1* and *mode 2*. These modes of operation are discussed in significant details along with application problems in this section, so as to present a clear idea about 8255 operation and interfacing in different modes with 8086.

5.5.1 BSR Mode

In this mode, any of the 8-bits of port C can be set or reset depending on B_0 of the control word. The bit to be set or reset is selected by bit select flags B_3, B_2 and B_1 of the CWR as given in Table 5.10. The CWR format is shown in Fig. 5.18(a).

Table 5.10

B_3	B_2	B_1	*Selected Bits of port C*
0	0	0	B_0
0	0	1	B_1
0	1	0	B_2
0	1	1	B_3
1	0	0	B_4
1	0	1	B_5
1	1	0	B_6
1	1	1	B_7

the maximum mode so that $\overline{IORD}$ and $\overline{IOWR}$ are readily available. If the 8086 is in minimum mode, $\overline{RD}$ and $\overline{WR}$ of 8086 are to be connected accordingly to 8255 and M/ $\overline{IO}$ pin is combined with the chip select of above hardware suitably so as to select the 8255 when M/$\overline{IO}$ is low.

The ALP for the problem is developed as follows:

```
MOV DX, 0746 H          ; Initialise CWR with
MOV AL, 82 H            ; control word 82H
OUT DX, AL              ;
SUB DX,04               ; Get address of port B in DX
IN AL, DX               ; Read port B for switch
SUB DX,02               ; positions in to AL and get port A address
                        ; in DX.
OUT DX, AL              ; Display switch positions on port A
MOV BL, 00 H            ; Initialise BL for switch count
MOV CH, 08H             ; Initialise CH for total switch number
YY: ROL AL              ; Rotate AL through carry to check,
    JNC XX              ; whether the switches are on or
    INC BL              ; off, i.e. either 1 or 0
XX :DEC CH              ; Check for next switch. If
    JNZ YY              ; all switch are checked, the
    MOV AL, BL          ; number of on switches are
    ADD DX, 04          ; in BL.Display it on port C
    OUT DX,AL           ; lower.
    HLT                 ; Stop
```

Program 5.5 *ALP for Problem 5.10*

Scanned by CamScanner

Fig. 5.19 *8255 Interfacing with 8086 for Problem 5.10*

Scanned by CamScanner

The circuit arrangment for this interfacing problem is as shown below:

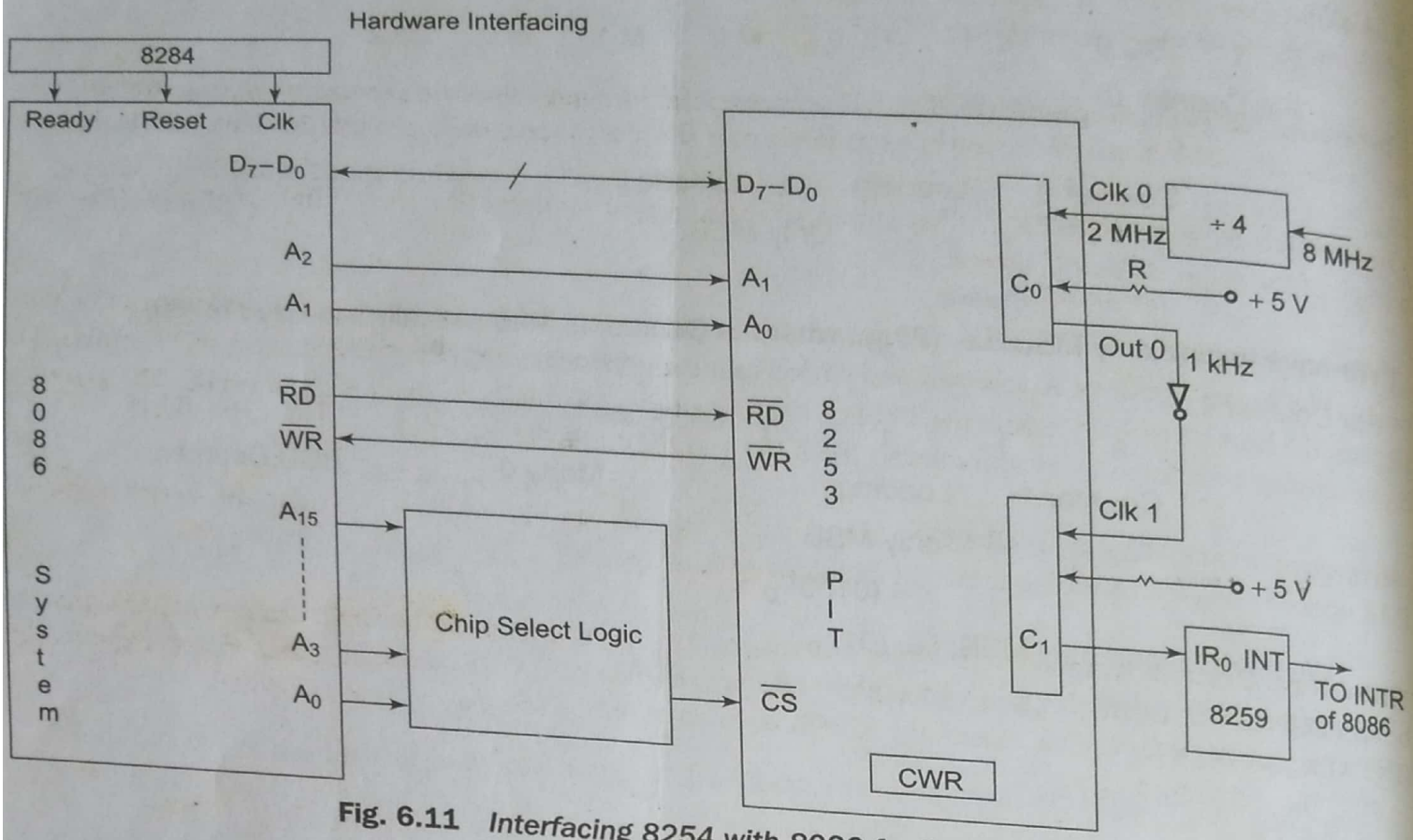


Fig. 6.11 *Interfacing 8254 with 8086 for Problem 6.2*

In all the above programs, the counting down starts as soon as the writing operation to the count register is over, and the $\overline{WR}$ pin goes high after writing. In case of 'read on fly' operation, the READ ON FLY control word for the specific counter is to be loaded in the control word register followed by the successive read operations. The 8254 and 8254 are the most widely used peripherals to generate accurate delays for industrial or laboratory purposes. The 8254 is similar to 8254 in architecture, programming and operation, hence 8254 is not discussed in this text.

6.2 PROGRAMMABLE INTERRUPT CONTROLLER 8259A

The processor 8085 had five hardware interrupt pins. Out of these five interrupt pins, four pins were allotted fixed vector addresses but the pin INTR was not allotted any vector address, rather an external device was supposed to hand over the type of the interrupt, i.e. (Type 0 to 7 for RST0 to RST7), to the microprocessor. The microprocessor then gets this type and derives the interrupt vector address from that. Consider an application, where a number of I/O devices connected with a CPU desire to transfer data using interrupt driven data transfer mode. In these types of applications, more number of interrupt pins are required than available in a typical microprocessor. Moreover, in these multiple interrupt systems, the processor will have to take care of the priorities for the interrupts, simultaneously occurring at the interrupt request pins.

To overcome all these difficulties, we require a programmable interrupt controller which is able to handle a number of interrupts at a time. This controller takes care of a number of simultaneously appearing interrupt requests along with their types and priorities. This relieves the processor from all these tasks. The programmable interrupt controller 8259A from Intel is one such device. Its predecessor 8259 was designed to operate only with 8-bit processors like 8085. A modified version, 8259A was later introduced that is compatible with 8-bit as well as 16-bit processors.

Scanned by CamScanner

6.2.1 Architecture and Signal Descriptions of 8259A

The architectural block diagram of 8259A is shown in Fig. 6.12. The functional explanation of each block is given in the following text in brief:

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

In-Service Register (ISR) This 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 $\overline{INTA}$ pulse. The IR_0 has the highest priority while the IR_7 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 the 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.

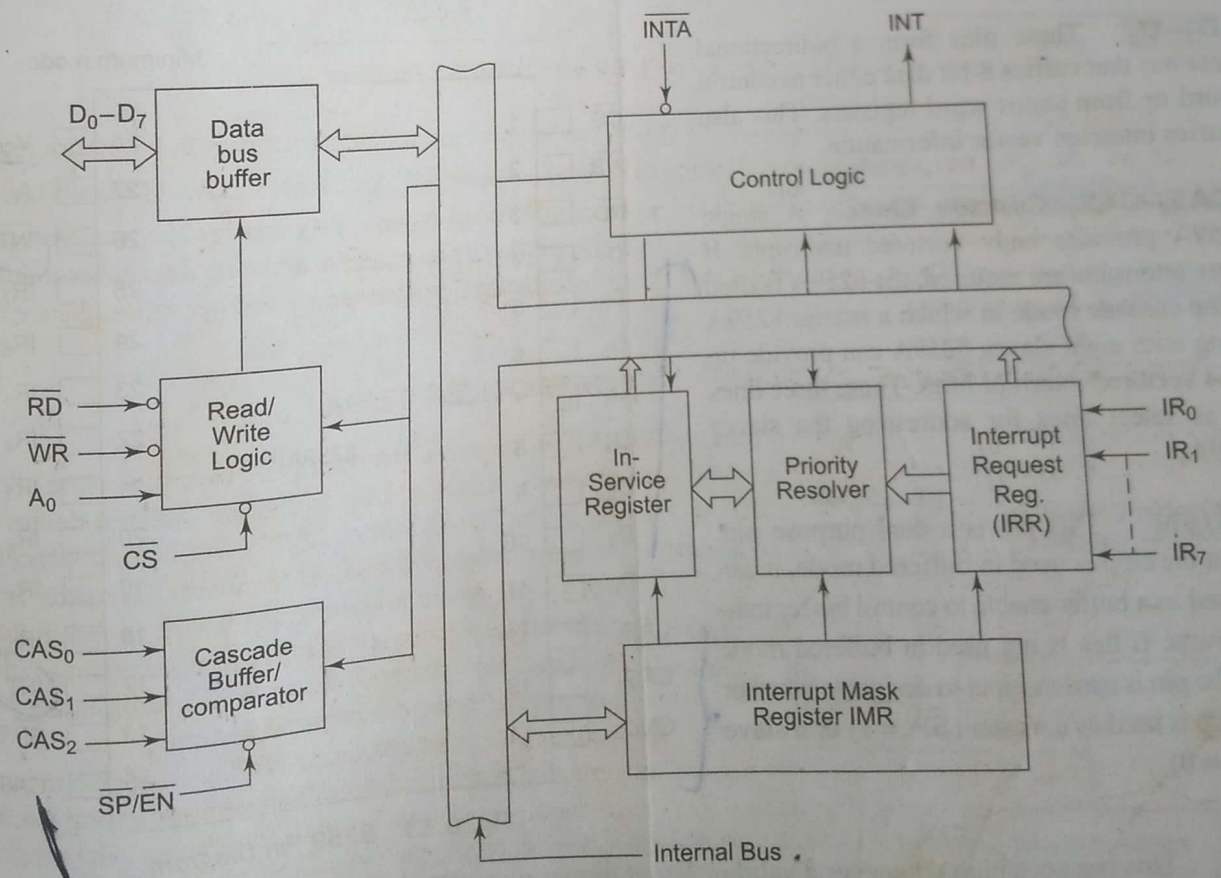


Fig. 6.12 *8259A Block Diagram*

Cascade Line

Scanned by CamScanner

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 IDs of all the 8259As used in the system. The three I/O pins CAS0-2 are outputs when the 8259A is used as a master. The same pins act as inputs when the 8259A is in the slave mode. The 8259A in the master mode, sends the ID of the interrupting slave device on these lines. The slave thus selected, will send its pre-programmed vector address on the data bus during the next $\overline{\text{INTA}}$ pulse.

Figure 6.13 shows the pin configuration of 8259A, followed by their functional description of each of the signals in brief.

CS This is an active-low chip select signal for enabling $\overline{\text{RD}}$ and $\overline{\text{WR}}$ operations of 8259A. $\overline{\text{INTA}}$ function is independent of $\overline{\text{CS}}$.

$\overline{\text{WR}}$ This pin is an active-low write enable input to 8259A. This enables it to accept command words from CPU.

$\overline{\text{RD}}$ 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.

D_7–D_0 These pins form a bidirectional data bus that carries 8-bit data either to control word or from status word registers. This also carries interrupt vector information.

CAS_0–CAS_2 Cascade Lines A single 8259A provides eight vectored interrupts. If more interrupts are required, the 8259A is used in the cascade mode in which 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 slaves 8259A.

$\overline{\text{PS}}/\overline{\text{EN}}$ This pin is a dual purpose pin. When the chip is used in buffered mode, it can be used as a buffer 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 ($\overline{\text{SP}} = 1$) or a slave ($\overline{\text{EN}} = 0$).

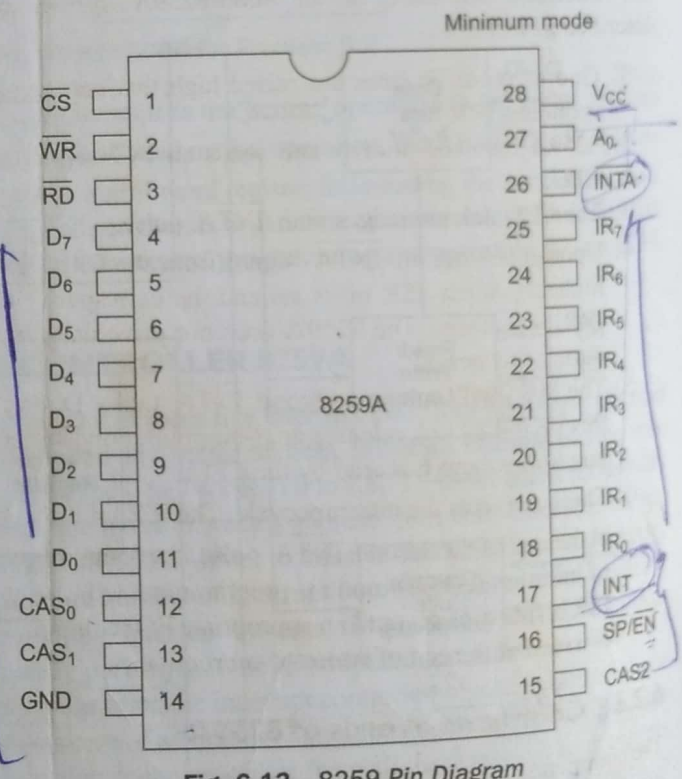


Fig. 6.13 8259 Pin Diagram

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.

Scanned by CamScanner

IR_0–IR_7(Interrupt requests) These pins act as inputs to accept interrupt requests to the CPU. In the edge triggered mode, an interrupt service is requested by raising an IR pin from a low to a high state. It is held high until it is acknowledged, and just by latching it to high level, if used in the level triggered mode.

$\overline{INTA}$ (Interrupt acknowledge) This pin is an input used to strobe-in 8259A interrupt vector data on to the data bus. In conjunction with $\overline{CS}$, $\overline{WR}$, and $\overline{RD}$ pins, this selects the different operations like, writing command words, reading status word, etc.

The device 8259A can be interfaced with any CPU using either polling or interrupt. In polling, the CPU keeps on checking each peripheral device in sequence to ascertain if it requires any service from the CPU. If any such service request is noticed, the CPU serves the request and then goes on to the next device in sequence. After all the peripheral devices are scanned as above the CPU again starts from the first device. This type of system operation results in the reduction of processing speed because most of the CPU time is consumed in polling the peripheral devices.

In the interrupt driven method, the CPU performs the main processing task till it is interrupted by a service requesting peripheral device. The net processing speed of these type of systems is high because the CPU serves the peripheral only if it receives the interrupt request. If more than one interrupt requests are received at a time, all the requesting peripherals are served one by one on priority basis. This method of interfacing may require additional hardware if number of peripherals to be interfaced is more than the interrupt pins available with the CPU.

6.2.2 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 acknowledges with $\overline{INTA}$ pulse.
4. Upon receiving an $\overline{INTA}$ signal from the CPU, the highest priority ISR bit is set and the corresponding IRR bit is reset. The 8259A does not drive data bus during this period.
5. The 8086 will initiate a second $\overline{INTA}$ pulse. During this period 8259A releases an 8-bit pointer on to data bus from where it is read by the CPU.
6. This completes the interrupt cycle. The ISR bit is reset at the end of the second $\overline{INTA}$ 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.

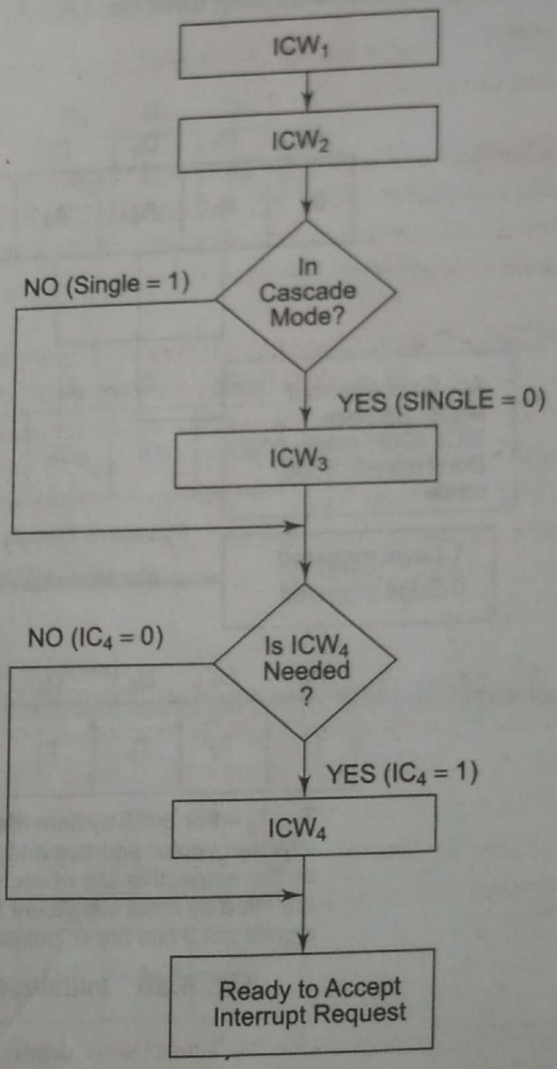


Fig. 6.14 *Initialization Sequence of 8259A*

6.2.3 Command Words of 8259A

The command words of 8259A are classified in two groups, viz. Initialization Command Words (ICWs) and Operation Command Words (OCWs).

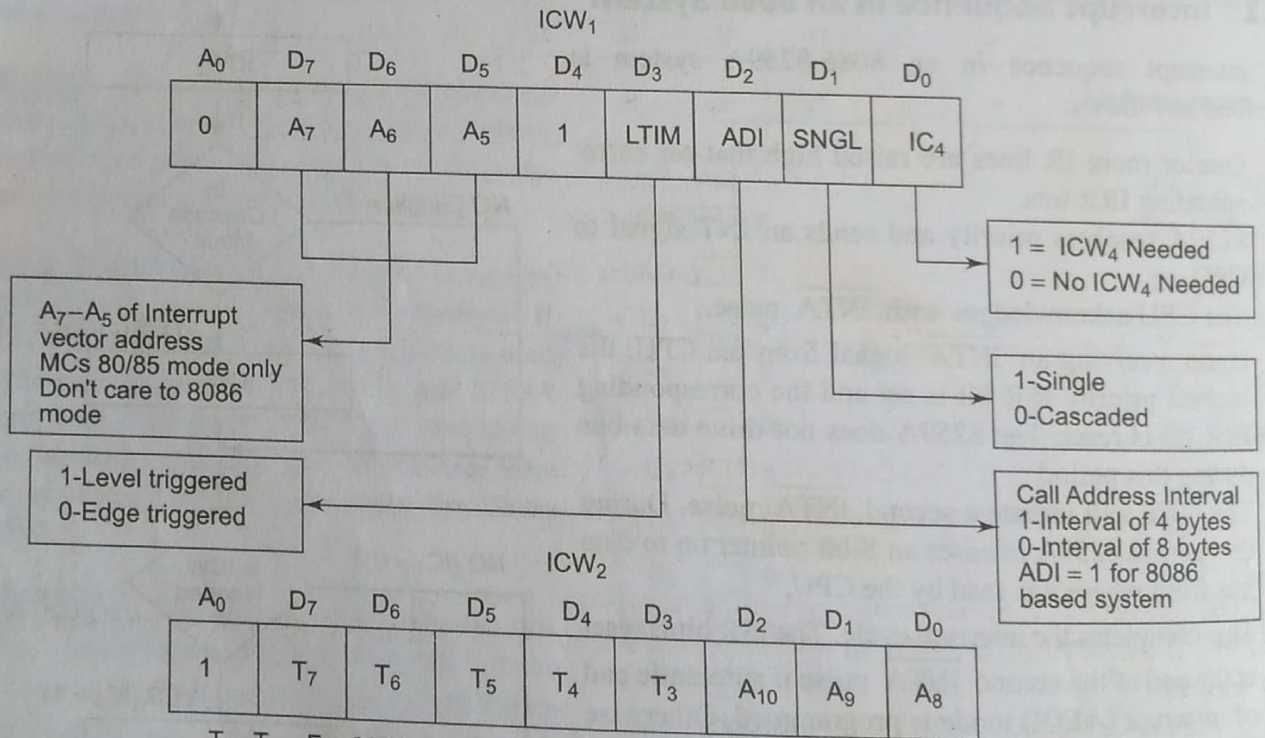
Scanned by CamScanner

Initialization Command Words (ICWs) 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 Initialization Command Words (ICWs). If $A_0 = 0$ and $D_4 = 1$, the control word is recognized as ICW_1. It contains the control bits for edge/level triggered mode, single/cascade mode, call address interval and whether ICW_4 is required or not, etc. If $A_0 = 1$, the control word is recognized as ICW_2. The ICW_2 stores details regarding interrupt vector addresses. The initialisation sequence of 8259A is described in from of a flow chart in Fig. 6.14. The bit functions of the ICW_1 and ICW_2 are self explanatory as shown in Fig. 6.15.

Once ICW_1 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 the status read is set to IRR
(f) If $IC_4 = 0$, all the functions of ICW_4 are set to zero. Master/slave bit in ICW_4 is used in the buffered mode only.

In an 8085 based system, $A_{15} - A_8$ of the interrupt vector address are the respective bits of ICW_2. In 8086/88 based system, five most significant bits of the interrupt type byte are inserted in place of $T_7 - T_3$ respectively and the remaining three bits (A_8, A_9 and A_{10}) are inserted internally as 000 (as if they are pointing



$T_7 - T_3$ – For 8085 system they are filled by A_{15}–A_{11} of the interrupt vector address and the least significant 3 bits are same as the respective bits of vector address. For 8086 system they are filled by most significant 5 bits of interrupt type and the least significant 3 bits are 0, pointing to Ir_0

Fig. 6.15 *Initialization Command Words ICW_1 and ICW_2*

Scanned by CamScanner

to IR_0). Other seven interrupt levels' vector addresses are internally generated automatically by 8259 using IR_0 vector. Address interval is always four in an 8086 based system.

ICW_1 and ICW_2 are compulsory command words in initialization sequence of 8259A as is evident from Fig. 6.14, while ICW_3 and ICW_4 are optional. The ICW_3 is read only when there are more than one 8259As in the system, i.e. cascading is used (SNGL = 0). The SNGL bit in ICW_1 indicates whether the 8259A is in the cascade mode or not. The ICW_3 loads an 8-bit slave register. Its detailed functions are as follows:

In the master mode (i.e. $\overline{SP}$ = 1 or in buffer mode M/S = 1 in ICW_4), the 8-bit slave register will be set bit-wise to '1' for each slave in the system, as shown in Fig. 6.16. The requesting slave will then release the second byte of a CALL sequence.

In slave mode (i.e. $\overline{SP}$ = 0 or if BUF = 1 and M/S = 0 in ICW_4) bits D_2 to D_0 identify the slave, i.e. 000 to 111 for slave1 to slave8. The slave compares the cascade inputs with these bits and if they are equal, the second byte of the CALL sequence is released by it on the data bus.

ICW_4 The use of this command word depends on the IC_4 bit of ICW_1. If IC_4 = 1, ICW_4 is used, otherwise it is neglected. The bit functions of ICW_4 are described as follows:

SFNM Special fully nested mode is selected, if SFNM = 1.

BUF If BUF = 1, the buffered mode is selected. In the buffered mode, SP/EN acts as enable output and the master/slave is determined using the M/S bit of ICW_4.

M/S If M/S = 1, 8259A is a master. If M/S = 0, 8259A is a slave. If BUF = 0, M/S is to be neglected.

Master mode ICW_3

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

Sn = 1-IRn Input has a slave
= 0-IRn Input does not have a slave

Slave mode ICW_3

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	ID_2	ID_1	ID_0

$D_2D_1D_0$ - 000 to 111 for IR_0 to IR_7 or slave 1 to slave 8

Fig. 6.16 *ICW_3 in Master and Slave Mode*

AEOI If AEOI = 1, the automatic end of interrupt mode is selected.

mPM If the mPM bit is 0, the Mcs-85 system operation is selected and if mPM =1, 8086/88 operation is selected.

Figure 6.17 shows the ICW_4 bit positions:

ICW_4

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	SFNM	BUF	M/$\overline{S}$	AEOI	mPM

Scanned by CamScanner

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 ways of handling the received interrupts called as modes of operation. These modes of operations

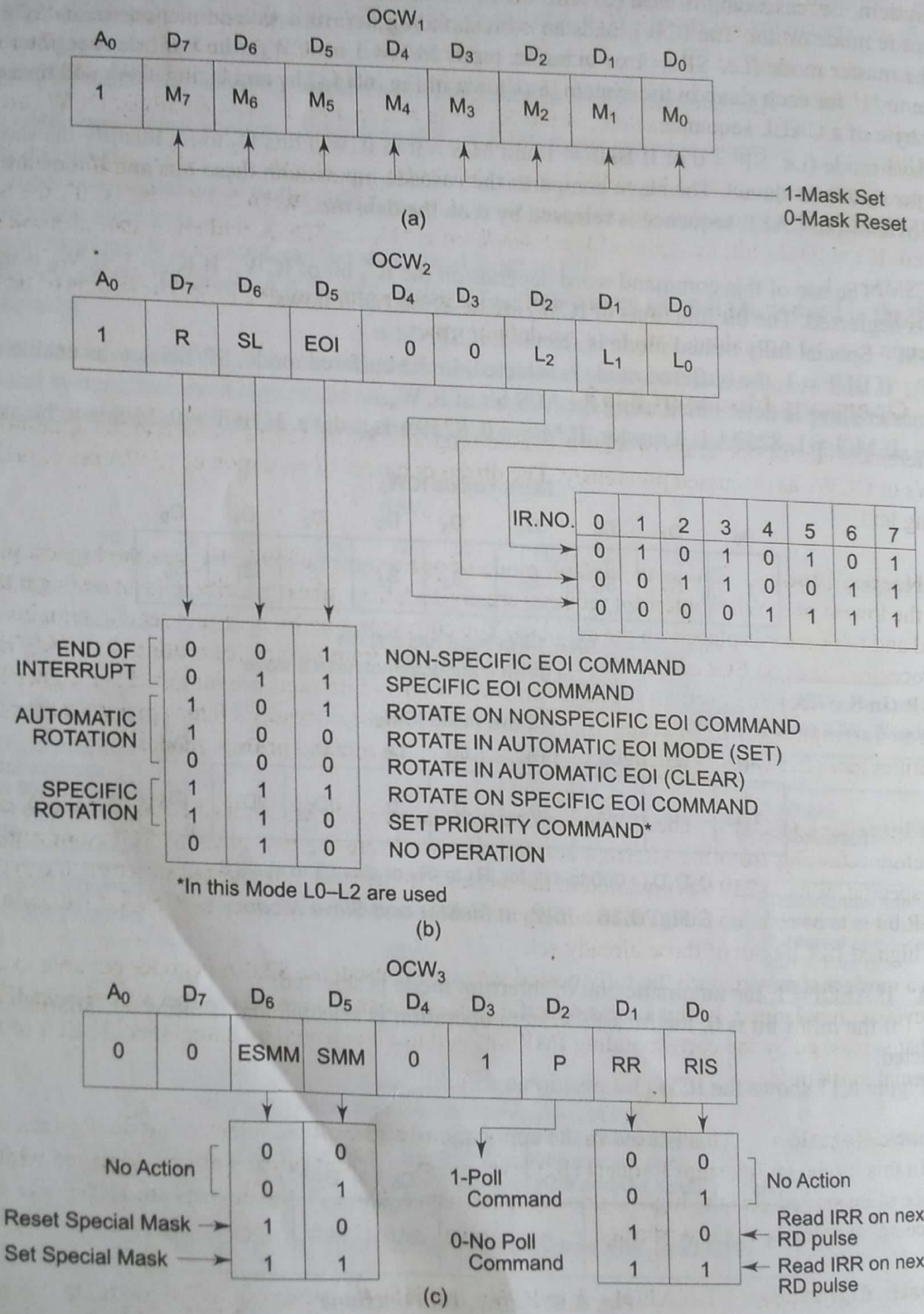


Fig. 6.18 *Operation Command Words*

Scanned by CamScanner

can be selected by programming, i.e. writing three internal registers called as *operation command word registers*. The data written into them (bit pattern) is called as *operation command words*. In the three operation command words OCW_1, OCW_2 and OCW_3, 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. 6.18 (a), (b) and (c) with the bit selection details. OCW_1 is used to mask the unwanted interrupt requests. If the mask bit is '1', the corresponding interrupt request is masked, and if it is '0', the request is enabled. In OCW_2 the three bits, viz. R,SL and EOI control the end of interrupt, the rotate mode and their combinations as shown in Fig. 6.18 (b). The three bits L_2, L_1 and L_0 in OCW_2 determine the interrupt level to be selected for operation , if the SL bit is active, i.e. '1'. The details of OCW_2 are shown in Fig. 6.18(b).

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 enabled to select or mask the *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 OCW_3 are given in Fig. 6.18 (c) along with their bit definitions.

6.2.4 Operating Modes of 8259

The different modes of operation of 8259A can be programmed by setting or resting the appropriate bits of the ICWs or OCWs as discussed previously. The different modes of operation of 8259A are explained in the following text:

Fully Nested Mode This is the default mode of operation of 8259A. IR_0 has the highest priority and IR_7 has the lowest one. When interrupt requests are noticed, the highest priority request amongst them is determined and the vector is placed on the data bus. The corresponding bit of ISR is set and remains set till the microprocessor issues an EOI command just before returning from the service routine or the AEOI bit is set. If the ISR (In Service) bit is set, all the same or lower priority interrupts are inhibited but higher levels will generate an interrupt, that will be acknowledged only if the microprocessor's Interrupt enable Flag (IF) is set. The priorities can afterwards be changed by programming the rotating priority modes.

End of Interrupt (EOI) The ISR bit can be reset either with AEOI bit of ICW_1 or by EOI command, issued before returning from the interrupt service routine. There are two types of EOI commands specific and non-specific. When 8259A is operated in the modes that preserve fully nested structure, it can determine which ISR bit is to be reset on EOI. When non-specific EOI command is issued to 8259A it will automatically reset the highest ISR bit out of those already set.

When a mode that may disturb the fully nested structure is used, the 8259A is no longer able to determine the last level acknowledged. In this case a specific EOI command is issued to reset a particular ISR bit. An ISR bit that is masked by the corresponding IMR bit, will not be cleared by a non-specific EOI of 8259A, if it is in special mask mode.

Automatic Rotation This is used in the applications where all the interrupting devices are of equal priority. In this mode, an Interrupt Request (IR) level receives lowest 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.

Automatic EOI Mode Till AEOI = 1 in ICW_4, the 8259A operates in AEOI mode. In this mode, the 8259A performs a non-specific EOI operation at the trailing edge of the last $\overline{\text{INTA}}$ pulse automatically. This mode should be used only when a nested multilevel interrupt structure is not required with a single 8259A.

Scanned by CamScanner

Specific Rotation In this mode a bottom priority level can be selected, using L_2, L_1 and L_0 in OCW_2 and R = 1, SL = 1, EOI = 0. The selected bottom priority fixes other priorities. If IR_5 is selected as a bottom priority, then IR_5 will have least priority and IR_4 will have a next higher priority. Thus IR_6 will have the highest priority. These priorities can be changed during an EOI command by programming the rotate on specific EOI command in OCW_2.

Special Mask Mode In the special mask mode, when a mask bit is set in OCW_1, it inhibits further interrupts at that level and enables interrupt from other levels, which are not masked.

Edge and Level Triggered Mode This mode decides whether the interrupt should be edge triggered or level triggered. If bit LTIM of $ICW_1 = 0$, they are edge triggered, otherwise the interrupts are level triggered.

Reading 8259 Status The status of the internal registers of 8259A can be read using this mode. The OCW_3 is used to read IRR and ISR while OCW_1 is used to read IMR. Reading is possible only in no polled mode.

Poll Command In the polled mode of operation, the INT output of 8259A is neglected, though it functions normally, by not connecting INT output or by masking INT input of the microprocessor. The poll mode is entered by setting P = 1 in OCW_3. The 8259A is polled by using software execution by microprocessor instead of the requests on INT input. The 8259A treats the next $\overline{RD}$ pulse to the 8259A as an interrupt acknowledge. An appropriate ISR bit is set, if there is a request. The priority level is read and a data word is placed on to data bus, after $\overline{RD}$ is activated. The data word is shown in Fig. 6.19.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
I	X	X	X	X	W_2	W_1	W_0
					↑	↑	↑
					Binary code		

Scanned by CamScanner

Specific Rotation In this mode a bottom priority level can be selected, using L_2, L_1 and L_0 in OCW_2 and R = 1, SL = 1, EOI = 0. The selected bottom priority fixes other priorities. If IR_4 is selected as a bottom priority, then IR_5 will have least priority and IR_6 will have a next higher priority. Thus IR_4 will have the highest priority. These priorities can be changed during an EOI command by programming the rotate on specific EOI command in OCW_2.

Special Mask Mode In the special mask mode, when a mask bit is set in OCW_1, it inhibits further interrupts at that level and enables interrupt from other levels, which are not masked.

Edge and Level Triggered Mode This mode decides whether the interrupt should be edge triggered or level triggered. If bit LTIM of $ICW_1 = 0$, they are edge triggered, otherwise the interrupts are level triggered.

Reading 8259 Status The status of the internal registers of 8259A can be read using this mode. The OCW_3 is used to read IRR and ISR while OCW_1 is used to read IMR. Reading is possible only in no polled mode.

Poll Command In the polled mode of operation, the INT output of 8259A is neglected, though it functions normally, by not connecting INT output or by masking INT input of the microprocessor. The poll mode is entered by setting P = 1 in OCW_2. The 8259A is polled by using software execution by microprocessor instead of the requests on INT input. The 8259A treats the next $\overline{RD}$ pulse to the 8259A as an interrupt acknowledge. An appropriate ISR bit is set, if there is a request. The priority level is read and a data word is placed on to data bus, after $\overline{RD}$ is activated. The data word is shown in Fig. 6.19.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
I	X	X	X	X	W_2	W_1	W_0

← If = 1, there is an interrupt

Binary code of highest priority level

Fig. 6.19 *Data Word of 8259*

A poll command may give you more than 64 priority levels. Note that this has nothing to do with the 8086 interrupt structure and the interrupt priorities.

Special Fully Nested Mode This mode is used in more complicated systems, where cascading is used and the priority has to be programmed in the master using ICW_4. This is somewhat similar to the normal nested mode. In this mode, when an interrupt request from a certain slave is in service, this slave can further send requests to the master, if the requesting device connected to the slave has higher priority than the one being currently served. In this mode, the master interrupts the CPU only when the interrupting device has a higher or the same priority than the one currently being served. In normal mode, other requests than the one being served are masked out.

When entering the interrupt

Scanned by CamScanner

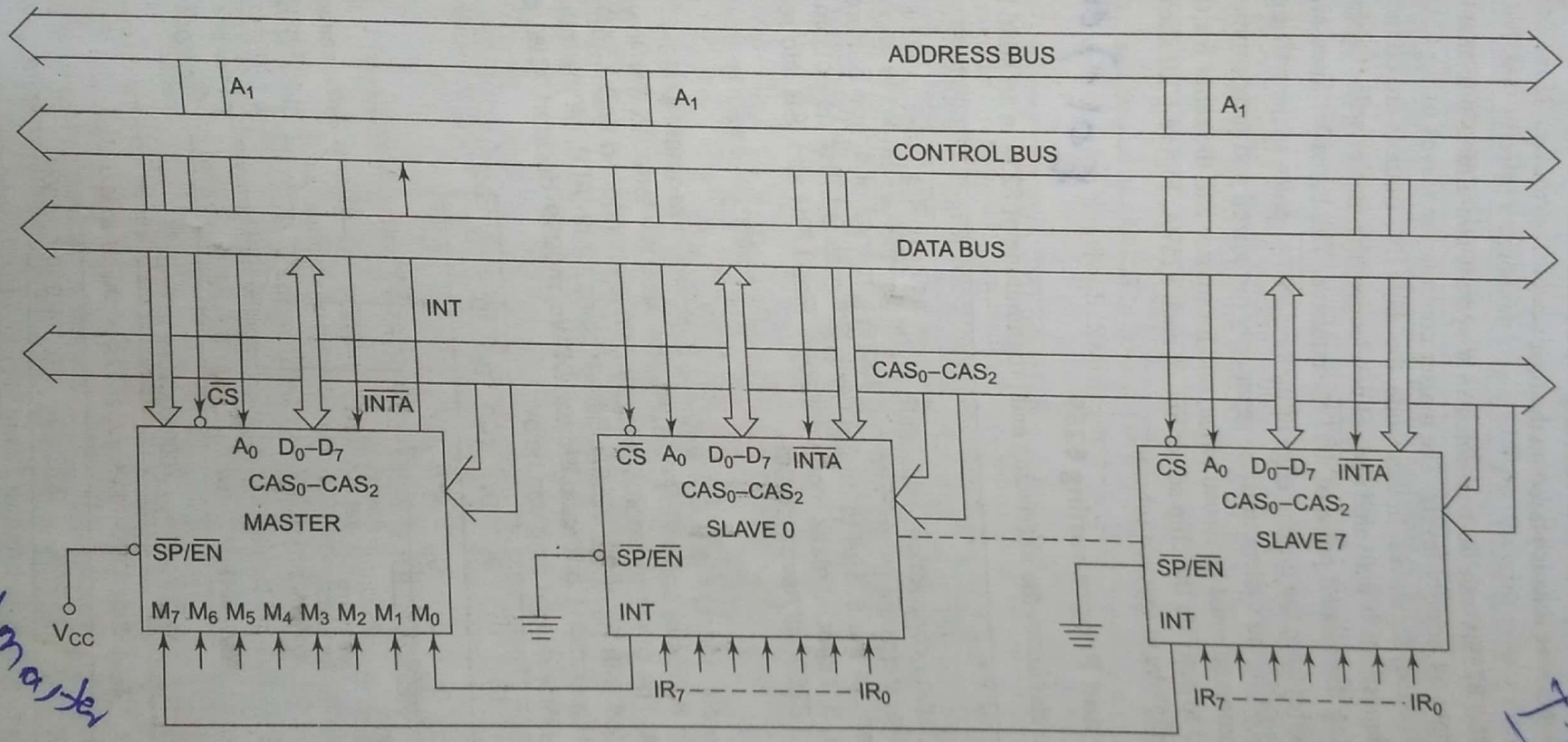


Fig. 6.20 8259A *in Cascaded Mode*

Scanned by CamScanner

Buffered Mode When the 8259A is used in the systems in which bus driving buffers are used on data buses (e.g. cascade systems), the problem of enabling the buffers arises. The 8259A sends a buffer enable signal on $\overline{SP}/\overline{EN}$ pin, whenever data is placed on the bus.

Cascade Mode The 8259A can be connected in a system containing one master and eight slaves (maximum) to handle upto 64 priority levels. The master controls the slaves using CAS_0–CAS_2 which act as chip select inputs (encoded) for slaves. In this mode, the slave INT outputs are connected with master IR inputs. When a slave request line is activated and acknowledged, the master will enable the slave to release the vector address during the second pulse of $\overline{INTA}$ sequence. The cascade lines are normally low and contain slave address codes from the trailing edge of the first $\overline{INTA}$ pulse to the trailing edge of the second $\overline{INTA}$ pulse. Each 8259A in the system must be separately initialized and programmed to work in different modes. The EOI command must be issued twice, one for master and the other for the slave. A separate address decoder is used to activate the chip select line of each 8259A. Figure 6.20 shows the details of the circuit connections of 8259As in cascade scheme.

6.2.5 Interfacing and Programming 8259

The following example elucidates the interfacing and programming of 8259 in an 8086 based system.

Problem 6.3

Show 8259A interfacing connections with 8086 at the address 074x. Write an ALP to initialize the 8259A in single level triggered mode, with call address interval of 4, non-buffered, no special fully nested mode. Then set the 8259A to operate with IR6 masked, IR4 as bottom priority level, with special EOI mode. Set special mask mode of 8259A. Read IRR and ISR into registers BH and BL respectively. IR_0 of 8259 will have type 80H.

Solution

Let the starting vector address is 0000:0200H (80H X 4 in segment 0000H). The interconnections of 8259A with 8086 are shown in Fig. 6.21. The 8259 is interfaced with lower byte of the 8086 data bus, hence A_0 line of the microprocessor system is abandoned and A_1 of the microprocessor system is connected with A_0 of the 8259A. Before going for an ALP, all the initialization command words (ICWs) and Operation Command Words (OCWs) must be decided. ICW_1 decides single level triggered, address interval of 4 as given below.

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

Always set to 0 — Don't care for 8086 system — Always set to 1

$ICW_1 = 1F$

- D_0 = 1 → ICW_4 needed
- D_1 = 1 → Single 8259A
- D_2 = 1 → Call address interval of 4
- D_3 = 1 → Level triggered mode

ICW_2 → vector address = 0000:0010 for IR3

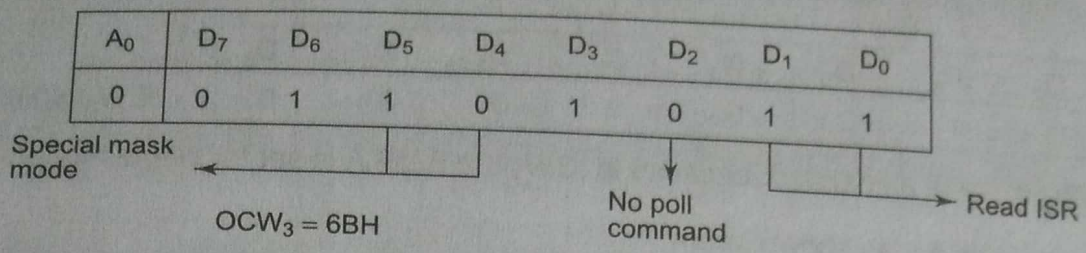
T_7	T_6	T_5	T_4	T_3	A_{10}	A_9	A_8
1	0	0	0	0	0	0	0

= 80H

IR3 Selected

Scanned by CamScanner

For reading IRR



The following ALP writes these commands to initialize the operation of the 8259A as required in the problem. Program 6.5 gives an ALP for the required initialization of 8259A.

```
CODE    SEGMENT
ASSUME  CS : CODE
START:  MOV AL, 1FH          ; Set the 8259A in single, level
        MOV DX, 0740H
        OUT DX, AL           ; triggered mode with call address of
                                   interval of 4

        MOV DX, 0742H
        MOV AL, 80H          ; Select vector address 0000:0200H
        OUT DX, AL           ; for IR3 (ICW2)
        MOV AL, 01H          ; ICW4 for 8086 system, normal
        OUT DX, AL           ; EOI, non-buffered, special fully nested
                                   mode masked

        MOV AL, 40H          ;
        OUT DX, AL           ; OCW1 for IR6 masked,
        MOV AL, 0E4H         ; Specific EOI with rotating
        MOV DX, 0740H
        OUT DX, AL           ; Priority and bottom level of IR4 with
                                   OCW2

        MOV AL, 6AH          ; Write OCW3 for reading
        OUT DX, AL           ; IRR and store in BH
        IN AL, DX            ;
        MOV BH, AL
        MOV AL, 6BH          ; Write OCW3 to read
        OUT DX, AL           ; ISR and store in
        IN AL, DX            ; BL
        MOV BL, AL
        MOV AH, 4CH          ; Return to DOS
        INT 21H
CODE    ENDS
        END START
```

Program 6.5 *ALP to Initialize 8259A for Problem 6.3*

Scanned by CamScanner

The interfacing circuit for Problem 6.3 has been shown below in Fig. 6.21.

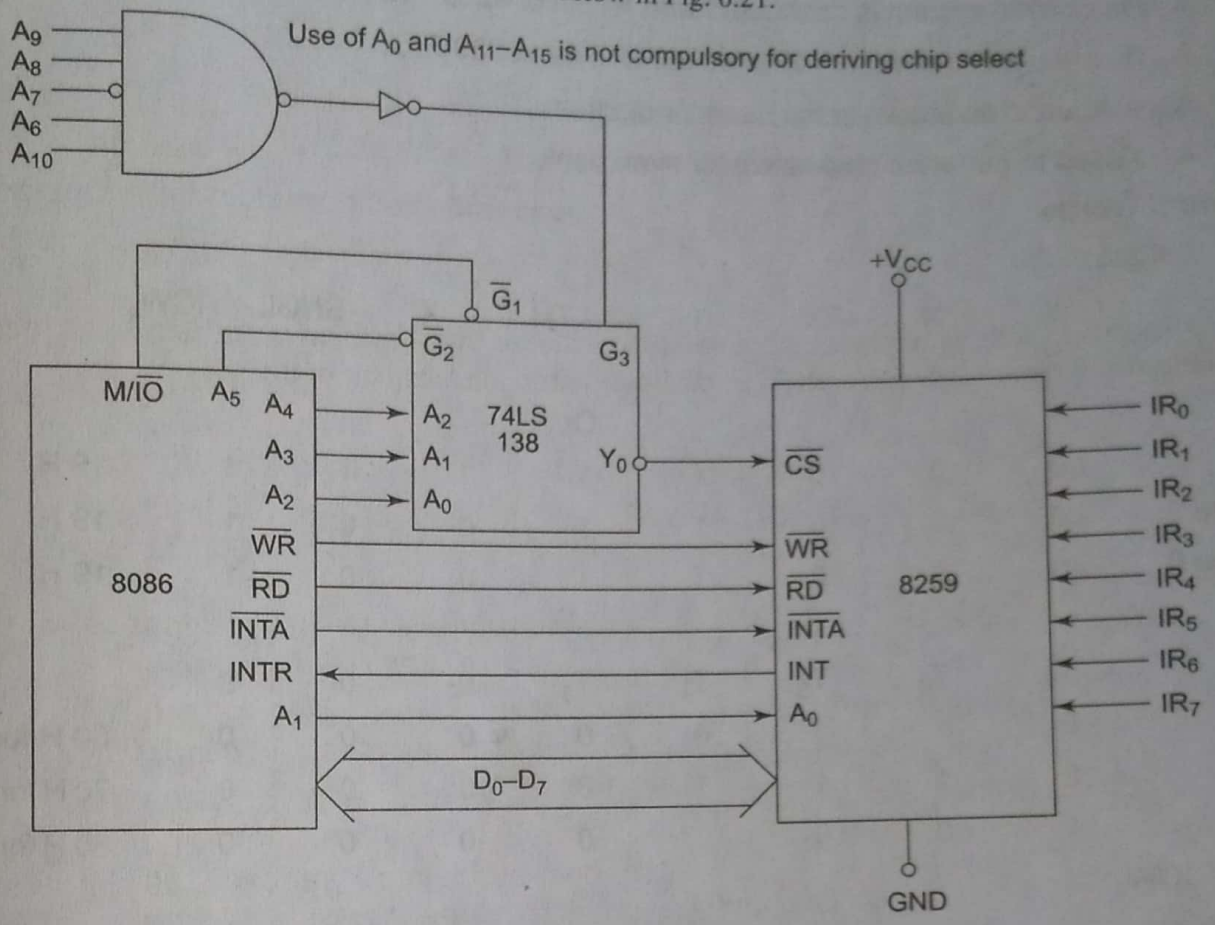


Fig. 6.21 *Interfacing 8259A with 8086*

Problem 6.4

Interface 3 ICS of 8259 PIC with 8086 system in such a way that one is master and rest two are slaves connected at IR_3 and IR_6 interrupt request level of the master. The 8259s are having vector address 60H, 70H and 80H. Write a program to initialize 8259 PIC so that IR_2 and IR_7 levels of master are masked. Initialize master in AEOI mode and automatic rotation mode in minimum mode of operation.

Solution

Address Decoding table for 8259 interfaced at even addresses.

Table 6.3

	Port	*Hex. Address*	A_{15}	A_{14}	A_1		A_0
Master	Port 0	C000H	1100	0000	0000	000	0
8259	Port 1	C002H	1100	0000	0000	001	0
$Slave_3$	Port 0	0004H	1100	0000	0000	010	0
	Port 1	C006H	1100	0000	0000	011	0
$Slave_6$	Port 0	C008H	1100	0000	0000	100	0
	Port 1	C00AH	1100	0000	0000	101	0

Scanned by CamScanner

```
        OUT 82H, AL          ; 100 kHz
        MOV AL, 0D3H         ; Clear display ram
        OUT 82H, AL          ; command
WAIT:   MOV AL, 40H          ; Read FIFO command
        OUT 82H, AL          ; for checking display RAM
        IN AL, 82H           ; Wait for clearing of
        AND AL, 80H          ; Display RAM by reading
        CMP AL, 80H          ; FIFO Du bit of the status word i.e.
        JNZ WAIT             ; If DU bit is not set wait, else proceed
        IN AL, 82H           ; Read FIFO status
        AND AL, 07H          ; Mask all bits except the
        CMP AL, 00           ; number of characters bits
        JNZ KEYCODE          ; If any key is pressed, take
WRAM:   MOV AL, 90H          ; required action, otherwise
        OUT 82H, AL          ; proceed to write display
        MOV AL, 55H          ; RAM by using write display
        MOV CH, 10H          ; command.Write the byte
NEXT:   OUT 80H, AL          ; 55H TO ALL DISPLAY RAM
        DEC CH               , locations
        JNZ NEXT             ;
        JMP STOP             ;
KEYCODE: CALL READCODE       ; Call routine to read the key
        MOV CL, AL           ; store the keycode in CL.
        JMP WRAM             ; code of the pressed key is assumed available
READCODE: MOV AL, 40H
        OUT 82H, AL
        IN AL, 80H
        RET
STOP:   MOV AH, 4CH          ; stop
        INT21H
CODE    ENDS
        END START
```

Program 6.7 *Initialisation of 8279 using an 8086 ALP for Problem 6.6*

Scanned by CamScanner

In simplex mode, data is transmitted only in one direction over a single communication channel. For example, a computer (CPU) may transmit data for a CRT display unit in this mode. In duplex mode, data may be transferred between two transreceivers in both directions simultaneously. In the half duplex mode, on the other hand, data transmission may take place in either direction, but at a time data may be transmitted only in one direction. For example, a computer may communicate with a terminal in this mode. When the terminal sends data (i.e. terminal is sender), the message is received by the computer (i.e. the computer is receiver). However, it is not possible to transmit data from the computer to the terminal and from terminal to the computer simultaneously.

6.4.2 Architecture and Signal Descriptions of 8251

The architectural block diagram of 8251A is shown in Fig. 6.26, followed by the functional description of each block.

The data buffer interfaces the internal bus of the circuit with the system bus. The read write control logic controls the operation of the peripheral depending upon the operations initiated by the CPU. This unit also selects one of the two internal addresses those are control address and data address at the behest of the C/$\overline{D}$ signal. The modem control unit handles the modem handshake signals to coordinate the communication between the modem and the USART. The transmit control unit transmits the data byte received by the data buffer from the CPU for further serial communication. This decides the transmission rate which is controlled by the $\overline{TXC}$ input frequency. This unit also derives two transmitter status signals namely

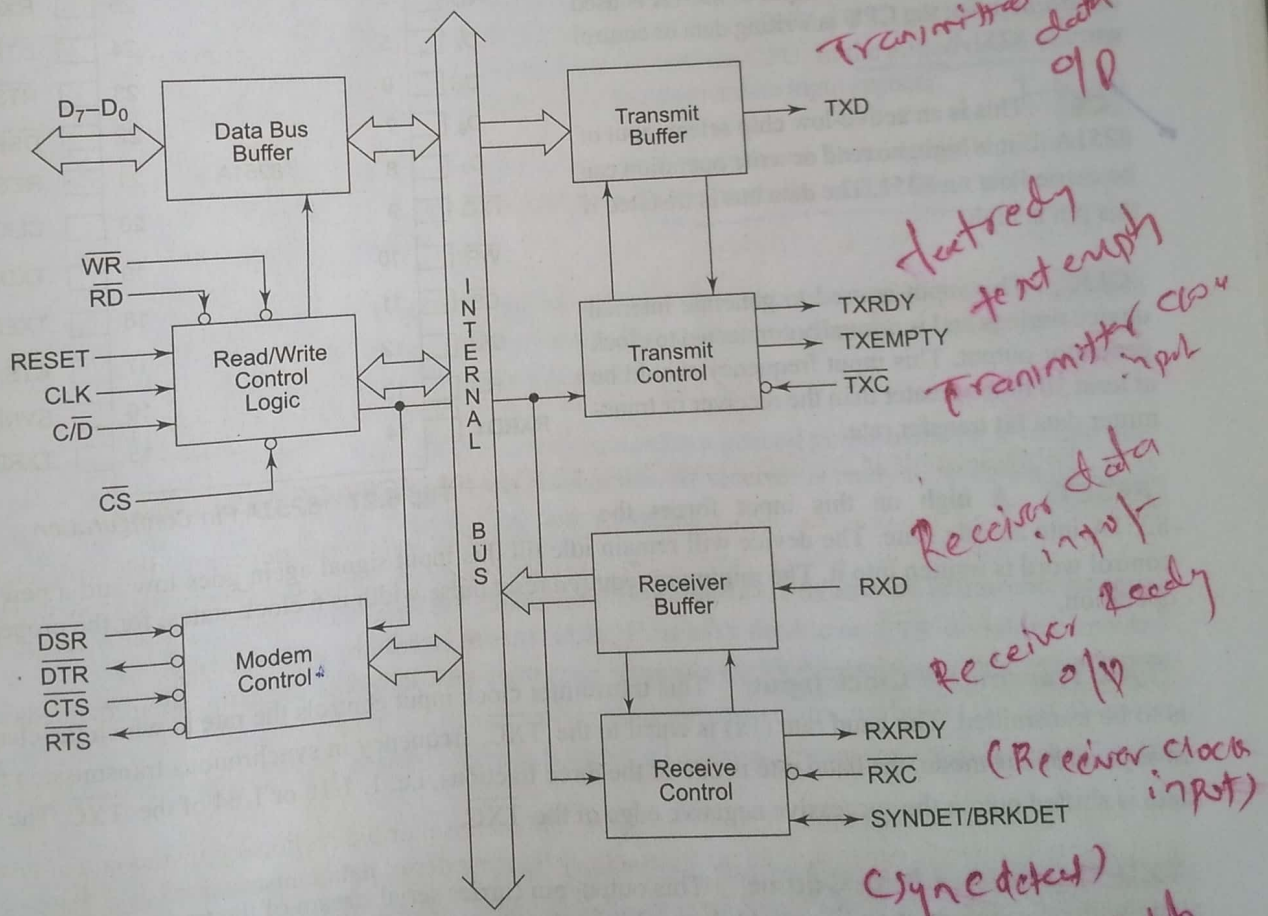


Fig. 6.26 8251A Internal Architecture

Scanned by CamScanner

TXRDY and TXEMPTY. These may be used by the CPU for handshaking. The transmit buffer is a parallel to serial converter that receives a parallel byte for conversion into a serial signal and further transmission onto the communication channel. The receive control unit decides the receiver frequency as controlled by the $\overline{RXC}$ input frequency. This unit generates a receiver ready (RXRDY) signal that may be used by the CPU for handshaking. This unit also detects a break in the data string while the 8251 is in asynchronous mode. In synchronous mode, the 8251 detects SYNC characters using SYNDET/BD pin.

The pin configuration of 8251A is shown in Fig. 6.27. The following text describes the signal descriptions of 8251A:

D_0–D_7 This is an 8-bit data bus used to read or write status, command word or data from or to the 8251A.

C/$\overline{D}$–Control Word/Data This input pin, together with $\overline{RD}$ and $\overline{WR}$ inputs, informs the 8251A that the word on the data bus is either a data or control word/status information. If this pin is 1, control/status is on the bus, otherwise data is on the bus.

$\overline{RD}$ This active-low input to 8251A is used to inform it that the CPU is reading either data or status information from its internal registers.

$\overline{WR}$ This active-low input to 8251A is used to inform it that the CPU is writing data or control word to 8251A.

$\overline{CS}$ This is an active-low chip select input of 8251A. If it is high, no read or write operation can be carried out on 8251. The data bus is tristated if this pin is high.

CLK This input is used to generate internal device timings and is normally connected to clock generator output. This input frequency should be at least 30 times greater than the receiver or transmitter data bit transfer rate.

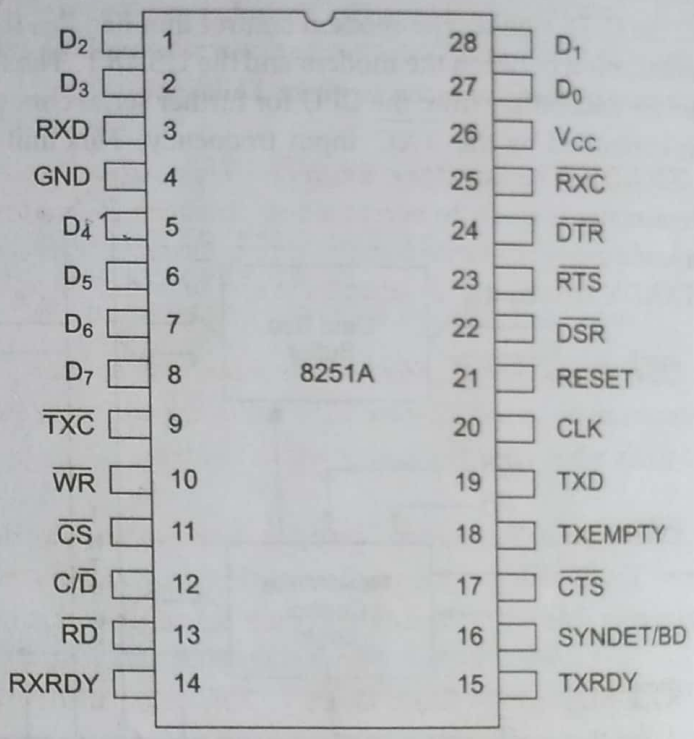


Fig. 6.27 *8251A Pin Configuration*

RESET A high on this input forces the 8251A into an idle state. The device will remain idle till this input signal again goes low and a new set of control word is written into it. The minimum required reset pulse width is 6 clock states, for the proper reset operation.

Scanned by CamScanner

$\overline{RXC}$ Receiver Clock Input This receiver clock input pin controls the rate at which the character is to be received. In synchronous mode, the baud rate is equal to the $\overline{RXC}$ frequency. In asynchronous mode, the baud rate is one of the three fractions, i.e. 1, 1/16 and 1/64th of the $\overline{RXC}$ frequency. The received data is read into the 8251 on rising edge of $\overline{RXC}$. In most of the systems, the $\overline{RXC}$ and $\overline{RXC}$ frequencies are equal.

RXD-Receive Data Input This input pin of 8251A receives a composite stream of the data to be received by 8251A.

RXRDY-Receiver Ready Output This output indicates that the 8251A contains a character to be read by the CPU. The RXRDY signal may be used either to interrupt the CPU or may be polled by the CPU. To set the RXRDY signal in asynchronous mode, the receiver must be enabled to sense a start bit and a complete character must be assembled and then transferred to the data output register. In synchronous mode, to set the RXRDY signal, the receiver must be enabled and a character must finish assembly and then be transferred to the data output register. If the data is not successfully read from the receiver data output register before assembly of the next data byte, the overrun condition error flag is set and the previous byte is over written by the next byte of the incoming data and hence it is lost.

TXRDY-Transmitter Ready This output signal indicates to the CPU that the internal circuit of the transmitter is ready to accept a new character for transmission from the CPU. The TXRDY signal is set by a leading edge of write signal if a data character is loaded into it from the CPU. In the polled operation, the TXRDY status bit will indicate the empty or full status of the transmitter data input register.

$\overline{DSR}$-Data Set Ready This input may be used as a general purpose one bit inverting input port. Its status can be checked by the CPU using a status read operation. This is normally used to check if the data set is ready when communicating with a modem.

$\overline{DTR}$-Data Terminal Ready This output may be used as a general purpose one bit inverting output port. This can be programmed low using the command word. This is used to indicate that the device is ready to accept data when the 8251 is communicating with a modem.

$\overline{RTS}$-Request to Send Data This output also may be used as a general purpose one bit inverting output port that can be programmed low to indicate the modem that the receiver is ready to receive a data byte from the modem. This signal is used to communicate with a modem.

$\overline{CTS}$-Clear to Send If the clear to send the input line is low, the 8251A is enabled to transmit the serial data, provided the enable bit in the command byte is set to '1'. If a Tx disable or $\overline{CTS}$ divsable command occurs, while the 8251A is transmitting data, the transmitter transmits all the data written to the USART prior to disabling the $\overline{CTS}$ or Tx. If the $\overline{CTS}$ disable or Tx disable command occurs just before the last character appears in the serial bit string, the character will be retransmitted again whenever the $\overline{CTS}$ is enabled or the Tx enable occurs.

TXE-Transmitter Empty If the 8251A, while transmitting, has no characters to transmit, the TXE output goes high and it automatically goes low when a character is received from the CPU, for further transmission. In synchronous mode, a 'high' on this output line indicates that a character has not been loaded and

Scanned by CamScanner

the SYNC character or characters are about to be or are being transmitted automatically as 'fillers'. The TXE signal can be used to indicate the end of a transmission mode.

SYNDET/BD-Synch Detect/Break Detect This pin is used in the synchronous mode for detecting SYNC characters (SYNDET) and may be used as either input or output. This can be programmed using the control word. After resetting, it is in the output mode. When used as an output, the SYNDET pin will go high to indicate that the 8251A has located a SYNC character in the receive mode. The SYNDET signal is automatically reset upon a following status read operation. When this is used as input, a positive going signal will cause the 8251A to start assembling a data character on the rising edge of the next RXC.

In the asynchronous mode, the pin acts as a break detect output. This goes high whenever the RXD pin remains low through two consecutive stop bit sequences. A stop bit sequence contains a stop bit, a start bit, data bits and parity bits. This is reset only with master chip reset or the RXD returning high. If the RXD returns to '1', during the last bit of the next character after the break, the break detect is latched up. The 8251A can now be cleared only with chip reset.

6.4.3 Description of 8251A Operating Modes

The 8251A can be programmed to operate in its various modes using its mode control words. A set of control words is written into the internal registers of 8251A to make it operate in the desired mode.

Once the 8251A is programmed as required, the TXRDY output is raised 'high' to signal the CPU that the 8251A is ready to receive a data byte from it that is to be further converted to serial format and transmitted. This automatically goes low when CPU writes a data byte into 8251A. In receiver mode, the 8251A receives a serial data byte from a modem or an I/O device. After receiving the entire data byte, the RXRDY signal is raised high to inform the CPU that the 8251A has a character ready for it. The RXRDY signal is automatically reset after the CPU reads the received byte from the 8251A . The 8251A cannot initiate transmission until the TX enable bit in the command word is set and a $\overline{\text{CTS}}$ signal is received from the modem or receiving I/O device.

The control words of 8251A are divided into two functional types:

1. Mode Instruction control word
2. Command Instruction control word

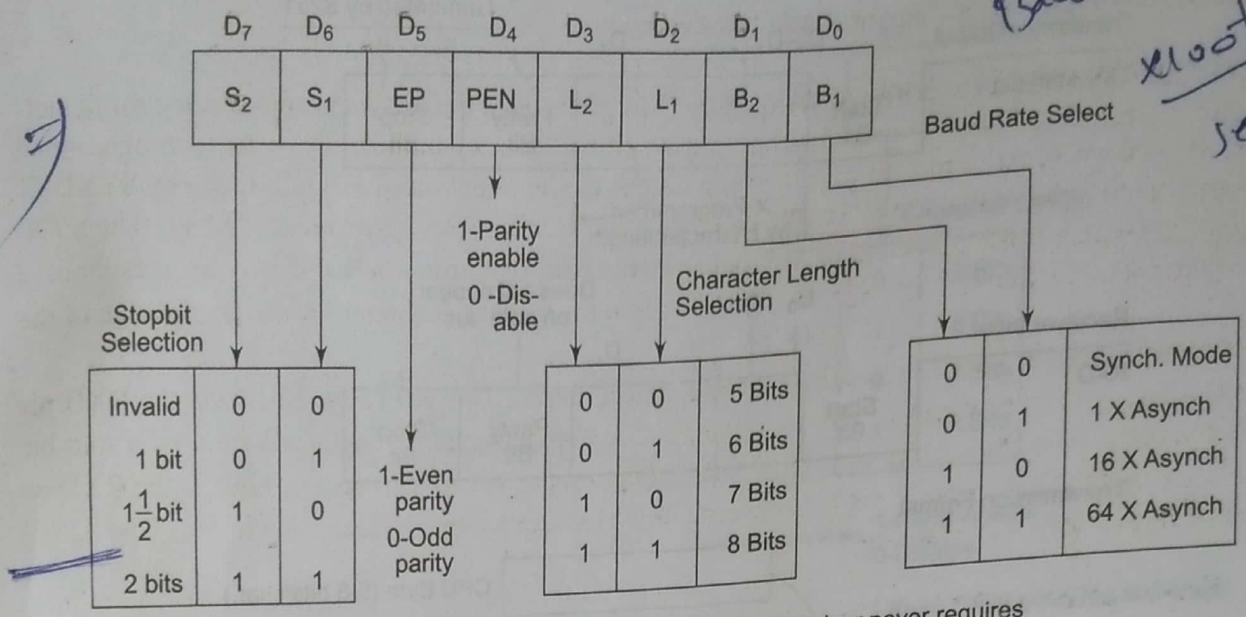
Asynchronous Mode

Mode Instruction Control Word This defines the general operational characteristics of 8251A. After internal (reset command) or external (reset input pin) reset, this must be written to configure the 8251A as per the required operation. Once this has been written into 8251A, SYNC characters or command instructions may be programmed further as per the requirements. To change the mode of operation from synchronous to asynchronous or vice-a-versa, the 8251A has to be reset using master chip reset.

Figure 6.28 shows asynchronous mode instruction control word format.

Asynchronous Mode (Transmission) When a data character is sent to 8251A by the CPU, it adds start bits prior to the serial data bits, followed by optional parity bit and stop bits using the asynchronous mode instruction control word format. This sequence is then transmitted using TXD output pin on the falling edge of $\overline{\text{TXC}}$. When no data characters are sent by the CPU to 8251A the TXD output remains 'high', if a 'break' has not been detected.

Scanned by CamScanner



N.B.—Stop bit selection as above is only for transmitter. Receiver never requires more than one stop bit

Fig. 6.28 *Mode Instruction Format Asynchronous Mode*

Asynchronous Mode (Receive) A falling edge on RXD input line marks a start bit. At baud rates of 16x and 64x, this start bit is again checked at the center of start bit pulse and if detected low, it is a valid start bit which starts counting. The bit counter locates the data bits, parity bit and stop bit. If a parity error occurs, the parity error flag is set. If a low level is detected as the stop bit, the framing error flag is set. The receiver requires only one stop bit to mark end of the data bit string, regardless of the stop bit programmed at the transmitting end. This 8-bit character is then loaded into parallel I/O buffer of 8251A. RXRDY pin is then raised high to indicate to the CPU that a character is ready for it. If the previous character has not been read by the CPU, the new character replaces it, and the overrun flag is set indicating that the previous character is lost. These error flags can be cleared using an error reset instruction. Figure 6.29 shows asynchronous mode transmission and receiver data formats. If character length is 5 to 7 bits then the remaining bits are set to zero.

Synchronous Mode

Figure 6.30 shows the synchronous mode instruction format with its bit definitions.

Synchronous Mode (Transmission) The TXD output is high until the CPU sends a character to 8251A which usually is a SYNC character. When $\overline{CTS}$ line goes low, the first character is serially transmitted out. All the characters are shifted out on the falling edges of $\overline{TXC}$. Data is shifted out at the same rate as $\overline{TXC}$, over TXD output line. If the CPU buffer becomes empty, the SYNC character or characters are inserted in the data stream over TXD output. The TXEMPTY pin is raised high to indicate that the 8251A is empty (i.e. it does not have any byte to transmit) and is transmitting SYNC characters. The TXEMPTY pin is reset, automatically when a data character is written to 8251A by the CPU. Figure 6.31 shows the relation between TXEMPTY and SYNC character insertion.

Synchronous Mode (Receiver) In this mode, the character synchronization can be achieved internally or externally. If this mode is programmed, then 'ENTER HUNT' command should be included in the first command instruction word written into the 8251A. The data on RXD pin is sampled on rising edge of the $\overline{RXC}$.

Scanned by CamScanner

```
        DEC CL              ; Decrement counter
        JNZ NXTBT           ; Repeat,if CL is not zero
        MOV AH,4CH          ; If CL is 0, return to DOS
        INT 21H
CODE    ENDS
        END START
```

Program 6.9 *ALP to Receive 100 Bytes*

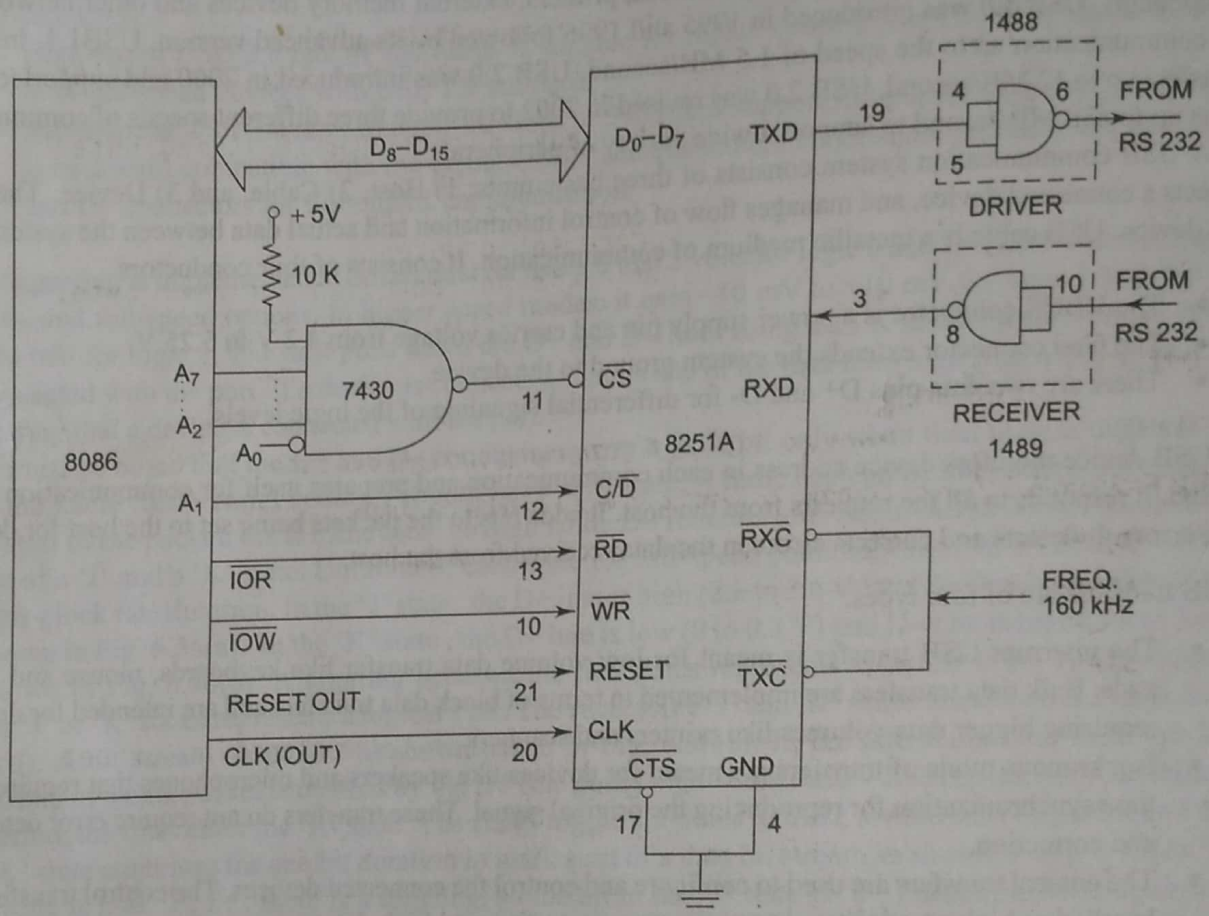


Fig. 6.35 *Interfacing of 8251A with 8086*

6.4.5 High-Speed Communication Standard; Universal Serial Bus (USB) Functional Description

Universal serial bus is the most popular serial communication standard introduced by USB Implementers Forum (USB-IF). RS-232c had been used for long and has a few serous disadvantages like limited speed of communication, high-voltage level signaling and big-size communication adapters. The USB standards are available with speeds from a few hundred kilobytes per second (USB1.x) to around 5 GB per second (USB3.x). The logic levels used for representing the bits use voltages less than 5 volts and currents less than 500 mA. A USB uses comparatively small size of connectors for establishing connection with compatible

Scanned by CamScanner

devices. In fact, in addition to the standard USB connector available with PCs, many small-size versions are introduced specially for embedded products. Recent USB ports are used for many other activities like battery charging, communication with PC and even device-to-device communications.

USB implements an asynchronous serial communication. Initially, it used to be half-duplex. However, the new USB standard supports full-duplex communication at the cost of additional two conductors. The data is transmitted in the form of packets containing start, data, parity and stop bits. The attached devices are detected and configured automatically, provided they are USB compatible. USB standards provide automatic detection and correction of errors. Design and implementation of a USB compatible system requires design certification and licensing from the USB-IF.

In its original form, USB provides a serial bus standard for connecting peripherals such as mouse, keyboard, gamepads and joysticks, scanners, cameras, printers, external memory devices and other networking components. USB 1.0 was introduced in 1995 and 1996 followed by its advanced version, USB1.1, in 1998 for communication up to the speed of 1.5 MB/second. USB 2.0 was introduced in 2000 and supported data transfers up to 12 MB/second. USB 2.0 was revised in 2002 to provide three different speeds of communications up to 480 MB/second to support a wide variety of peripherals.

A USB communication system consists of three basic units: 1) Host, 2) Cable, and 3) Device. The host detects a connected device, and manages flow of control information and actual data between the system and the device. USB cable is a metallic medium of communication. It consists of four conductors.

- The VBUS conductor is a power supply pin and carries voltage from 4.2 V to 5.25 V.
- The Gnd connector extends the system ground to the device.
- There are two data pins D+ and D- for differential signaling of the logic levels.

A USB device monitors device address in each communication and prepares itself for communication if selected. It responds to all the requests from the host. It adds bits to the packets being set to the host for detecting errors. It detects and corrects errors in the data received from the host.

USB transfers are of four types.

- The interrupt USB transfer is meant for low-volume data transfer like keyboards, mouse and touch pads. Bulk data transfers are implemented in terms of block data transfers and are intended for devices requiring bigger data volumes like printers and scanners.
- Isochronous mode of transfers are meant for devices like speakers and microphones that require real-time synchronization for reproducing the original signal. These transfers do not require error detection and correction.
- The control transfers are used to configure and control the connected devices. The control transfers are handled as highest priority, automatic error protection and high data rate transfers.
- The USB asynchronous communication transfers four types of bit packets over the cable.
  - The first type of packet is called 'Handshake Packet'. The handshake packet consists of a packet identification, i.e., PID byte, and are used for reliable communication of data packets.
  - The second type of packet is called 'Token Packet'. The token packets are always sent by the host and contain 2 bytes including 11-bit address and 5 bits of cyclic redundancy code. The token packet commands the device either to receive data or transmit data in response to specific demands from the host.
  - Data packets contain actual bytes of data up to 1024 bytes. It also includes a 16-bit CRC code.

Scanned by CamScanner

- A fourth type of packets called PRE packets are only of importance to low-speed USB devices. They contain a special PID value allotted to low-speed devices. High-speed devices neglect the PRE packets. However, the PRE values are used by network hubs while communicating with USB devices over a network. Over networks, the hubs control the data flow rate even to the high-speed devices subject to the instantaneous network traffic.

Signaling in USB Systems Available USB standards support four signaling rates.

- Low-speed USB 1.0 rate of 1.5 Mbps is suitable for human interface devices.
- Full-speed USB 1.0 rate of 12 Mbps is suitable for communication with network hubs using networks like LAN.
- High-speed USD 2.0 rate of 480 Mbps is suitable for higher speed devices but it is also compatible with full-speed devices with USB 1.0 standard.
- Super-speed USB 3.0 rates of up to 5.0 Gbps or 596 Mbps support full-duplex operation. However, it is backward compatible with the earlier USB standards though they require an additional pair of D+ and D– connectors for full-duplex communication.

USB signaling is implemented in differential for using 0 to 0.3 volts for logic 0 and 2.6 to 3.6 volts for logic 1 in low- and full-speed options. In higher speed modes, it uses –10 mV to +10 mV for logic 1 and 360 mV to 400 mV for logic 1. The host pulls down the D+ and D– lines using a 15 K resistor indicating no device is connected with the port. If a device is connected, it pulls one of the data lines high with a pull up of 1.5 K indicating that a device is connected with the port.

It must be noted that the D+ and D– conductors carry a valid bit only when their state is different and thus the name 'differential coding'. If both have zero state, it indicates end of the packet. To mark the next start of the packet, the D+ line must go high while D– remains low. Serial bus states are described in terms of a 'J' and a 'K' state. Duration of each state for full-speed communication in 1/12 MHz = 83 ns, i.e., bit-clock rate duration. In the 'J' state , the D+ line is high (2.6 to 3.6 V)and D– line is low (0 to 0.3 V) as shown in Fig. 6.36(a). In the 'K' state , the D+ line is low (0 to 0.3 V) and D– line is high (2.6 to 3.6 V) as shown in Fig. 6.36(b). A toggling between the two states represents a logical 0 bit. A repetition of either 'J' or 'K' states represents a logical 1 bit. The successive 'J' and 'K' states are shown in Figs 6.36 (c) and (d). A bit stream '0100100' is shown coded in Fig. 6.36(e). In the idle state, both lines are low. A starting of a data packet is marked by the D+ line going high while the D– line remains low. In the next bit period, the lines enter the 'K' state. The states toggle six times to mark a valid start of packet and then the 'K' state continues for one bit duration to mark start of a data bit stream as shown in Fig. 6.36(e). The next state is now 'J', i.e., there is a toggling, so the bit in the 'J' state is '0'. Further, there is again a 'J' state, i.e., a state continues, it represents a '1' bit. Thus, if there is a toggling between J to K or K to J, it represents a '0' bit and if there is a continuation of states, either J to J or K to K, it represents a '1' bit. A data packet may contain up to 1024 bits followed by a 16-bit CRC code. The end of the packet is marked by both the lines D+ and D– being pulled low for more than one-bit duration. If there are more than six successive '1' bits in the stream, a zero is inserted by the host or a USB compatible device after the sixth '1'. This is called bit stuffing. Thus, received seven consecutive '1' are considered an error. In case of an error, the same data packet may be requested again by the device using a special type of the handshake packet.

Scanned by CamScanner

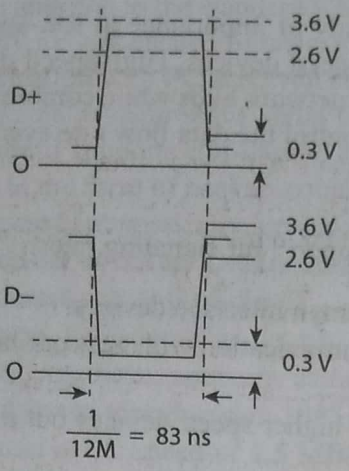


Fig. 6.36(a) *J State*

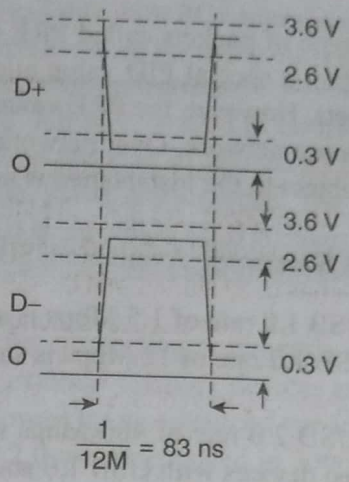


Fig. 6.36(b) *K State*

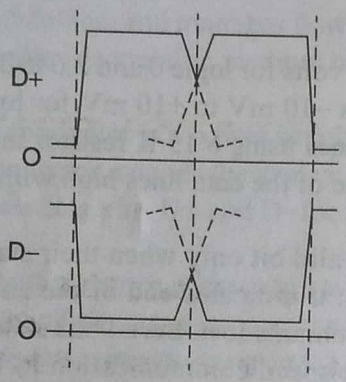


Fig. 6.36(c) *Successive J States*

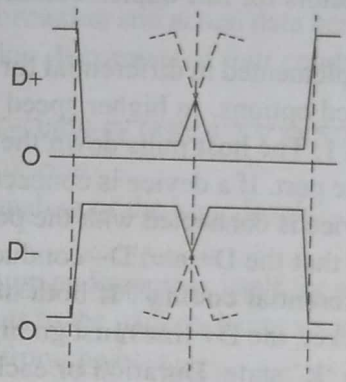


Fig. 6.36 (d) *Successive K States*

- Toggling between J & K states indicates a logics
- Successive states either J or K indicate logic 1

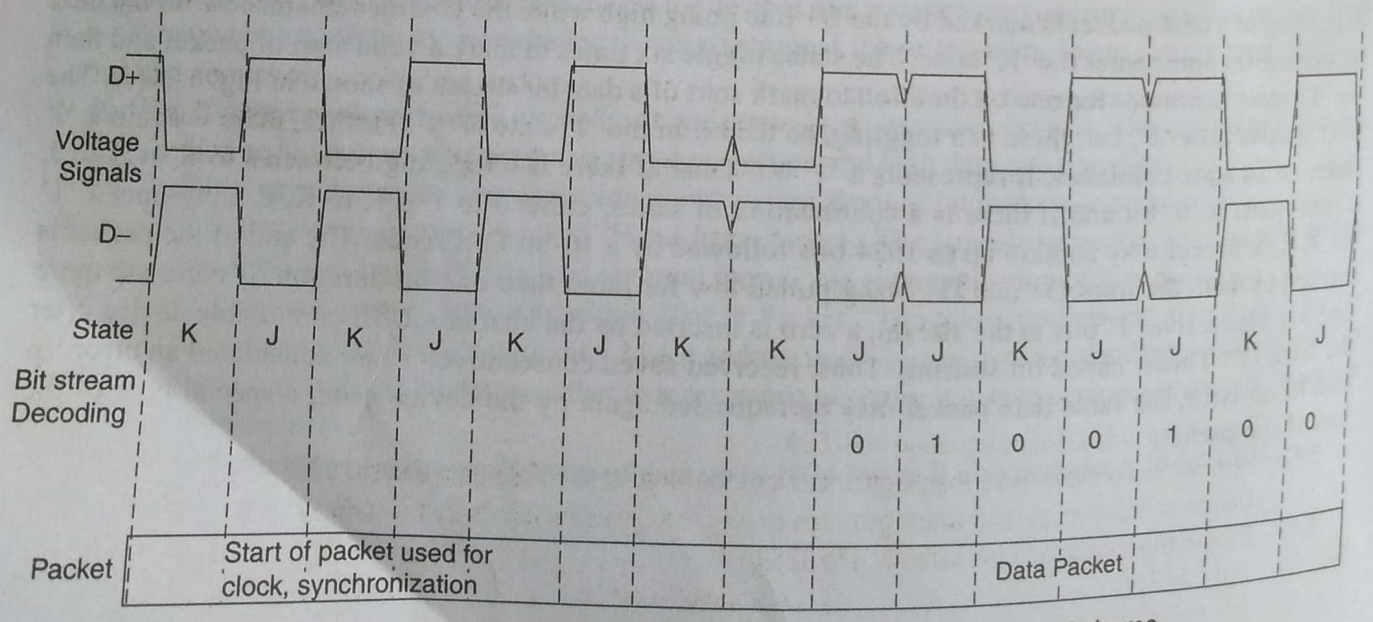


Fig. 6.36(e) *Coding of a Bit Stream Using the USB Signalling Scheme.*

Scanned by CamScanner

In the previous chapter, we studied some of the dedicated peripherals and their interfacing techniques with 8086. In this chapter, we will further discuss a few advanced peripherals and their interfacing techniques with 8086. In the applications where the CPU is to transfer bulk data, it may be a waste of time to transfer the data from source to destination using program controlled data transfer or interrupt driven data transfer. The alternate way of transferring the bulk data is the Direct Memory Access (DMA) technique in which the data is transferred under the control of a DMA controller, after it is properly initialised by the CPU. A DMA controller is designed to complete the bulk data transfer task much faster than the CPU. One such application which involves bulk data transfer is the storage of programs or data into secondary memories. At the end we have presented a brief overview of high capacity memory devices like old days floppy, Compact disk, Digital video disk and Hard disk drive.

Scanned by CamScanner

7.1.1 Internal Architecture of 8257

The internal architecture of 8257 is depicted in Fig. 7.1. The chip supports four DMA channels, i.e. four peripheral devices can independently request for DMA data transfer through these channels at a time. The DMA controller has 8-bit internal data buffer, a read/write unit, a control unit, a priority resolving unit along with a set of registers. We will discuss each one of them in the following sections. Next, we describe the register organisation of 8257.

Register Organisation of 8257 The 8257 performs the DMA operation over four independent DMA channels. Each of the four channels of 8257 has a pair of two 16-bit registers, viz. DMA *address register* and *terminal count register*. Also, there are two common registers for all the channels, namely, *mode set register* and *status register*. Thus there are a total of ten registers. The CPU selects one of these ten registers using address lines A_0–A_3. Table 7.1 shows how the A_0–A_3 bits may be used for selecting one of these registers. We will now describe each register as follows:

DMA Address Registers Each DMA channel has one DMA address register. The function of this register is to store the address of the starting memory location, which will be accessed by the DMA channel. Thus the starting address of the memory block which will be accessed by the device is first loaded in the DMA address register of the channel. Naturally, the device that wants to transfer data over a DMA channel, will access the block of memory with the starting address stored in the DMA Address Register.

Terminal Count Register As in the previous case, each of the four DMA channels of 8257 has one terminal count register (TC). This 16-bit register is used for ascertaining that the data transfer through a DMA channel ceases or stops after the required number of DMA cycles. Thus this register should be appropriately written before the actual DMA operation starts. The low order 14-bits of the terminal count register are initialised with the binary equivalent of the number of required DMA cycles minus one. After each DMA cycle, the terminal count register content will be decremented by one and finally it becomes zero after the required number of DMA cycles are over.

The bits 14 and 15 of this register indicate the type of the DMA operation (transfer). If the device wants to write data into the memory, the DMA operation is called DMA write operation. Bit 14 of the register in this case will be set to one and bit 15 will be set to zero. Table 7.2 gives details of DMA operation selection and the corresponding bit configuration of the bits 14 and 15 of the TC register.

Mode Set Register The mode set register is used for programming the 8257 as per the requirements of the system. The function of the mode set register is to enable the DMA channels individually and also to set the various modes of operation. A DMA channel should not be enabled till the DMA address register and the terminal count register contain valid information, otherwise, an unwanted DMA request may initiate a DMA cycle, probably destroying the valid memory data.

Table 7.1 *8257 Register Selection*

Register	*Byte*	*Address Inputs*			*F/L*		*BI-Directional Data Bus*							
		A_3	A_2	A_1	A_0		D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CH-0 DMA Address	LSB	0	0	0	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	MSB	0	0	0	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-0 Terminal Count	LSB	0	0	0	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0

(Contd.)

Scanned by CamScanner

Register	Byte	Address Inputs				F/L	BI-Directional Data Bus							
		A_3	A_2	A_1	A_0		D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CH-1 DMA Address	MSB	0	0	0	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
	LSB	0	0	1	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
CH-1 Terminal Count	MSB	0	0	1	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
	LSB	0	0	1	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
CH-2 DMA Address	MSB	0	0	1	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
	LSB	0	1	0	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
CH-2 Terminal Count	MSB	0	1	0	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
	LSB	0	1	0	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
CH-3 DMA Address	MSB	0	1	0	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
	LSB	0	1	1	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
CH-3 Terminal Count	MSB	0	1	1	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
	LSB	0	1	1	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	MSB	0	1	1	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
MODE SET (Programme only)	—	1	0	0	0	0	AL	TCS	EW	RP	EN3	EN2	EN1	EN0
STATUS (Read only)	—	1	0	0	0	0	0	0	0	UP	TC3	TC2	TC1	TC0

Scanned by CamScanner

Table 7.1 (Contd.)

Register	Byte	Address Inputs				F/L	BI-Directional Data Bus							
		A_3	A_2	A_1	A_0		D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
	MSB	0	0	0	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
CH-1 DMA Address	LSB	0	0	1	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	MSB	0	0	1	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-1 Terminal Count	LSB	0	0	1	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	MSB	0	0	1	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
CH-2 DMA Address	LSB	0	1	0	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	MSB	0	1	0	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-2 Terminal Count	LSB	0	1	0	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	MSB	0	1	0	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
CH-3 DMA Address	LSB	0	1	1	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	MSB	0	1	1	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-3 Terminal Count	LSB	0	1	1	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	MSB	0	1	1	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
MODE SET	—	1	0	0	0	0	AL	TCS	EW	RP	EN3	EN2	EN1	EN0

Scanned by CamScanner

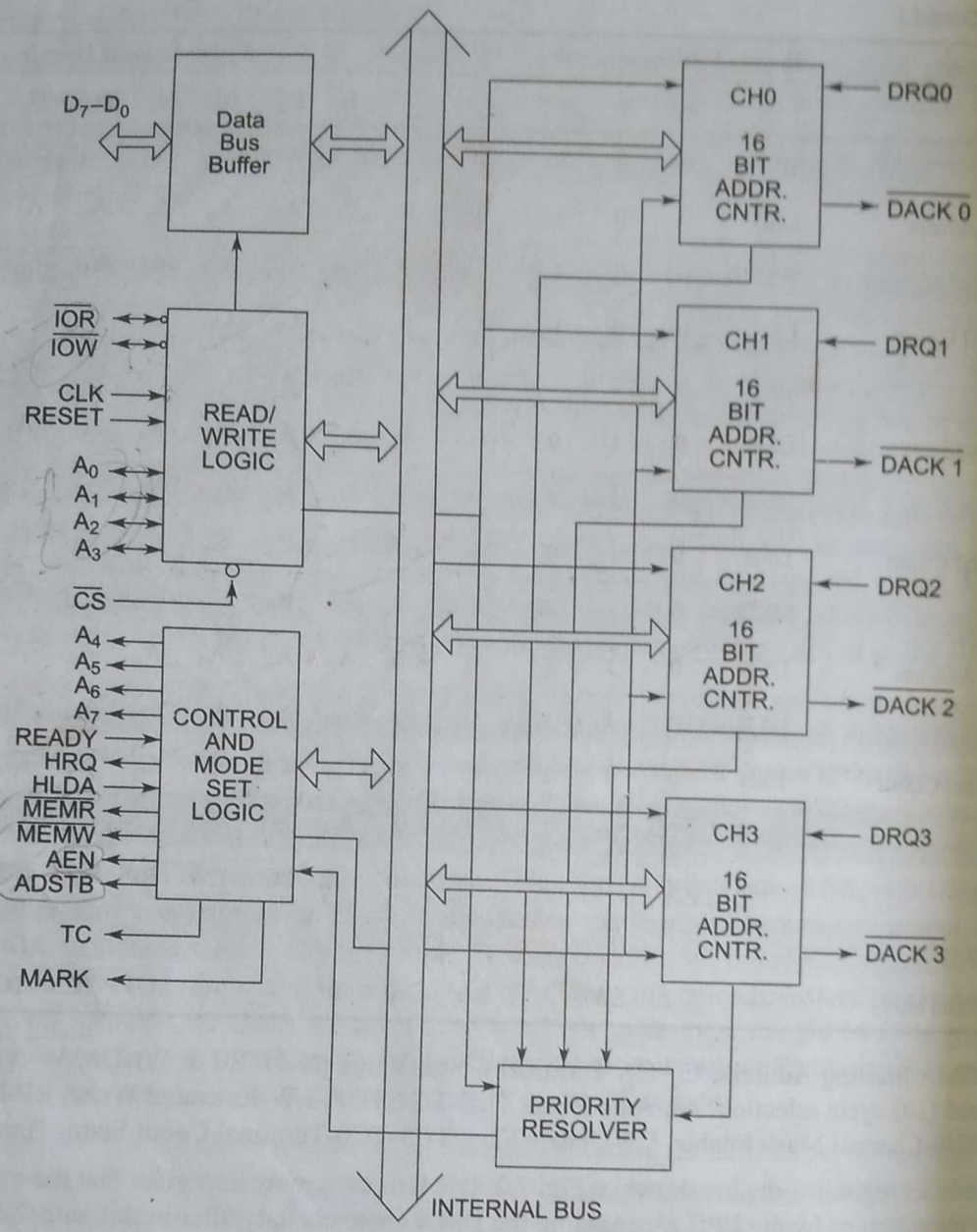


Fig. 7.1 *Internal Architecture of 8257*

The bits D_0–D_3 enable one of the four DMA channels of 8257. For example, if D_0 is '1', channel 0 is abled. If bit D_4 is set, rotating priority is enabled, otherwise, the normal, i.e. fixed priority is enabled. The rmal and rotating priorities will be explained later in this text.

Scanned by CamScanner

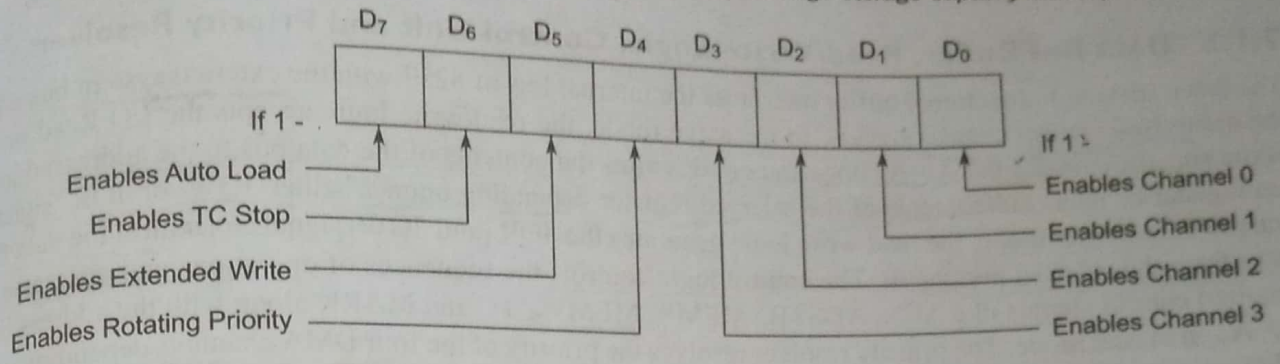


Fig. 7.2 *Bit Definitions of the Mode Set Register*

If the TC STOP bit is set, the selected channel is disabled after the *terminal count* condition is reached, and it further prevents any DMA cycle on the channel. To enable the channel again, this bit must be reprogrammed. If the TC STOP bit is programmed to be zero, the channel is not disabled, even after the count reaches zero and further requests are allowed on the same channel.

The auto load bit, if set, enables channel 2 for the repeat block chaining operations, without immediate sofware intervention between the two successive blocks. The channel 2 registers are used as usual, while the channel 3 registers are used to store the block reinitialisation parameters, i.e. the DMA starting address and the terminal count. After the first block is transferred using DMA, the channel 2 registers are reloaded with the corresponding channel 3 registers for the next block transfer, if the *Update* flag is set.

The extended write bit, if set to '1', extends the duration of $\overline{\text{MEMW}}$ and $\overline{\text{IOW}}$ signals by activating them earlier. This is useful in interfacing the peripherals with different access times. If the peripheral is not accessed within the stipulated time, it is expected to give the 'NOT READY' indication to 8257, to request it to add one or more wait states in the DMA cycle. The mode set register can only be written into.

Status Register The status register of 8257 is shown in Fig. 7.3. The lower order 4-bits of this register contain the terminal count status for the four individual channels. If any of these bits is set, it indicates that the specific channel has reached the terminal count condition. These bits remain set till either the status is read by the CPU or the 8257 is reset. The update flag is not affected by the read operation. This flag can only be cleared by resetting 8257 or by resetting the auto load bit of the mode set register. If the update flag is set, the contents of the channel 3 registers are reloaded to the corresponding registers of

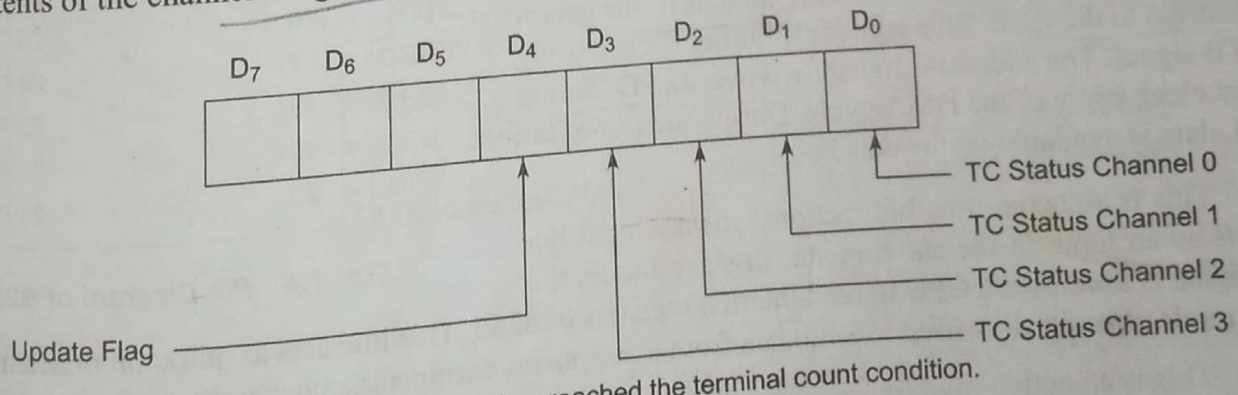


Fig. 7.3 *Bit Definitions of Status Register of 8257*

channel 2, whenever the channel 2 reaches a terminal count condition, after transferring one block and the next block is to be transferred using the autoload feature of 8257. The update flag is set every time, the channel 2 registers are loaded with contents of the channel 3 registers. It is cleared by the completion of the first DMA cycle of the new block. This register can only be read.

Scanned by CamScanner

7.1.2 Data Bus Buffer, Read/Write Logic, Control Unit and Priority Resolver

The 8-bit, tristate, bidirectional buffer interfaces the internal bus of 8257 with the external system bus under the control of various control signals. In the slave mode, the read/write logic accepts the I/O Read or I/O Write signals, decodes the A_0–A_3 lines and either writes the contents of the data bus to the addressed internal register or reads the contents of the selected register depending upon whether $\overline{IOW}$ or $\overline{IOR}$ signal is activated. In master mode, the read/write logic generates the $\overline{IOR}$ and $\overline{IOW}$ signals to control the data flow to or from the selected peripheral. The control logic controls the sequences of operations and generates the required control signals like AEN, ADSTB, $\overline{MEMR}$, $\overline{MEMW}$, TC and MARK along with the address lines A_4–A_7, in master mode. The priority resolver resolves the priority of the four DMA channels depending upon whether normal priority or rotating priority is programmed.

7.1.3 Signal Descriptions of 8257

Figure 7.4 shows pin configuration of 8257, followed by the functional description of each signal.

DRQ_0–DRQ_3 These are the four individual channel DMA request inputs, used by the peripheral devices for requesting DMA services. The DRQ_0 has the highest priority while DRQ_3 has the lowest one, if the fixed priority mode is selected.

$\overline{DACK_0}$ – $\overline{DACK_3}$ These are the active-low DMA acknowledge output lines which inform the requesting peripheral that the request has been honoured and the bus is relinquished by the CPU. These lines may act as strobe lines for the requesting devices.

D_0–D_7 These are bidirectional, data lines used to interface the system bus with the internal data bus of 8257. These lines carry command words to 8257 and status word from 8257, in slave mode, i.e. under the control of CPU. The data over these lines may be transferred in both the directions. When the 8257 is the bus master (master mode, i.e. not under CPU control), it uses D_0–D_7 lines to send higher byte of the generated address to the latch. This address is further latched using ADSTB signal. The address is transferred over D_0–D_7 during the first clock cycle of the DMA cycle. During the rest of the period, data is available on the data bus.

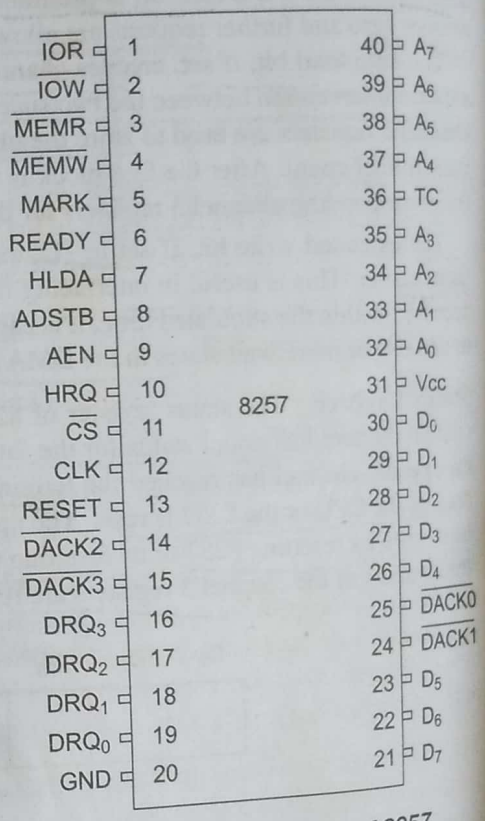


Fig. 7.4 *Pin Diagram of 8257*

$\overline{IOR}$ This is an active-low bidirectional tristate input line that acts as an input in the slave mode. In slave mode, this input signal is used by the CPU to read internal registers of 8257. This line acts as output in master mode. In master mode, this signal is used to read data from a peripheral during a memory write cycle.

$\overline{IOW}$ This is an active-low, bidirectional tristate line that acts as input in slave mode to load the contents of the data bus to the 8-bit mode register or upper/lower byte of a 16-bit DMA address register or terminal count register. In the master mode, it is a control output that loads the data to a peripheral during DMA memory read cycle (write to peripheral).

CLK This is a clock frequency input required to derive basic system timings for the internal operation of 8257.

Scanned by CamScanner

Data pointer

RESET This active-high asynchronous input disables all the DMA channels by clearing the mode register and tristates all the control lines.

A_0–A_3 These are the four least significant address lines. In slave mode, they act as input which select one of the registers to be read or written. In the master mode, they are the four least significant memory address output lines generated by 8257.

$\overline{CS}$ This is an active-low chip select line that enables the read/write operations from/to 8257, in slave mode. In the master mode, it is automatically disabled to prevent the chip from getting selected (by CPU) while performing the DMA operation.

A_4–A_7 This is the higher nibble of the lower byte address generated by 8257 during the master mode of DMA operation.

READY This is an active-high asynchronous input used to stretch memory read and write cycles of 8257 by inserting wait states. This is used while interfacing slower peripherals.

HRQ The hold request output requests the access of the system bus. In the non-cascaded 8257 systems, this is connected with HOLD pin of CPU. In the cascade mode, this pin of a slave is connected with a DRQ input line of the master 8257, while that of the master is connected with HOLD input of the CPU.

HLDA The CPU drives this input to the DMA controller high, while granting the bus to the device. This pin is connected to the HLDA output of the CPU. This input, if high, indicates to the DMA controller that the bus has been granted to the requesting peripheral by the CPU.

$\overline{MEMR}$ This active-low memory read output is used to read data from the addressed memory locations during DMA read cycles.

$\overline{MEMW}$ This active-low three state output is used to write data to the addressed memory location during DMA write operation.

ADSTB This output from 8257 strobes the higher byte of the memory address generated by the DMA controller into the latches.

AEN This output is used to disable the system data bus and the control the bus driven by the CPU. This may be used to disable the system address and data bus by using the enable input of the bus drivers to inhibit the non-DMA devices from responding during DMA operations. This also may be used to transfer the higher byte of the generated address over the data bus. If the 8257 is I/O mapped, this should be used to disable the other I/O devices, when the DMA controller address is on the address bus.

TC Terminal count output indicates to the currently selected peripheral that the present DMA cycle is the last for the previously programmed data block. If the TC STOP bit in the mode set register is set, the selected channel will be disabled at the end of the DMA cycle. The TC pin is activated when the 14-bit content of the terminal count register of the selected channel becomes equal to zero. The lower order 14 bits of the terminal count register are to be programmed with a 14-bit equivalent of (n-1), if n is the desired number of DMA cycles.

MARK The modulo 128 mark output indicates to the selected peripheral that the current DMA cycle is the 128th cycle since the previous MARK output. The mark will be activated after each 128 cycles or integral multiples of it from the beginning of the data block (the first DMA cycle), if the total number of the required DMA cycles (n) is completely divisible by 128.

V_{cc} This is a +5V supply pin required for operation of the circuit.

GND This is a return line for the supply (ground pin of the IC).

Scanned by CamScanner

7.2 DMA TRANSFERS AND OPERATIONS

The 8257 is able to accomplish three types of operations, viz. verify DMA operation, write operation and read operation. The complete operational sequence of 8257 is described using a state diagram in Fig. 7.5 for a single channel.

A single byte transfer using 8257 may be requested by an I/O device using any one of the 8257 DRQ inputs. In response, the 8257 sends HRQ signal to the CPU at its HLD input and waits for acknowledgement at the HLDA input. If the HLDA signal is received by the DMA controller, it indicates that the bus is available for the transfer. The $\overline{DACK}$ line of the used channel is pulled down by the DMA controller to indicate the I/O device that its request for the DMA transfer has been honoured by the CPU. The DMA controller generates the read and write commands to transfer the byte from/to the I/O device. The $\overline{DACK}$ line is pulled transfer is over, to indicate the DMA controller that the transfer, as requested by the device, is over. The HRQ line is lowered by the DMA controller to indicate the CPU that it may regain the control of the bus. The DRQ must be high until acknowledged and must go low before S_4 state of the DMA operation state diagram to avoid another unwanted transfer.

If more than one channel requests service simultaneously, the transfer will occur as a *burst transfer*. This will be discussed further in case of 8237. No overhead is required in switching from one channel to another. In each S_4, the DRQ lines are sampled and the highest priority request is recognized during the next transfer. Once the higher priority transfer is over; the lower priority transfer requests may be served, provided their DRQ lines are still active. The HRQ line is maintained active till all the DRQ lines go low.

The burst or continuous transfer, described above may be interrupted by an external device by pulling down the HLDA line. After each transfer, the 8257 checks the HLDA line. If it is found active, it completes the current transfer and releases the HRQ line (i.e. sends it low) and returns to its idle state. If the DRQ line is still active, the 8257 will again activate HRQ and proceed as already described. The 8257 uses four clock cycles to complete a transfer. The 8257 has a READY input to interface it with low speed devices. The READY pin status is checked in S_3 of the state diagram. If it is low, the 8257 enters a wait state. The status is sampled in every state till it goes high. Once the READY pin goes high, the 8257 continues from state S_4 to complete the transfer. The 8257 can be interfaced as a memory mapped device or an I/O mapped device. If it is connected as memory mapped device, proper care must be taken while programming Rd/A_{15} and Wr/A_{14} bits in the terminal count register.

7.2.1 Priorities of the DMA Requests

The 8257 can be programmed to select any of the two priority schemes using the command register. The first is the *fixed priority scheme*, while the second is the *rotating priority scheme*. In the fixed priority scheme, each device connected to a channel is assigned a fixed priority. In this scheme, the $DREQ_3$ has the lowest priority followed by DRQ_2 and DRQ_1. DRQ_0 has the highest priority. In the rotating priority mode, the priorities assigned to the channels are not fixed. At any point of time, suppose DRQ_0 has highest priority and DRQ_3 the lowest, then after the device at channel 0 gets the service, its priority goes down and the channel 0 becomes the lowest priority channel. Channel1 now becomes the highest priority channel, and remains the highest priority channel till it gets the service. Once channel 1 is served, it becomes the lowest priority channel and the channel 2 now becomes the highest priority channel. If you select the rotating priority, in a single chip DMA system any device requesting the service is guaranteed to be recognized after no more than three higher priority requests, thus avoiding dominance of any one channel. The priority allotment in the rotating priority mode is as shown in Fig. 7.6.

Scanned by CamScanner

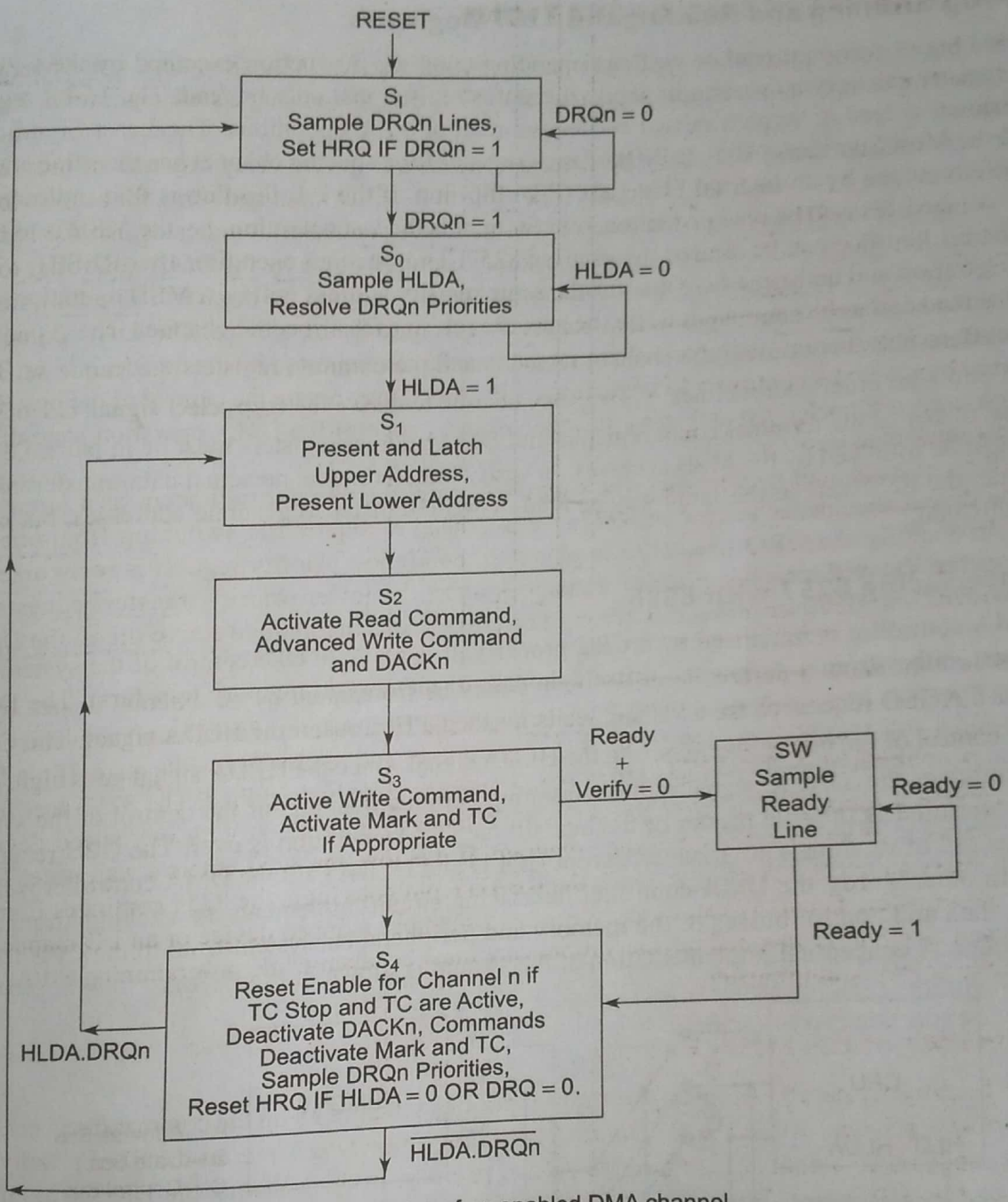


Fig. 7.5 *DMA Operation State Diagram*

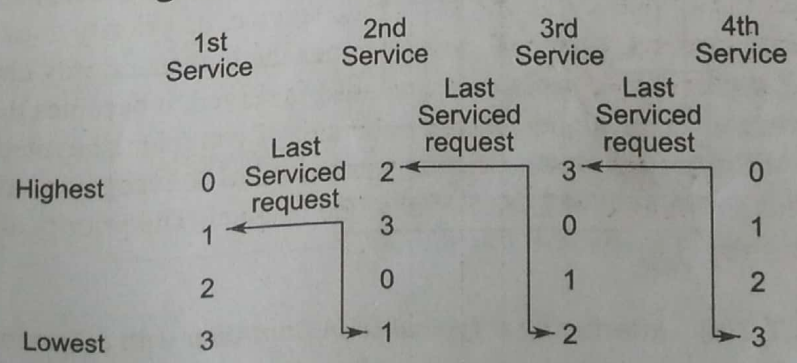


Fig. 7.6 *Priority Allotment in Rotating Priority Mode*

Scanned by CamScanner

7.2.2 Programming and Reading the 8257 Registers

The selected register may be read or written depending upon the instruction executed by the CPU but the mode set register can only be written in, while the status register can only be read. The 16-bit register pair of each channel is read or written in two successive read or write operations. The Least Significant Byte (LSB) and the Most Significant Byte (MSB) of each register for a specific channel has the same address, but they are differentiated by an internal First/Last (F/L) flip-flop. If the F/L flip-flop is 0, it indicates the first operation, i.e. the LSB is to be read or written, otherwise, it is the last operation, i.e. the MSB is to be read or written. The F/L flip-flop can be cleared by resetting 8257. Thus the first operation after RESET will always be a LSB operation and the successive one for the same register address will be a MSB operation. The least significant three address bits A_0–A_2 indicate the specific register for a specific channel. The A_3 address line is used to differentiate between all the channel registers and the common registers, i.e. mode set and status registers. The higher order address lines A_4–A_{15} may be used to derive the chip select signal $\overline{CS}$ of 8257. All the accesses to any of the terminal count registers and DMA address registers must be in pairs, i.e. the LSB accesses must be followed by the MSB accesses. In verify transfer mode, no actual data transfer takes place. In this mode, the 8257 acts in the same way as read or write transfer to generate addresses, but no control lines are activated.

7.2.3 Interfacing 8257 with 8086

Once a DMA controller is initialised by a CPU properly, it is ready to take control of the system bus on a DMA request, either from a peripheral or itself (in case of memory-to-memory transfers). The DMA controller sends a HOLD request to the CPU and waits for the CPU to assert the HLDA signal. The CPU relinquishes the control of the bus before asserting the HLDA signal. Once the HLDA signal goes high, the DMA controller activates the $\overline{DACK}$ signal to the requesting peripheral and gains the control of the system bus. The DMA controller is the sole master of the bus, till the DMA operation is over. The CPU remains in the HOLD status (all of its signals are tristated except HOLD and HLDA), till the DMA controller is the master of the bus. In other words, the DMA controller interfacing circuit implements a switching arrangement for the address, data and control busses of the memory and peripheral subsystem from/to the CPU to/from the DMA controller. A conceptual implementation of the system is shown in Fig. 7.7(a).

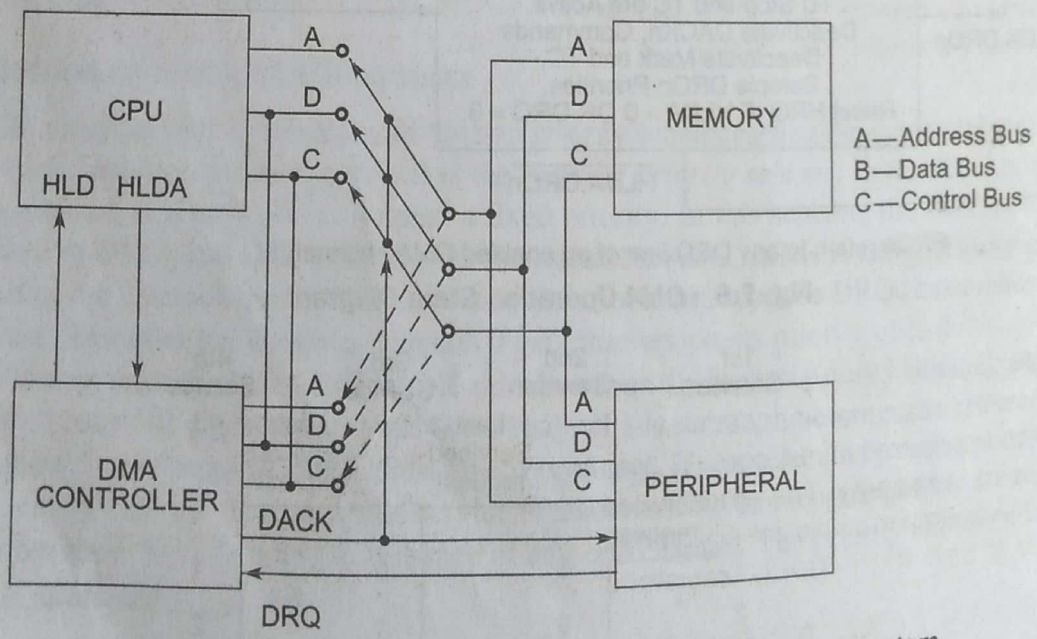


Fig. 7.7(a) *Interfacing a Typical DMA Controller with a System*

To explain the interfacing of 8257 with 8086 let us consider an interfacing example.

Scanned by CamScanner

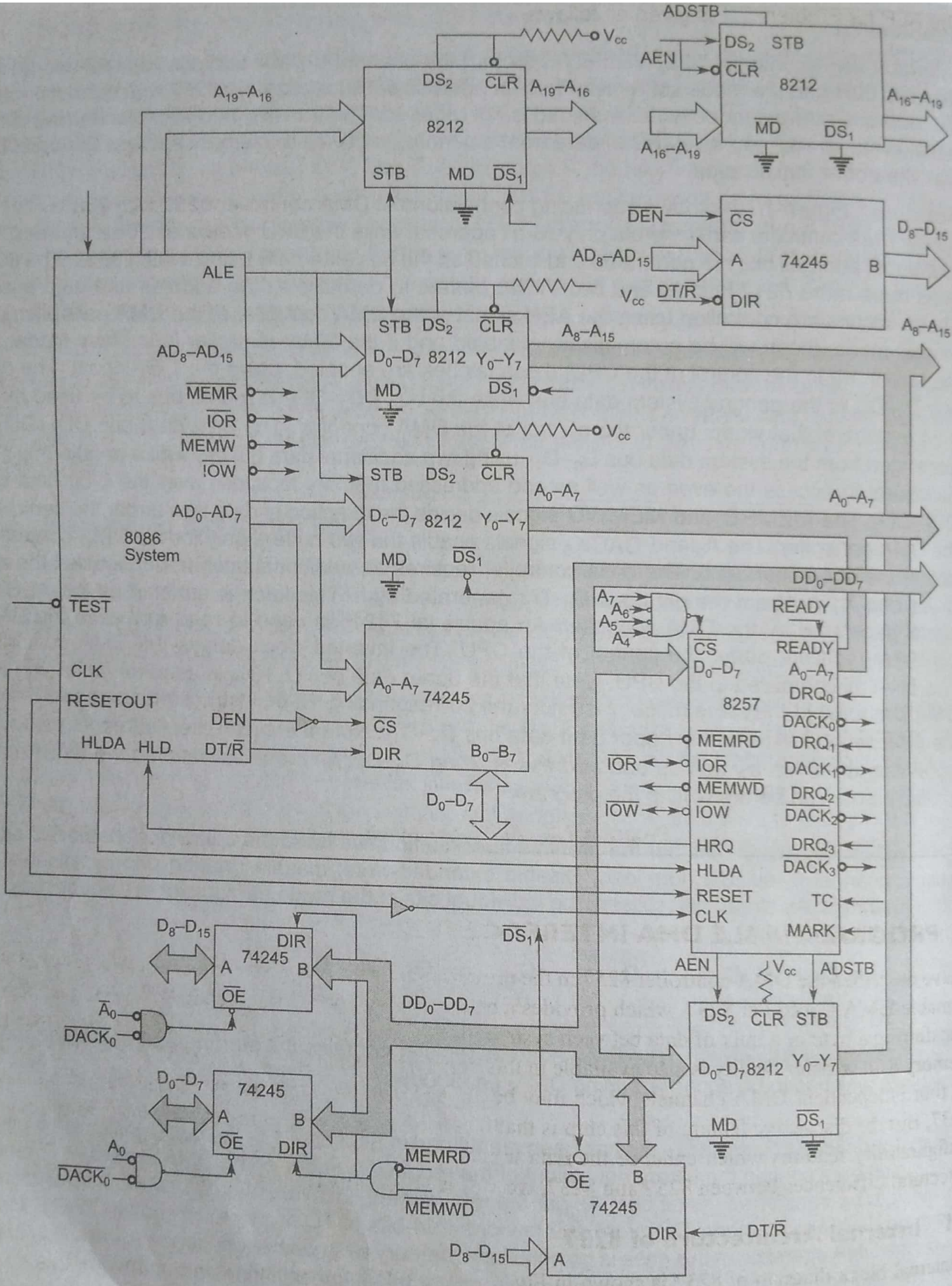


Fig. 7.7(b) *Interfacing 8257 with 8086*

Scanned by CamScanner

The ALP for Problem 7.1 is given as follows.

```
ASSUME      CS:CODE,DS:DATA
CODE        SEGMENT
START:      MOV AX, DATA            ; Initialize Data Segment
            MOV DS, AX              ;
            MOV AX, DMAL            ; Load DMA address register with
            OUT 80H, AX             ; lower byte of DMA address
            MOV AX, DMAH            ; Load higher byte of DMA address
            OUT 80H, AX             ; register of Channel 0
            MOV AX, TCL             ; Load lower byte TC register of
            OUT 81H, AX             ; channel 0
            MOV AX, TCH             ; Load higher byte of TC register
            OUT 81H, AX             ;
            MOV AX, MSR             ; MODE SET Register
                                    ; Initialization, The F/L flip-flop
            OUT 88H, AX             ; is assumed to be cleared
            MOV AH, 4CH             ; Latch segment address on A16-A19
            INT 21H                 ; externally, i.e. 0010(2H) and wait
                                      for the
                                    ; DMA request
                                    ; After the request is served, the
CODE ENDS                           ; CPU may continue the execution

DATA SEGMENT
MSR EQU 4141H                       ; Mode Set Register content
DMAL EQU 0000H                      ; DMA Address lower byte
DMAH EQU 5050H                      ; DMA Address higher byte
TCL EQU FFFFH                       ; Terminal count lower byte
TCH EQU 4747H                       ; Terminal count higher byte
DATA    ENDS
        END START
```

Program 7.1 *ALP for Problem 7.1*

Scanned by CamScanner