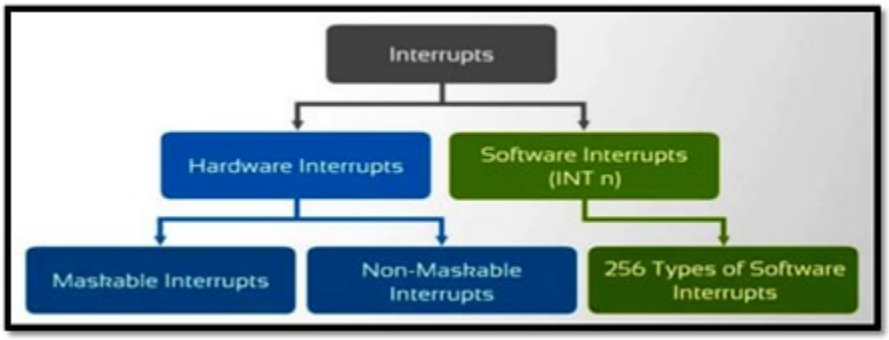
UNIT – III

8086 INTERRUPTS:

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



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

Interrupt can divide to five groups: 1. hardware interrupt

2. Non-maskable interrupts

3. Software interrupt

4. Internal interrupt

5. Reset

Hardware, software and internal interrupt are service on priority basis. Each interrupt is given a different priority level by assign it a type number. Type 0 identifies the highest-priority and type 255 identifies the lowest- priority interrupts. The 80x86 chips allow up to 256 vectored interrupts. This means that you can have up to 256 different sources for an interrupt and the 80x86 will directly call the service routine for that interrupt without any software processing. This is in contrast to no vectored interrupts that transfer control directly to a single interrupt service routine, regardless of the interrupt source.

The 80x86 provides a 256 entry interrupt vector table beginning at address 0:0 in memory. This is a 1K table containing 256 4-byte entries. Each entry in this table contains a segmented address that points at the interrupt service routine in memory. The lowest five types are dedicated to specific interrupts such as the divide by zero interrupt and the non maskable interrupt. The next 27 interrupt types, from 5 to 31 are High priority 3 reserved by Intel for use in future microprocessors. The upper 224 interrupt types, from32 to 255, are available to use for hardware and software interrupts.

When an interrupt occurs (shown in figure 1), regardless of source, the 80x86 does the following:

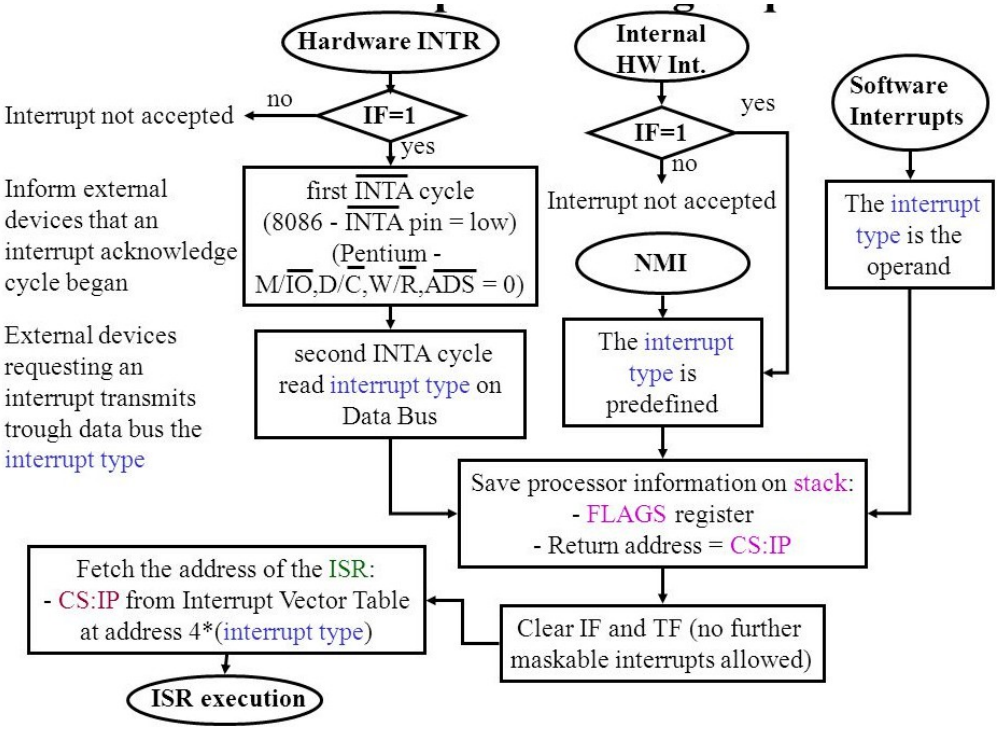
1. The CPU pushes the flags register onto the stack.

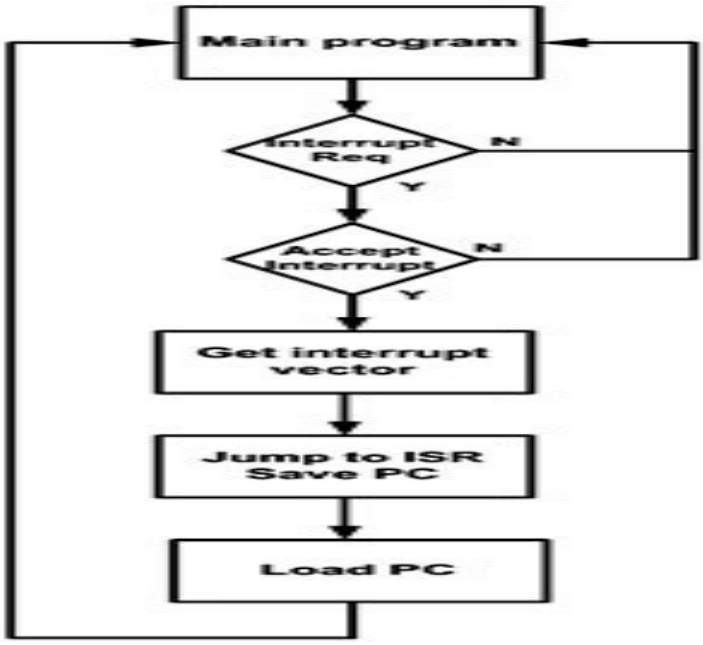
2. The CPU pushes a far return address (segment: offset) onto the stack, segment value first.

3. The CPU determines the cause of the interrupt (i.e., the interrupt number) and fetches the four byte interrupt vector from address 0: vector*4.

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

PROCESSING OF AN INTERRUPT:





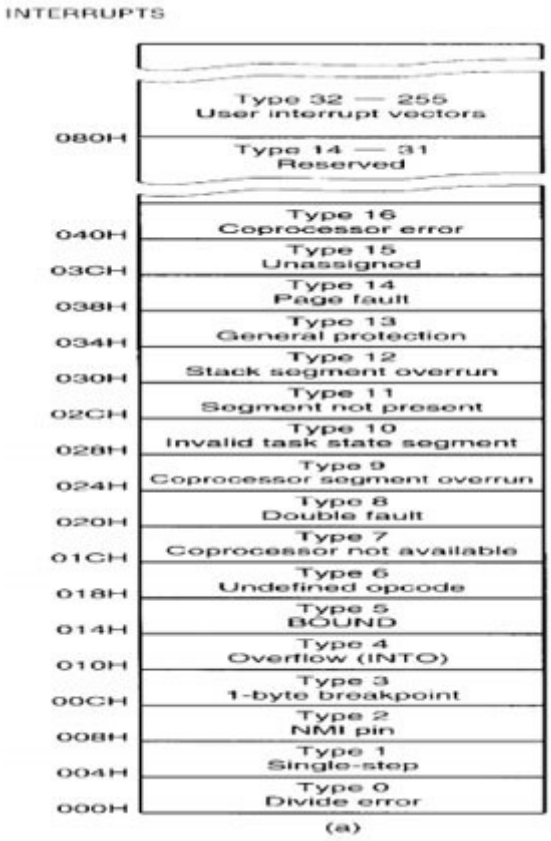
Hardware and Software Interrupts:

Hardware interrupts:

- The nonmaskable interrupt is generated by en external device, trough a rising edge on the NMI pin
- An external device, trough a high logic level on the INTR pin (the external device has to specify the interrupt number).

Software interrupts: Software interrupts (exceptions) using the INT instruction (followed by the interrupt number (type)).

Interrupt Vector Table:



An "**interrupt vector table"** (IVT) is a data structure that associates a list of interrupt handlers with a list of interrupt requests in a table of interrupt vectors. An entry in the interrupt vector is the address of the interrupt handler. While the concept is common across processor architectures, each IVT may be implemented in an architecture-specific fashion. For example, a dispatch table is one method of implementing an interrupt vector table.

The first 1Kbyte of memory of 8086 (00000 to003FF) is set aside as a table for storing the starting addresses of Interrupt Service Procedures (ISP). Since 4-bytes are required for storing starting addresses of ISPs, the table can hold 256 Interrupt procedures. The starting address of an ISP is often called the Interrupt Vector or Interrupt Pointer. Therefore the table is referred as Interrupt Vector Table. In this table, IP value is put in as low word of the vector & CS is put in high vector.

Dedicated interrupts of 8086:

The following are the various types of interrupts:
- Type 0 interrupts: This interrupt is also known as the divide by zero interrupt. For cases

where the quotient becomes particularly large to be placed / adjusted an error might occur.
- Type 1 interrupts: This is also known as the single step interrupt. This type of interrupt is primarily used for debugging purposes in assembly language.
- Type 2 interrupts: also known as the non-maskable NMI interrupts. These type of interrupts are used for emergency scenarios such as power failure.
- Type 3 interrupts: These type of interrupts are also known as breakpoint interrupts. When this interrupt occurs a program would execute up to its break point.

-Type 4 interrupts: Also known as overflow interrupts is generally existent after an arithmetic operation was performed.

Interrupt Priority Structure

Interrupt	Priority
Divide Error, INT(n),INTO	Highest
NMI	↓
INTR	↓
Single Step	Lowest

Bio

use
to i ers.
Trac uch
as b hat
boc nly
run her
mu: ling
BIO the
CPU ugh
son arly
stag

isk,
key d in
ROM on the main board, takes control immediately after the processor is reset, including during power-up or when a hardware reset button is pressed. The BIOS initializes the hardware, finds, loads and runs the boot program (usually, but not necessarily, an OS loader), and provides basic hardware control to the operating system running on the machine, which is usually an operating system but may be a directly booting single software application.

For IBM's part, they provided all the information needed to use their BIOS fully or to directly utilize the hardware and avoid BIOS completely, when programming the early IBM PC models (prior to the PS/2). From the beginning, programmers had the choice of using BIOS or not, on a per-hardware-peripheral basis. Today, the BIOS in a new PC still supports most, if not all, of the BIOS interrupt function calls defined by IBM for the IBM

AT (introduced in 1984), along with many more newer ones, plus extensions to some of the originals (e.g. expanded parameter ranges). This, combined with a similar degree of hardware compatibility, means that most programs written for an IBM AT can still run correctly on a new PC today, assuming that the faster speed of execution is acceptable (which it typically is for all but games that use CPU-based timing). Despite the considerable limitations of the services accessed through the BIOS interrupts, they have proven extremely useful and durable to technological change.

<table>
<tr><th>Interrupt vector</th><th>Description</th></tr>
<tr><td>05h</td><td>Executed when Shift-Print screen is pressed, as well as when the BOUND instruction detects a bound failure.</td></tr>
<tr><td>10h</td><td>Video Services
<table>
<tr><th>AH</th><th>Description</th></tr>
<tr><td>00h</td><td>Set Video Mode</td></tr>
<tr><td>01h</td><td>Set Cursor Shape</td></tr>
<tr><td>02h</td><td>Set Cursor Position</td></tr>
<tr><td>03h</td><td>Get Cursor Position And Shape</td></tr>
<tr><td>04h</td><td>Get Light Pen Position</td></tr>
<tr><td>05h</td><td>Set Display Page</td></tr>
<tr><td>06h</td><td>Clear/Scroll Screen Up</td></tr>
<tr><td>07h</td><td>Clear/Scroll Screen Down</td></tr>
<tr><td>08h</td><td>Read Character and Attribute at Cursor</td></tr>
<tr><td>09h</td><td>Write Character and Attribute at Cursor</td></tr>
</table></td></tr>
</table>

	<table><tr><td></td><td></td></tr><tr><td>0Ah</td><td>Write Character at Cursor</td></tr><tr><td>0Bh</td><td>Set Border Color</td></tr><tr><td>0Ch</td><td>Write Graphics Pixel</td></tr><tr><td>0Dh</td><td>Read Graphics Pixel</td></tr><tr><td>0Eh</td><td>Write Character in TTY Mode</td></tr><tr><td>0Fh</td><td>Get Video Mode</td></tr><tr><td>10h</td><td>Set Palette Registers (EGA, VGA, SVGA)</td></tr><tr><td>11h</td><td>Character Generator (EGA, VGA, SVGA)</td></tr><tr><td>12h</td><td>Alternate Select Functions (EGA, VGA, SVGA)</td></tr><tr><td>13h</td><td>Write String</td></tr><tr><td>1Ah</td><td>Get or Set Display Combination Code (VGA, SVGA)</td></tr><tr><td>1Bh</td><td>Get Functionality Information (VGA, SVGA)</td></tr><tr><td>1Ch</td><td>Save or Restore Video State (VGA, SVGA)</td></tr><tr><td>4Fh</td><td>VESA BIOS Extension Functions (SVGA)</td></tr></table>
11h	Returns equipment list
12h	Return conventional memory size

13h	Low Level Disk Services **AH** — **Description** 00h — Reset Disk Drives 01h — Check Drive Status 02h — Read Sectors 03h — Write Sectors 04h — Verify Sectors 05h — Format Track 08h — Get Drive Parameters 09h — Init Fixed Drive Parameters 0Ch — Seek To Specified Track 0Dh — Reset Fixed Disk Controller 15h — Get Drive Type 16h — Get Floppy Drive Media Change Status 17h — Set Disk Type 18h — Set Floppy Drive Media Type

<table>
<tr><td></td><td>
<table>
<tr><td>41h</td><td>Extended Disk Drive (EDD) Installation Check</td></tr>
<tr><td>42h</td><td>Extended Read Sectors</td></tr>
<tr><td>43h</td><td>Extended Write Sectors</td></tr>
<tr><td>44h</td><td>Extended Verify Sectors</td></tr>
<tr><td>45h</td><td>Lock/Unlock Drive</td></tr>
<tr><td>46h</td><td>Eject Media</td></tr>
<tr><td>47h</td><td>Extended Seek</td></tr>
<tr><td>48h</td><td>Extended Get Drive Parameters</td></tr>
<tr><td>49h</td><td>Extended Get Media Change Status</td></tr>
<tr><td>4Eh</td><td>Extended Set Hardware Configuration</td></tr>
</table>
</td></tr>
<tr><td>14h</td><td>Serial port services
<table>
<tr><th>AH</th><th>Description</th></tr>
<tr><td>00h</td><td>Serial Port Initialization</td></tr>
<tr><td>01h</td><td>Transmit Character</td></tr>
<tr><td>02h</td><td>Receive Character</td></tr>
<tr><td>03</td><td>Status</td></tr>
</table>
</td></tr>
</table>

	h	
15h	Miscellaneous system services

AH	AL	Description
00h		Turn on cassette drive motor
01h		Turn off cassette drive motor
02h		Read data blocks from cassette
03h		Write data blocks to cassette
4Fh		Keyboard Intercept
83h		Event Wait
84h		Read Joystick
85h		Sysreq Key Callout
86h		Wait
87h		Move Block
88h		Get Extended Memory Size
89h		Switch to Protected Mode
C0h		Get System Parameters
C1h		Get Extended BIOS Data Area Segment

	C2h		Pointing Device Functions
	C3h		Watchdog Timer Functions - PS/2 systems only
	C4h		Programmable Option Select - MCA bus PS/2 systems only
	D8h		EISA System Functions - EISA bus systems only
	E8h	01h	Get Extended Memory Size (Newer function, since 1994). Gives results for memory size above 64 Mb.
	E8h	20h	Query System Address Map. The information returned from E820 supersedes what is returned from the older AX=E801h and AH=88hinterfaces.

16h

Keyboard services

AH	Description
00h	Read Character
01h	Read Input Status
02h	Read Keyboard Shift Status
05h	Store Keystroke in Keyboard Buffer
10h	Read Character Extended
11	Read Input Status Extended

<table>
<tr><td></td><td><table>
<tr><td>h</td><td></td></tr>
<tr><td>12h</td><td>Read Keyboard Shift Status Extended</td></tr>
</table></td></tr>
<tr><td>17h</td><td>Printer services<table>
<tr><th>AH</th><th>Description</th></tr>
<tr><td>00h</td><td>Print Character to Printer</td></tr>
<tr><td>01h</td><td>Initialize Printer</td></tr>
<tr><td>02h</td><td>Check Printer Status</td></tr>
</table></td></tr>
<tr><td>18h</td><td>Execute Cassette BASIC: Very early true IBM computers contain Microsoft Cassette BASIC in the ROM, to be started by this routine in the event of a failure to boot from disk (called by the BIOS). On virtually all clones and later models in the PC line from IBM, which lack BASIC in ROM, this interrupt typically displays a message such as "No ROM BASIC" and halts.</td></tr>
<tr><td>19h</td><td>After POST this interrupt is used by BIOS to load the operating system. A program can call this interrupt to reboot the computer (but must ensure that hardware interrupts or DMA operations will not cause the system to hang or crash during either the reinitialization of the system by BIOS or the boot process).</td></tr>
<tr><td>1Ah</td><td>Real Time Clock Services<table>
<tr><th>AH</th><th>Description</th></tr>
<tr><td>00h</td><td>Read RTC</td></tr>
</table></td></tr>
</table>

<table>
<tr><td></td><td>
<table>
<tr><td>01h</td><td>Set RTC</td></tr>
<tr><td>02h</td><td>Read RTC Time</td></tr>
<tr><td>03h</td><td>Set RTC Time</td></tr>
<tr><td>04h</td><td>Read RTC Date</td></tr>
<tr><td>05h</td><td>Set RTC Date</td></tr>
<tr><td>06h</td><td>Set RTC Alarm</td></tr>
<tr><td>07h</td><td>Reset RTC Alarm</td></tr>
</table>
</td></tr>
<tr><td>1Ah</td><td>PCI Services - implemented by BIOSes supporting PCI 2.0 or later
<table>
<tr><th>AX</th><th>Description</th></tr>
<tr><td>B101h</td><td>PCI Installation Check</td></tr>
<tr><td>B102h</td><td>Find PCI Device</td></tr>
<tr><td>B103h</td><td>Find PCI Class Code</td></tr>
<tr><td>B106h</td><td>PCI Bus-Specific Operations</td></tr>
<tr><td>B108h</td><td>Read Configuration Byte</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td><table>
<tr><td>B109h</td><td>Read Configuration Word</td></tr>
<tr><td>B10Ah</td><td>Read Configuration Dword</td></tr>
<tr><td>B10Bh</td><td>Write Configuration Byte</td></tr>
<tr><td>B10Ch</td><td>Write Configuration Word</td></tr>
<tr><td>B10Dh</td><td>Write Configuration Dword</td></tr>
<tr><td>B10Eh</td><td>Get IRQ Routine Information</td></tr>
<tr><td>B10Fh</td><td>Set PCI IRQ</td></tr>
</table></td></tr>
<tr><td>1Bh</td><td>Ctrl-Break handler - called by INT 09 when Ctrl-Break has been pressed</td></tr>
<tr><td>1Ch</td><td>Timer tick handler - called by INT 08</td></tr>
<tr><td>1Dh</td><td>Not to be called; simply a pointer to the VPT (Video Parameter Table), which contains data on video modes</td></tr>
<tr><td>1Eh</td><td>Not to be called; simply a pointer to the DPT (Diskette Parameter Table), containing a variety of information concerning the diskette drives</td></tr>
<tr><td>1Fh</td><td>Not to be called; simply a pointer to the VGCT (Video Graphics Character Table), which contains the data for ASCII characters 80h to FFh</td></tr>
<tr><td>41h</td><td>Address pointer: FDPT = Fixed Disk Parameter Table (1st hard drive)</td></tr>
<tr><td>46h</td><td>Address pointer: FDPT = Fixed Disk Parameter Table (2nd hard drive)</td></tr>
<tr><td>4Ah</td><td>Called by RTC for alarm</td></tr>
</table>

MEMORY AND I/O INTERFACING:

I/O Interface

Introduction:

Any application of a microprocessor based system requires the transfer of data between external circuitry to the microprocessor and microprocessor to the external circuitry. User can give information to the microprocessor based system using keyboard and user can see the result or output information from the microprocessor based system with the help of display device. The transfer of data between keyboard and microprocessor, and microprocessor and display device is called input/output data transfer or I/O data transfer. This data transfer is done with the help of I/O ports.

Input port:

It is used to read data from the input device such as keyboard. The simplest form of input port is a buffer. The input device is connected to the microprocessor through buffer, as shown in the fig.1. This buffer is a tri-state buffer and its output is available only when enable signal is active. When microprocessor wants to read data from the input device (keyboard), the control signals from the microprocessor activates the buffer by asserting enable input of the buffer. Once the buffer is enabled, data from the input device is available on the data bus. Microprocessor reads this data by initiating read command.

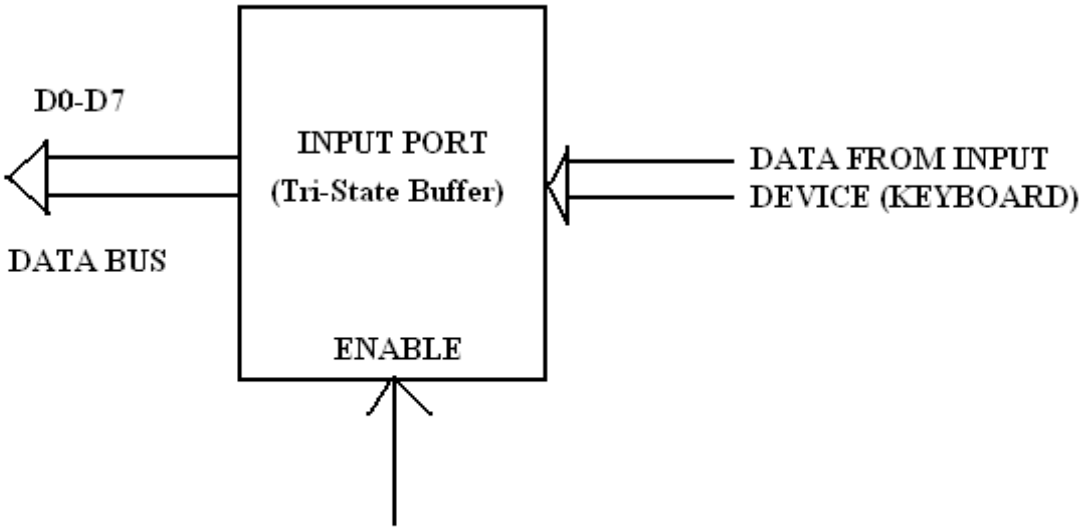


FIG.1 INPUT PORT

It is used to read data from the input device such as keyboard. The simplest form of input port is a buffer. The input device is connected to the microprocessor through buffer, as shown in the fig.1. This buffer is a tri-state buffer and its output is available only when enable signal is active. When microprocessor wants to read data from the input device (keyboard), the control signals from the microprocessor activates the buffer by asserting enable input of the buffer. Once the buffer is enabled, data from the input device is available on the data bus. Microprocessor reads this data by initiating read command.

Output port:

It is used to send data to the output device such as display from the microprocessor. The simplest form of output port is a latch. The output device is connected to the microprocessor through latch, as shown in the fig.2. When microprocessor wants to send data to the output device is puts the data on the data bus and activates the clock signal of

the latch, latching the data from the data bus at the output of latch. It is then available at the output of latch for the output device.

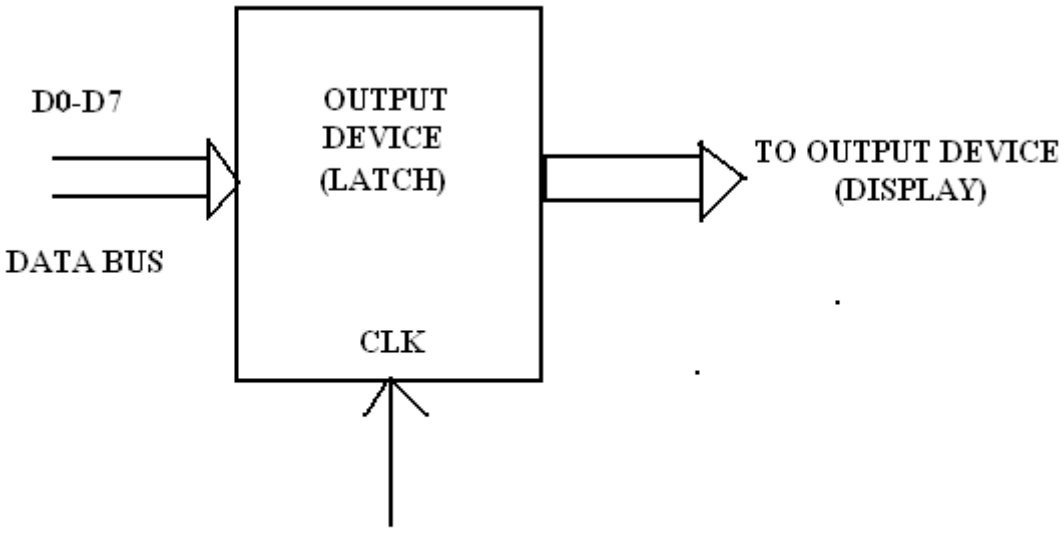


FIG.2 OUTPUT PORT

There are three different ways that the data transfer can take place. They are

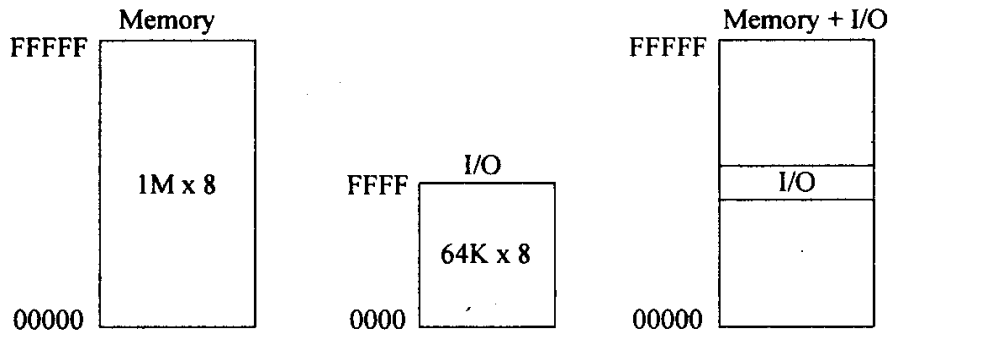
(1) Program controlled I/O
(2) Interrupt Program Controlled I/O
(3) Hardware controlled I/O

In program controlled I/O data transfer scheme the transfer of data is completely under the control of the microprocessor program. In this case an I/O operation takes place only when an I/O transfer instruction is executed. In an interrupt program controlled I/O an external device indicates directly to the microprocessor its readiness to transfer data by a signal at an interrupt input of the microprocessor. When microprocessor receives this signal the control is transferred to ISS (Interrupt service subroutine) which performs the data transfer. Hardware controlled I/O is also known as direct memory access DMA. In this case the data transfer takes place directly between an I/O device and memory but not through microprocessors. Microprocessor only initializes the process of data transfer by indicating the starting address and the number of words to be transferred. The instruction .set of any microprocessor contains instructions that transfer information to an I/O device and to read information from an I/O device.

In 8086 we have IN, OUT instructions for this purpose. OUT instruction transfers information to an I/O device whereas IN instruction is used to read information from an I/O device. Both the instructions perform the data transfer using accumulator AL or AX. The I/O address is stored in register DX. The port number is specified along with IN or OUT instruction. The external I/O interface decodes to find the address of the I/O device. The 8 bit fixed port number appears on address bus A0 - A7 with A8 - A15 all zeros. The address connections above A15 are undefined for an I/O instruction. The 16 bit variable port number appears on address connections A0 - A15. The above notation indicates that first 256 I/O port addresses 00 to FF are accessed by both the fixed and variable I/O instructions. The I/O addresses from 0000 to FFFF are accessed by the variable I/O address.

I/O devices can be interfaced to the microprocessors using two methods. They are I/O mapped I/O and memory mapped I/O. The I/O mapped I/O is also known as isolated I/O

or direct I/O. In I/O mapped I/O the IN and OUT instructions transfer data between the accumulator or memory and I/O device. In memory mapped I/O the instruction that refers memory can perform the data transfer.



(a) Isoloated I/O map (b) Memory map I/O

Memory and I/O map of 8086

I/O mapped I/O is the most commonly used I/O transfer technique. In this method I/O locations are placed separately from memory. The addresses for isolated I/O devices are separate from memory. Using this method user can use the entire memory. This method allows data transfer only by using instructions IN, OUT. The pins M/ IO and W/R are used to indicate I/O read or an I/O write operations. The signals on these lines indicate that the address on the address bus is for I/O devices. Memory mapped I/O does not use the IN, OUT instruction it uses only the instruction that transfers data between microprocessor and memory.

A memory mapped I/O device is treated as memory location. The disadvantage in this system is the overall memory is reduced. The advantage of this system is that any memory transfer instruction can be used for data transfer and control signals like I/O read and I/O write are not necessary which simplify the hardware.

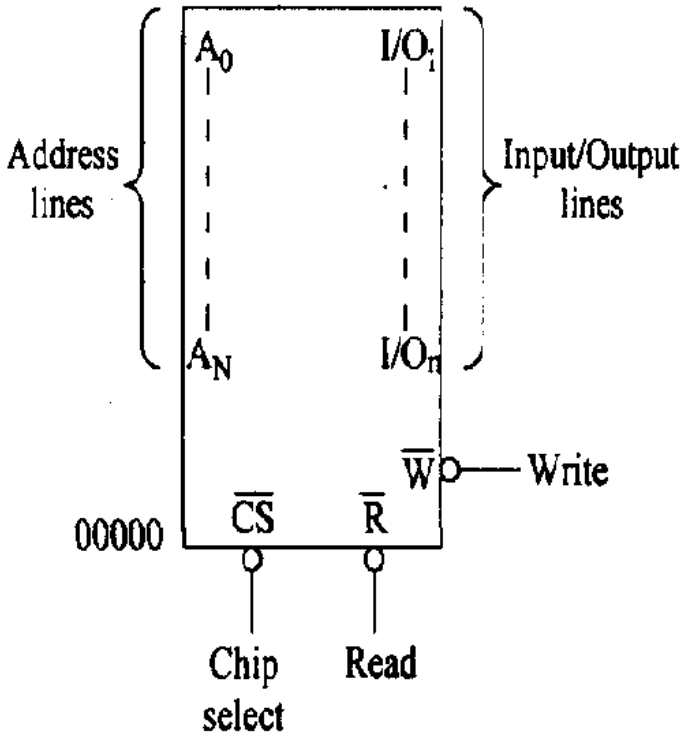
Memory interfacing

Memory is an integral part of a microcomputer system. There are two main types of memory.

(i) **Read only memory (ROM):** As the name indicates this memory is available only for reading purpose. The various types available under this category are PROM, EPROM, EEPROM which contain system software and permanent system data.

(ii) **(ii) Random Access memory (RAM):** This is also known as Read Write Memory. It is a volatile memory. RAM contains temporary data and software programs generally for different applications.

While executing particular task it is necessary to access memory to get instruction codes and data stored in memory. Microprocessor initiates the necessary signals when read or write operation is to be performed. Memory device also requires some signals to perform read and write operations using various registers. To do the above job it is necessary to have a device and a circuit, which performs this task is known as interfacing device and as this is involved with memory it-is known as memory interfacing device. The basic concepts of memory interfacing involve three different tasks. The microprocessor should be able to read from or write into the specified register. To do this it must be able to select the required chip, identify the required register and it must enable the appropriate buffers.



Simple memory device

Any memory device must contain address lines and Input, output lines, selection input, control input to perform read or write operation. All memory devices have address inputs that select memory location within the memory device. These lines are labeled as AO ..... AN. The number of address lines indicates the total memory capacity of the memory device. A 1K memory requires 10 address lines A0-A9. Similarly a 1MB requires 20 lines A0-A19 (in the case of 8086). The memory devices may have separate I/O lines or a common set of bidirectional I/O lines.

Using these lines data can be transferred in either direction. Whenever output buffer is activated the operation is read whenever input buffers are activated the operation is write. These lines are labeled as I/O,..... I/On or DO .....Dn. The size of a memory location is dependent upon the number of data bits. If the number of data lines is eight D0 - D7 then 8 bits or 1 byte of data can be stored in
each location. Similarly if numbers of data bits are 16 (D0 - D15) then the memory size is 2 bytes. For
Example 2K x 8 indicates there are 2048 memory locations and each memory location can store 8 bits
of data.

Memory devices may contain one or more inputs which are used to select the memory device or to enable the memory device. This pin is denoted by CS (Chip select) or CE (Chip enable). When this pin is at logic '0' then only the memory device performs a read or a write operation. If this pin is at logic '1' the memory chip is disabled. If there are more than one CS input then all these pins must be activated to perform read or write operation.

All memory devices will have one or more control inputs. When ROM is used we find OE output enable pin which allows data to flow out of the output data pins. To perform this

task both CS and OE must be active. A RAM contains one or two control inputs. They are R /W or RD and WR. If there is only one input R/W then it performs read operation when R/W pin is at logic 1. If it is at logic 0 it performs write operation. Note that this is possible only when CS is also active.

Memory Interface using RAMS, EPROMS and EEPROMS:

Semiconductor Memory Interfacing:

Semiconductor memories are of two types, viz. RAM (Random Access Memory) and ROM (Read Only Memory).

Static RAM Interfacing:

The semiconductor RAMs are of broadly two types-static RAM and dynamic RAM. The semiconductor memories are organised as two dimensional arrays of memory locations. For example, 4K x 8 or 4K byte memory contains 4096 locations, where each location contains 8-bit data and only one of the 4096 locations can be selected at a time. Obviously, for addressing 4K bytes of memory, twelve address lines are required. In general, to address a memory location out of N memory locations , we will require at least *n* bits of address, i.e. *n* address lines where *n* = Log2 N. Thus if the microprocessor has *n* address lines, then it is able to address at the most N locations of memory, where *2n* = N. However, if out of *N* locations only *P* memory locations are to be interfaced, then the least significant *p* address lines out of the available *n* lines can be directly connected from the microprocessor to the memory chip while the remaining *(n-p)* higher order address lines may be used for address decoding (as inputs to the chip selection logic). The memory address depends upon the hardware circuit used for decoding the chip select (CS).

The output of the decoding circuit is connected with the CS pin of the memory chip. The general procedure of static memory interfacing with 8086 is briefly described as follows:

1. Arrange the available memory chips so as to obtain 16-bit data bus width. The upper 8-bit bank is called ‘odd address memory bank’ and the lower 8-bit bank is called ‘even address memory bank’.

2. Connect available memory address lines of memory chips with those of the microprocessor and also connect the memory RD and WR inputs to the corresponding processor control signals. Connect the 16-bit data bus of the memory bank with that of the microprocessor 8086.

3. The remaining address lines of the microprocessor, BHE and A0 are used for decoding the required chip select signals for the odd and even memory banks. CS of memory is derived from the O/P of the decoding circuit.

As a good and efficient interfacing practice, the address map of the system should be continuous as far as possible, i.e. there should be no windows in the map. A memory location should have a single address corresponding to it, i.e. absolute decoding should be preferred, and minimum hardware should be used for decoding. In a number of cases, linear decoding may be used to minimize the required hardware. Let us now consider a few example problems on memory interfacing with 8086.

Problem 5.1

Interface two 4K × 8 EPROMS and two 4K × 8 RAM chips with 8086. Select suitable maps.

Solution We know that, after reset, the IP and CS are initialised to form address FFFF0H. Hence, this address must lie in the EPROM. The address of RAM may be selected any where in the 1MB address space of 8086, but we will select the RAM address such that the address map of the system is continuous, as shown in Table 5.1.

Table 5.1 *Memory Map for Problem 5.1*

Address	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
						EPROM						8K × 8								
FE000H	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
FDFFFH	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
						RAM						8K × 8								
FC000H	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Total 8K bytes of EPROM need 13 address lines $A_0 - A_{12}$ (since $2^{13} = 8K$). Address lines $A_{13} - A_{19}$ are used for decoding to generate the chip select. The $\overline{BHE}$ signal goes low when a transfer is at odd address or higher byte of data is to be accessed. Let us assume that the latched address, $\overline{BHE}$ and demultiplexed data lines are readily available for interfacing. Figure 5.1 shows the interfacing diagram for the memory system.

The memory system in this example contains in total four 4K × 8 memory chips.

The two 4K × 8 chips of RAM and ROM are arranged in parallel to obtain 16-bit data bus width. If A_0 is 0, i.e. the address is even and is in RAM, then the lower RAM chip is selected indicating 8-bit transfer at an even address. If A_0 is 1, i.e. the address is odd and is in RAM, the $\overline{BHE}$ goes low, the upper RAM chip is selected, further indicating that the 8-bit transfer is at an odd address. If the selected addresses are in ROM, the respective ROM chips are selected. If at a time A_0 and $\overline{BHE}$ both are 0, both the RAM or ROM chips are selected, i.e. the data transfer is of 16 bits. The selection of chips here takes place as shown in Table 5.2.

Table 5.2 *Memory Chip Selection for Problem 5.1*

Decoder I/P → *Address/$\overline{BHE}$ →*	A_2 A_{13}	A_1 A_0	A_0 $\overline{BHE}$	*Selection/ Comment*
Word transfer on $D_0 - D_{15}$	0	0	0	Even and odd addresses in RAM
Byte transfer on $D_7 - D_0$	0	0	1	Only even address in RAM
Byte transfer on $D_8 - D_{15}$	0	1	0	Only odd address in RAM
Word transfer on $D_0 - D_{15}$	1	0	0	Even and odd addresses in ROM
Byte transfer on $D_0 - D_7$	1	0	1	Only even address in ROM
Byte transfer on $D_8 - D_{15}$	1	1	0	Only odd address in ROM

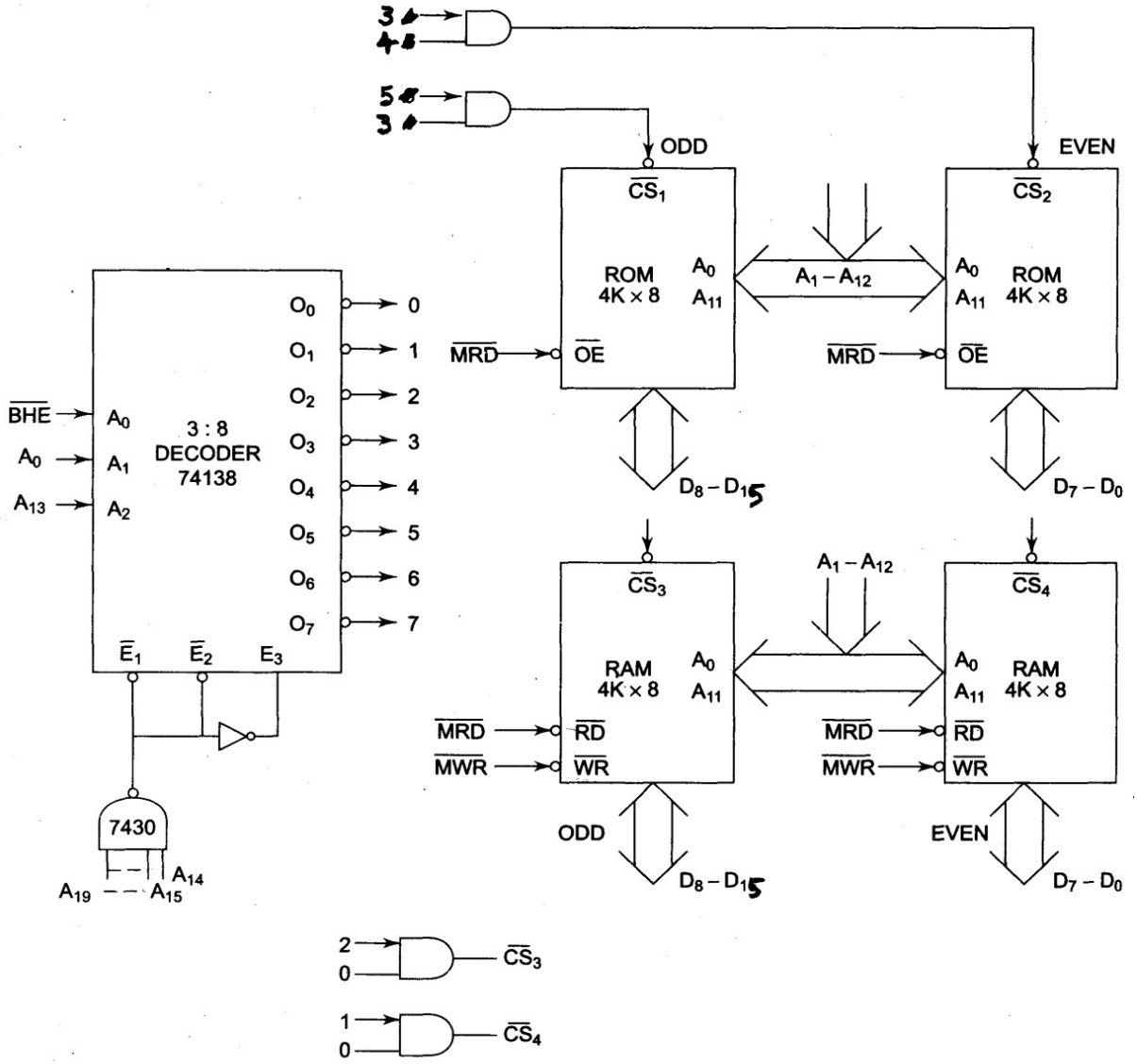


Fig. 5.1 *Interfacing Problem 5.1*

Problem 5.2

Design an interface between 8086 CPU and two chips of 16K × 8 EPROM and two chips of 32K × 8 RAM. Select the starting address of EPROM suitably. The RAM address must start at 00000H.

Solution: The last address in the map of 8086 is FFFFFH. After resetting, the processor starts from FFFF0H. Hence this address must lie in the address range of EPROM. Figure 5.2 shows the interfacing diagram, and Table 5.3 shows complete map of the system.

Table 5.3 *Address Map for Problem 5.2*

Addresses	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_{09}	A_{08}	A_{07}	A_{06}	A_{05}	A_{04}	A_{03}	A_{02}	A_{01}	A_{00}
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
						32KB EPROM														
F8000H	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0FFFFH	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
						64KB RAM														
00000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

It is better not to use a decoder to implement the above map because it is not continuous, i.e. there is some unused address space between the last RAM address (0FFFFH) and the first EPROM address F8000H). Hence the logic is implemented using logic gates, as shown in Fig. 5.2.

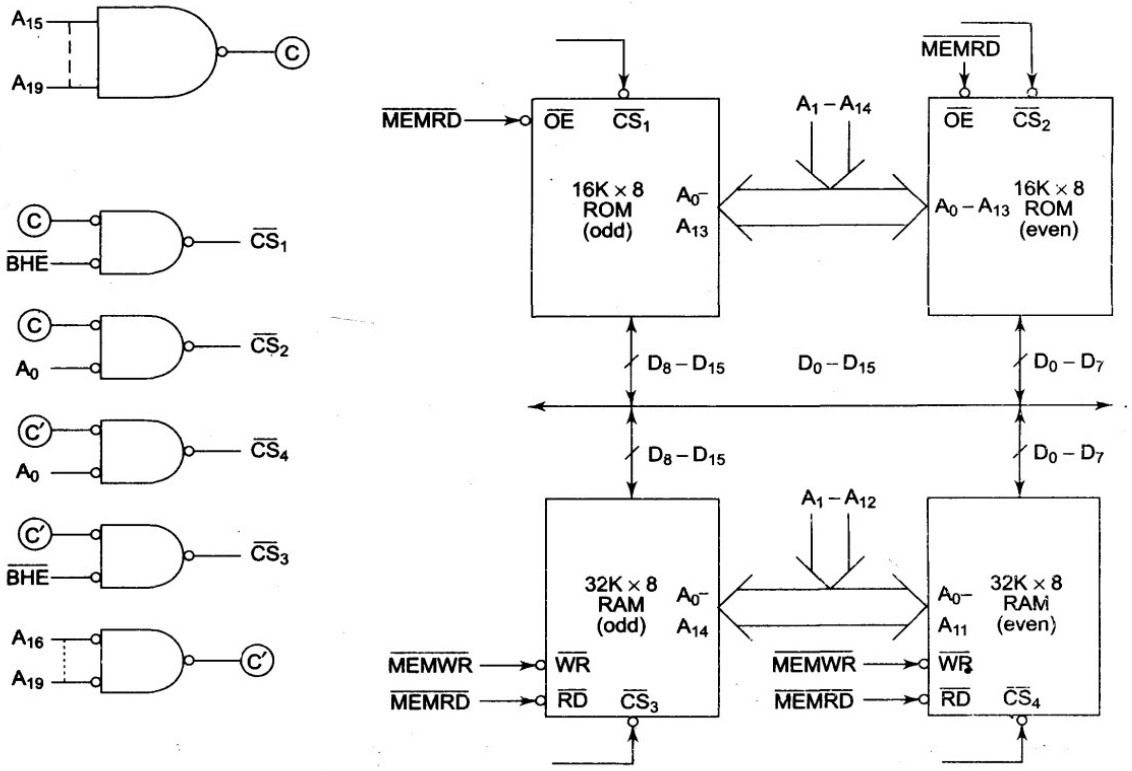


Fig. 5.2 *Interfacing Problem 5.2*

Problem 5.3

It is required to interface two chips of 32K × 8 ROM and four chips of 32K × 8 RAM with 8086, according to the following map.

```
ROM 1 and 2 F0000H - FFFFFH, RAM 1 and 2 D0000H - DFFFFH
RAM 3 and 4 E0000H - EFFFFH
```

Show the implementation of this memory system.

Solution Let us write the memory map of the system as shown in Table 5.6.

The implementation of the above map is shown in Fig. 5.3 using the same technique as in Problem 5.1 and Problem 5.2. All the address, data and control signals are assumed to be readily available.

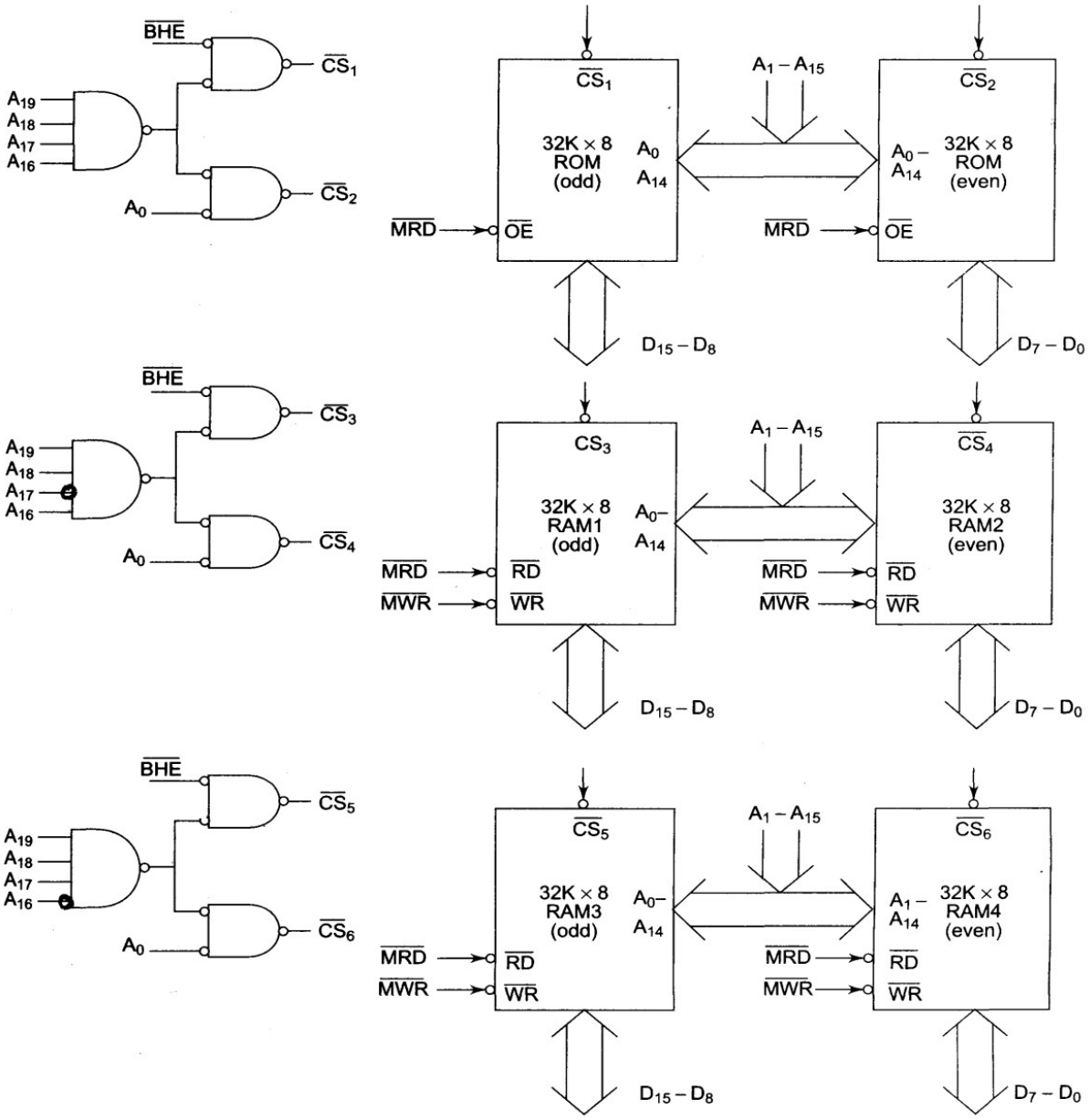


Fig. 5.3 *Interfacing Problem 5.3*

UNIT-IV

8255 PPI: