Macros:

- A **macro** is a group of repetitive instructions in a program which are codified only once and can be used as many times as necessary.
- A macro can be defined anywhere in program using the directives **MACRO** and **ENDM**
- Each time we call the macro in a program, the assembler will insert the defined group of instructions in place of the call.
- The assembler generates machine codes for the group of instructions each time the macro is called.
- Using a macro avoids the overhead time involved in calling and returning from a procedure.

Syntax of macro:

```
macroname MACRO

instruction1

instruction2

.

.

ENDM
```

- **Example:**

```
Read MACRO
 mov ah,01h
int 21h
ENDM
```

```
Display MACRO
mov dl,al
 Mov ah,02h
int 21h
 ENDM
```

ALP for Finding Factorial of number using procedures

```
CODE SEGMENT
ASSUME CS:CODE
    FACT MACRO
     MOV BX,AX
     DEC BX
    BACK: MUL BX
     DEC BX
     JNZ BACK
    ENDM
START: MOV AX,7
     FACT
     MOV AH,4CH
     INT 21H
     CODE ENDS
     END START
```

Advantage of Procedure and Macros:

Procedures:

Advantages

- The machine codes for the group of instructions in the procedure only have to be put once.

Disadvantages

- Need for stack
- Overhead time required to call the procedure and return to the calling program.

Macros:

Advantages

- Macro avoids overhead time involving in calling and returning from a procedure.

Disadvantages

- Generating in line code each time a macro is called is that this will make the program take up more memory than using a procedure.

Differences between Procedures and Macros:

PROCEDURES	MACROS
Accessed by CALL and RET mechanism during program execution	Accessed by name given to macro when defined during assembly
Machine code for instructions only put in memory once	Machine code generated for instructions each time called
Parameters are passed in registers, memory locations or stack	Parameters passed as part of statement which calls macro
Procedures uses stack	Macro does not utilize stack
A procedure can be defined anywhere in program using the directives PROC and ENDP	A macro can be defined anywhere in program using the directives MACRO and ENDM
Procedures takes huge memory for CALL (3 bytes each time CALL is used) instruction	Length of code is very huge if macro's are called for more number of times

8086 MEMORY INTERFACING:

- Most the memory ICs are byte oriented i.e., each memory location can store only one byte of data.
- The 8086 is a 16-bit microprocessor, it can transfer 16-bit data.
- So in addition to byte, word (16-bit) has to be stored in the memory.
- To implement this , the entire memory is divided into two memory banks: Bank0 and Bank1.
- Bank0 is selected only when A0 is zero and Bank1 is selected only when BHE' is zero.
- A0 is zero for all even addresses, so Bank0 is usually referred as even addressed memory bank.
- BHE' is used to access higher order memory bank, referred to as odd addressed memory bank.

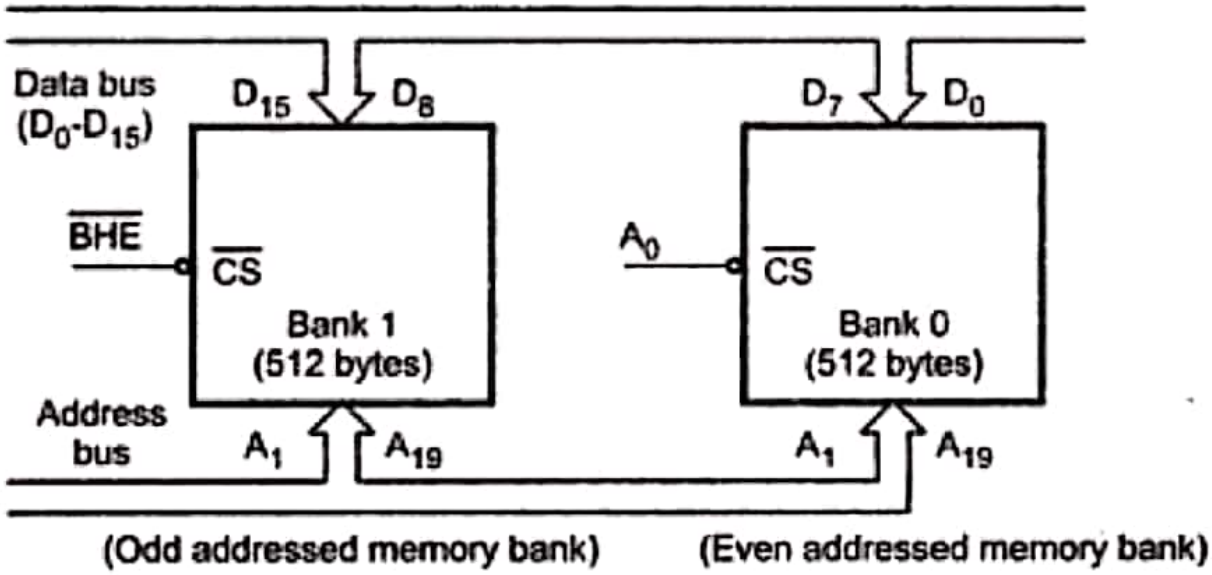


Fig. 5.2 Memory Interfacing

- Every microprocessor based system has a memory system.
- Almost all systems contain two basic types of memory, read only memory (ROM) and random access memory (RAM) or read/write memory.
- ROM contains system software and permanent system data such as lookup tables, IVT..etc.
- RAM contains temporary data and application software.
- ROMs/PROMs/EPROMs are mapped to cover the CPU's reset address, since these are non-volatile.
- When the 8086 is reset, the next instruction is fetched from the memory location FFFF0H.
- So in the 8086 system the location FFFF0H must be in ROM location.

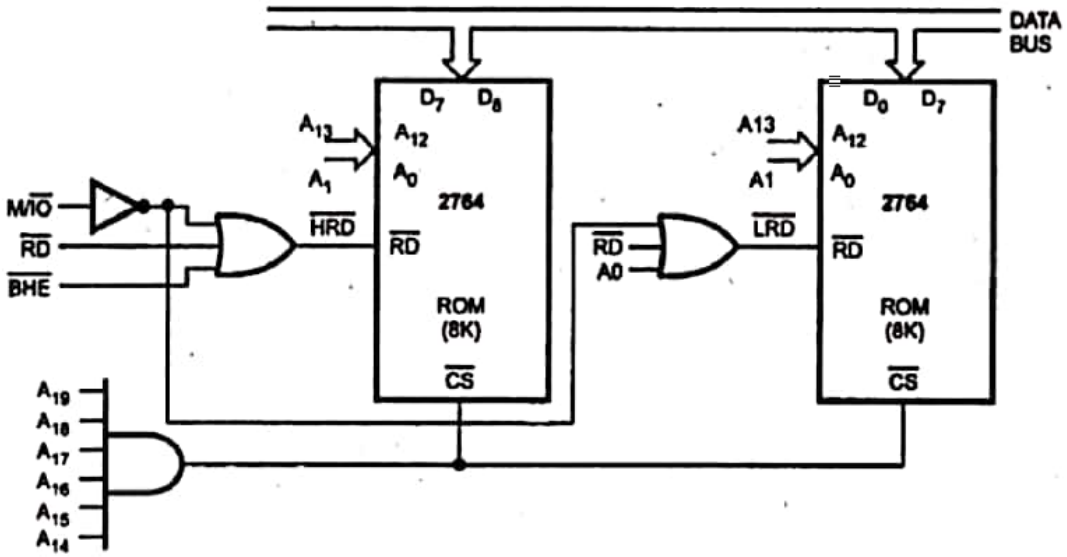
Address Decoding Techniques

1. Absolute decoding
2. Linear decoding
3. Block decoding

1. Absolute Decoding:

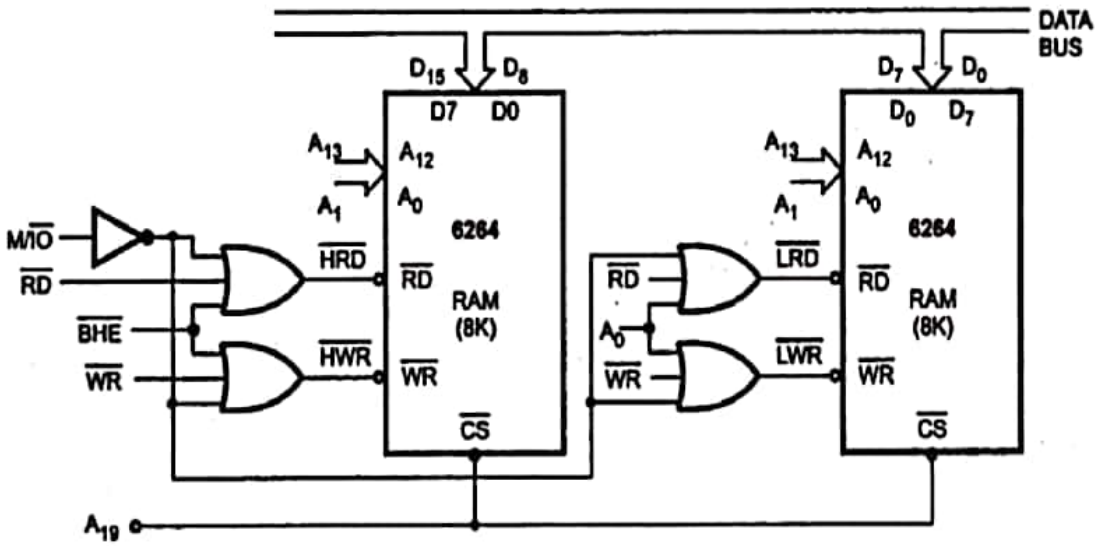
- In the absolute decoding technique the memory chip is selected only for the specified logic level on the address lines: no other logic levels can select the chip.
- Below figure the memory interface with absolute decoding. Two 8K EPROMs (2764) are used to provide even and odd memory banks.
- Control signals BHE and A0 are use to enable output of odd and even memory banks respectively. As each memory chip has 8K memory locations, thirteen address lines are required to address each locations, independently.

- All remaining address lines are used to generate an unique chip select signal. This address technique is normally used in large memory systems.



Linear Decoding:

In small system hardware for the decoding logic can be eliminated by using only required number of addressing lines (not all). Other lines are simple ignored. This technique is referred as linear decoding or partial decoding. Control signals BHE and Ao are used to enable odd and even memory banks, respectively. Figure shows the addressing of 16K RAM (6264) with linear decoding.

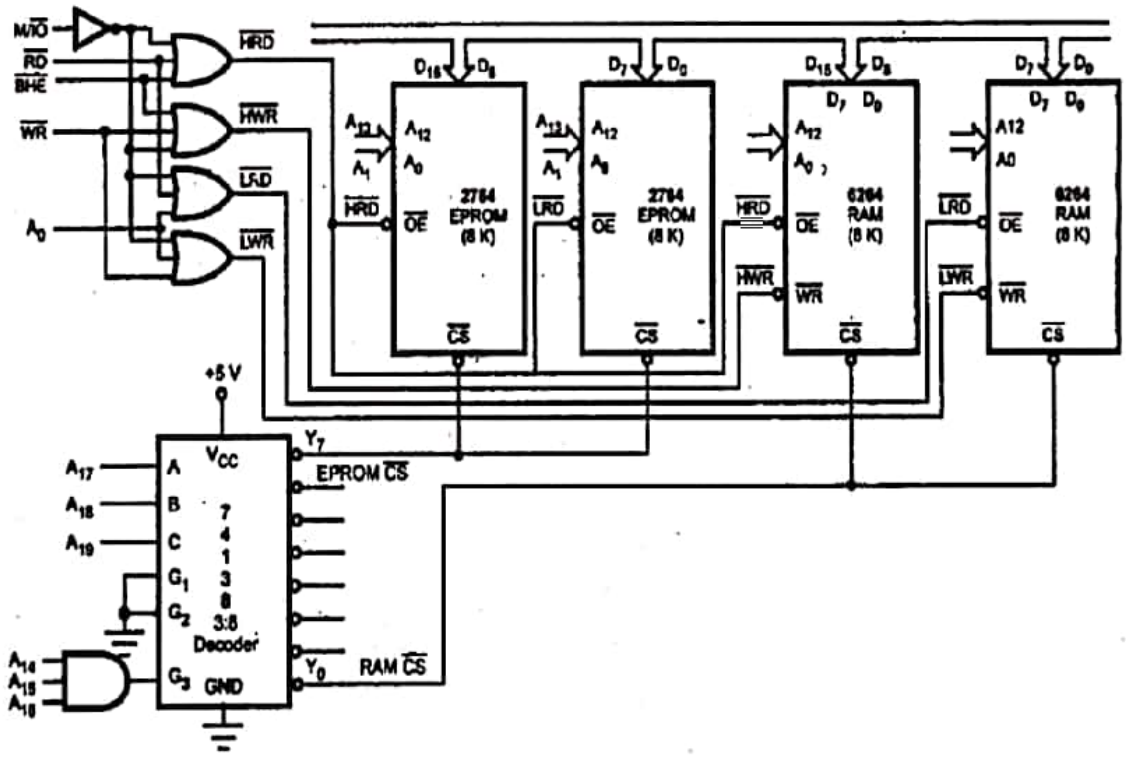


The address line A19 is used to select the RAM chips. When A19 is low, chip is selected, otherwise it is disabled. The status of A14 to A18 does not affect the chip selection logic. This gives you multiple addresses (shadow addresses). This technique reduces the cost of decoding circuit, but it gas drawback of multiple addresses.

Block Decoding:

In a microcomputer system the memory array is often consists of several blocks of memory chips. Each block of memory requires decoding circuit. To avoid separate decoding for each memory block special decoder IC is used to generate chip select signal for each block.

Figure shows the Block decoding technique using 74138, 3:8 decoder



Interfacing RAM, ROM, EPROM to 8086:

- The general procedure of static memory interfacing with 8086

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'.
    - 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 RD and WR inputs to the corresponding processor control signals.
3. Connect the 16-bit data bus of memory bank with that of the microprocessor 8086.
4. The remaining address lines of the microprocessor, BHE and A_0 are used for decoding the required chip select signals for the odd and even memory banks. The CS of memory is derived from the output of the decoding circuit.

Problem 1:

Interface two 4Kx8 EPROM and two 4Kx8 RAM chips with 8086. Select suitable maps.

Solution:

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

Memory Map Table

Address	A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	A00
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EPROM								8K X 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 X 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 A0-A12 (since z13 = 8K).

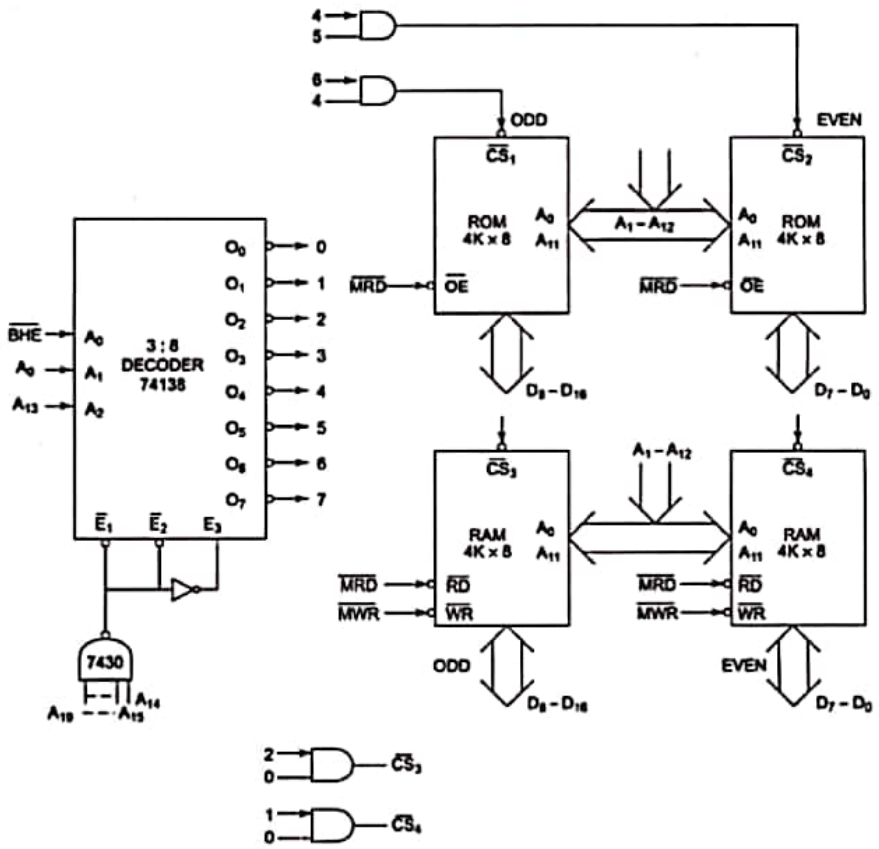
Address lines A13 - A19 are used for decoding to generate the chip select.

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.

The memory system in this problem contains in total four 4K x 8 memory chips.

The two 4K x 8 chips of RAM and ROM are arranged in parallel to obtain 16-bit data bus width. If A0 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 A0 is 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 A0 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 below.



Memory Chip Selection Table:

Decoder I/P --> Address/$\overline{BHE}$ -->	A2 A13	A1 A0	A0 $\overline{BHE}$	Selection/ Comment
Word transfer on D0 - D15	0	0	0	Even and odd address in RAM
Byte transfer on D7 - D0	0	0	1	Only even address in RAM
Byte transfer on D8 - D15	0	1	0	Only odd address in RAM
Word transfer on D0 - D15	1	0	0	Even and odd address in RAM
Byte transfer on D7 - D0	1	0	1	Only even address in RAM
Byte transfer on D8 - D15	1	1	0	Only odd address in ROM

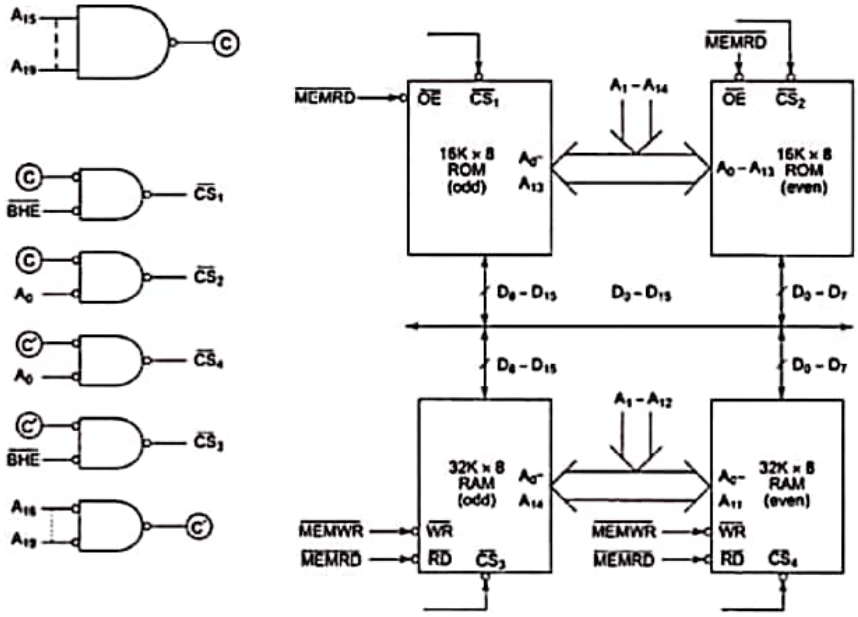
Problem2: Design an interface between 8086 CPU and two chips of 16K×8 EPROM and two chips of 32K×8 RAM. Select the starting address of EPROM suitably. The RAM address must start at 00000 H.

Solution: The last address in the map of 8086 is FFFFF H. after resetting, the processor starts from FFFF0 H. hence this address must lie in the address range of EPROM.

Address Map for Problem

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 (0FFFF H) and the first EPROM address (F8000 H). Hence the logic is implemented using logic gates.



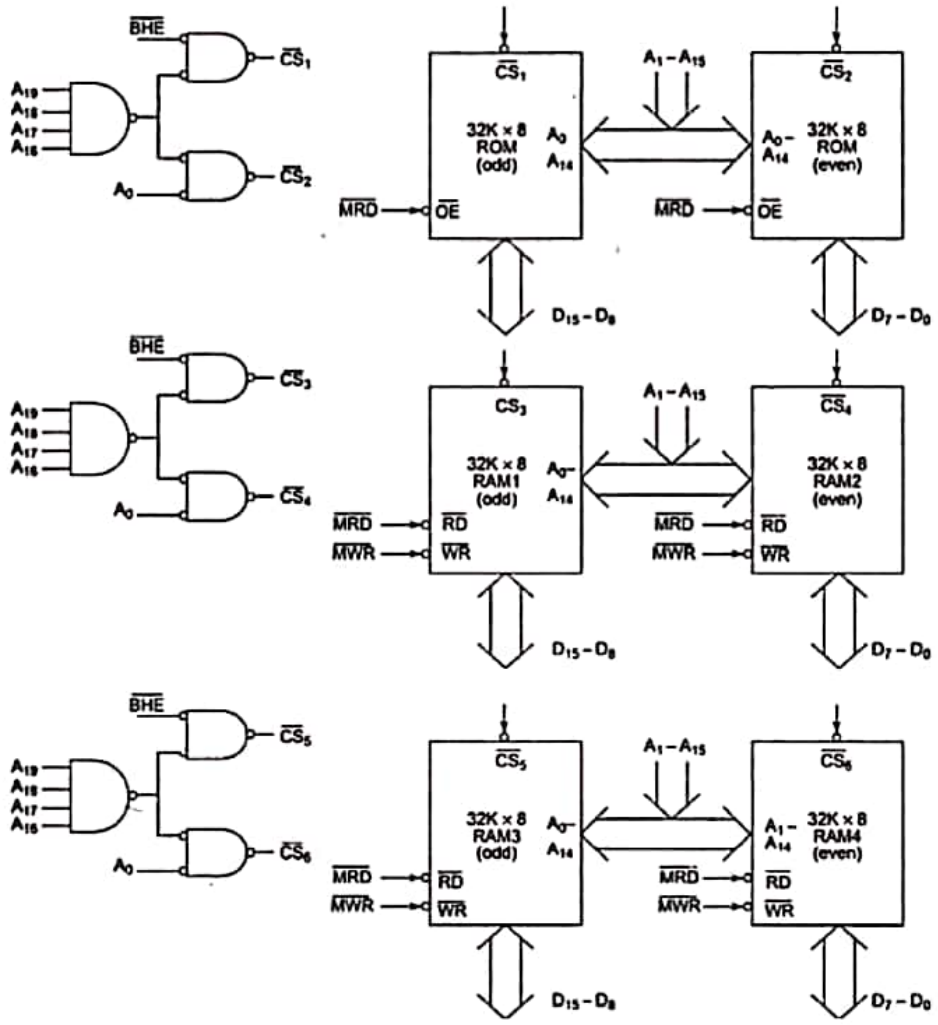
Problem3: It is required to interface two chips of 32K×8 ROM and four chips of 32K×8 RAM with 8086, according to following map.

ROM 1 and ROM 2 F0000H - FFFFFH, RAM 1 and RAM 2 D0000H - DFFFFH, RAM 3 and RAM 4 E0000H - EFFFFH. Show the implementation of this memory system.

Solution:

Address Map for Problem

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_{0}
F0000H	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ROM 1and2										64K										
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
D0000H	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RAM 1and2										64K										
DFFFFH	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
E0000H	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
RAM 3and4										64K										
EFFFFH	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



Methods of Interfacing I/O Devices

Memory Mapping	IO mapping
1. 20-bit addresses are provided for IO devices.	1. 8-bit or 16-bit address are provided for IO devices
2. The IO ports or peripherals can be treated like memory locations and so all instructions related to memory can be used for data transfer.	2. Only IN and OUT instructions can be used for data transfer between IO device and the processor.
3. In memory mapped ports, the data can be moved from any register to port and vice versa	3. In IO mapped ports, the data transfer can take only between the accumulator and the ports
4. When memory mapping is used for IO devices, the full memory address space cannot be used for addressing memory.	4. When IO mapping is used for IO devices, then the full address space can be used for addressing memory.