Memory Interface

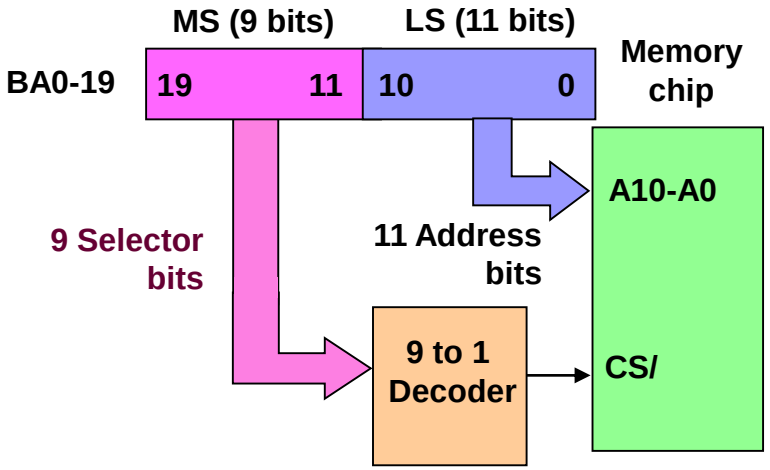
CEN433

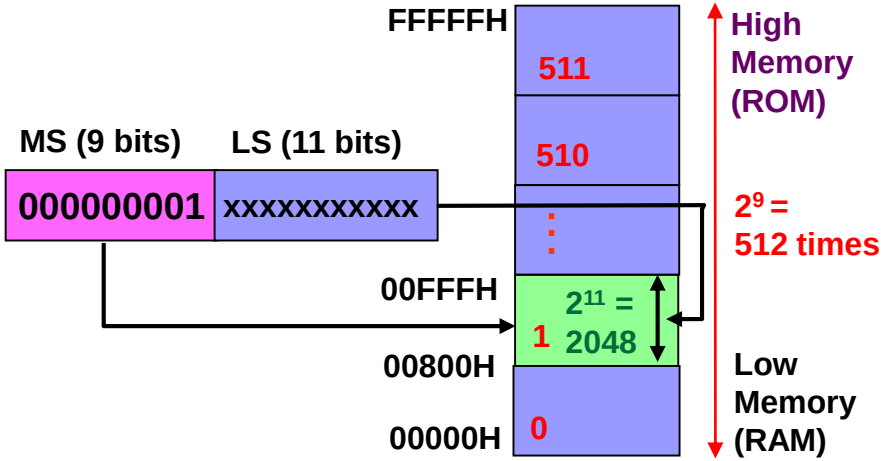
King Saud University

Dr. Mohammed Amer Arafah

Address Decoding

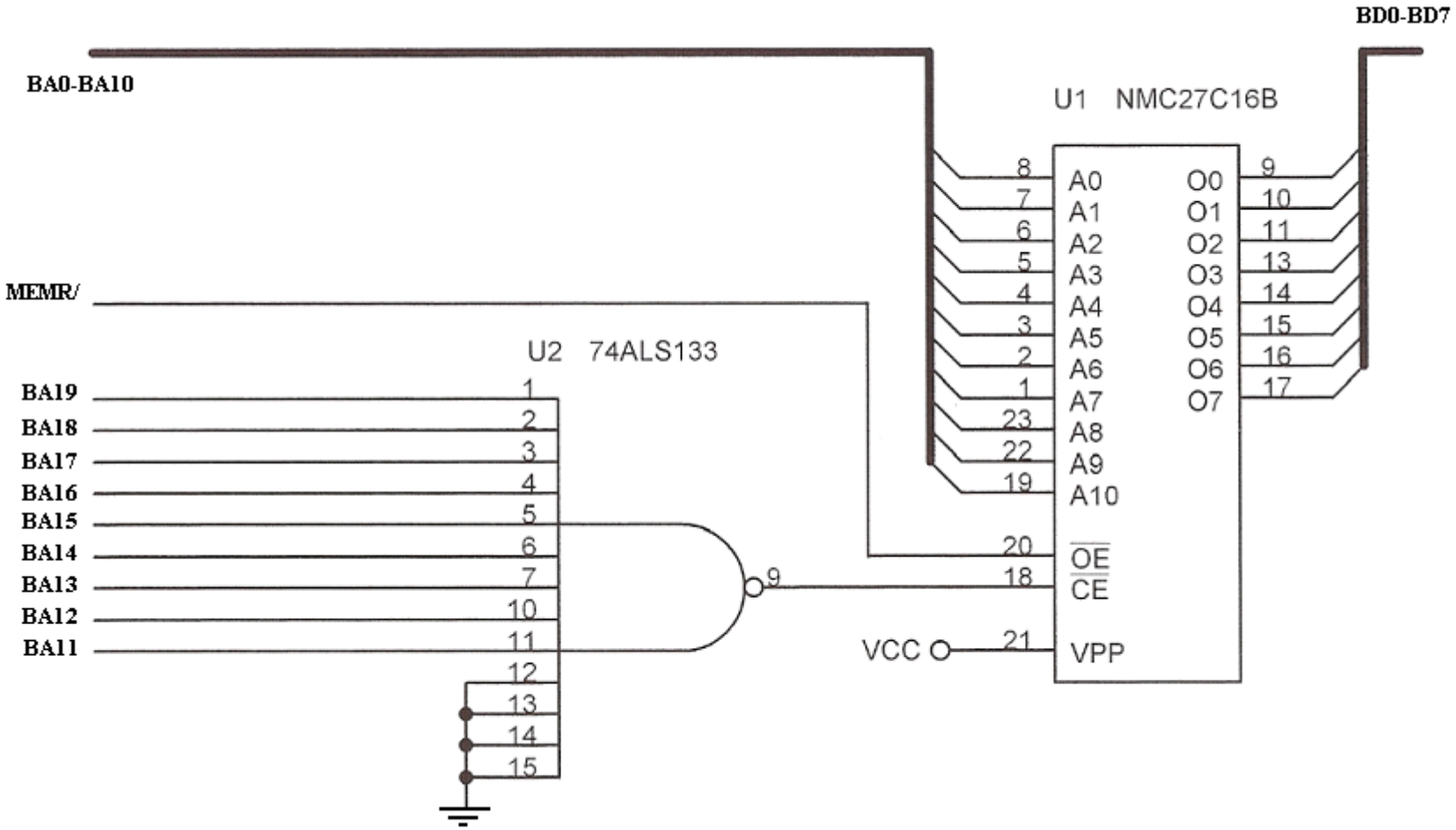
- Memory devices interfaced are usually of smaller storage capacity than the full address space of the processor
- For example, the 2716 is 2 K x 8 memory device has 11 (= 1 + 10) address inputs (A0-A10).
- When interfaced to a microprocessor with 20 address signals there is a mismatch.
- The extra 9 address pins (A11-A19) are decoded using a decoder such that they select the memory device for a unique position in the memory map of the processor.
- Here the μP address space is 2^9 times the size of the memory chip
- Decoding A11-A19 and using them for selecting the memory chip fixes the position of the memory locations in the μP address map





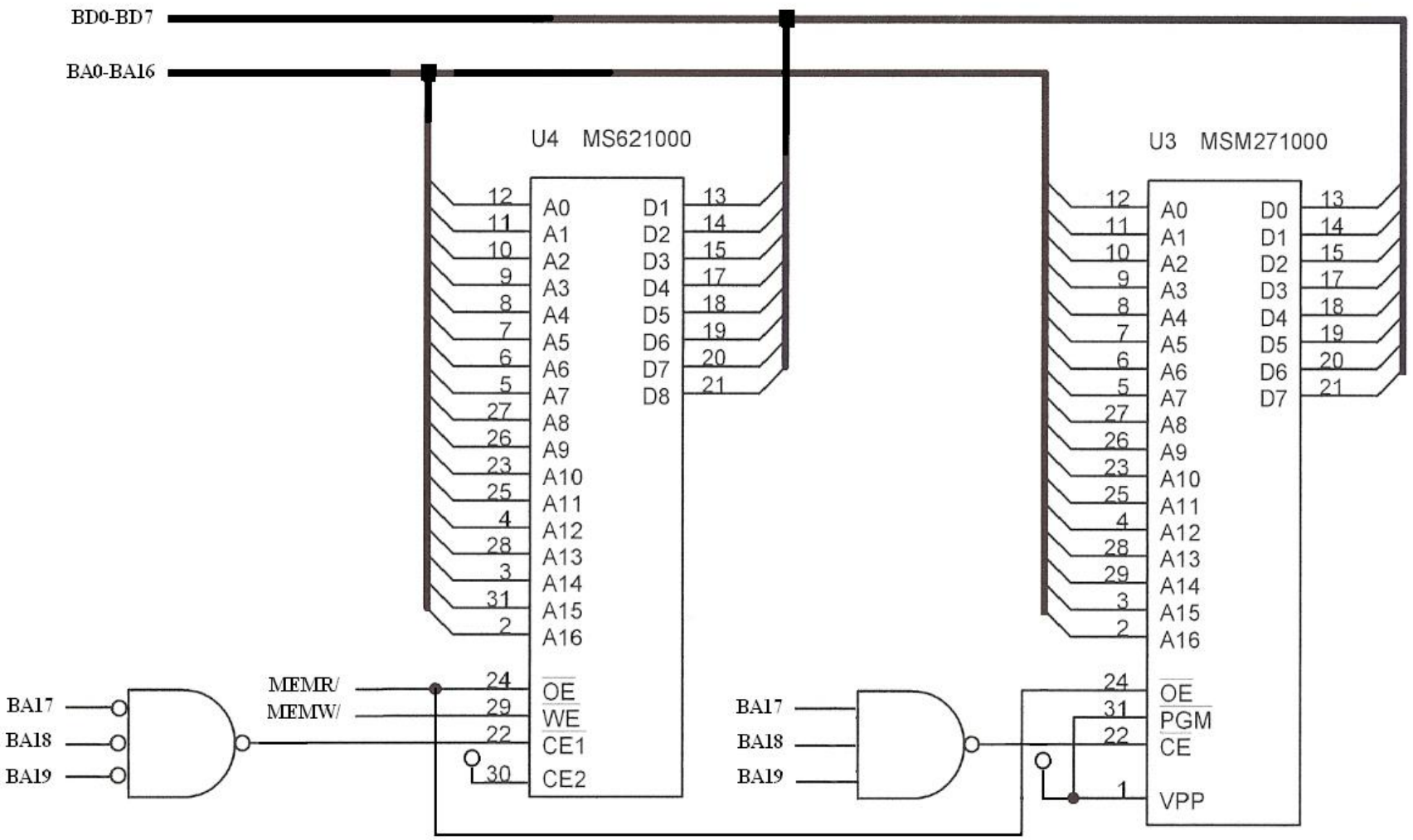
μP Memory Map

Example 1: Exhaustive Address Decoding (Simple NAND Gate Decoder)



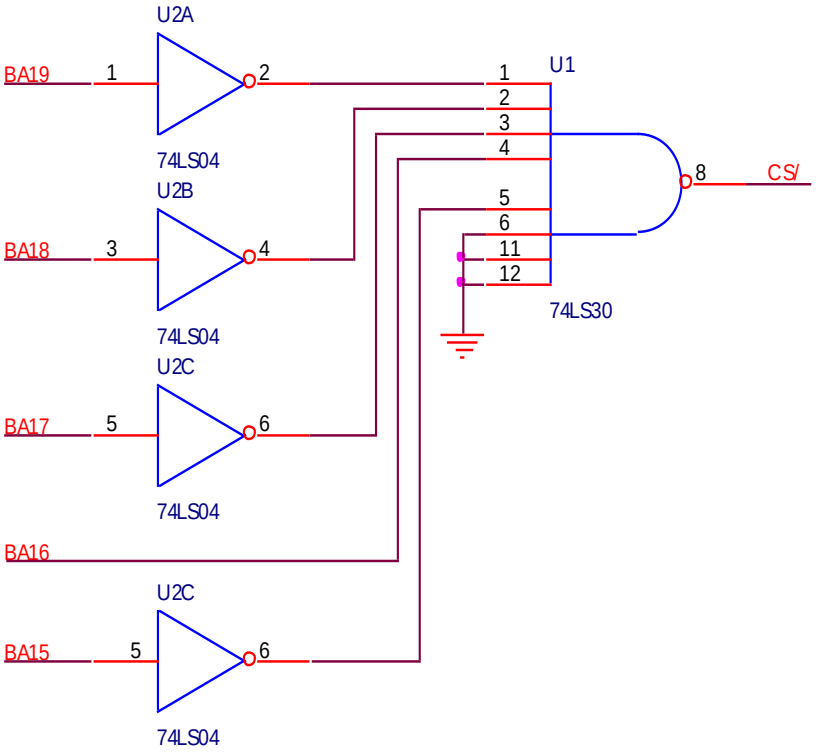
Memory locations:FF800H-FFFFFH

Example 2: Exhaustive Address Decoding

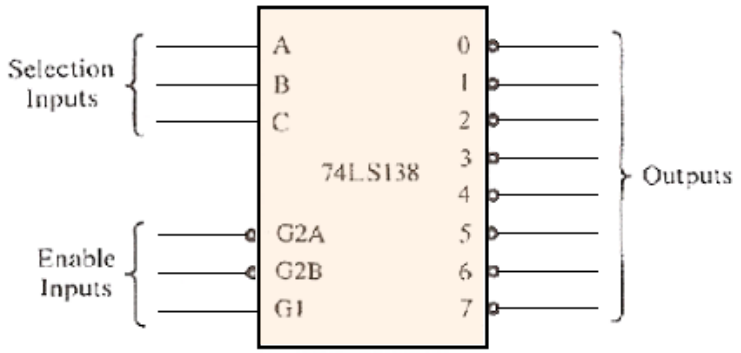


Example 3: Exhaustive Address Decoding (Simple NAND Gate Decoder)

- 32 K x 8 memory device: 15 bit address: 2^{15} locations
- Selector address: 20 – 15 = 5 bits
- If we want the memory locations to start at **10000H**, What is the selector address to decode?:
- Start Address
  00010000000000000000 = 10000H
- End address
  00010111111111111111 = 17FFFH
- Selector 5 bits: **00010** (Remain fixed)
- Check: These are 7FFF+1 = 8000H = 2^{15} locations

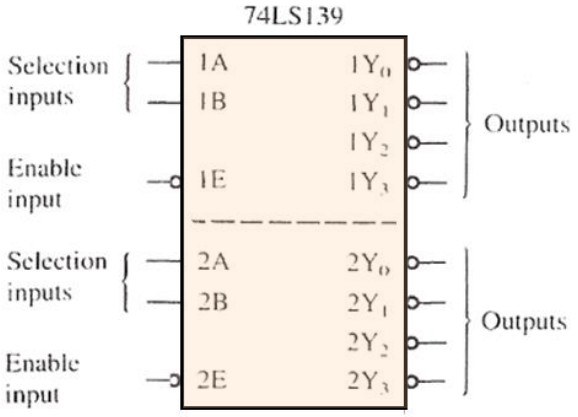


Exhaustive Address Decoding (74LS138 – 74LS139)



DM74LS138

Inputs						Outputs							
Enable			Select										
G1	$\overline{G2A}$	$\overline{G2B}$	C	B	A	YO	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	L	H	H	H	H	H	L	H	H	H	H
H	L	L	H	L	L	H	H	H	H	L	H	H	H
H	L	L	H	L	H	H	H	H	H	H	L	H	H
H	L	L	H	H	L	H	H	H	H	H	H	L	H
H	L	L	H	H	H	H	H	H	H	H	H	H	L



DM74LS139

Inputs			Outputs			
Enable	Select					
G	B	A	Y0	Y1	Y2	Y3
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

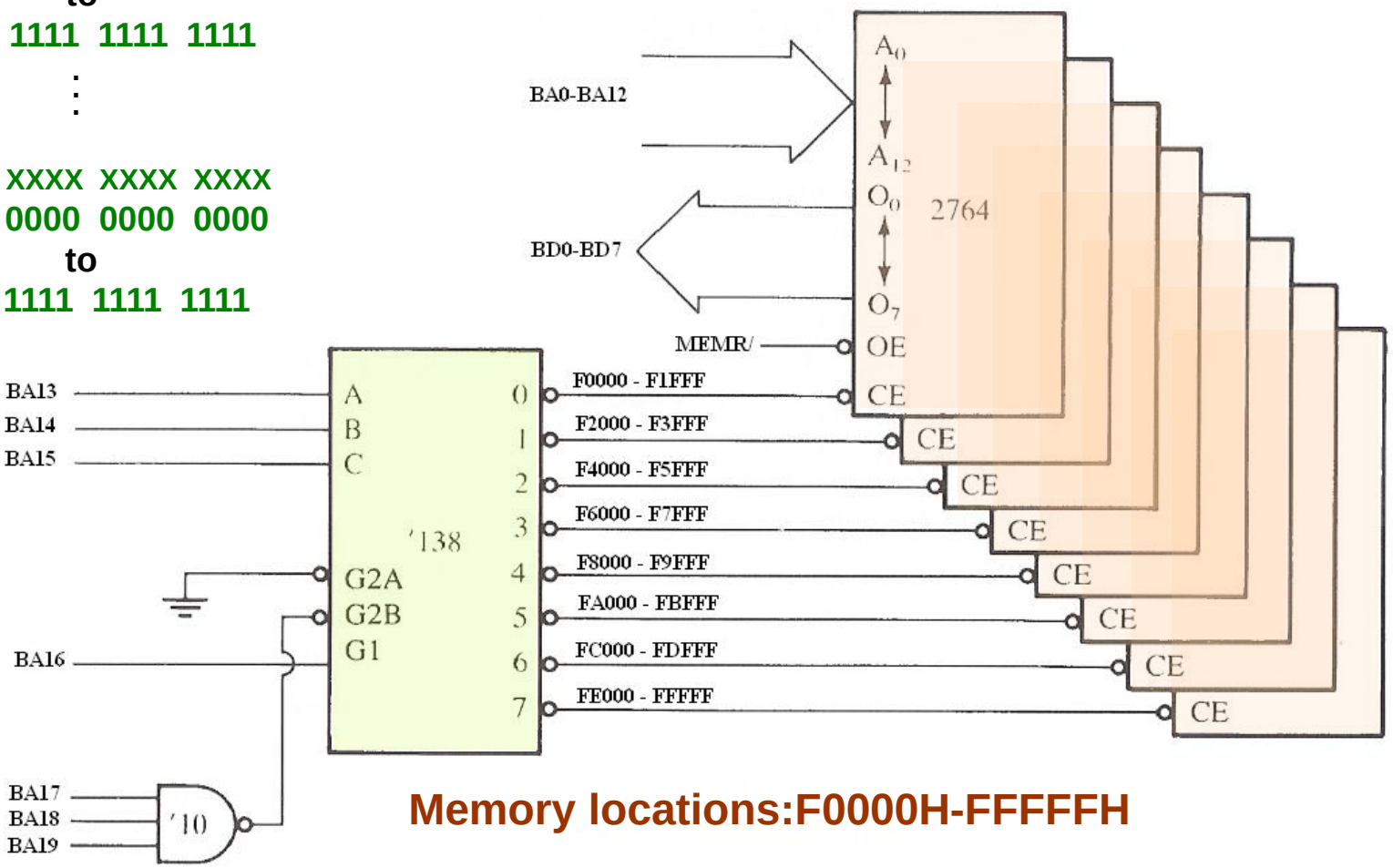
Example 4: Exhaustive Address Decoding (74LS138)

Module 0:

1111 000x xxxx xxxx xxxx
1111 0000 0000 0000 0000
to
1111 0001 1111 1111 1111
⋮

Module 7:

1111 111x xxxx xxxx xxxx
1111 1110 0000 0000 0000
to
1111 1111 1111 1111 1111



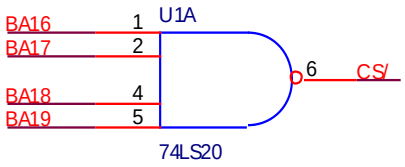
Memory locations:F0000H-FFFFFH

Example 5: Exhaustive Address Decoding (74LS138)

64 KB EPROM Starting at F0000H

Answer:

- → **F0000 – FFFFFH.**
- → **BA0-BA15: Location Addressing.**
- → **BA16-BA19: Space Addressing.**



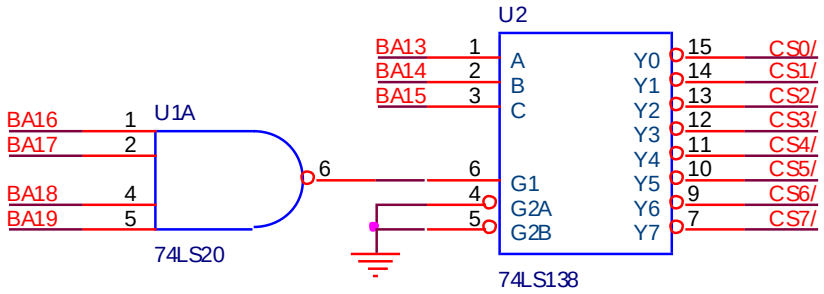
Assume that 64 KB EPROM is not found!

We can replace it with 8 of 8KB EPROM.

Starting address is F0000H.

Answer:

- → **BA16-BA19: Space Addressing.**
- → **8KB EPROM**
  - → **BA0-BA12: Location Addressing**
- → **BA13-BA15: 8KB Module Addressing**

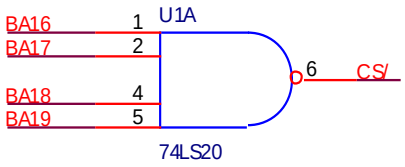


Example 6: Exhaustive Address Decoding (74LS139)

64 KB EPROM Starting at F0000H

Answer:

- → **F0000 – FFFFFH.**
- → **BA0-BA15: Location Addressing.**
- → **BA16-BA19: Space Addressing.**



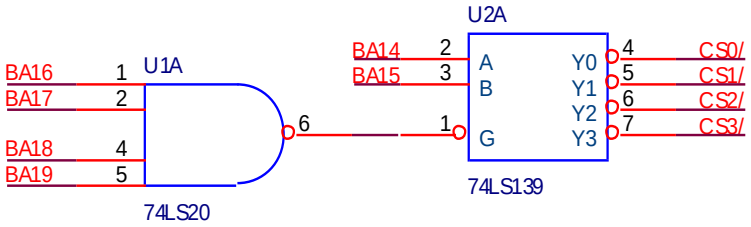
Assume that 64 KB EPROM is not found!

We can replace it with 4 of 16KB EPROM.

Starting address is F0000H.

Answer:

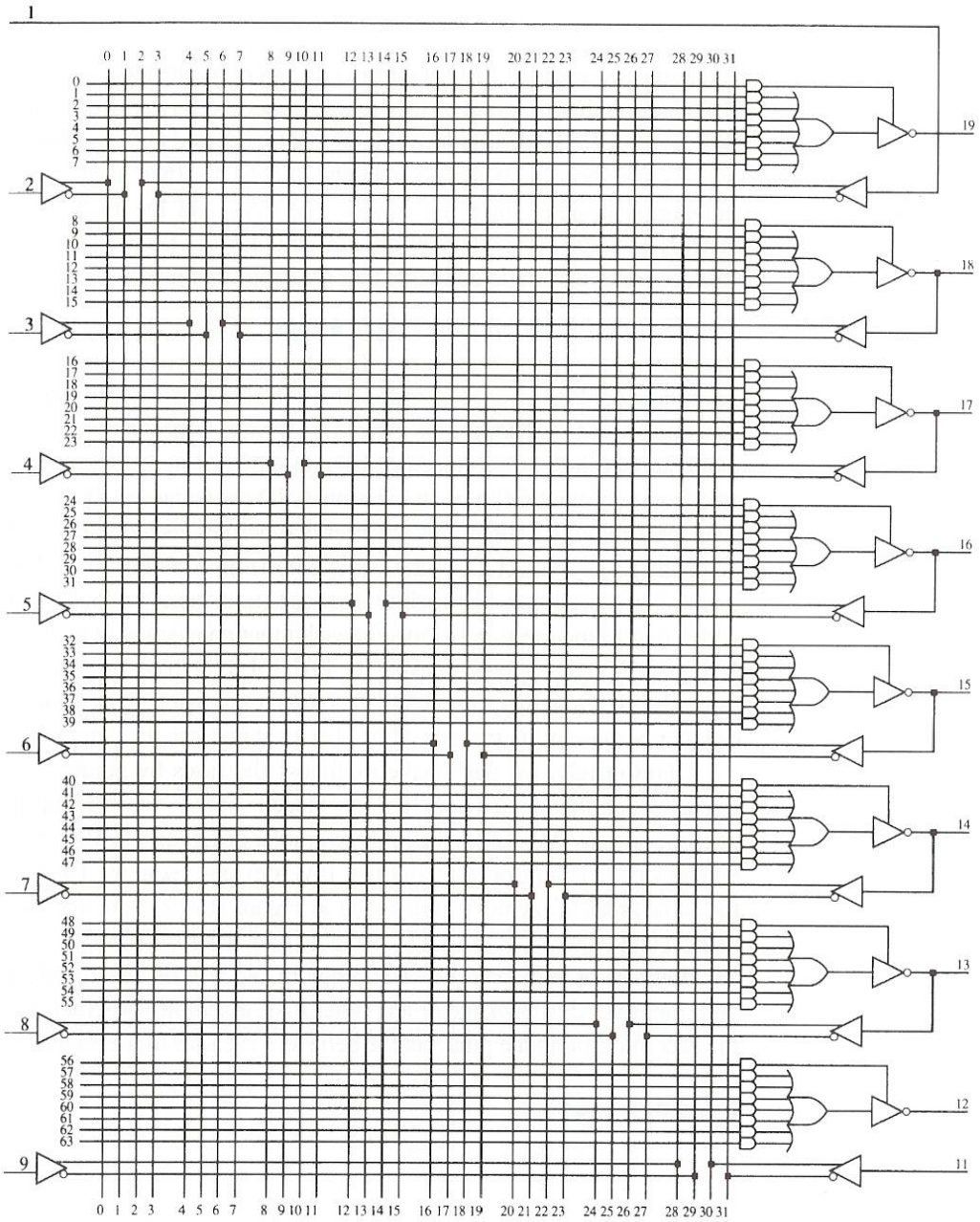
- → **BA16-BA19: Space Addressing.**
- → **16KB EPROM**
  - → **BA0-BA13: Location Addressing**
- → **BA14-BA15: 8KB Module Addressing**



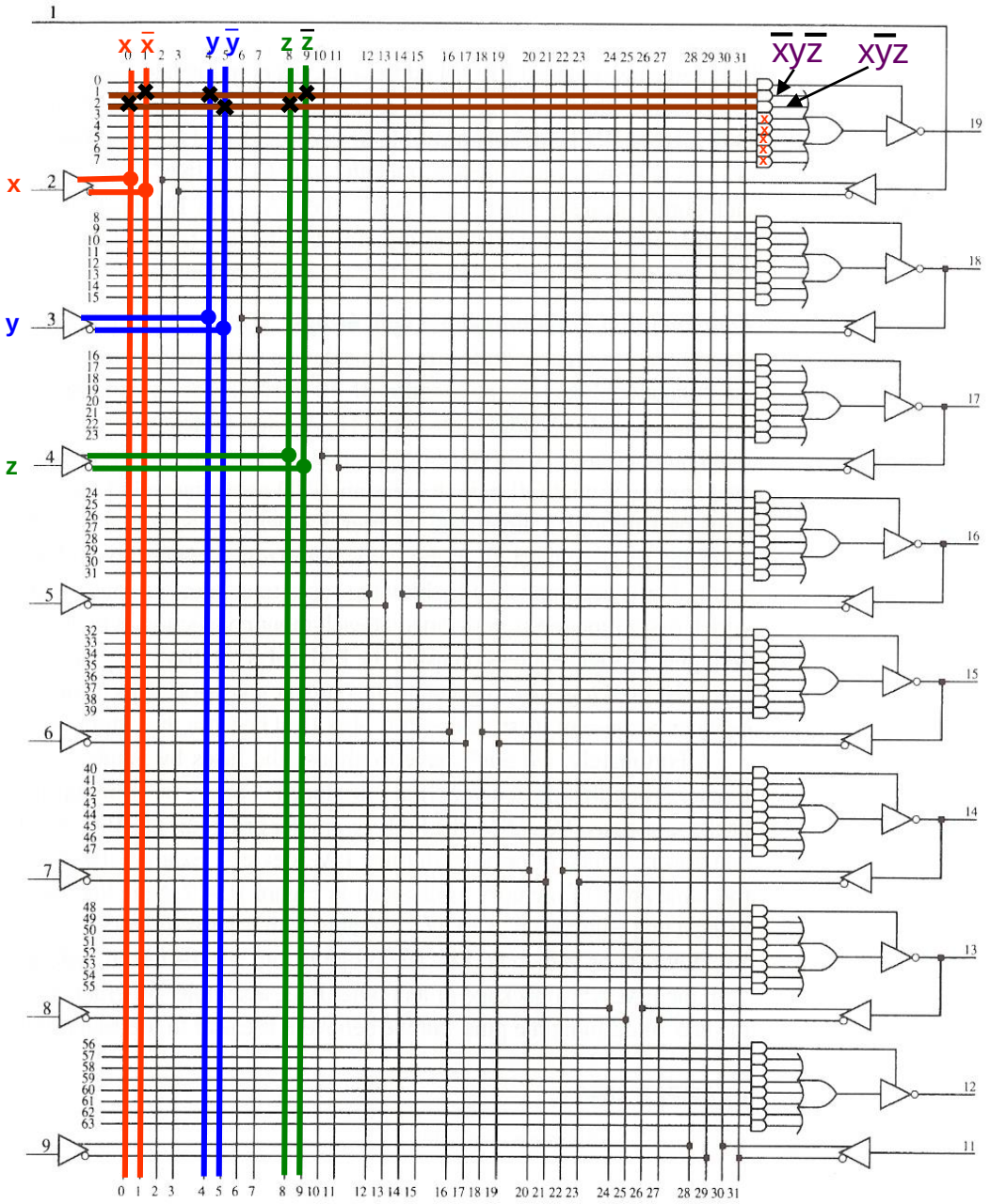
Exhaustive Address Decoding (PLD Decoders)

- Many modern systems use programmable logic decoders in place of integrated decoders
- They give total freedom in decoding different addresses for individual memory devices
- Programmable logic devices have may be called:
  - PLDs: Programmable logic devices
  - PLAs: Programmable logic array
  - PALs: Programmable array logic
  - GALs: Gate Array Logic
  - SPLDs: Simple programmable logic devices
  - CPLDs: Complex programmable logic devices
- They are all programmable logic devices. Nowadays they can be programmed using VHDL (Verilog Hardware Definition Language)
- Some types are programmed only once (fused links), similar to PROMs
- Some types are erasable like EPROMs
- The next slide shows one of the most common low cost devices: the PAL16L8

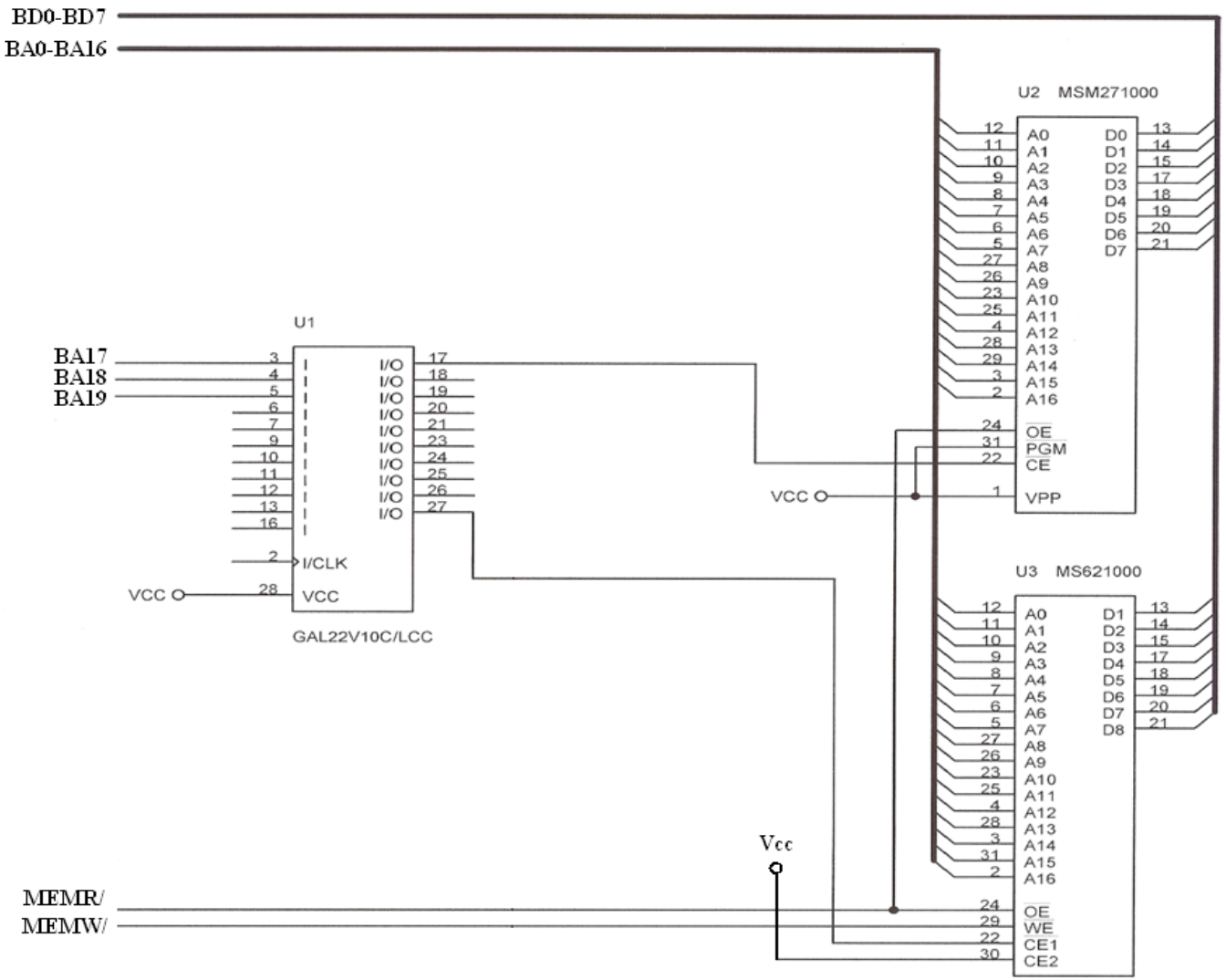
PAL16L8


1
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
2
3
4
5
6
7
8
9
19
18
17
16
15
14
13
12
11

PAL16L8



Example 7: Exhaustive Decoding

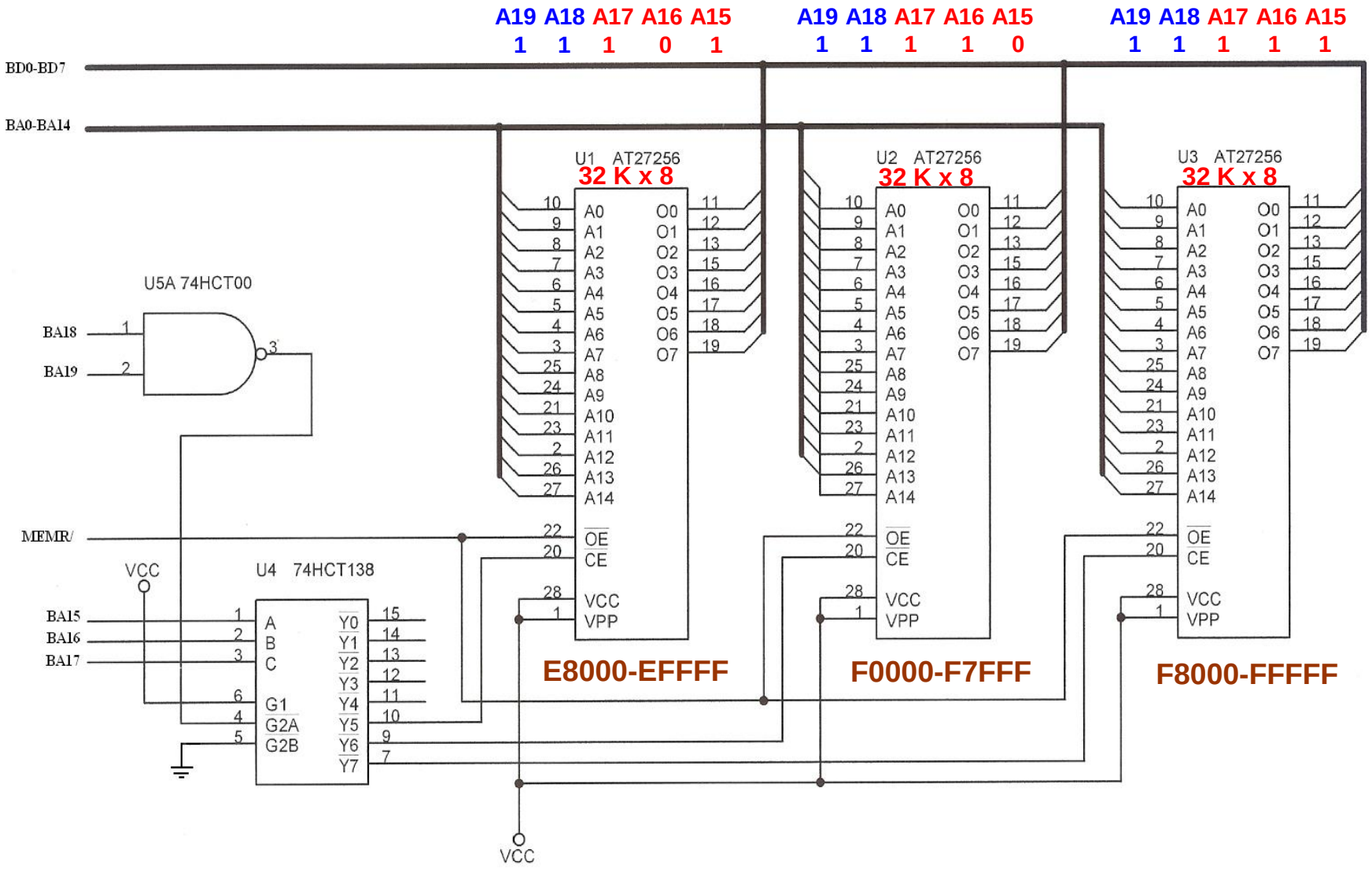


Example 7: Exhaustive Decoding

```
library ieee;
use ieee.std_logic_1164.all;
entity DECODER_10_17 is
port (
    BA19, BA18, BA17: in STD_LOGIC; Input declaration
    ROMCS/, RAMCS/ : out STD_LOGIC; output declaration
);
end;
architecture V1 of DECODER_10_17 is
begin

    ROMCS/  <= not BA19 or not BA18 or not BA17
                    ;  ROMCS/ = 0 for BA19BA18BA17 = 111
    RAMCS/  <= A17 or A18 or BA19
                    ; RAMCS/ = 0 for  BA19BA18BA17 = 000
    end V1;
```

Example 8: Exhaustive Decoding



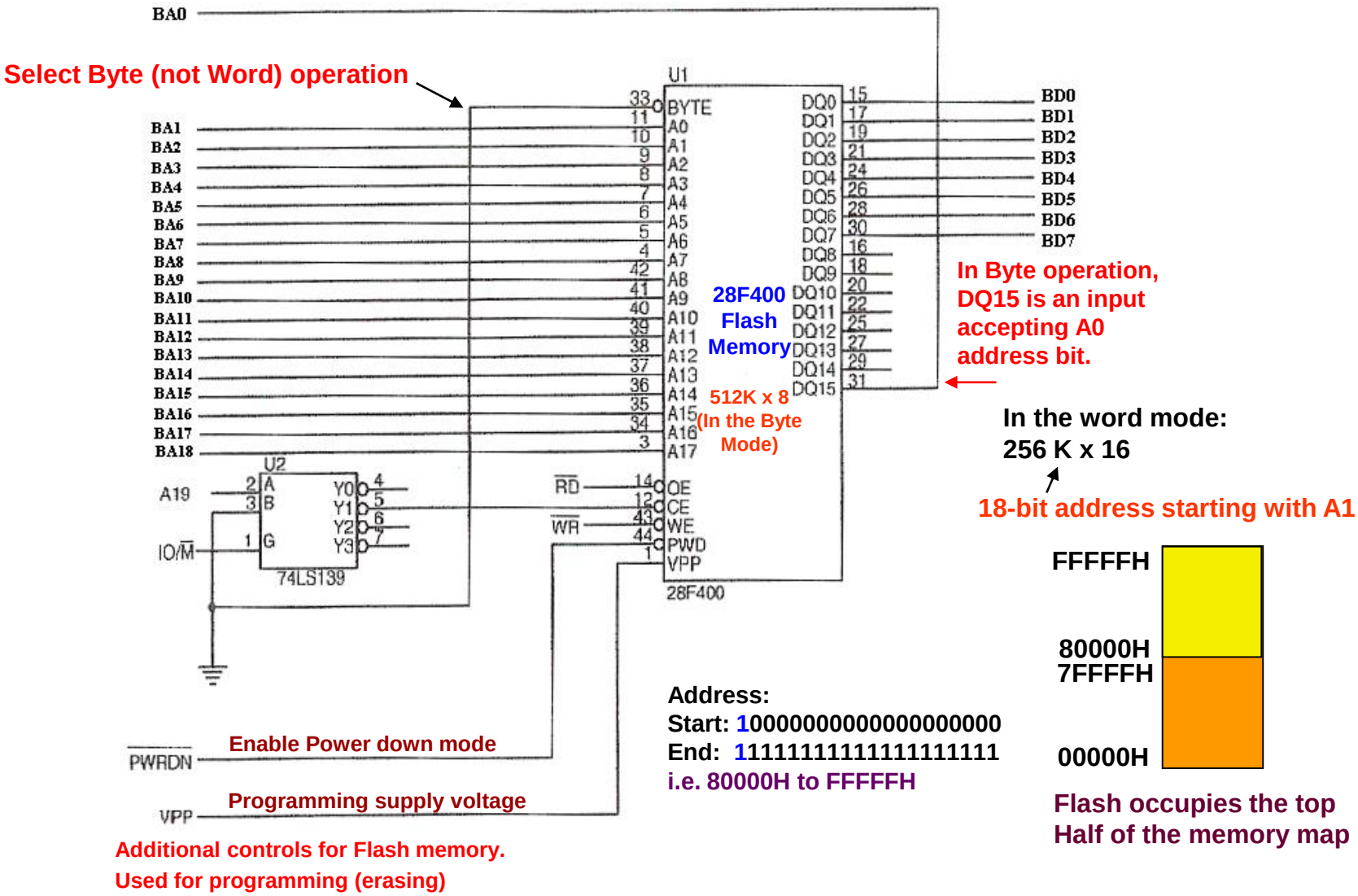
Interfacing EEPROM (Flash) Memories

- **Main Flash memory applications:**
  - Used when contents need to be changed only infrequently, e.g.:
    - Storing system BIOS
    - USB pen drives
    - MP3 audio players
- **Similarities with SRAMs:**
  - Both need the 3 basic memory control inputs: CE, OE, and WE
- **Differences with SRAMs:**

1. EEPROM needs an additional programming controls and programming (erasing) supply voltage. Used to be 25 or 12V, now 5 or even 3.3V.

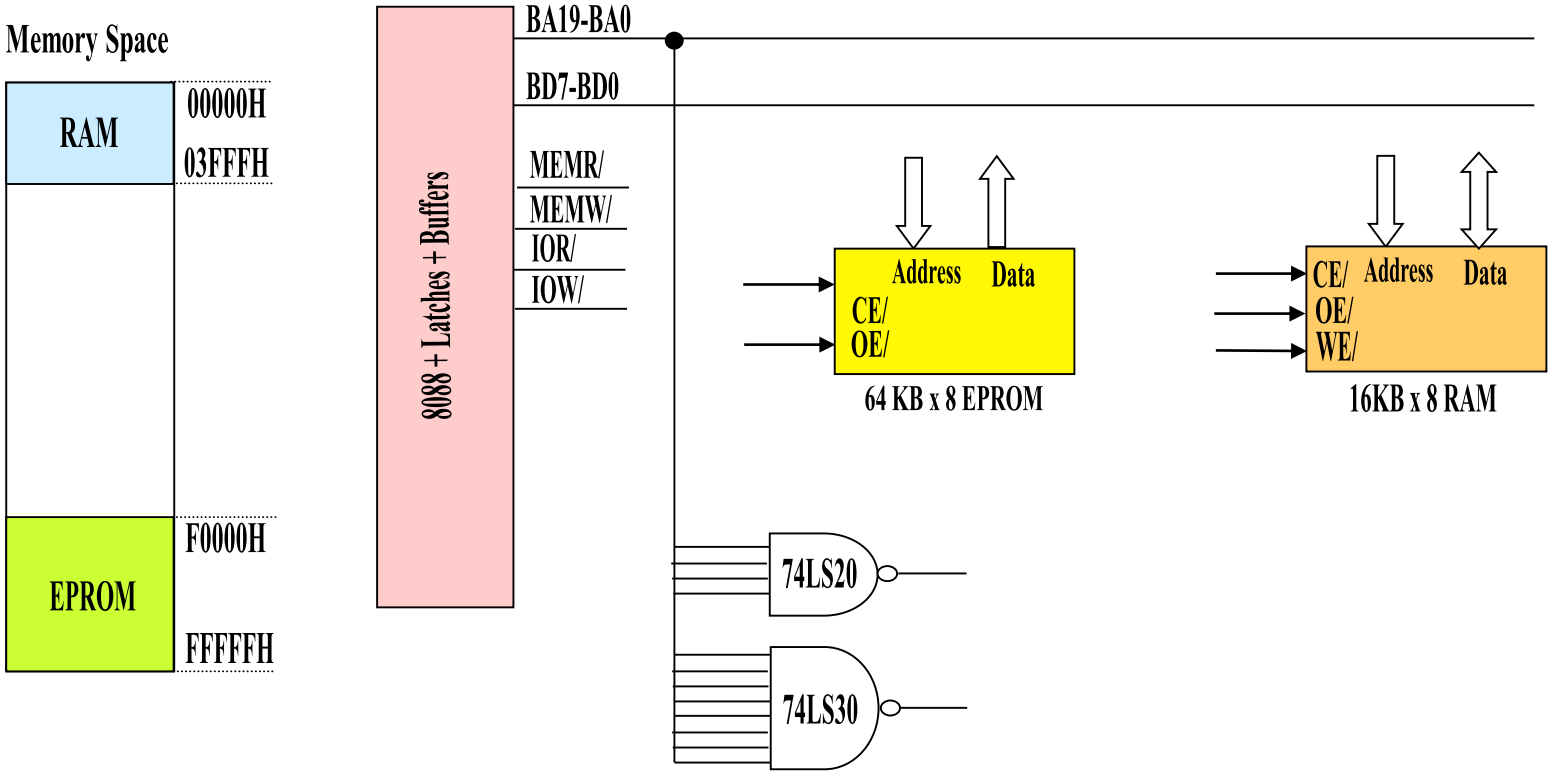
1. EEPROM is much slower to write (erase) a byte: 0.4 s Versus 10ns for SRAM

Example 9: Interfacing EEPROM

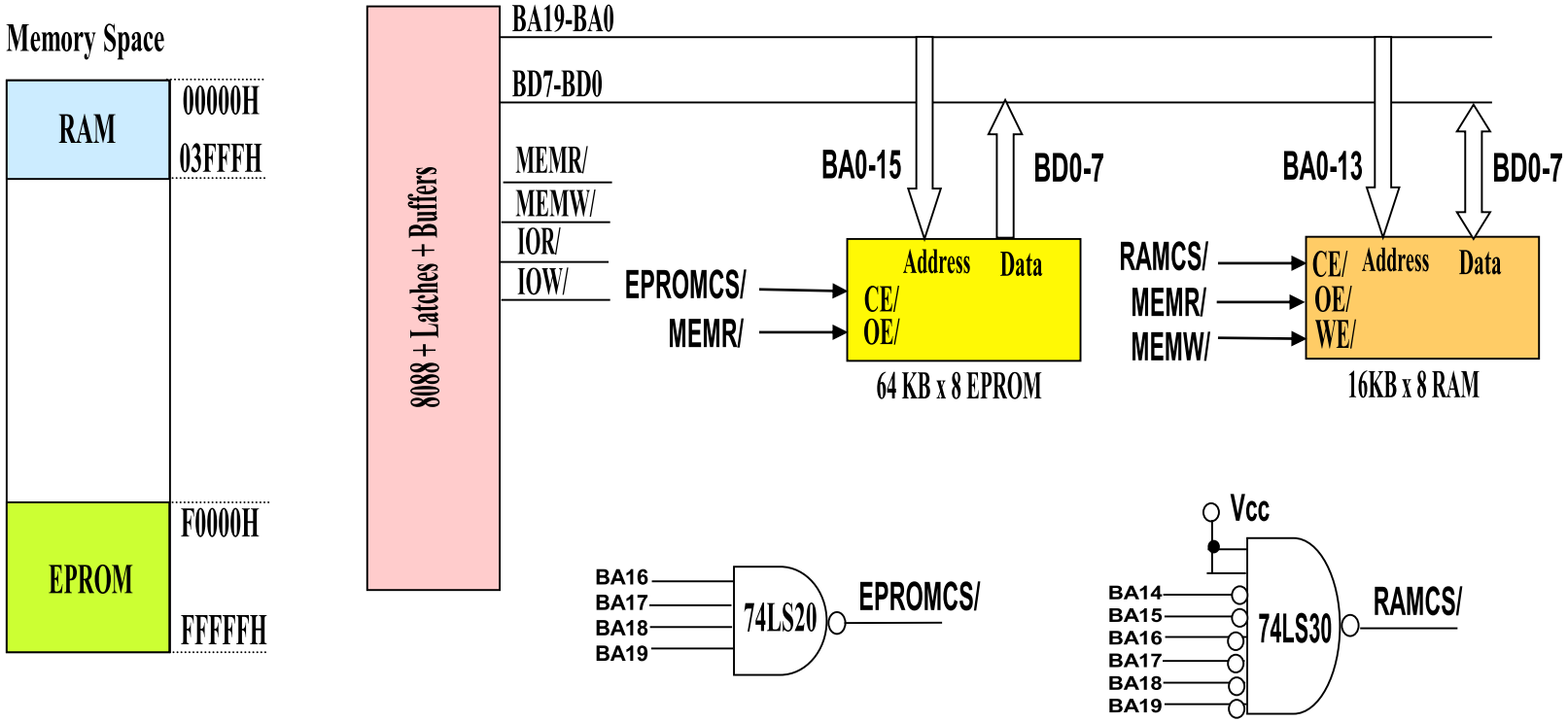


Example 10: Exhaustive Decoding

Interface a 16KB RAM and a 64KB EPROM to an 8088 buffered system. Assume the starting address of the RAM is 0H and the starting address of the EPROM is F0000H. Use NAND gates, as needed, to generate the chip selects.

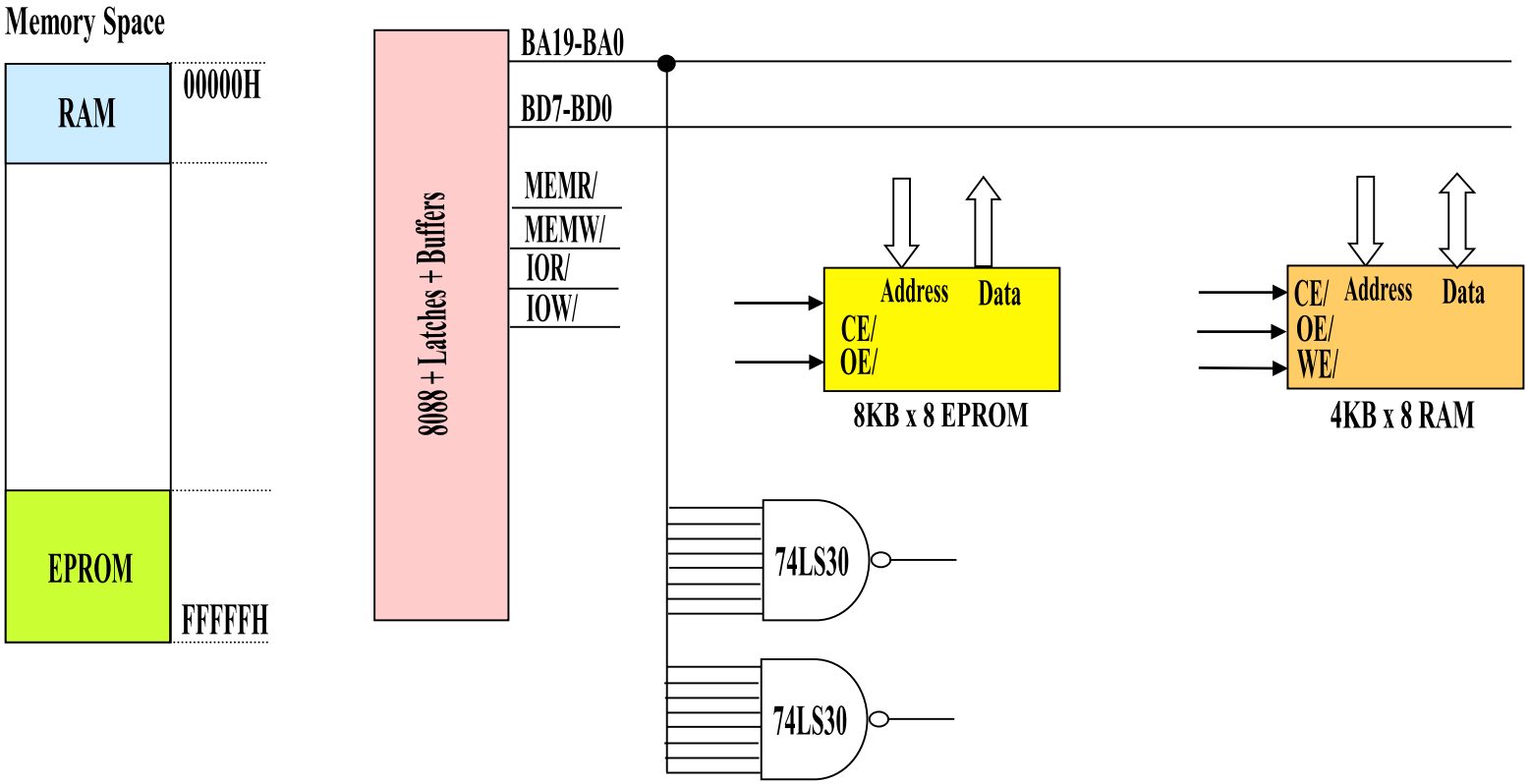


Solution of Example 10:

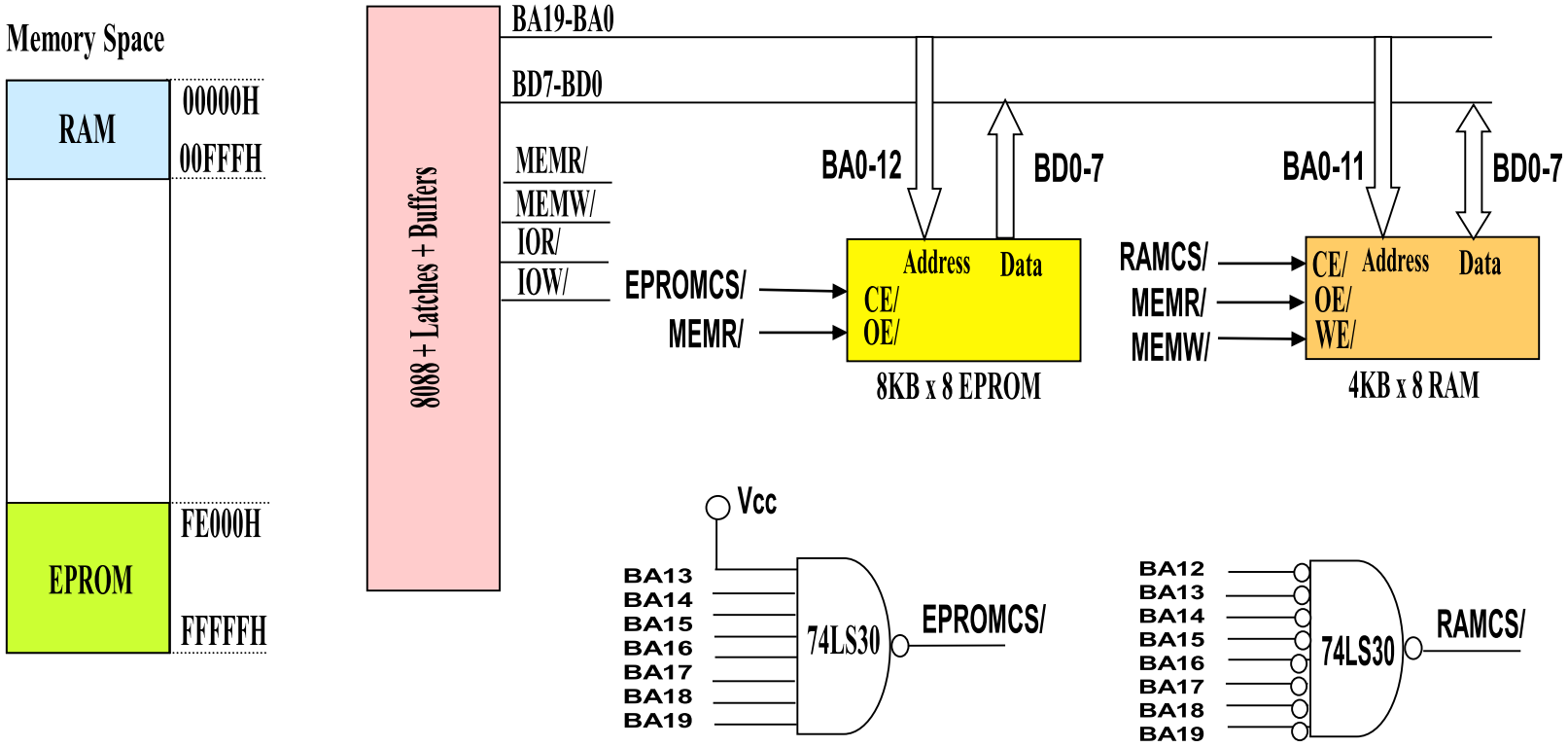


Example 11: Exhaustive Decoding

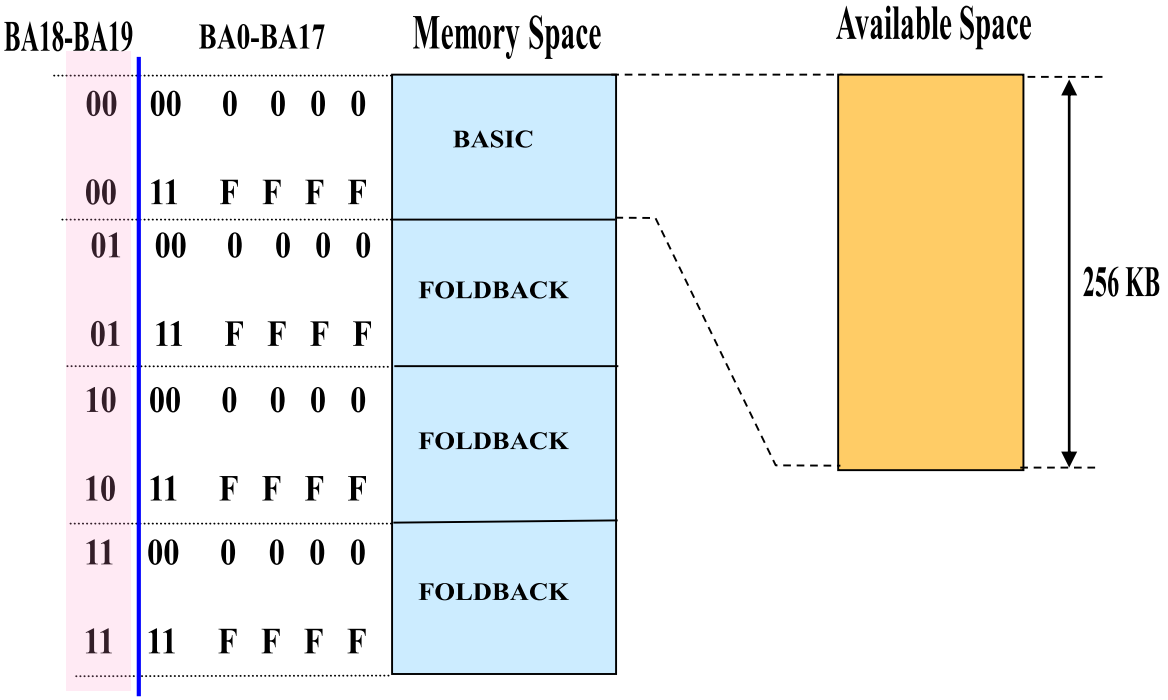
Interface a 4KB RAM and an 8KB EPROM to an 8088 buffered system. Assume the starting address of the RAM is 0H and the last address of the EPROM is FFFFFH. Use NAND gates, as needed, to generate the chip selects.



Solution of Example 11:

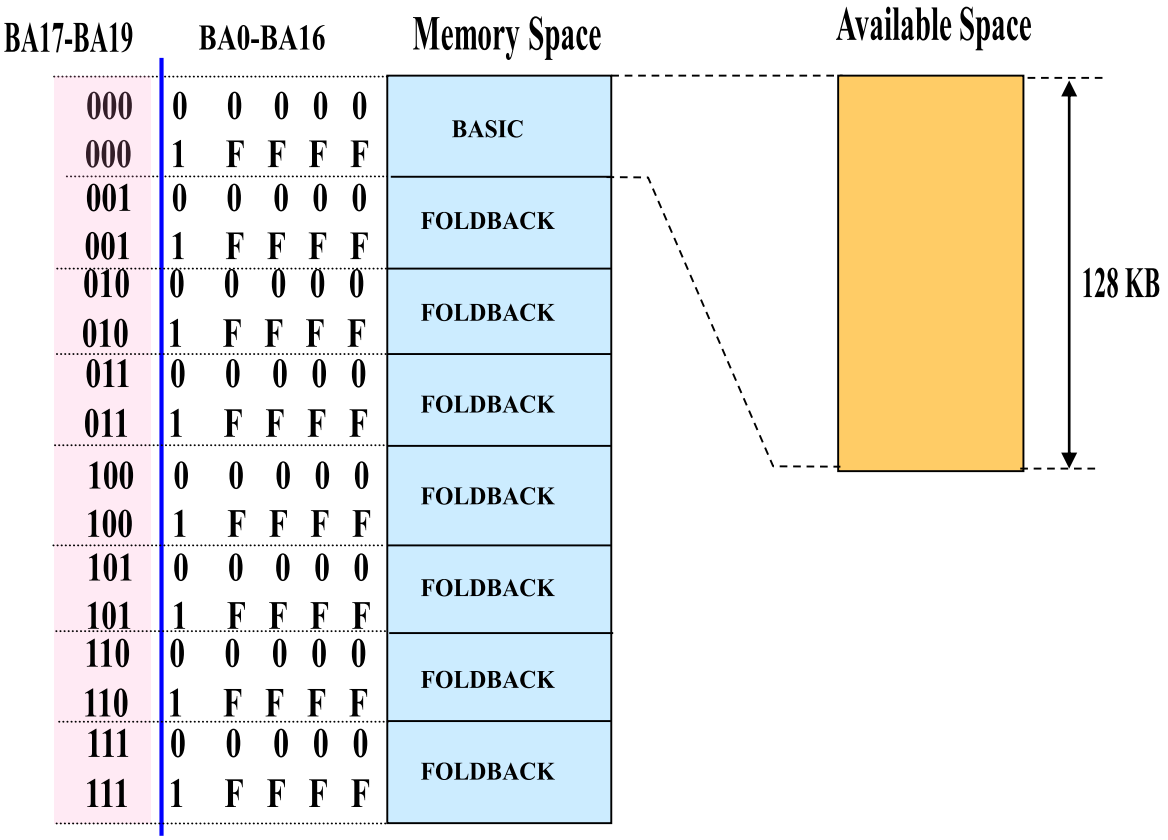


Example 1: Partial Decoding



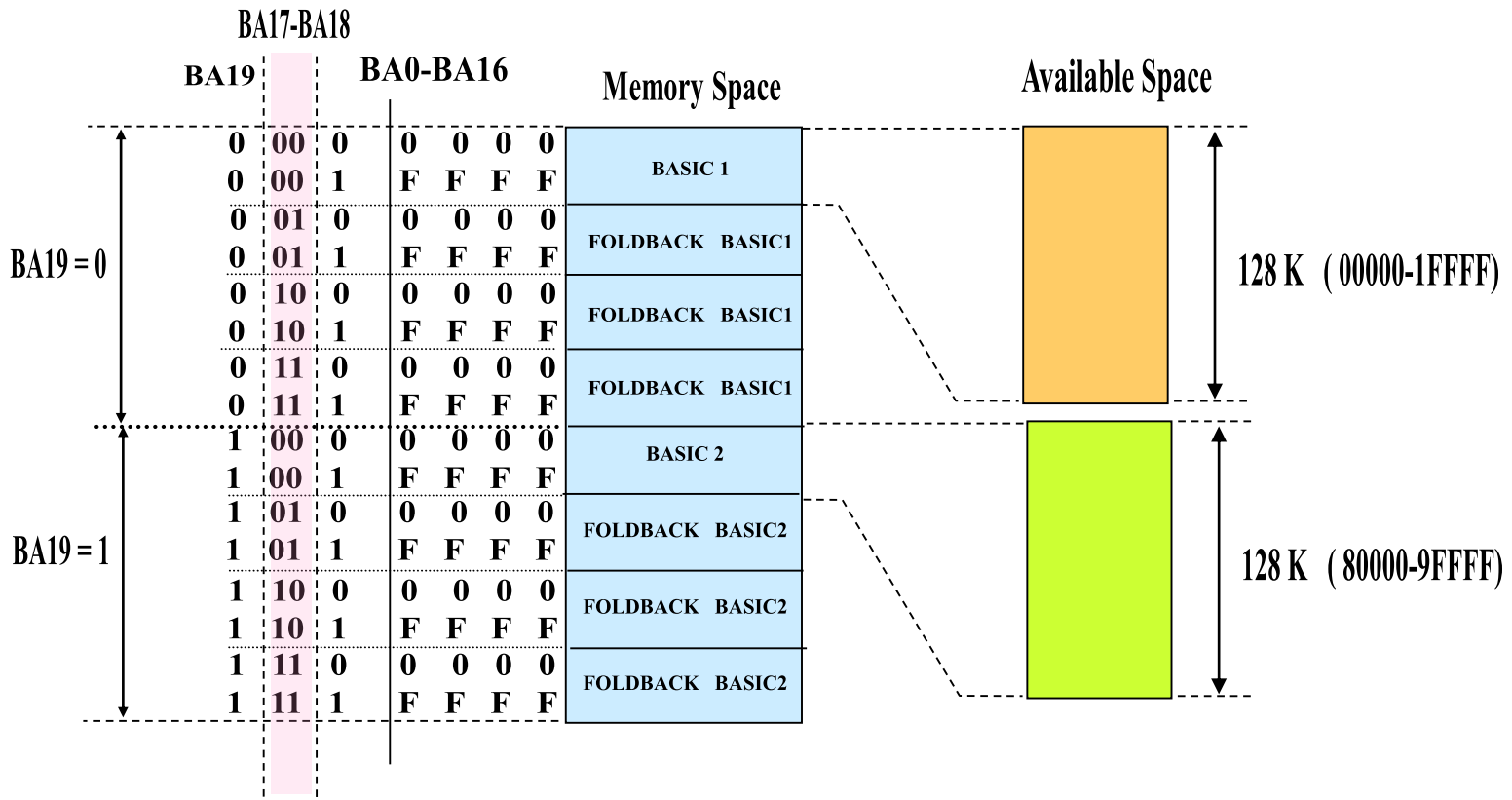
Ignoring both BA18 and BA19

Example 2: Partial Decoding



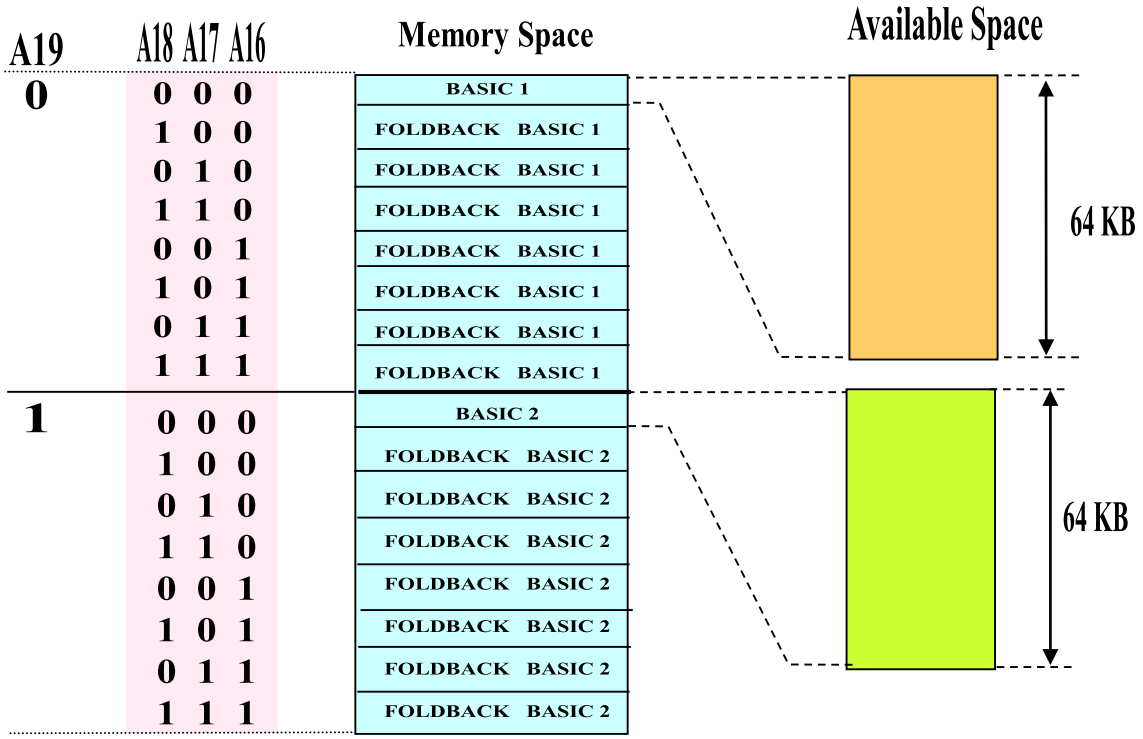
Ignoring both BA17, BA18 and BA19

Example 3: Partial Decoding



Ignoring both BA17 and BA18

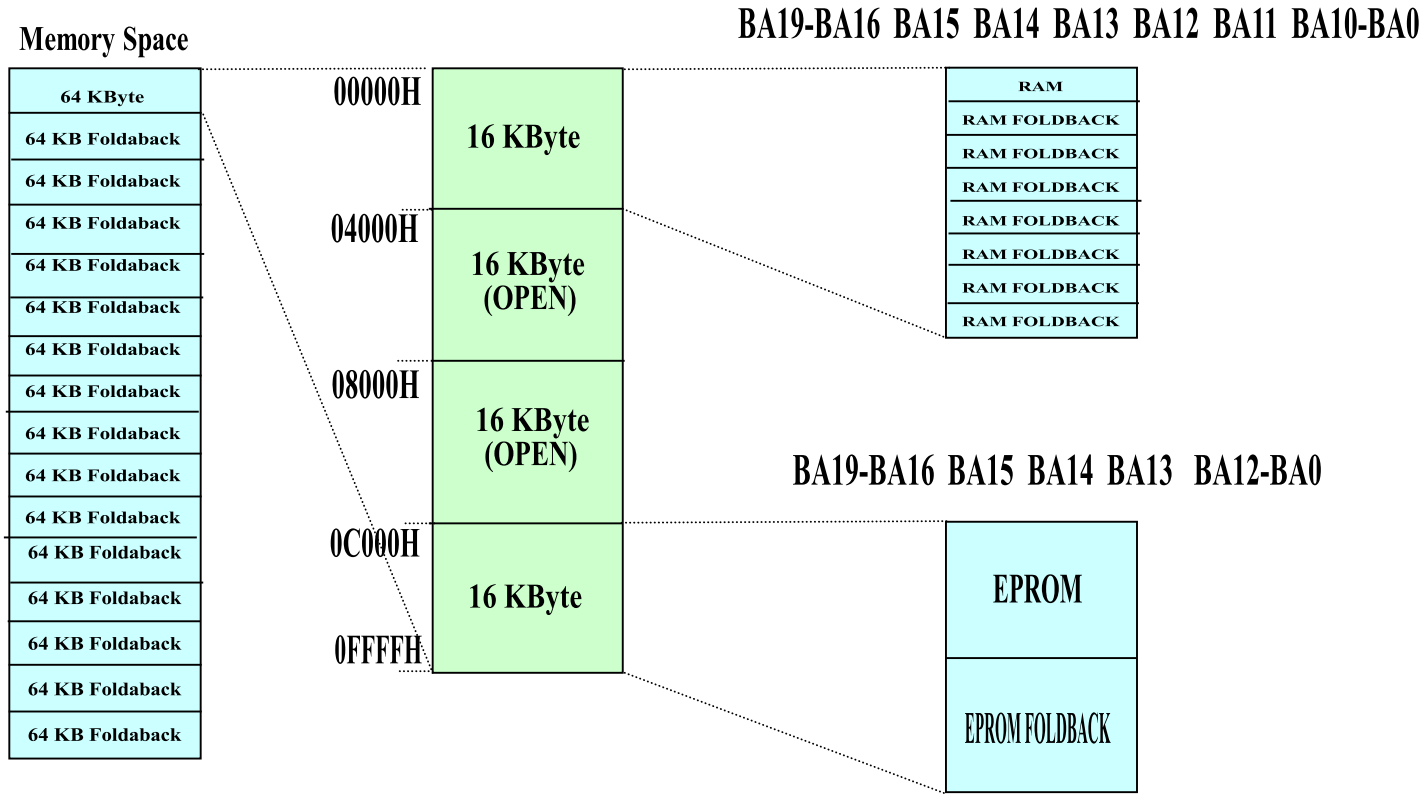
Example 4: Partial Decoding



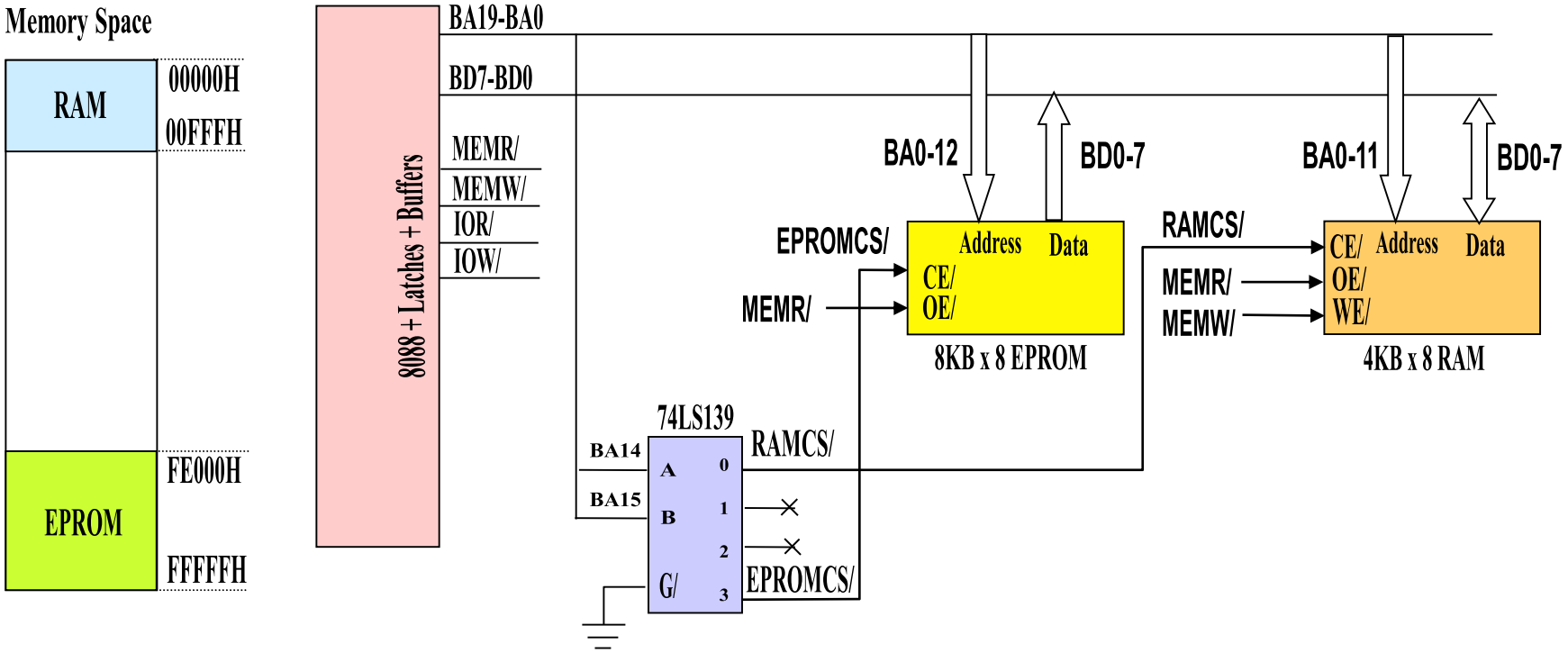
Ignoring BA16, BA17 and A18

Example 5: Partial Decoding

Assume that the 1M memory space is divided into 16 blocks. One of these blocks is used, and the rest are ignored. The block is divided further into 4 segments. The first segment is used only by a 2KB RAM, the second and the third segment are left for future expansion, and the fourth segment is used only by an 8KB EPROM. Interface both 2KB RAM and 8KB EPROM to the 8088 system using partial decoding.



Solution of Example 5:



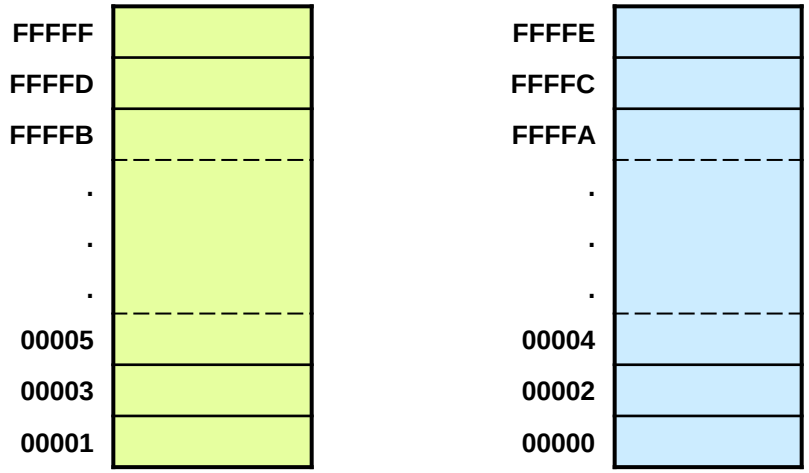
Interfacing 16-bit Memory to the 8086, 80286, and 80386SX

- **Main differences: 8086 from the 8088:**
  - The M/#IO replaces the IO/#M control signal
  - The data bus is now 16 bits not 8 bits
  - A new control output, BHE/ [Bus (byte) High Enable]
  - A0 has a special use as BLE/ [Bus (byte) Low Enable]
- **Main differences between 8086/186 and 80286/386SX:**
  - 80286/386SX has 24-bit address bus (A23-A0)
  - The M/#IO, RD/, WR/ are replaced with MRDC/, MWTC/, IORDC/ and IOWTC/- more specific signals

16-Bit Wide Memory

- 16-bit wide memory is organized in two separate 8-bit wide memory banks:
  - Low bank (even-numbered byte locations: 0, 2, 4, 6, …)
    → low 8 bits of the data bus (D0-D7): LS Byte
  - High bank (odd-numbered byte locations: 1, 3, 5, 7, …)
    → high 8 bits of the data bus (D8-D15): MS Byte
- Processor must be able to access any 16 or 8 bit locations
- Banks are selected by the microprocessor through bank selection (Bank Enable) signals
- On the 8086, these byte selection signals are
  - The BLE/ (A0) signal selecting low bank
  - The BHE/ signal selecting high bank
- Only with writes…. For Reads, the processor will take the byte it wants and no need to enforce byte (bank) selection

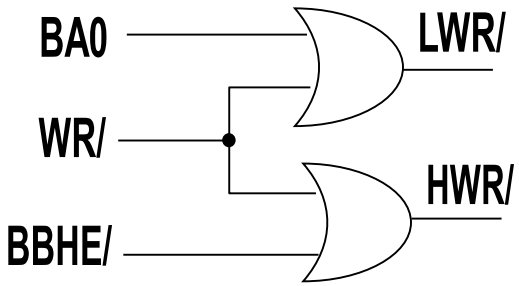
16-Bit Wide Memory



BBHE/	BA0	Function
0	0	Both banks enabled for a 16-bit transfer
0	1	High bank enabled for an 8-bit transfer
1	0	Low bank enabled for an 8-bit transfer
1	1	No banks enabled

16-Bit Wide Memory

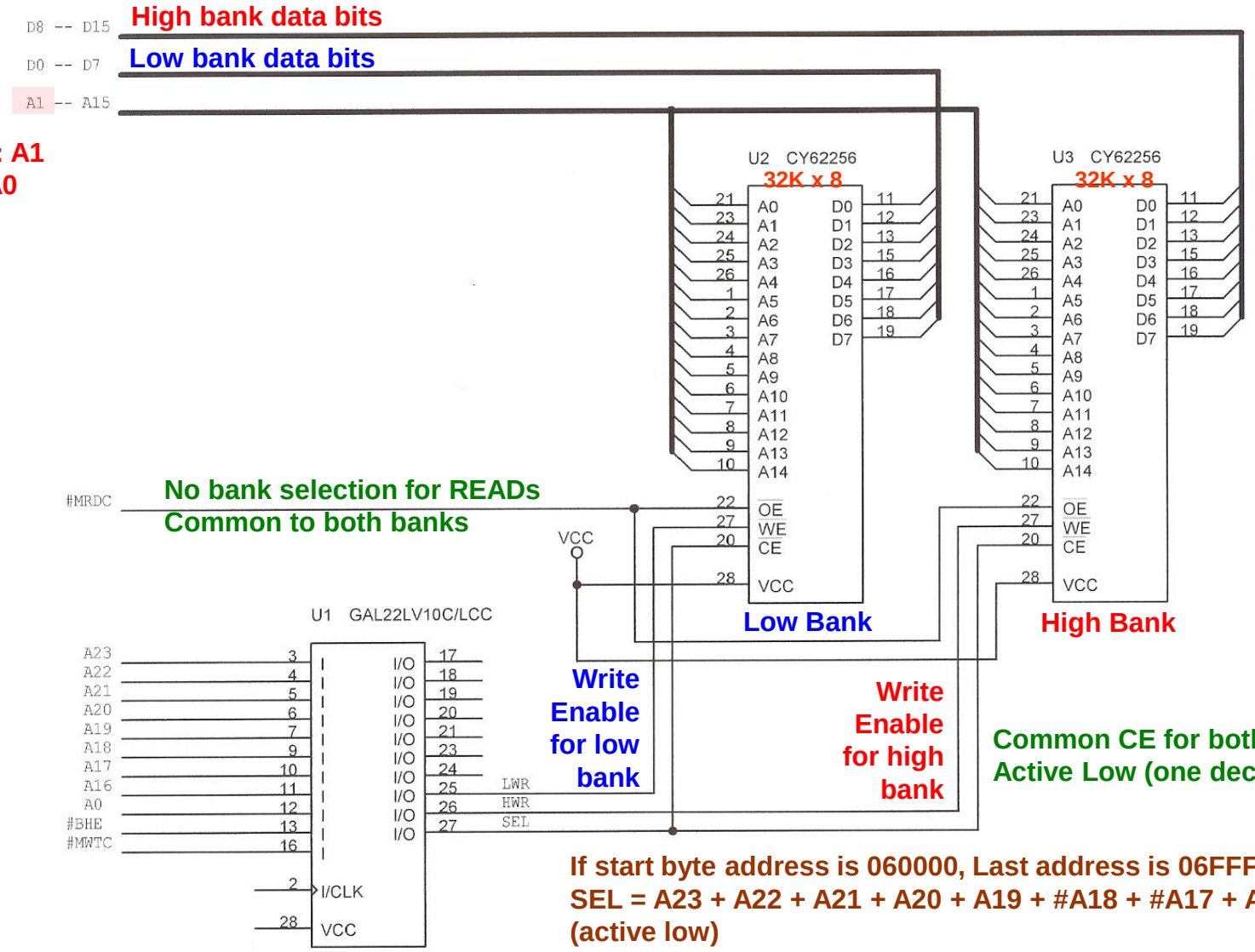
Encoding Separate Write Signals



- **Why consider this only with write access (not Reads)?**

  Because the processor can choose only the byte(s) it wants to read from the full 16-bit data placed on the data bus by the 2 banks. So always enable both banks for READs.

Example 1: 16-Bit Wide Memory

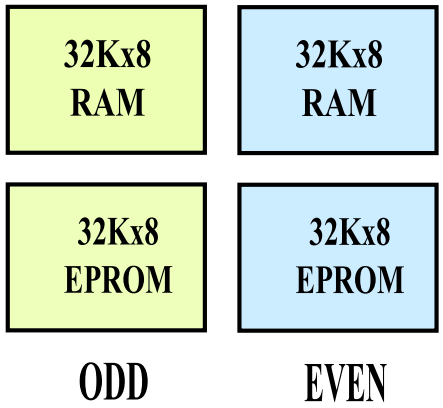


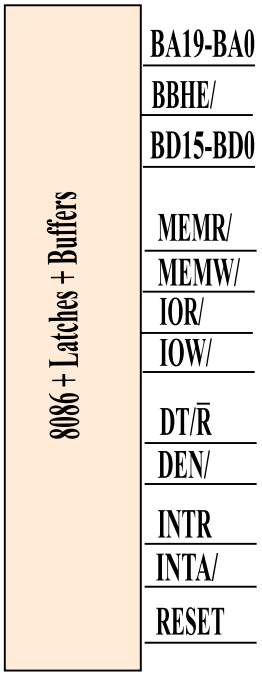
Example 1: 16-Bit Wide Memory

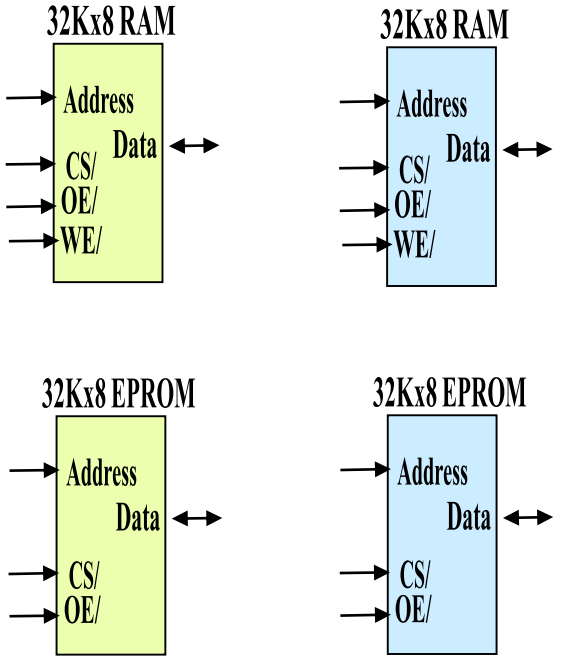
```
library ieee;
use ieee.std_logic_1164.all;
entity DECODER_10_28 is
port (
        A23, A22, A21, A20, A19, A18, A17, A16, A0, BHE, MWTC: in STD_LOGIC;
        SEL, LWR, HWR: out STD_LOGIC
);
end;
architecture V1 of DECODER_10_28 is
begin
        SEL <= A23 or A22 or A21 or A20 or A19 or (not A18) or (not A17) or A16;
        LWR <= A0 or MWTC;
        HWR <= BHE or MWTC;
end V1;
```

Example 2: Memory Interface to 8086

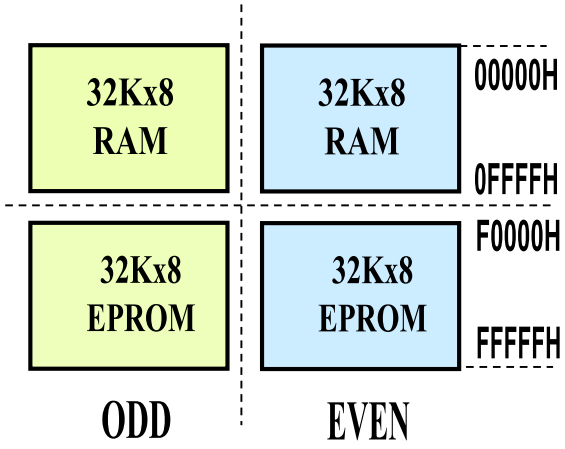
- We want to interface 64 KB RAM and 64KB EPROM chips to 8086 microprocessor according to the following assumptions:
  - The starting address of the RAM is 0H.
  - The last address of the EPROM is FFFFFH.
  - Exhaustive decoding is used.
  - The RAM and EPROM chips are organized as following:

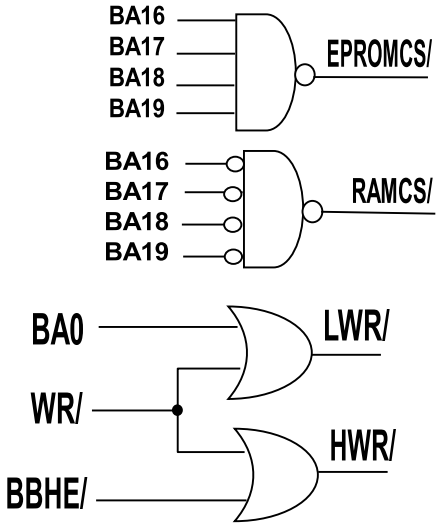


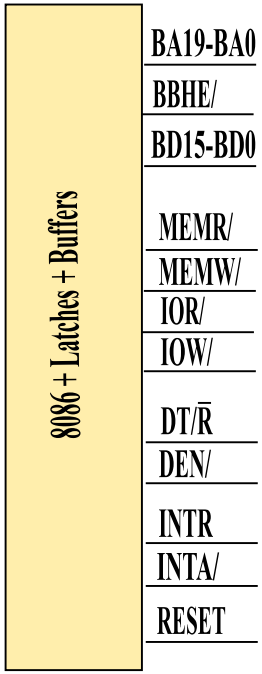


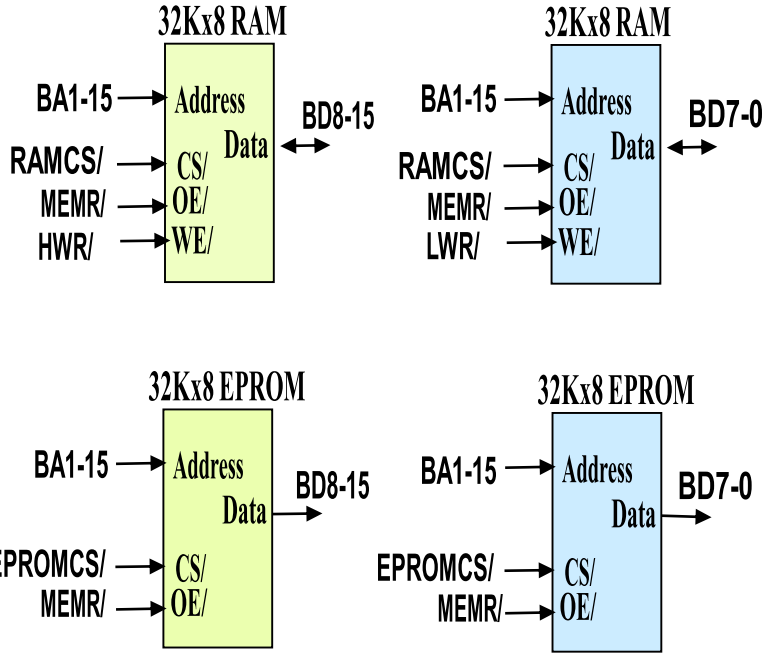


Example 2: Memory Interface to 8086



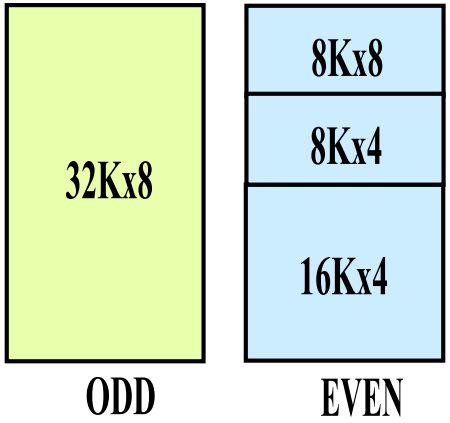


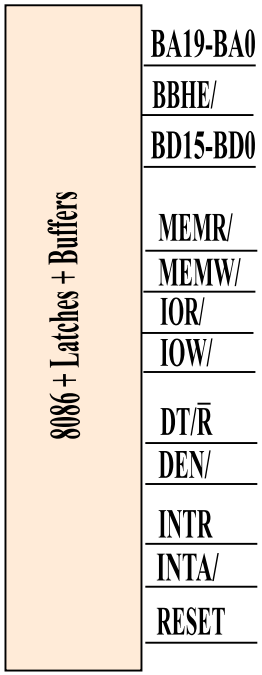


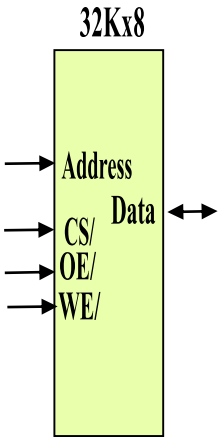


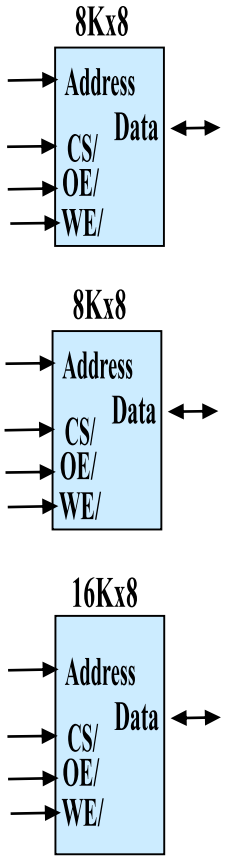
Example 3: Memory Interface to 8086

- We want to interface 64 KB RAM to 8086 microprocessor according to the following assumptions:
  - The starting address of 64 KB RAM in the memory space is 0H.
  - Exhaustive decoding is used.
  - The RAM chips are organized as following:









Example 3: Memory Interface to 8086