

Lecture 06: Memory Address Decoding

Reference Book:

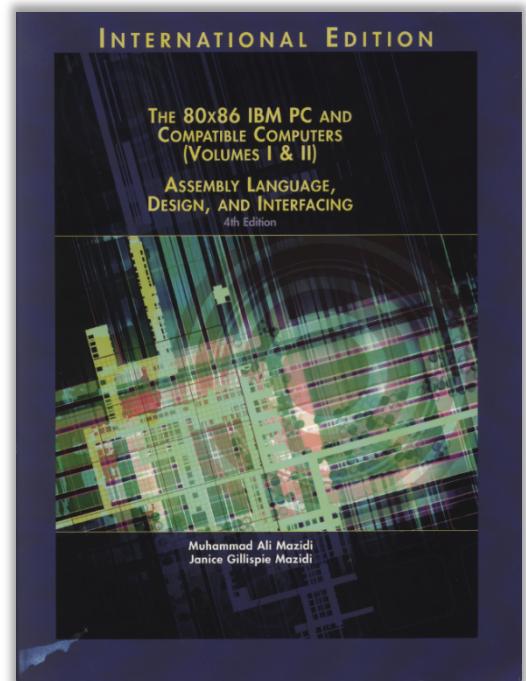
The 80x86 IBM PC and Compatible Computers

Chapter 10

**Memory and Memory
Interfacing**

Chapter 11

I/O and the 8255



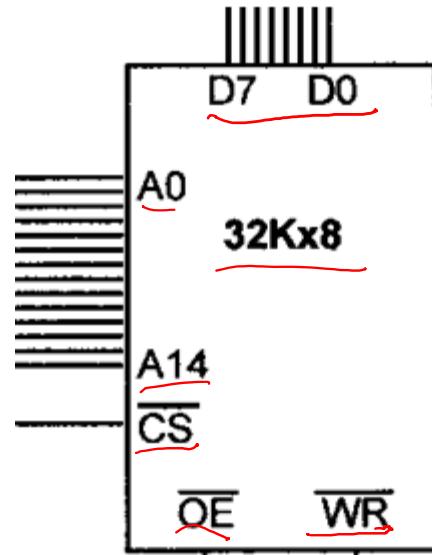
Review on Memory Chips

- Key concepts: capacity, organization (the number of locations X the size of addressable unit), access time

word size

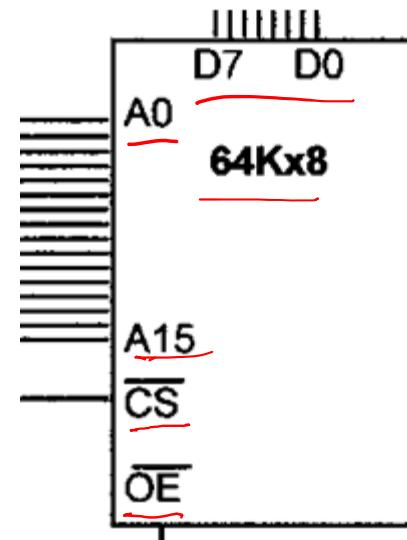
- Packaging

2^{15}



RAM chip

2^{16}



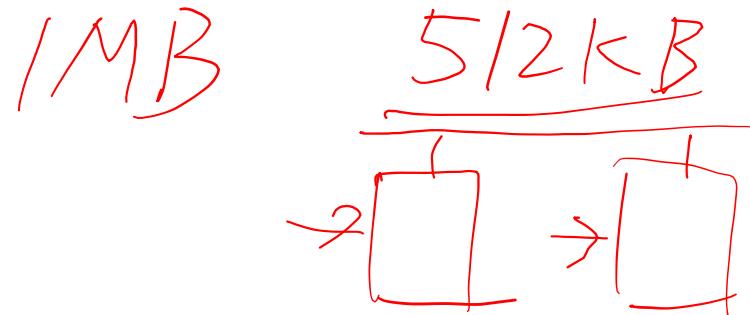
ROM chip

What to Learn?

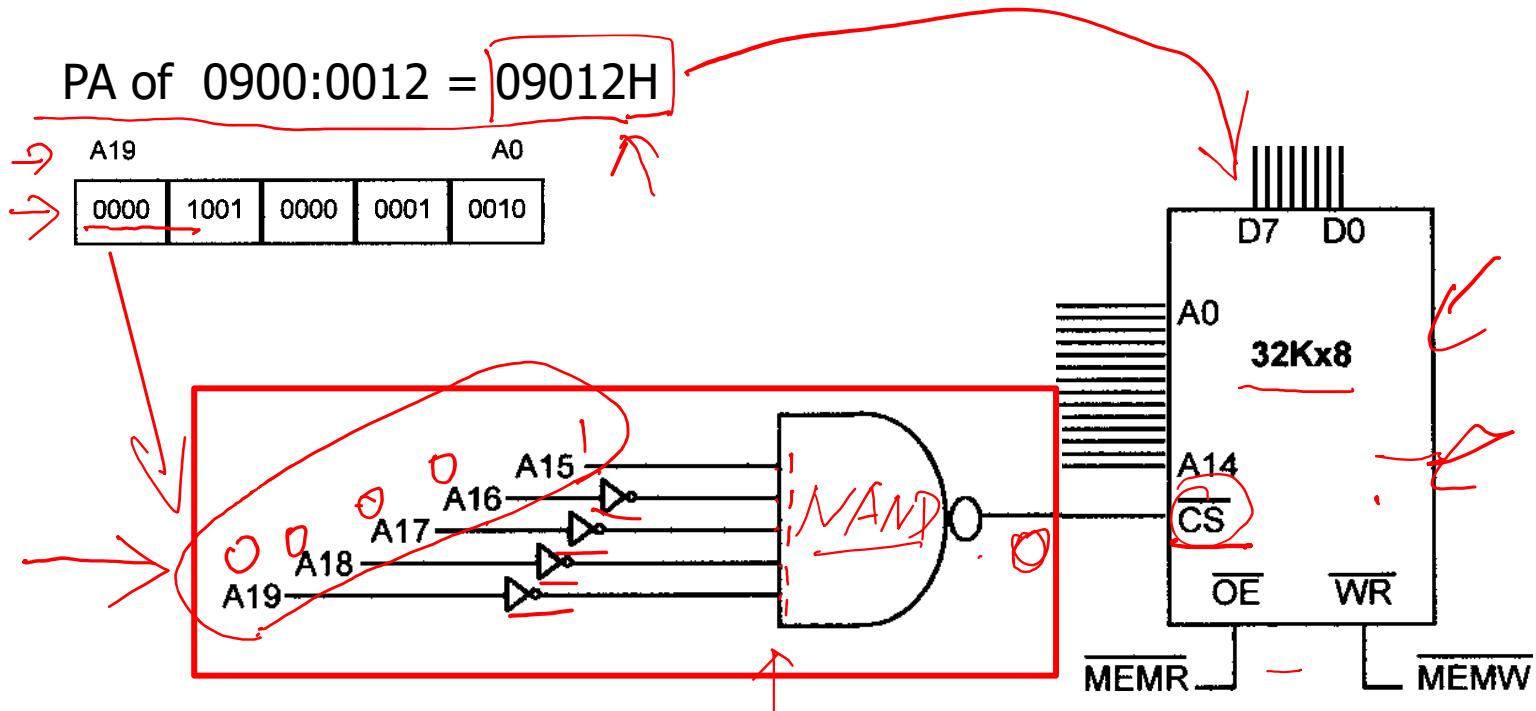
- How does the 8086 CPU execute an instruction like **MOV BX, [1000h]**?
- What is the difference between executing **MOV BX, [1000h]** and executing **MOV [1001h], BX** ?
- What is the difference between accessing memory and accessing I/O devices?

Memory Address Decoding

- According to **your instructions** that access memory
 - E.g., MOV AX, [0012H]
- **CPU** calculates the physical address of the operand and put corresponding signals on the address bus
 - E.g., if DS=0900H, what's the PA?
- **Memory address decoding circuitry** locates the specific memory chip that stores the desired data
 - Examine address decoding using logic gates and 74LS138 decoder chips



Memory Address Decoding

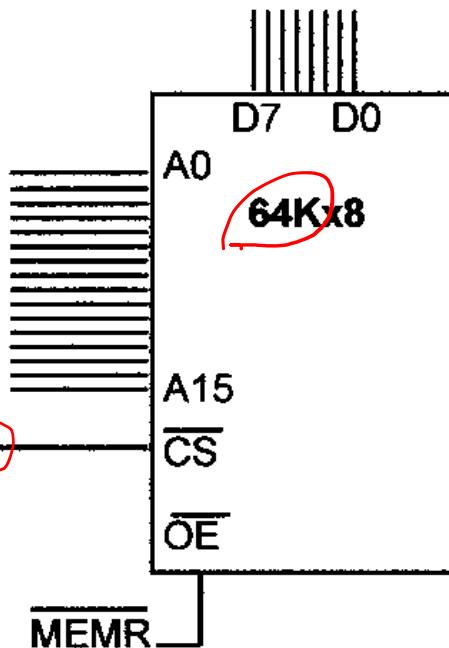
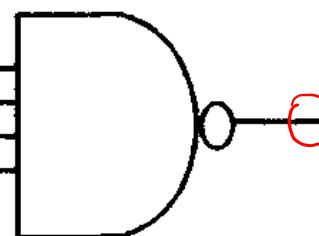


A19	A0					
0000	1000	0000	0000	0000		=08000H
0000	1111	1111	1111	1111		=FFFFH

Quiz

What's the type and the address range of this memory chip?

8000001H
A16 A17 A18 A19



8FFFF1H

ROM

$100 \rightarrow 4$

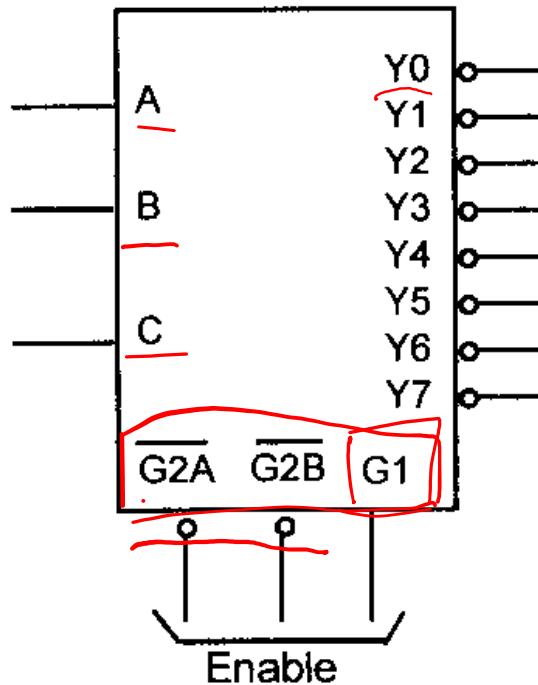
The 74LS138 Decoder Chip

$cBA \Rightarrow$

$3 \rightarrow 8$

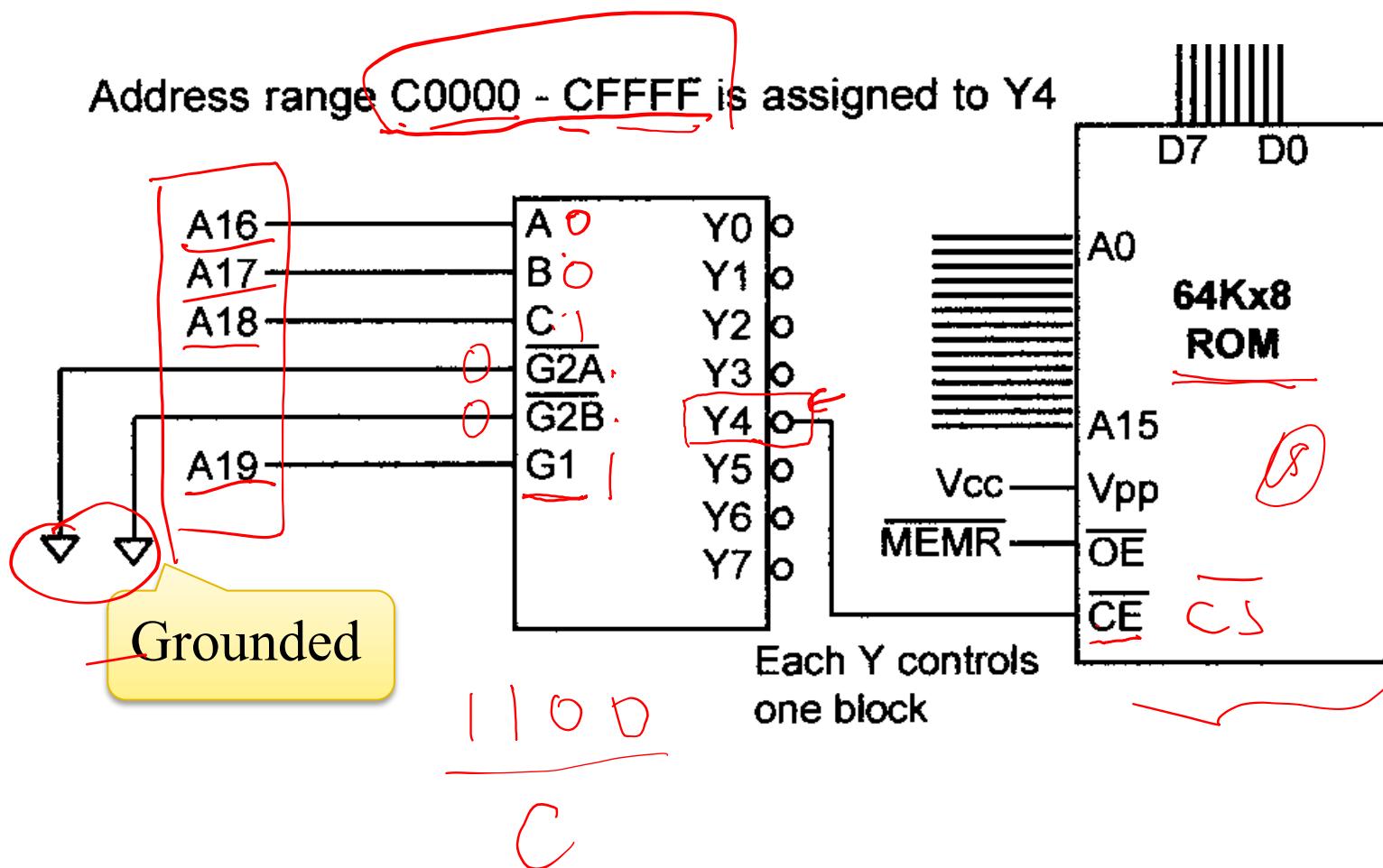
2^3

Function Table



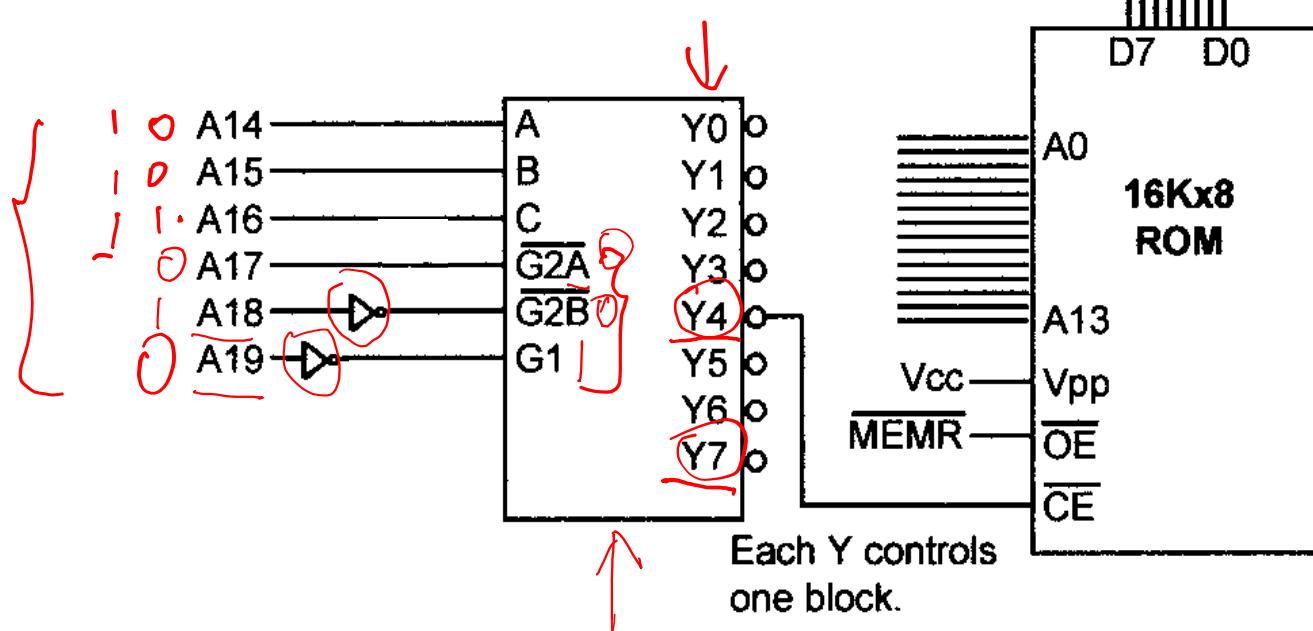
Inputs		Enable	Select	Outputs							
G1G2	CBA			Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H		XXX			H	H	H	H	H	H
L	X		XXX			H	H	H	H	H	H
H	L		LLL			L	H	H	H	H	H
H	L		LLH			H	L	H	H	H	H
H	L		LHL			H	H	L	H	H	H
H	L		LHH			H	H	H	L	H	H
H	L		HLL			H	H	H	H	L	H
H	L		HLH			H	H	H	H	L	H
H	L		HHL			H	H	H	H	H	L
H	L		HHH			H	H	H	H	H	H

Using 74LS138 to Decode



Quiz 2

What's the address range for Y4 and Y7?



More on Address Decoding

- Absolute address decoding 全译码
 - All address lines are decoded
- Linear select decoding 部分译码
 - Only selected lines are decoded
 - Cheap 易实现
 - But with aliases: the same memory unit (I/O port) with multiple addresses
 - *Why does this happen?*

001111111111
 ↙
 14 → 2¹⁴
 16KB

Example 1. In a particular computer, the address range from 0000h to 3FFFh is used for ROM, the range from 4000h to 5FFFh is reserved for future use, and the range from 6000h to 0FFFFh is used for RAM. Assume that the control signals for RAM are CS~ and WE~, and the CPU has 16 address pins (i.e., A15~A0), 8 data pins (i.e., D7~D0), and R/W~ and MREQ~ control signals.
Achieve the following requests.

↙ 16 → [64KB] →

- (1) draw the address decoding solution using a 74LS138 chip
- (2) if both ROM and RAM are built with $8K \times 1$ memory chips, try to draw the connection between the CPU and the memory.
- (3) if ROM is built with $8K \times 8$ memory chips and RAM is built with $4K \times 8$ chips, try to draw the connection between the CPU and the memory.
- (4) what if ROM is built with $16K \times 8$ memory chips and RAM is built with $8K \times 8$ memory chips?

(1) draw the address decoding solution using a 74LS138 chip

Solution.

The logic expression of each output of the 74LS138 chip:

$$\text{romsel0} = \overline{A_{15}} * \overline{A_{14}} * \overline{A_{13}} * \overline{\text{MREQ\#}}$$

$$\text{romsel1} = \overline{A_{15}} * \overline{A_{14}} * A_{13} * \overline{\text{MREQ\#}}$$

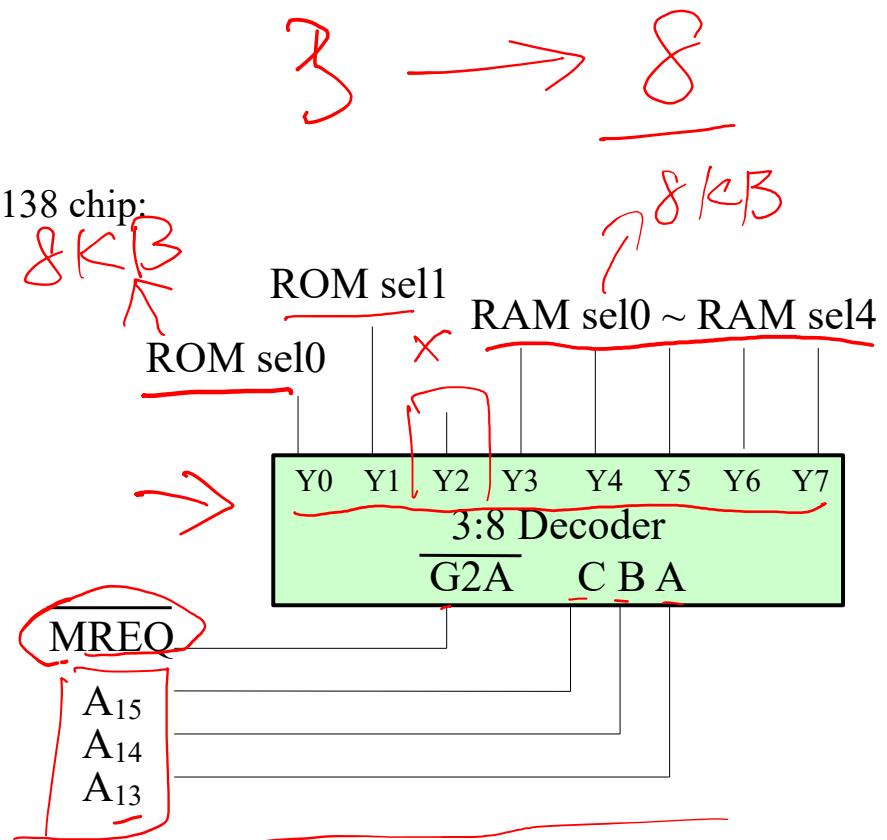
$$\text{ramsel0} = \overline{A_{15}} * A_{14} * A_{13} * \overline{\text{MREQ\#}}$$

$$\text{ramsel1} = A_{15} * \overline{A_{14}} * \overline{A_{13}} * \overline{\text{MREQ\#}}$$

$$\text{ramsel2} = A_{15} * \overline{A_{14}} * A_{13} * \overline{\text{MREQ\#}}$$

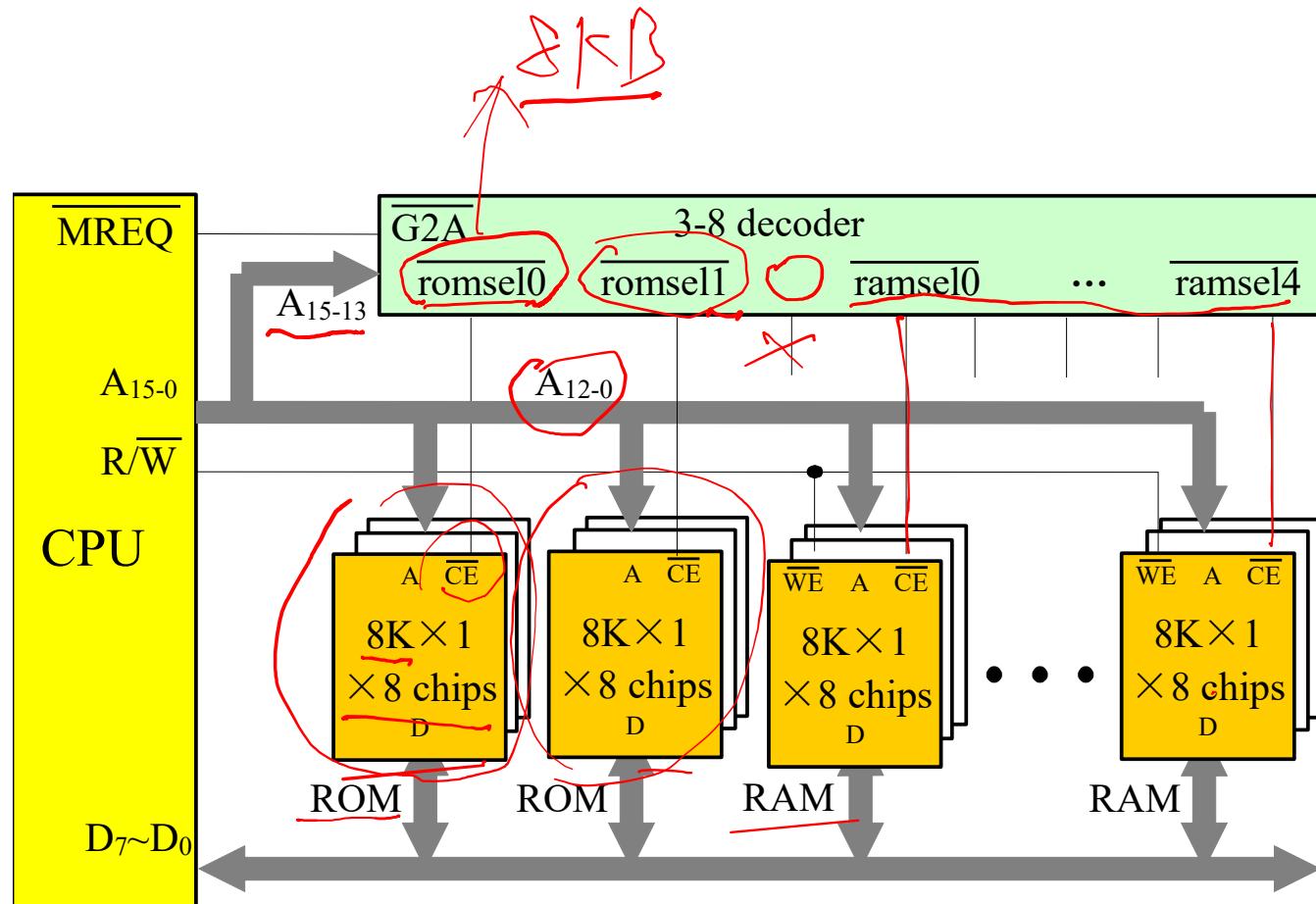
$$\text{ramsel3} = A_{15} * A_{14} * \overline{A_{13}} * \overline{\text{MREQ\#}}$$

$$\text{ramsel4} = A_{15} * A_{14} * A_{13} * \overline{\text{MREQ\#}}$$



(2) if both ROM and RAM are built with $8K \times 1$ memory chips, try to draw the connection between the CPU and the memory.

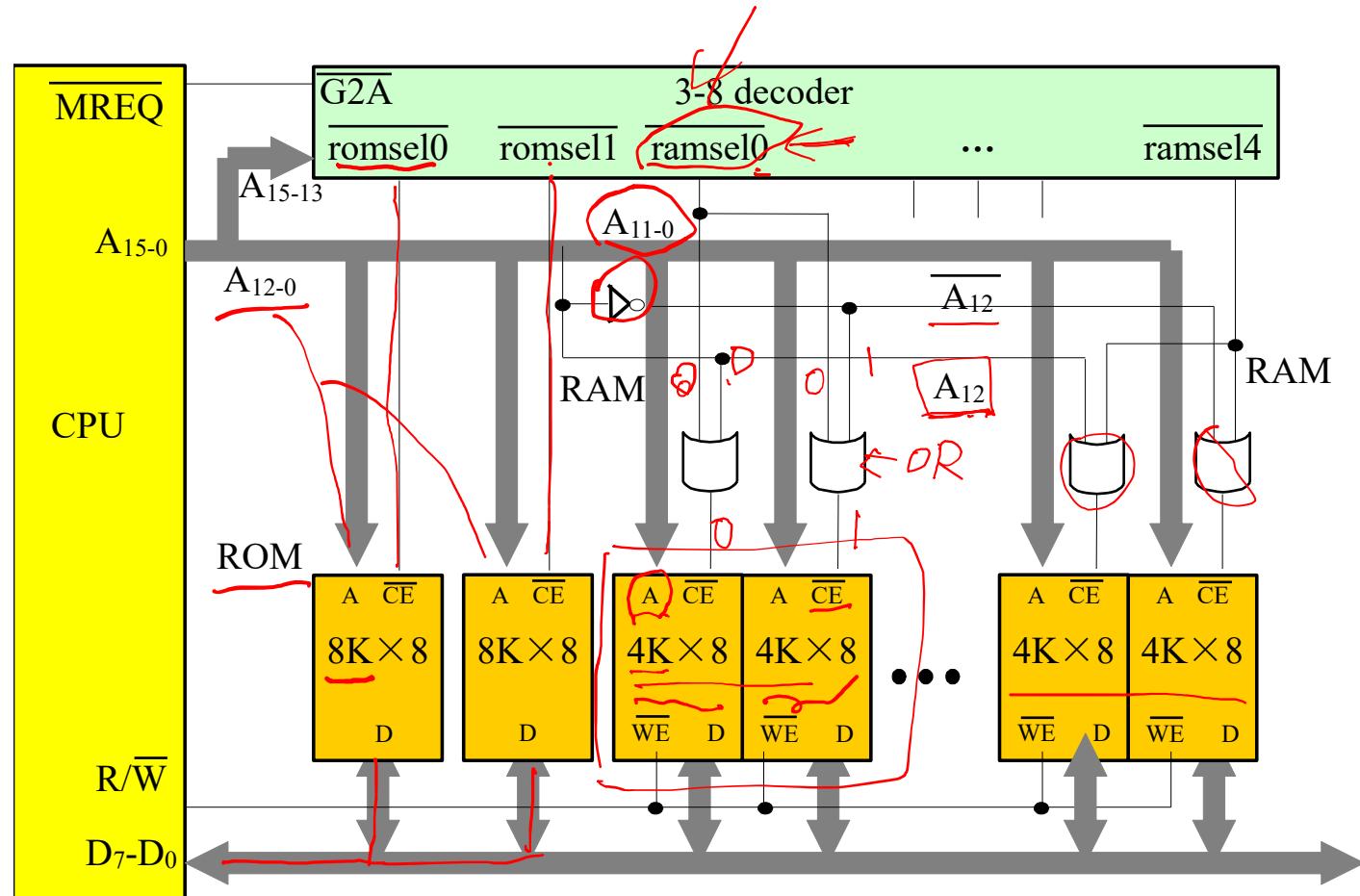
Solution.



(3) if ROM is built with $8K \times 8$ memory chips and RAM is built with $4K \times 8$ chips, try to draw the connection between the CPU and the memory.

$12 \rightarrow 4KB$

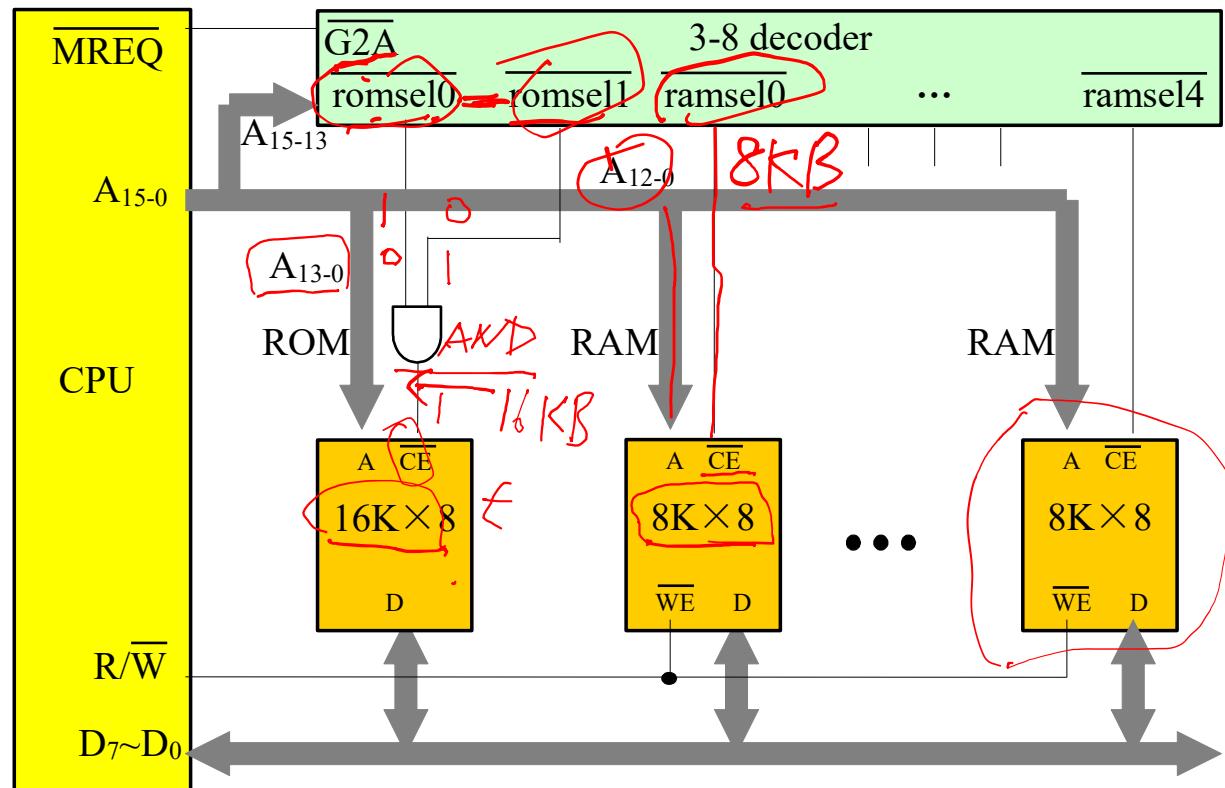
Solution.



(4) what if ROM is built with $16K \times 8$ memory chips and RAM is built with $8K \times 8$ memory chips?

$A_5 - A_0$

Solution.



Example 2. Assume one computer system needs 512 byte RAM and 512 byte ROM. If RAM is built with 128×8 memory chips and ROM is built with 512×8 memory chips, please specify the address range of each memory chip. Given that RAM chips need CS~ and WE~ control signals, ROM chips need only CS~ control signal, and the CPU has 16 address pins (A₁₅~A₀), 8 data pins (D₇~D₀) and R/W~ and MREQ~ control signals, draw the connection between the CPU and the memory.

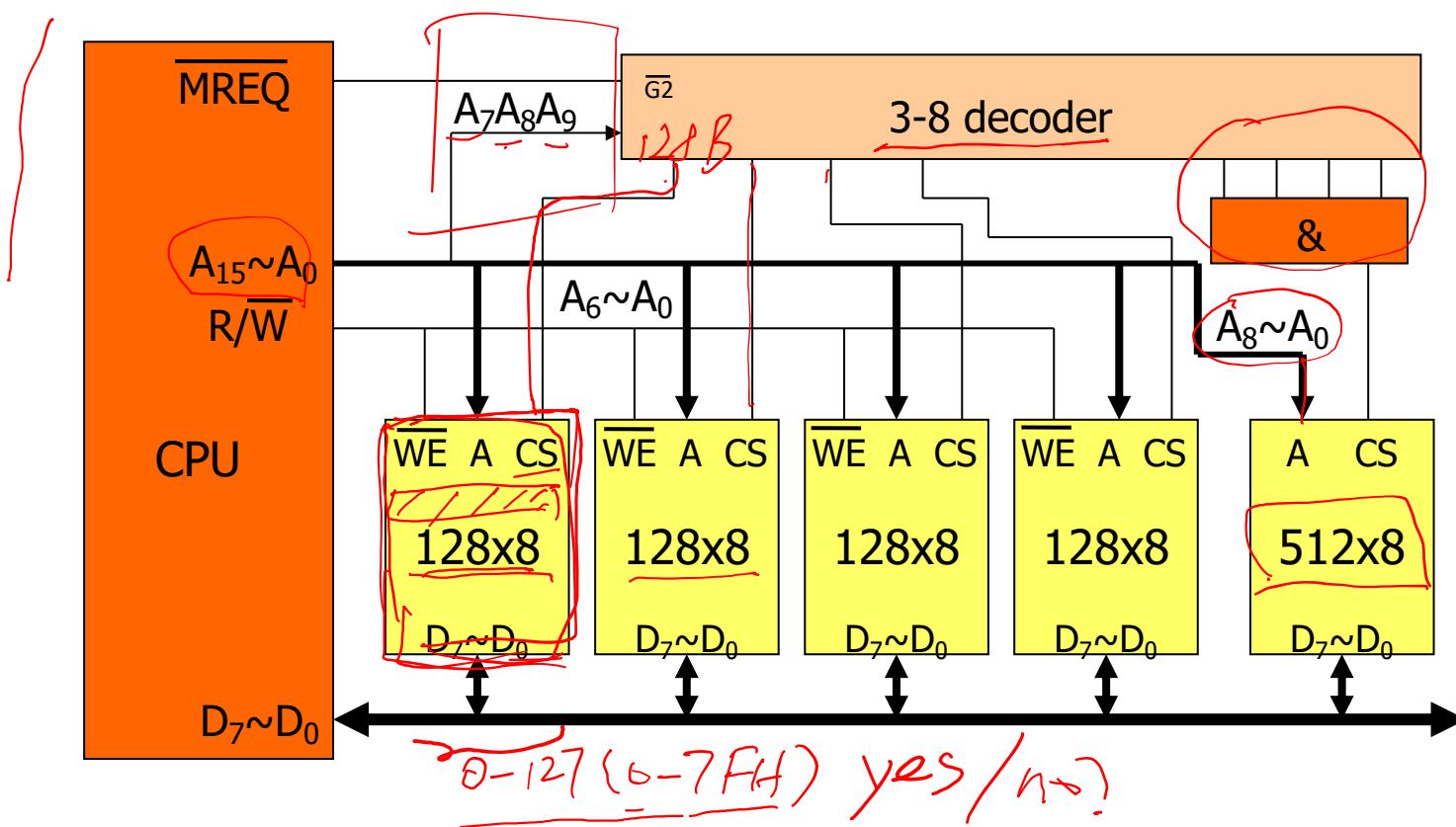
Solution. The address range of each memory chip:

Memory Chip	Address range (hex)	binary
RAM1	0000~007F	0 0 0 x x x x x x x x
RAM2	0080~00FF	0 0 1 x x x x x x x x
RAM3	0100~017F	0 1 0 x x x x x x x x
RAM4	0180~01FF	0 1 1 x x x x x x x x
ROM	0200~03FF	1 x x x x x x x x x x

$\{ \begin{array}{l} A_{15} - A_{10} \\ \hline 00 \quad 0000 \\ 00 \quad 0001 \end{array} \} \leftarrow 0 \leftarrow$

~~A_9~~ $\{ \begin{array}{l} A_{15} - A_{10} \\ \hline A_9 - A_0 \end{array} \}$

As the total volume of the memory is 1K, we need 10 address lines.
 RAM chips need 7 address lines and ROM chips need 9 address lines.

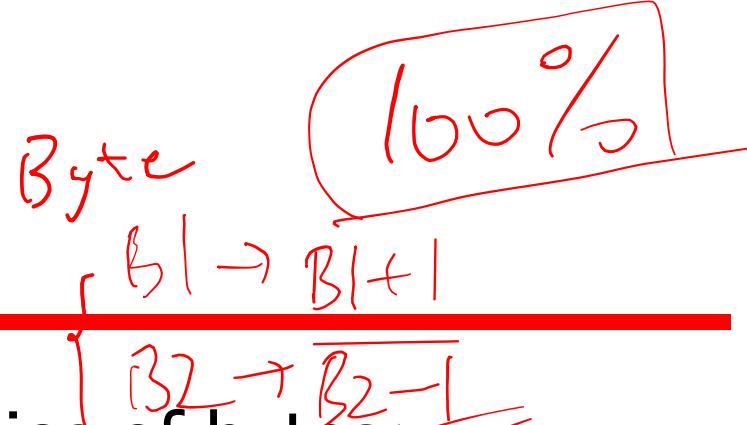


Data Integrity

数据完整性

- Checksum byte for ROM
- Parity bit for DRAM
- CRC for disks and the Internet

Checksum Byte



- Check the integrity of a series of bytes

- Calculation

- Add all bytes together and drop all carries → SUM ←
- Take the 2's complement of the sum → -SUM

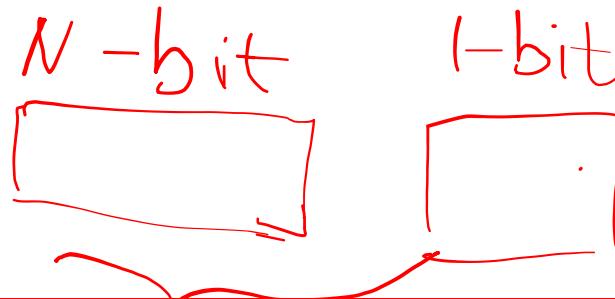
- Store the checksum byte together with data
- check the integrity by adding data and the checksum together

- Then how to prove the integrity of the data?

- E.g., 38H, 23H, 33H, 07H, what is the checksum byte?



Parity bit



- Check the integrity of a series of bits (a byte)
 - even parity: if the number of 1s in the series of bits is odd, then the parity bit is set to 1; otherwise, set to 0, making the total number of 1s even (Data + the parity bit)
 - odd parity: if the number of 1s in the series of bits is odd, then the parity bit is set to 0; otherwise, set to 1
 - For PF in 8086, odd parity is used, i.e., if data have even number of 1s, the parity bit is set

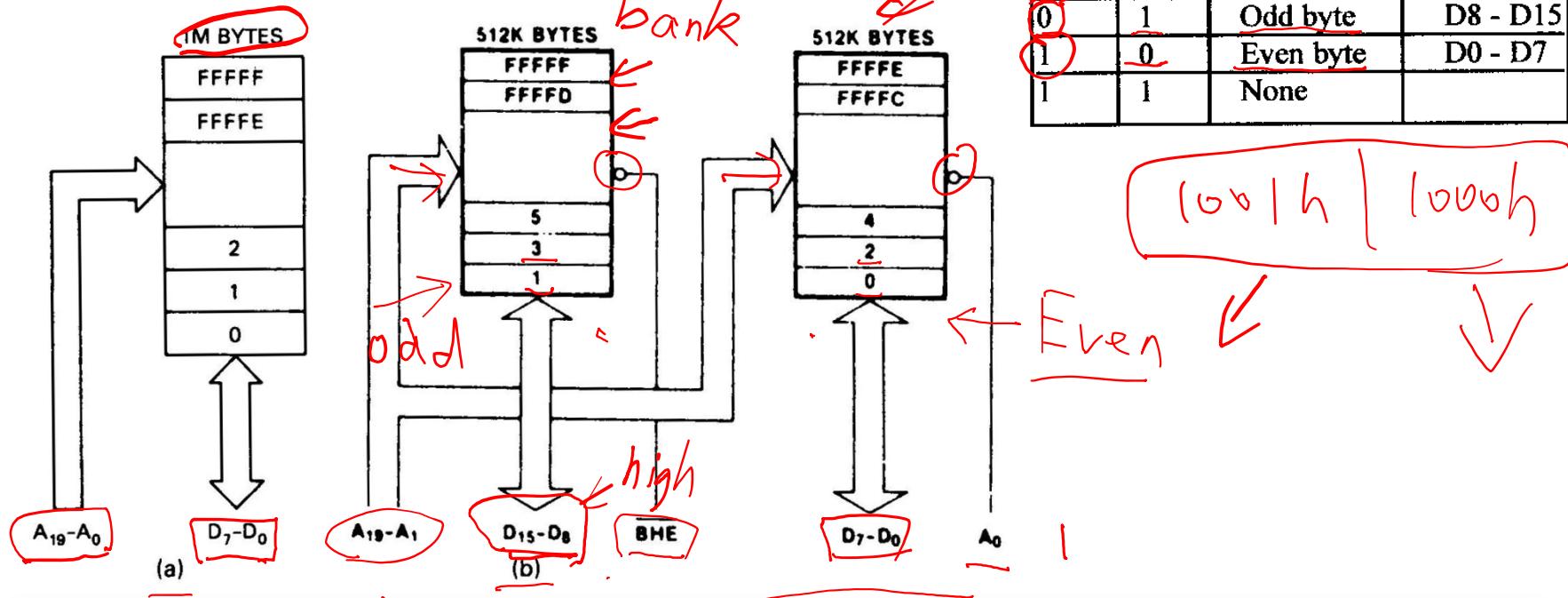
Size N | B
Addr E | O

$\Rightarrow \text{MOV BX, [100h]}$ | MOV BL, [100h]

Memory Organization in 8086

100h

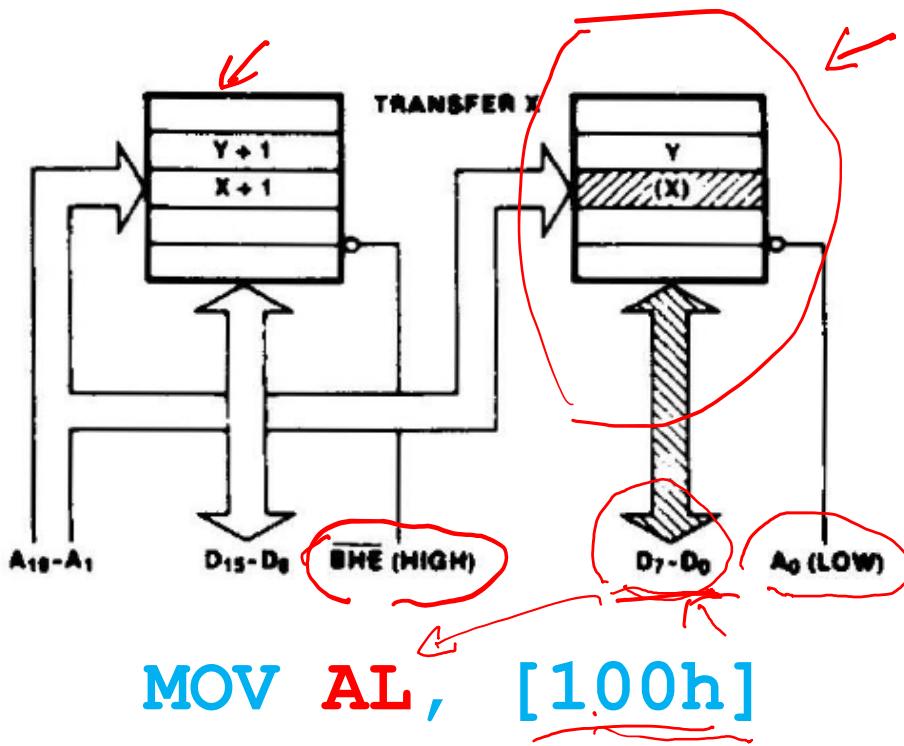
Even and odd banks



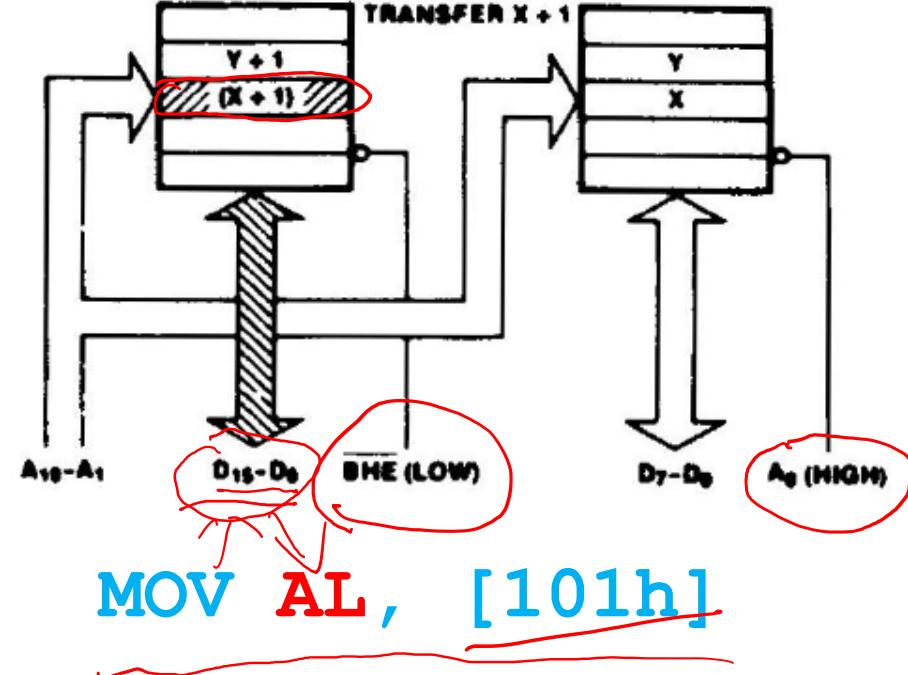
Address bits A₁ through A₁₉ select the storage location that is to be accessed. They are applied to both banks in parallel. A₀ and bank high enable (BHE) are used as **bank-select** signals.

Byte-Memory Operations

- Byte-memory operation at even address X

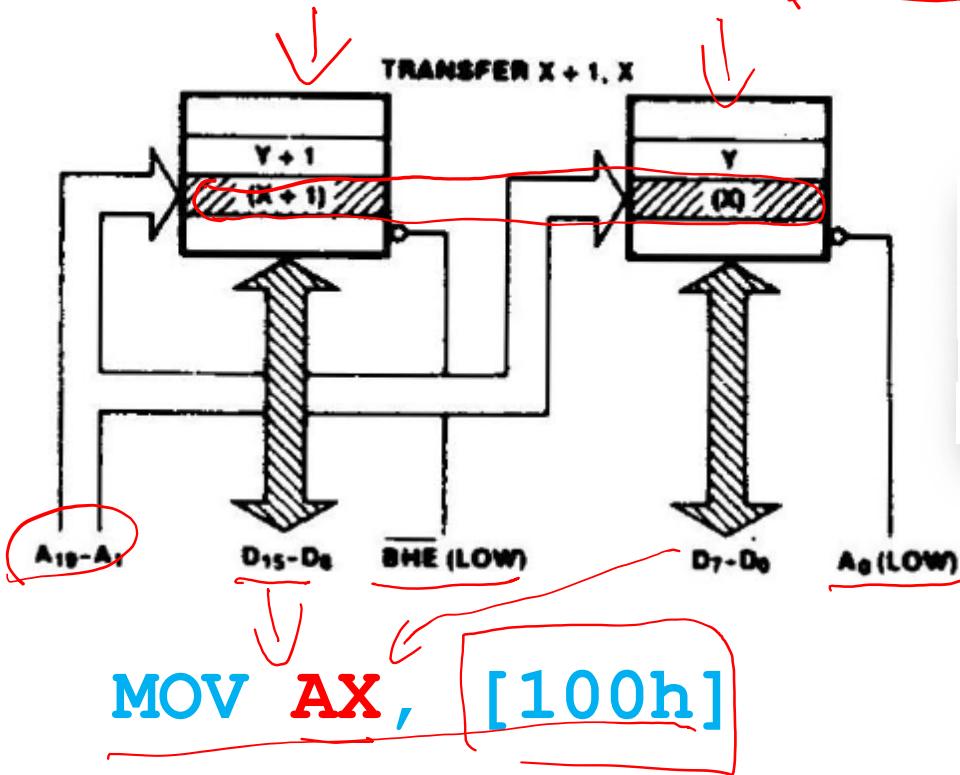


- Byte-memory operation at odd address $X+1$



Aligned Word-Memory Operations

- Accessing an **aligned word** at even address X

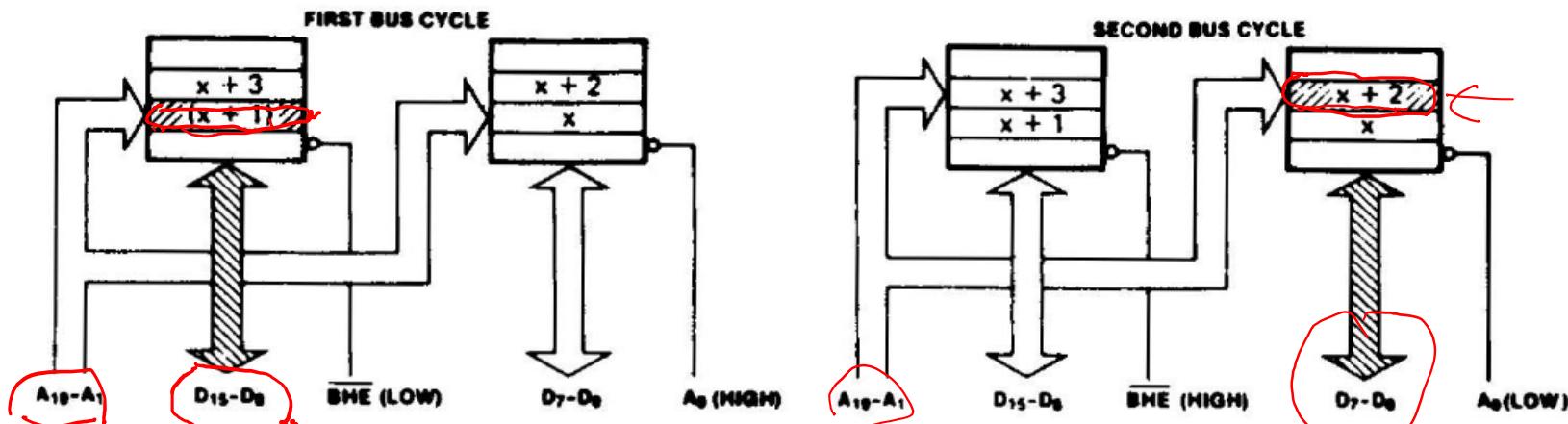


Both the high and low **banks** are accessed at the **same time**. Both A_0 and **BHE** are set to 0. This 16-bit word is transferred over the complete data bus **D₀** through **D₁₅** in just **one bus cycle**.

.ORG →

Misaligned Word-Memory Operations

Accessing an misaligned word at odd address X+1



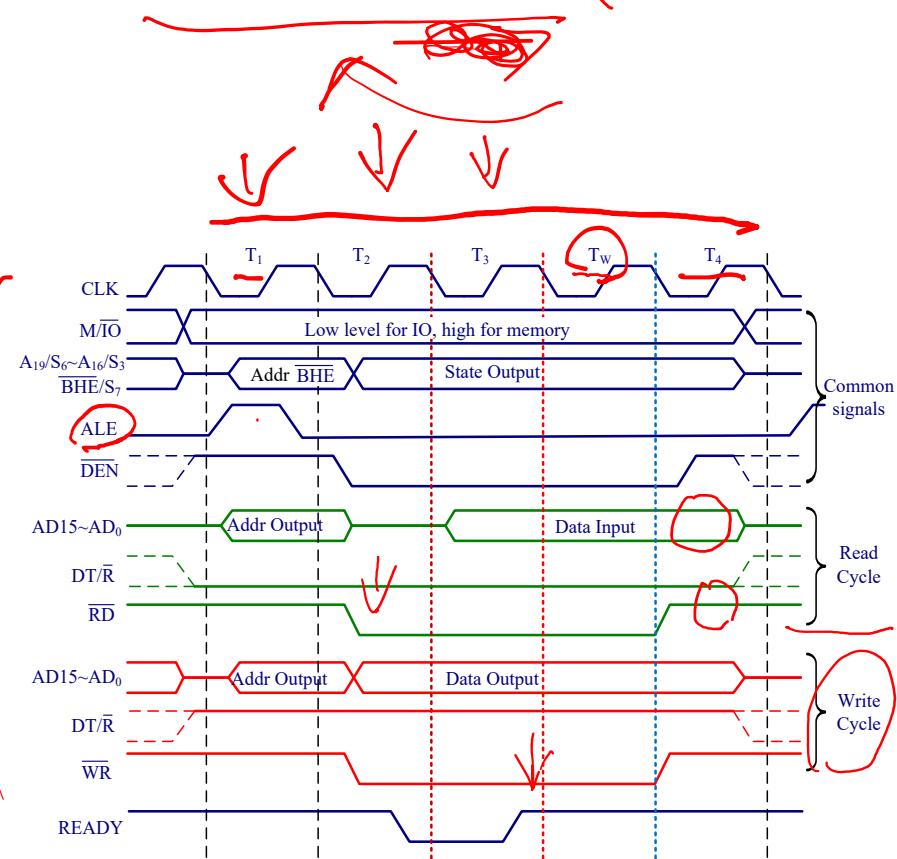
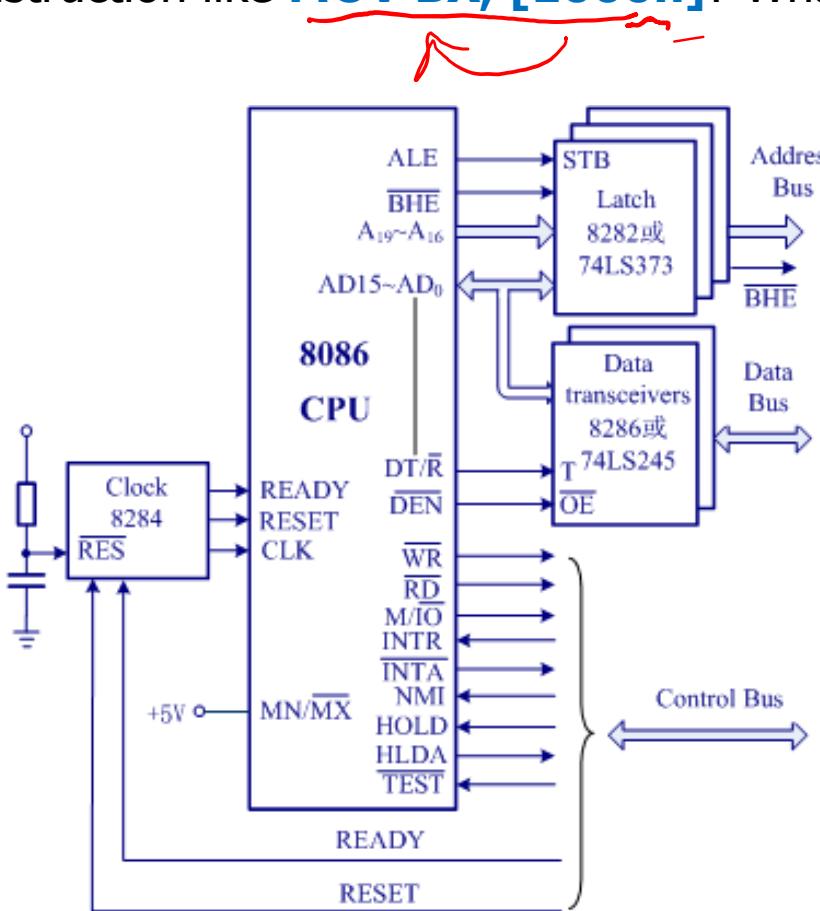
Two bus cycles are needed. During the first bus cycle, the byte of the word located at address $X + 1$ in the high bank is accessed over D_8 through D_{15} . Even though the data transfer uses data lines D_8 through D_{15} , to the processor it is the low byte of the addressed data word. In the second memory bus cycle, the even byte located at $X + 2$ in the low bank is accessed over bus lines D_0 through D_7 .

MOV AX, [101h] → 102h

$A_{19}-A_1$ A_0

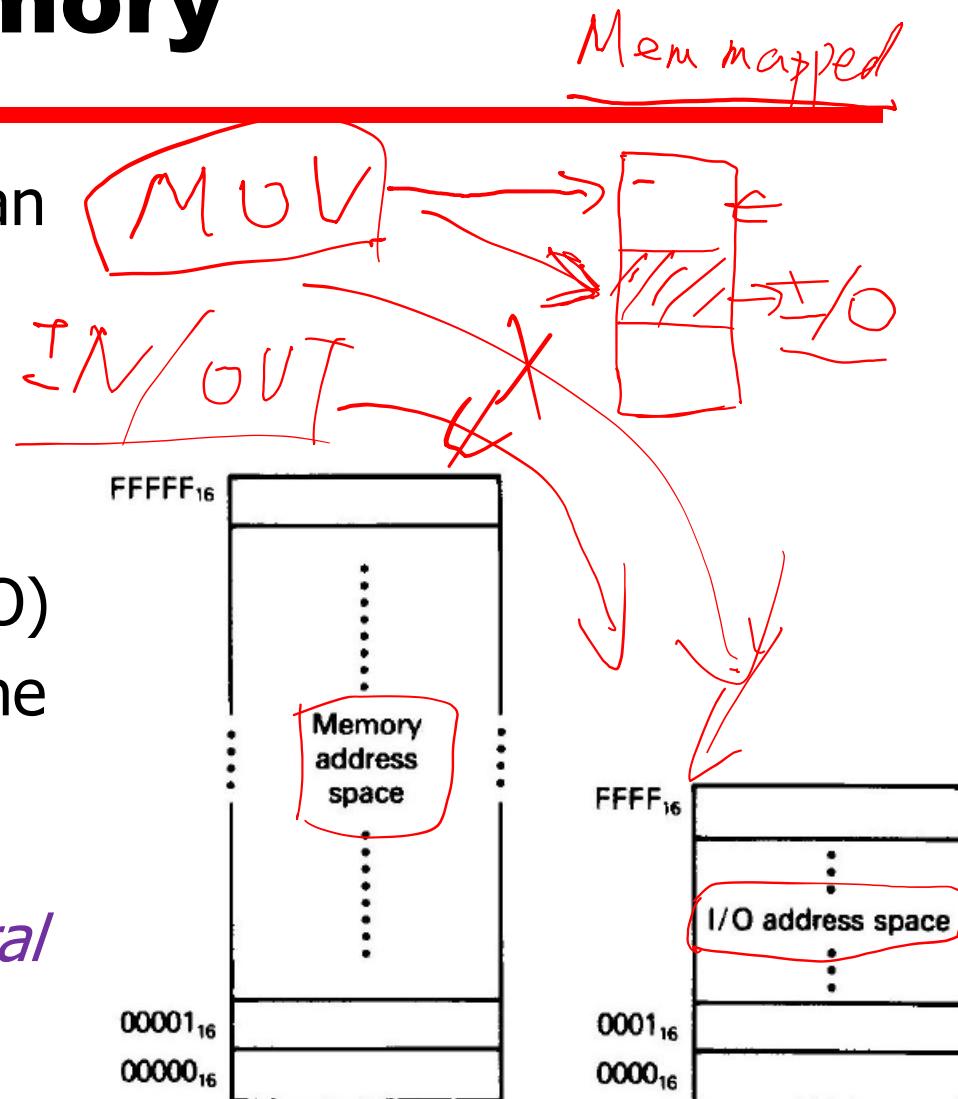
Let's Tell the Stories

Could you possibly tell the complete procedure of the 8086 CPU executing an instruction like **MOV BX, [1000h]**? What about **MOV [1001h], BX**? 



I/O in X86 family – The Other Space from Memory

- X86 microprocessors have an I/O space in addition to memory space
- Use special I/O instructions accessing I/O devices at **ports** (i.e., addresses for I/O)
- Memory can contain machine codes and data, I/O ports only contain data
- Also referred to as *peripheral I/O* or *isolated I/O*



I/O Instructions – 8-Bit Instance

Format: Inputting Data • Outputting Data
IN dest,source OUT dest,source

■ Direct I/O instructions:

- port# ranges from 00h to 0ffh, 256 ports in total

(1) IN AL, port# OUT port#, AL

■ Indirect I/O instructions:

- port# ranges from 0000h to 0ffffh, 65536 ports in all
- use a 16-bit address that resides in the DX register

(2) → MOV IN DX, port# AL, DX MOV OUT DX, port# DX, AL

- ## ■ Note: no segment concept for port addresses

while //

/ read port

I/O Example

In a given 8088-based system, port address 22H is an input port for monitoring the temperature. Write Assembly language instructions to monitor that port continuously for the temperature of 100 degrees. If it reaches 100, then BH should contain 'Y'

Solution:

```
BACK:  
      IN    AL, 22H  
      CMP   AL, 100  
      JNZ   BACK  
      MOV   BH, 'Y'
```

I/O Instructions – 16-Bit Instance

- | For 16-bit I/O modules

- | Direct I/O instructions:

- | port# ranges from 00h to 0ffh, 256 ports in total

IN AX, port#

OUT port#, AX

- | Indirect I/O instructions:

- | port# ranges from 0000h to 0ffffh, 65536 ports in all

- | use a 16-bit address that resides in the DX register

MOV DX, port#

IN AX, DX

MOV DX, port#

OUT DX, AX

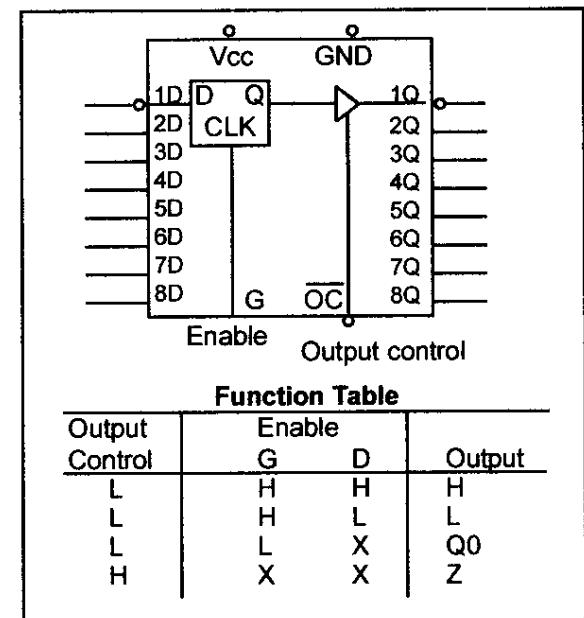
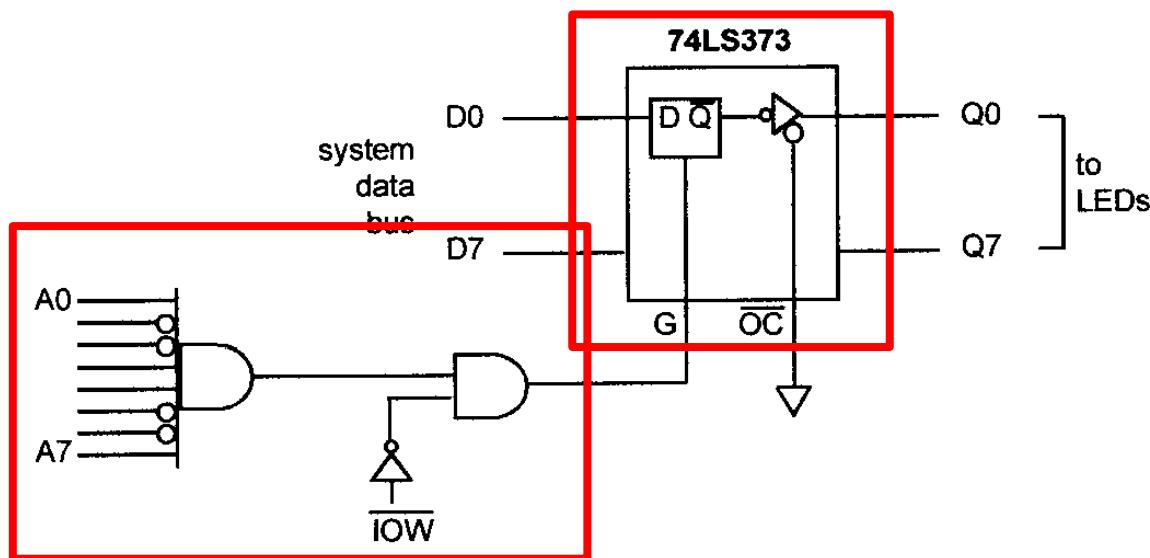
Interfacing 8-bit I/O Modules to a 16-bit Data Bus

- For 8086, data for even-address ports are carried on data bus D0-D7 and data for odd-address ports are carried on data bus D8-D15

<u>BHE</u>	<u>A0</u>	
0	0	Even-addressed words (uses D0 - D15)
0	1	Odd-addressed byte (uses D8 - D15)
1	0	Even-addressed byte (uses D0 - D7)

Output Port Design

- Latch the data coming from the CPU
- Address decoding

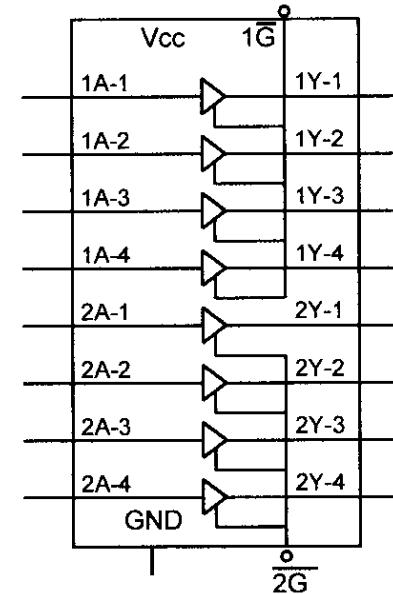
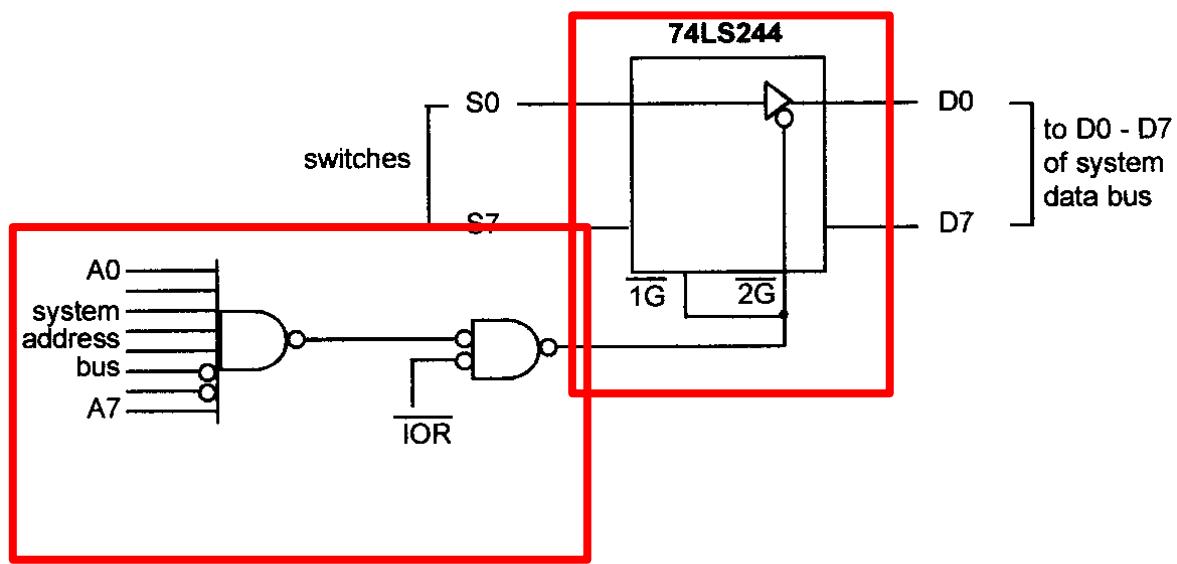


74LS373 D Latch

What is the address of this port?

Input Port Design

- Use *tri-state buffer* to connect to system data bus
- Address decoding



What is the address of this port?