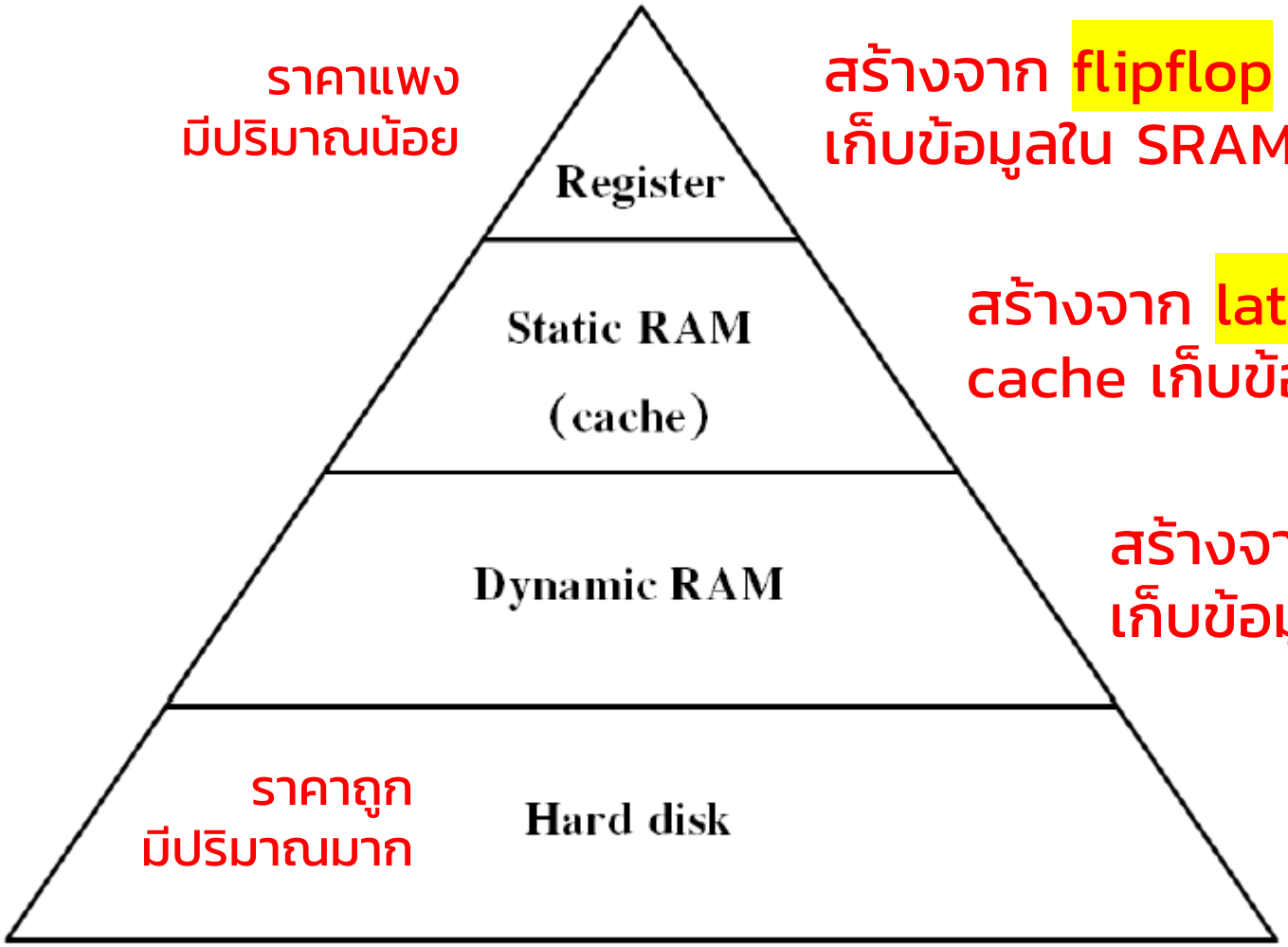
แก้ไขครั้งล่าสุดเมื่อวันที่ 19 กันยายน 2568

Chapter 7

Memory and Programmable Logic



สร้างจาก flipflop อยู่ใน microprocessor
เก็บข้อมูลใน SRAM ที่ใช้บ่อย ๆ

สร้างจาก latch ใช้ทำ cache ใน microprocessor
cache เก็บข้อมูลใน DRAM ที่ถูกใช้บ่อย ๆ

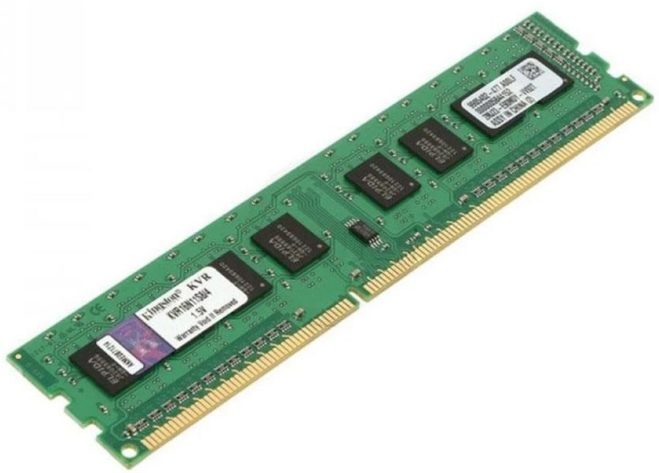
สร้างจาก capacitor (charged/discharged)
เก็บข้อมูลใน hard disk ที่ใช้บ่อย ๆ

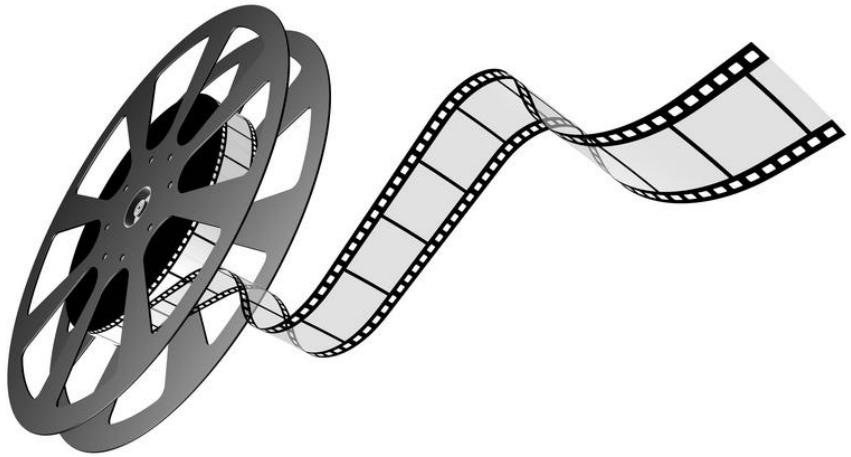
เก็บข้อมูลแบบ non-volatile
magnetic disk vs. solid-state disk
(แยกไปสอนในวิชา operating systems)

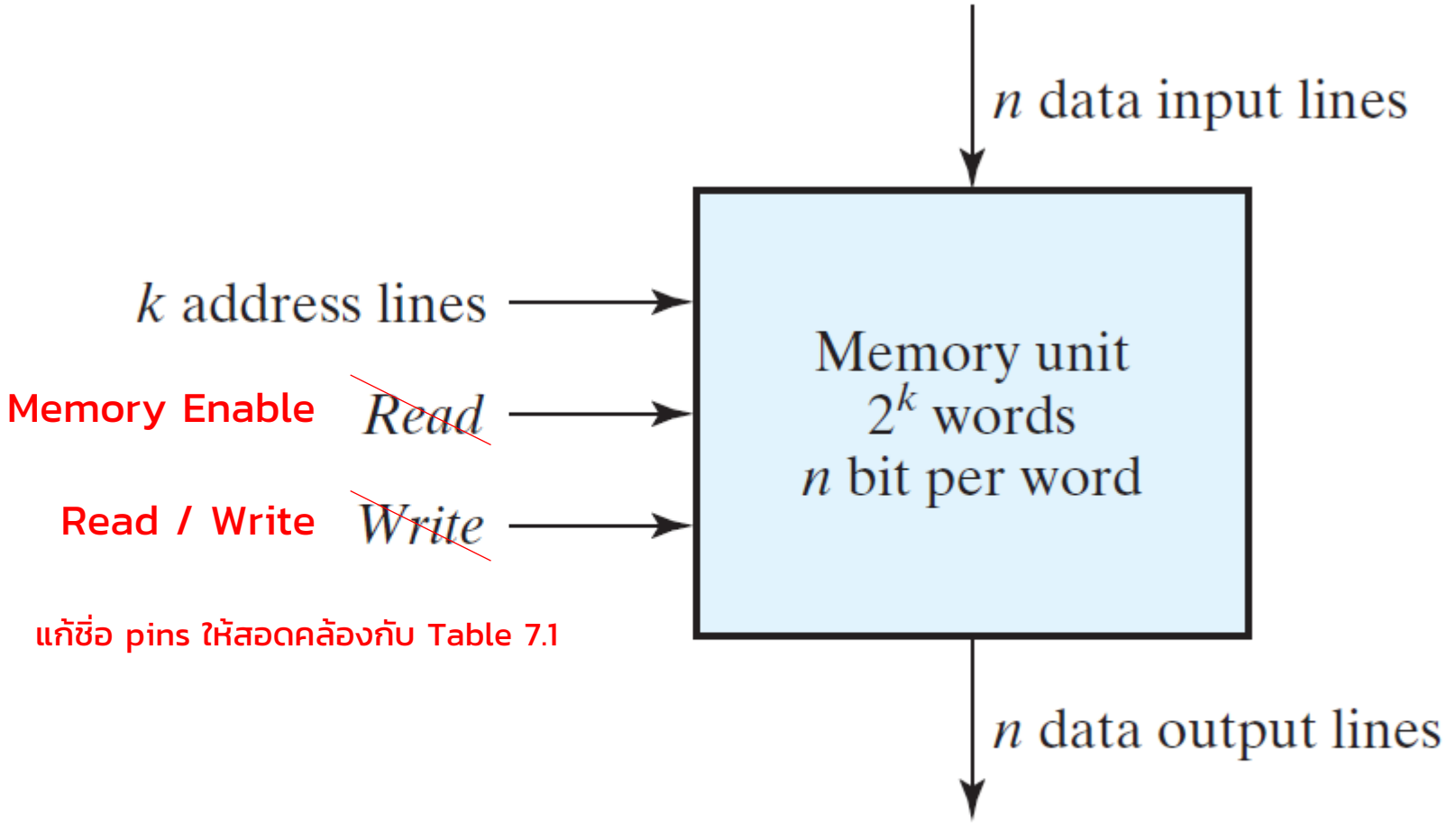
รูปที่ 7.1 ลำดับชั้นของหน่วยความจำ

Memory Hierarchy

Random vs. Non-random Access







แก้ชื่อ pins ให้สอดคล้องกับ Table 7.1

อาจจะยุบ input/output ports
รวมกันเป็น bi-directional ports

FIGURE 7.2
Block diagram of a memory unit

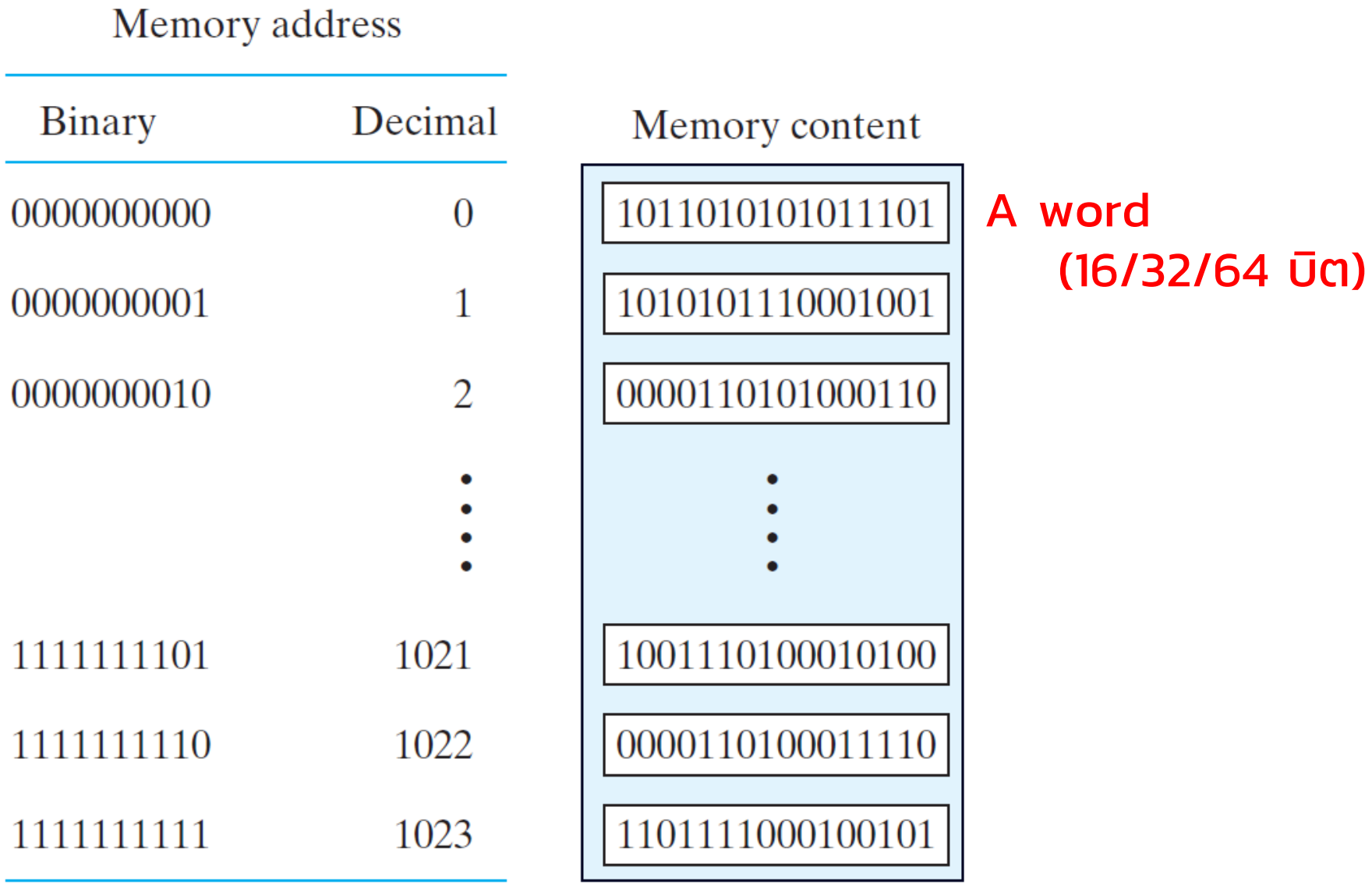


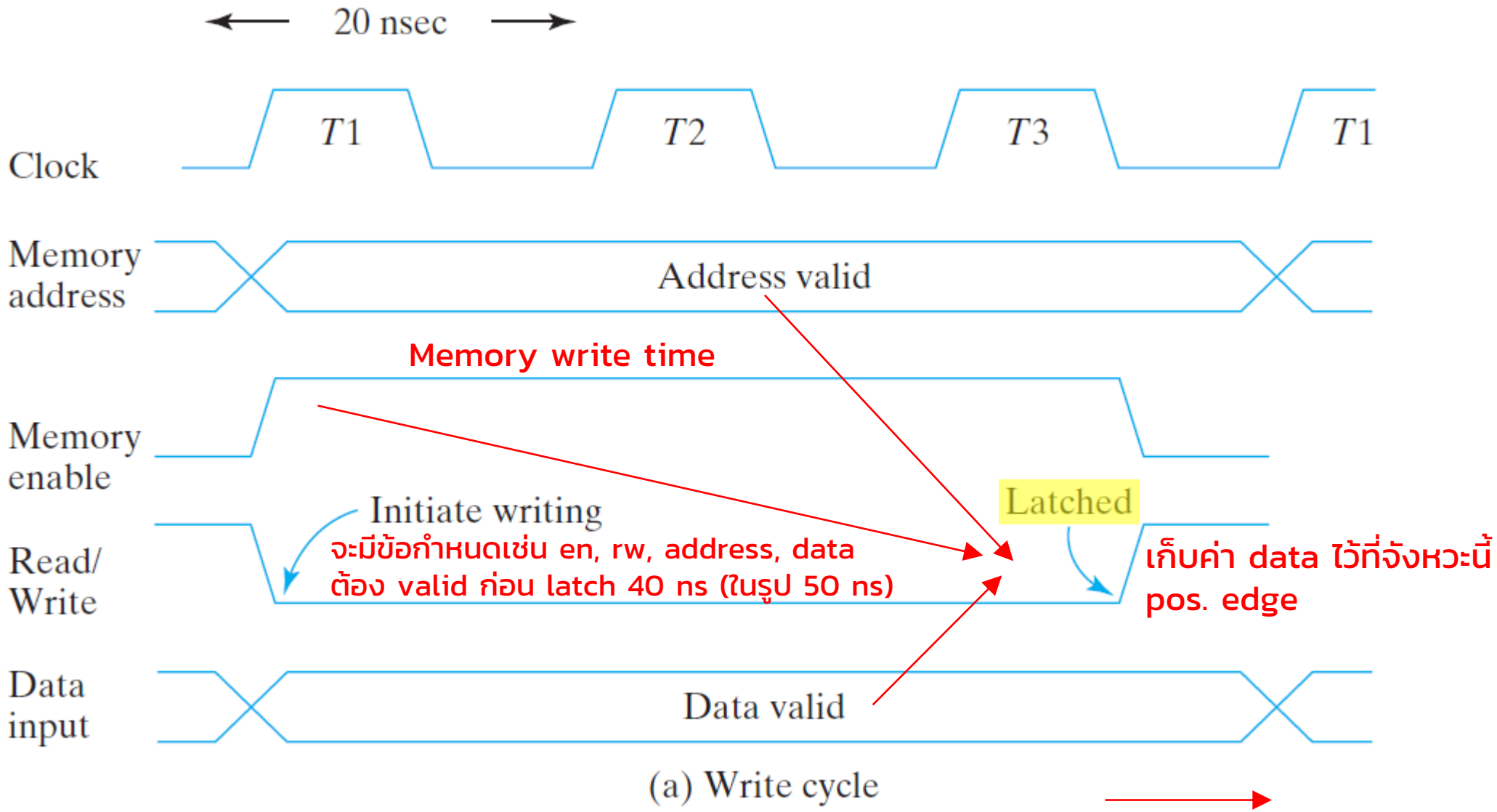
FIGURE 7.3
Contents of a 1024 × 16 memory

In computing, a **word** is the natural unit of data used by a particular processor design. A word is a fixed-sized piece of data handled as a unit by the instruction set or the hardware of the processor.

Table 7.1
Control Inputs to Memory Chip

Memory Enable	Read/Write	Memory Operation
0	X	None
1	0 ↑ ทำให้เป็น 0 ก่อน เพื่อสร้าง positive edge	Write to selected word
1	1	Read from selected word

memory ไม่ได้ต้องการ clock
แต่เราใช้ clock เพื่อสร้างสัญญาณควบคุม



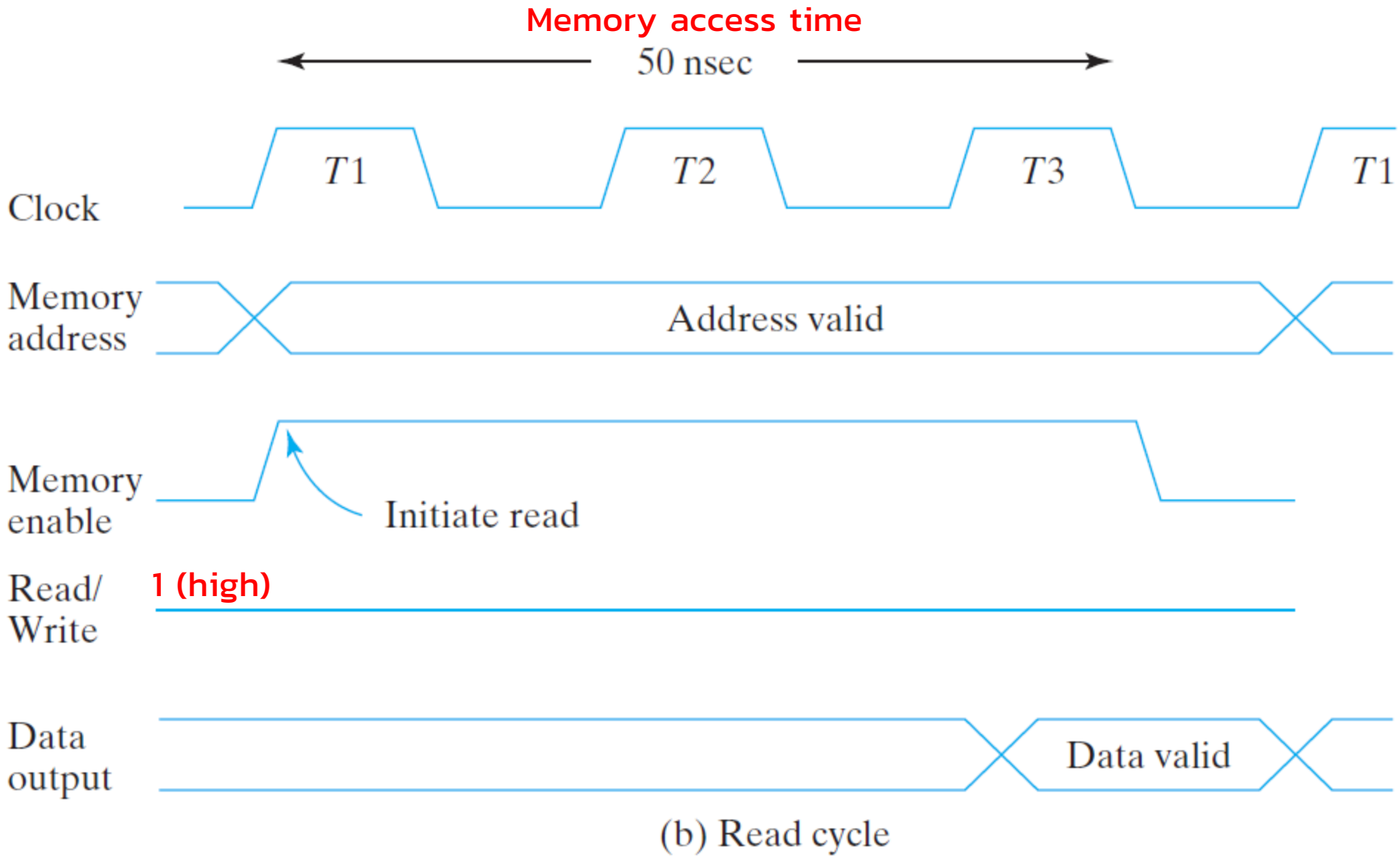
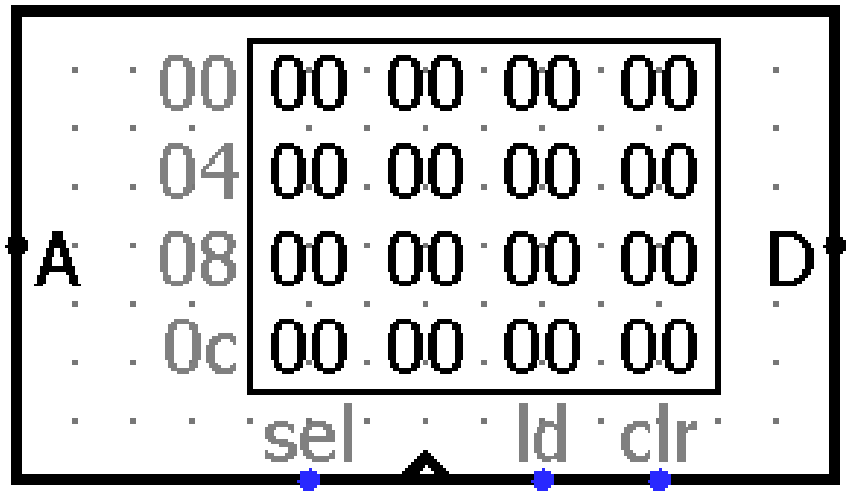


FIGURE 7.4
Memory cycle timing waveforms

Logisim

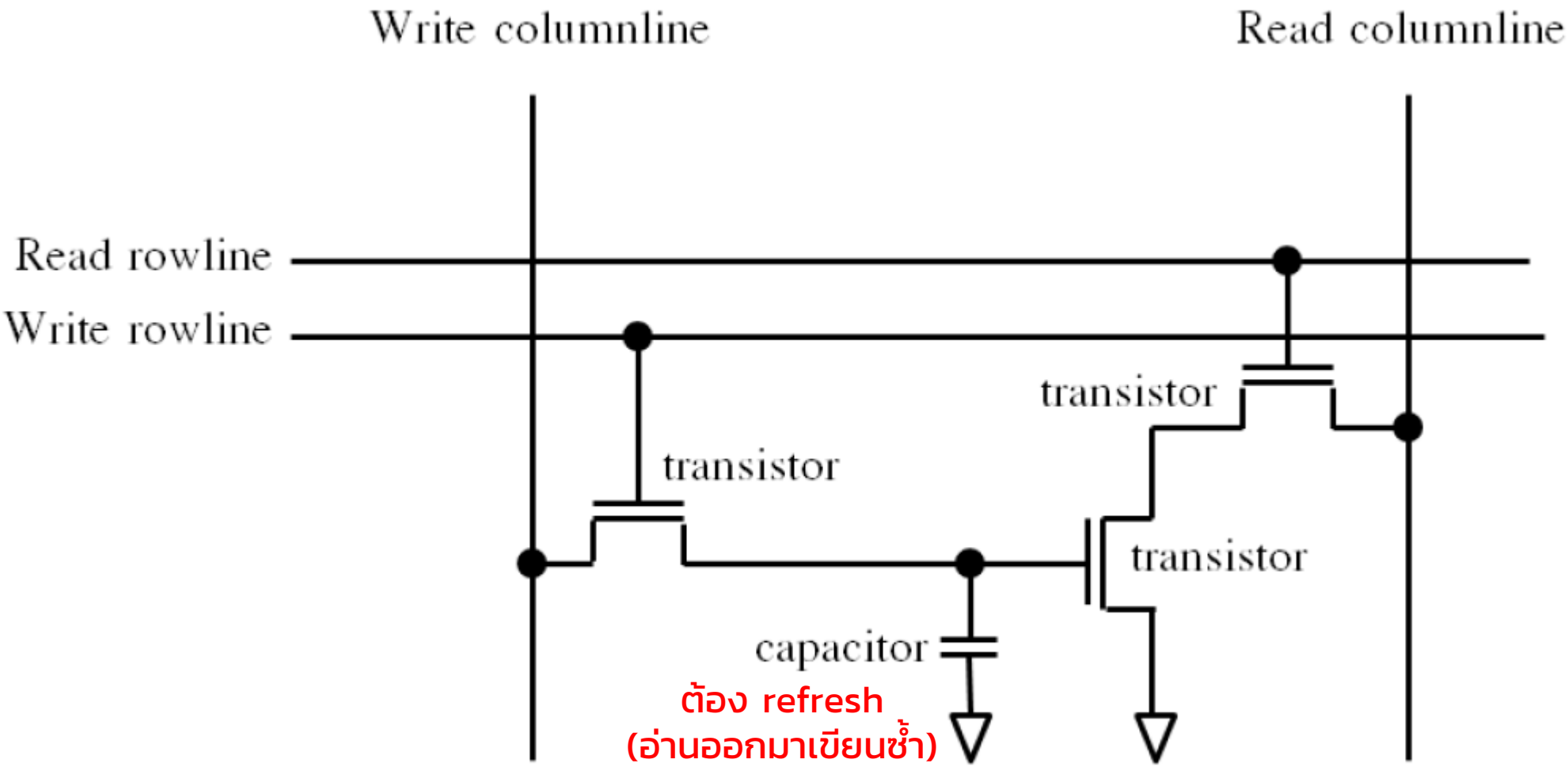
Address



Data
(bi-directional port)

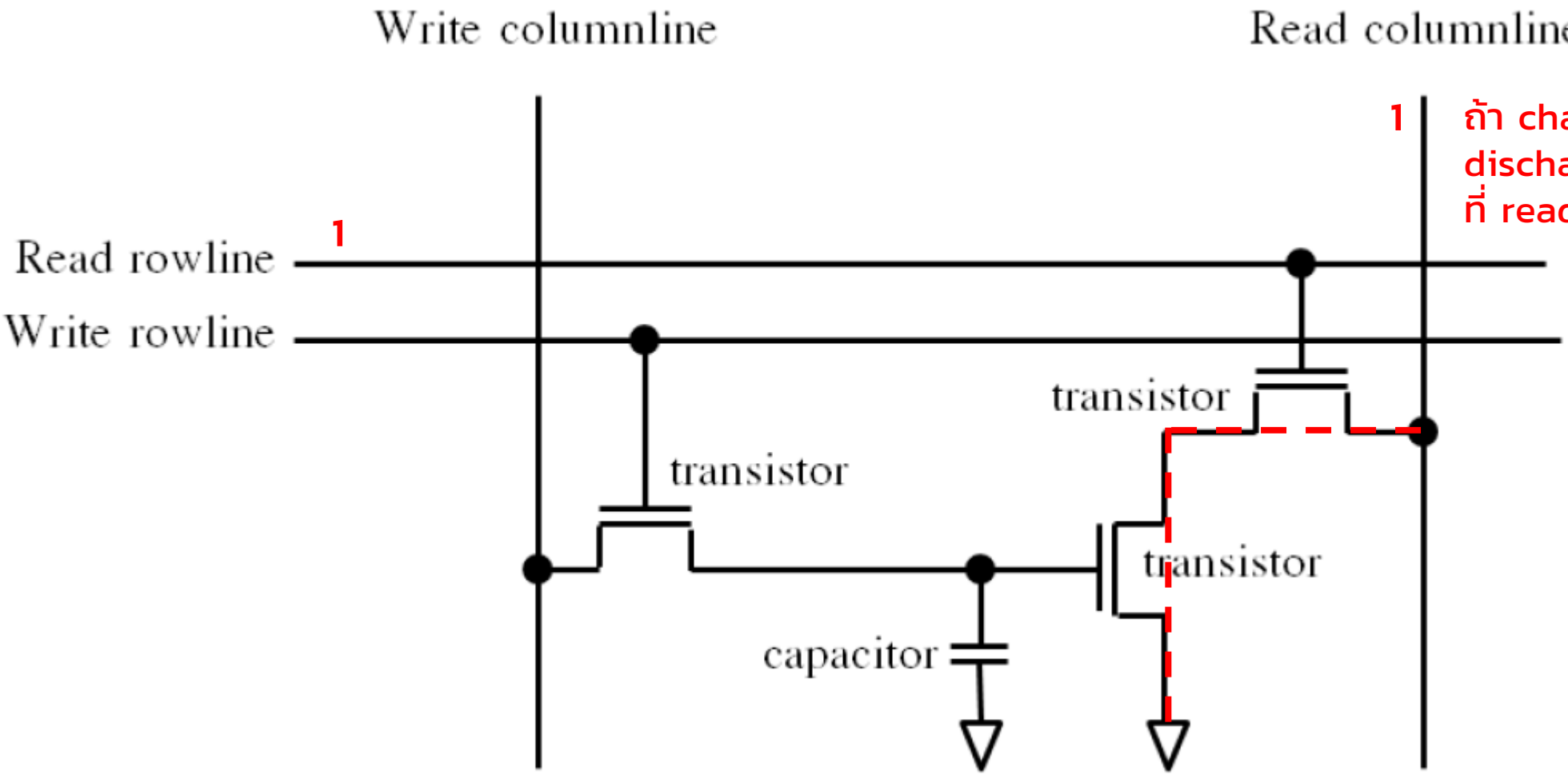
Chip Select (sel)	sel = 1 คือให้ memory ทำงาน
Clock	ที่ positive edge จะเอาค่า Data ไปเขียนลง memory ช่อง Address หรือการเขียน (write) memory
Load (ld)	ld = 1 จะเอาค่าใน memory ช่อง Address ไปออกที่ Data หรือการอ่าน (read) memory
Clear (clr)	clr = 1 คือให้รีเซ็ต memory ทุกช่อง

Dynamic RAM (DRAM)



รูปที่ 7.4 ดีแรมขนาด 1 บิต (ยุคแรกสุด)

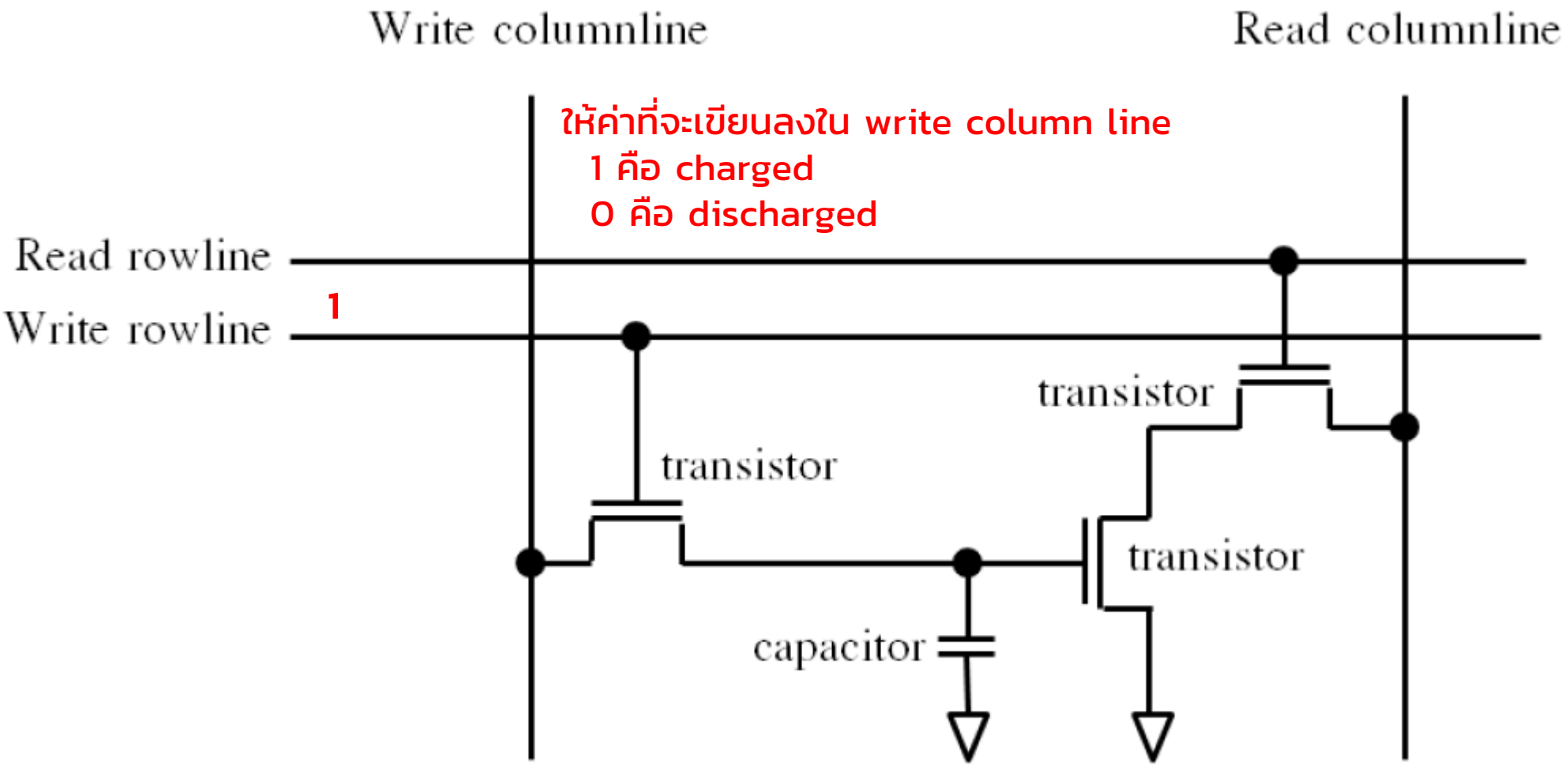
Read



1 ถ้า charged จะอ่านได้ 0
discharge จะอ่านได้ 1
ที่ read column line

รูปที่ 7.4 ดีแรมขนาด 1 บิต

Write (ต้องเขียนซ้ำเป็นระยะ)



รูปที่ 7.4 ดีแรมขนาด 1 บิต

Static RAM (SRAM)

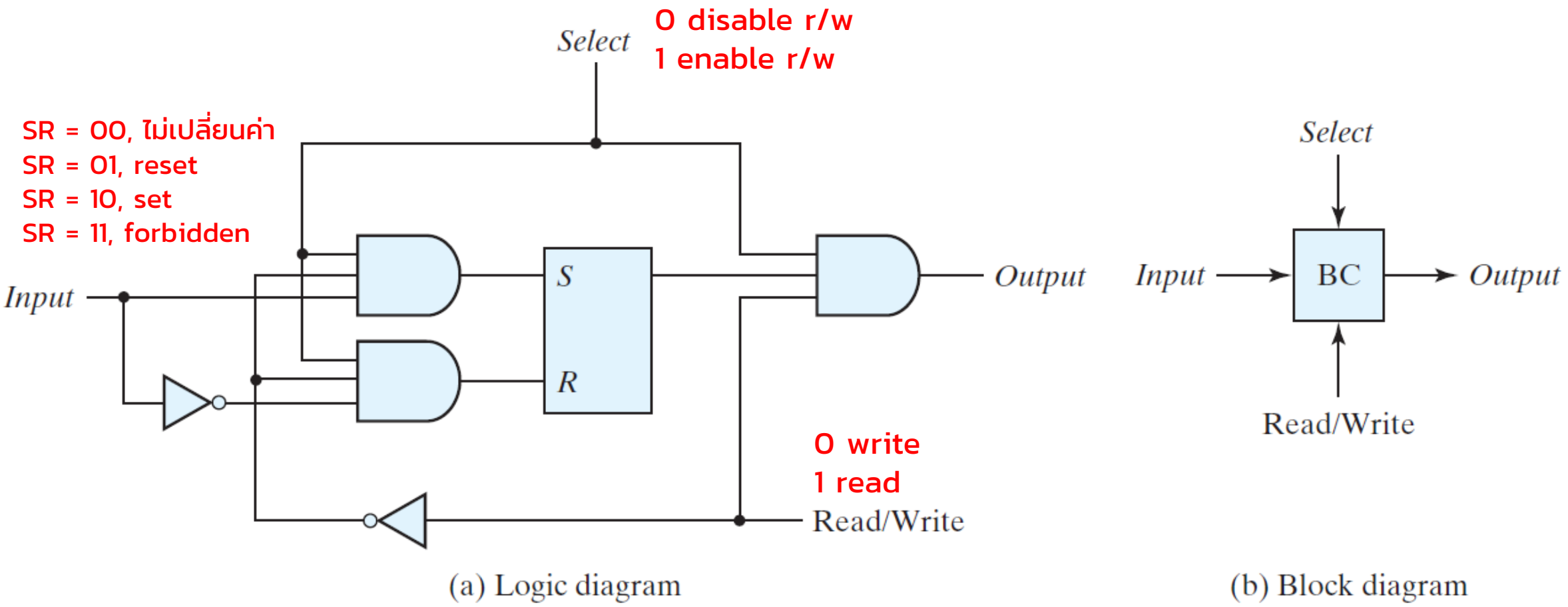


FIGURE 7.5
Memory cell

ใช้ decode ว่าจะเอา enable ไปออกที่ 0, 1, 2, 3

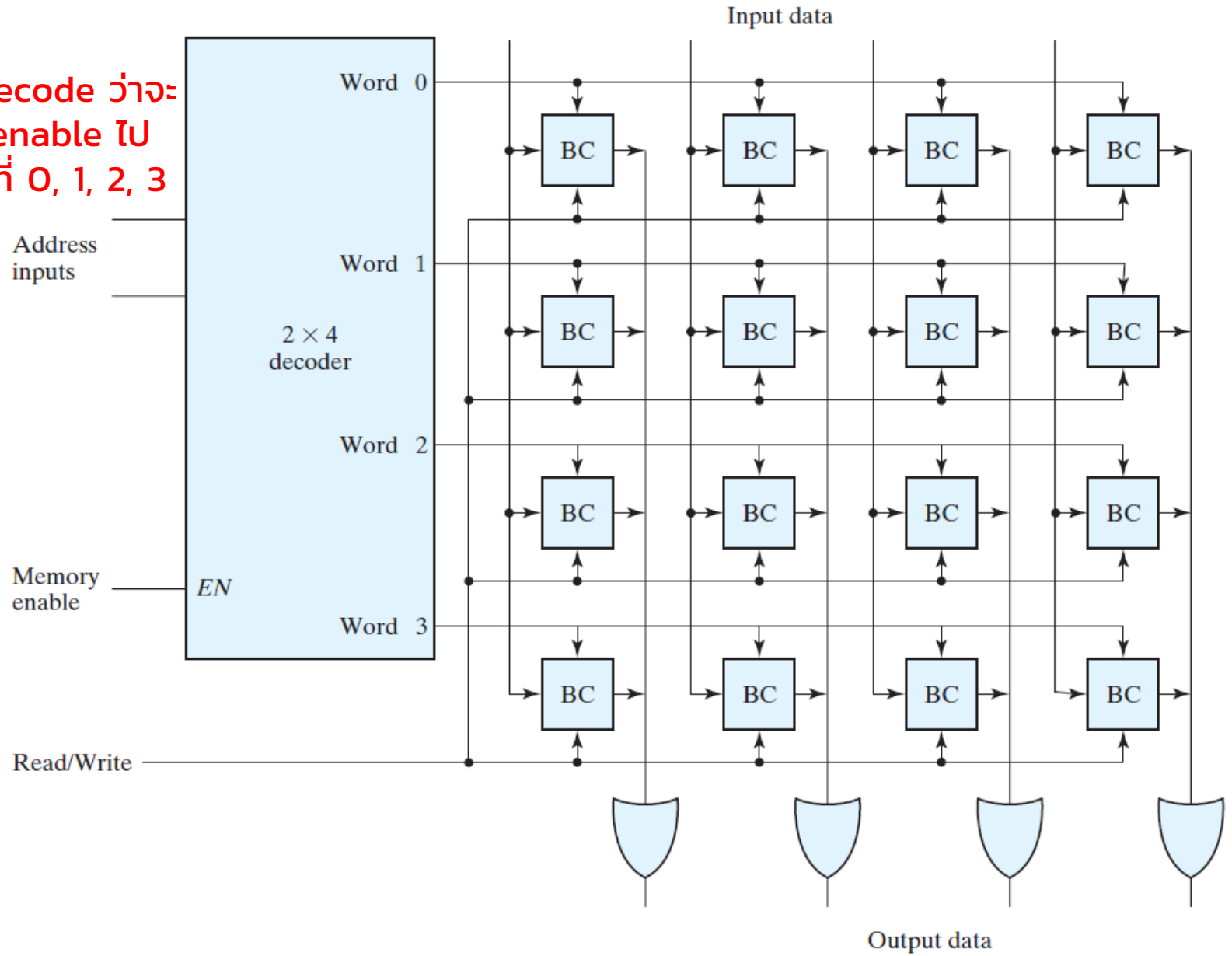


FIGURE 7.6
Diagram of a 4×4 RAM

M x N คือ M ช่อง ช่องละ N บิต

ทำเป็นโครงสร้าง 2 มิติ
Decoder จะได้ไม่ใหญ่มาก

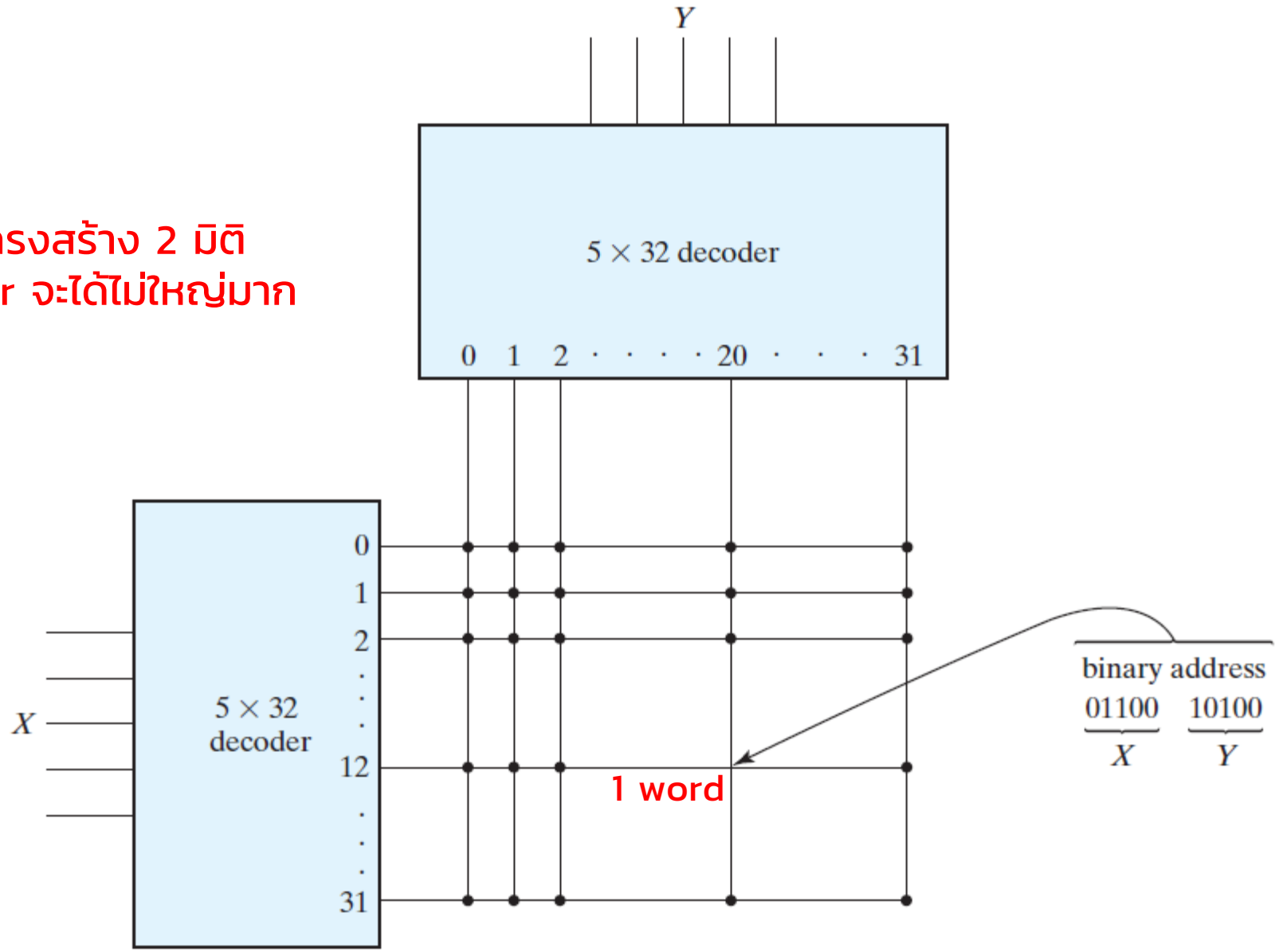


FIGURE 7.7
Two-dimensional decoding structure for a 1K-word memory

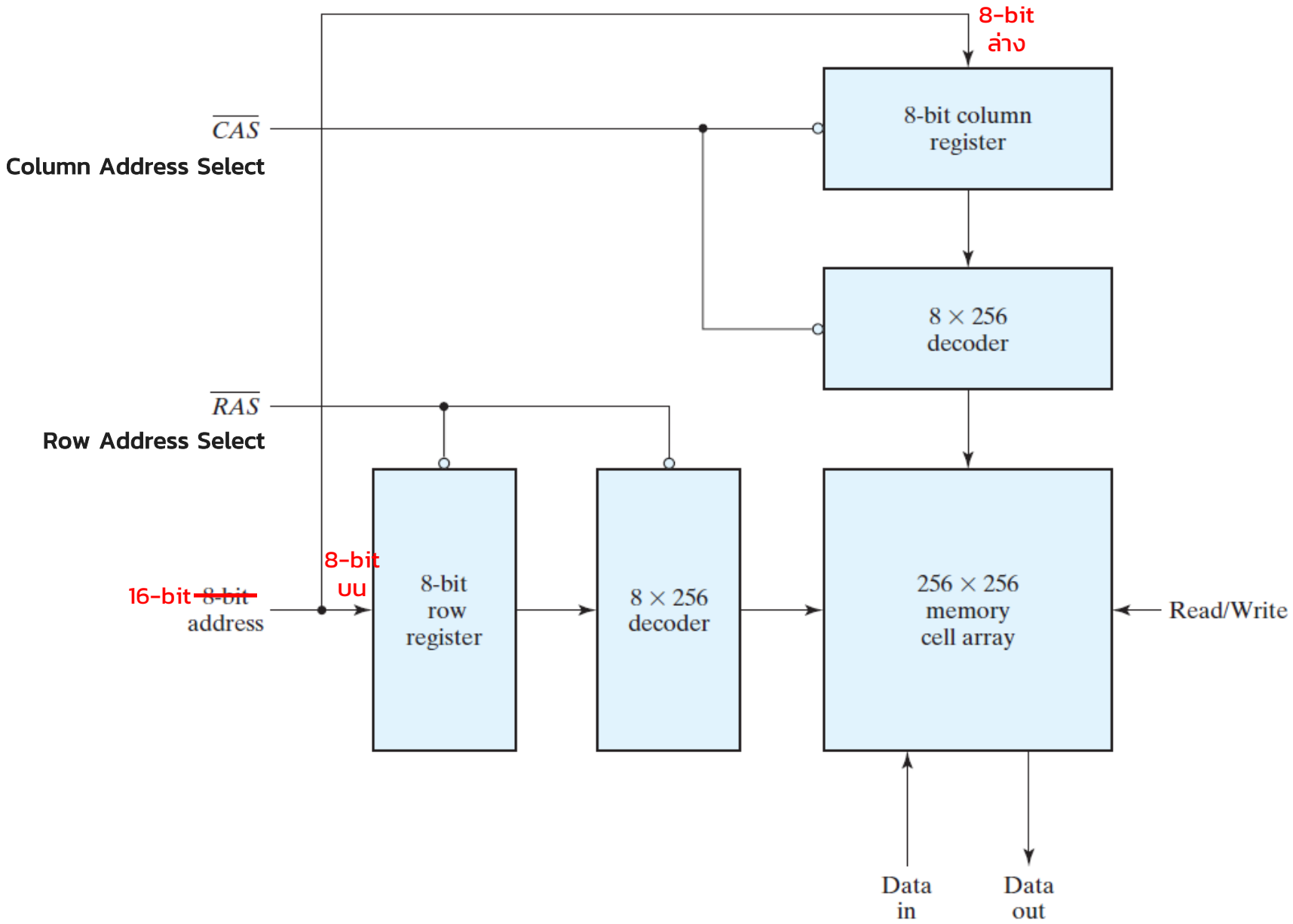
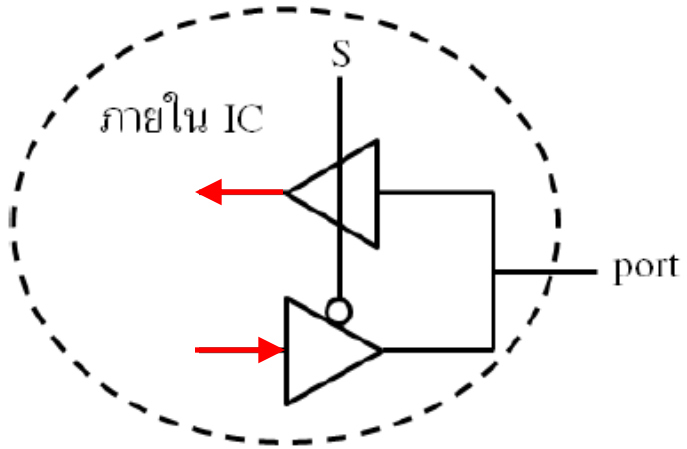


FIGURE 7.8
Address multiplexing for a 64K DRAM

2^{16} ช่อง ช่องละ 1 บิต (parallel ให้เป็น n บิตได้)

พอร์ตเข้าออกได้สองทาง (Bi-directional Port)

โดยปกติขาหรือพอร์ต (port) ของอุปกรณ์ดิจิทัลจะเป็นอินพุตหรือเอาต์พุตเท่านั้น แต่เราสามารถทำให้พอร์ตเดียวเป็นทั้งอินพุตและเอาต์พุตได้ ประโยชน์ของการใช้พอร์ตอินพุตและเอาต์พุตร่วมกันคือช่วยลดจำนวนขาของอุปกรณ์ลง โดยเฉพาะอุปกรณ์หน่วยความจำที่อาจจะมีขา data-in และ data-out มากถึง 32 หรือ 64 บิต รูปที่ 7.6 แสดงพอร์ตเข้าออกได้สองทางที่สร้างจากบัฟเฟอร์สามสถานะ ถ้า S = 0 จะทำให้พอร์ตเป็นเอาต์พุต ถ้า S = 1 จะเป็นพอร์ตเป็นอินพุต



รูปที่ 7.6 พอร์ตเข้าออกได้สองทาง (bi-directional port)

Big-endian และ Little-endian

การเรียงข้อมูลในหน่วยความจำมีสองแบบ แบบแรกเรียกว่า Big-endian คือเรียงข้อมูลจากไบต์แรกไปไบต์สุดท้าย แบบที่สองเรียกว่า Little-endian คือเรียงกลับกัน ตัวอย่างเช่น ข้อมูล (int) คือ 0x12345678 หรือ 305,419,896 ในฐานสิบ เก็บแบบ Big-endian และ Little-endian ได้ดังนี้

HEX	1234 5678
BIN	0001 0010 0011 0100 0101 0110 0111 1000

	Addr i	Addr i + 1	Addr i + 2	Addr i + 3
แบบ big-endian	12	34	56	78
แบบ little-endian	78	56	34	12

คอมพิวเตอร์ 2 เครื่อง อาจจะอ่านค่าที่ addr นี้ได้ไม่เหมือนกัน

กรณีที่หน่วยความจำต่ออยู่กับไมโครโพรเซสเซอร์ เวลาอ่านเขียนหน่วยความจำไมโครโพรเซสเซอร์จะเป็นตัวกำหนดว่าจะเก็บข้อมูลแบบ Big-endian หรือแบบ Little-endian ไมโครโพรเซสเซอร์ในปัจจุบัน เช่น Intel เก็บข้อมูลแบบ Little-endian

Processor	Endianness
Motorola 68000	Big Endian
PowerPC (PPC)	Big Endian
Sun Sparc	Big Endian
IBM S/390	Big Endian
Intel x86 (32 bit)	Little Endian
Intel x86_64 (64 bit)	Little Endian
Dec VAX	Little Endian
Alpha	Bi (Big/Little) Endian
ARM	Bi (Big/Little) Endian
IA-64 (64 bit)	Bi (Big/Little) Endian
MIPS	Bi (Big/Little) Endian

Network Byte Order:

Big endian byte ordering has been chosen as the "neutral" or standard for network data exchange and thus Big Endian byte ordering is also known as the "Network Byte Order". Thus Little Endian systems will convert their internal Little Endian representation of data to Big Endian byte ordering when writing to the network via a socket. This also requires Little Endian systems to swap the byte ordering when reading from a network connection. Languages such as Java manage this for you so that Java code can run on any platform and programmers do not have to manage byte ordering.

http://www.yolinux.com/TUTORIALS/Endian-Byte-Order.html

Error Detection & Error Correction

Hamming Code

One of the most common error-correcting codes used in RAMs was devised by R. W. Hamming. In the Hamming code, k parity bits are added to an n-bit data word, forming a new word of $n + k$ bits. The bit positions are numbered in sequence from 1 to $n + k$. Those positions numbered as a power of 2 are reserved for the parity bits. The remaining bits are the data bits. The code can be used with words of any length. Before giving the general characteristics of the code, we will illustrate its operation with a data word of eight bits.

Consider, for example, the 8-bit data word 11000100. We include 4 parity bits with the 8-bit word and arrange the 12 bits as follows:

Bit position:	1	2	3	4	5	6	7	8	9	10	11	12
	P_1	P_2	1	P_4	1	0	0	P_8	0	1	0	0

The 4 parity bits, P_1, P_2, P_4, and P_8, are in positions 1, 2, 4, and 8, respectively. The 8 bits of the data word are in the remaining positions. Each parity bit is calculated as follows:

$$P_1 = \text{XOR of bits } (3, 5, 7, 9, 11) = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0$$

$$P_2 = \text{XOR of bits } (3, 6, 7, 10, 11) = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0$$

$$P_4 = \text{XOR of bits } (5, 6, 7, 12) = 1 \oplus 0 \oplus 0 \oplus 0 = 1$$

$$P_8 = \text{XOR of bits } (9, 10, 11, 12) = 0 \oplus 1 \oplus 0 \oplus 0 = 1$$

0 Even
1 Odd

Remember that the exclusive-OR operation performs the odd function: It is equal to 1 for an odd number of 1's in the variables and to 0 for an even number of 1's. Thus, each parity bit is set so that the total number of 1's in the checked positions, including the parity bit, is always even.

The 8-bit data word is stored in memory together with the 4 parity bits as a 12-bit composite word. Substituting the 4 P bits in their proper positions, we obtain the 12-bit composite word stored in memory:

	0	0	1	1	1	0	0	1	0	1	0	0
Bit position:	1	2	3	4	5	6	7	8	9	10	11	12

When the 12 bits are read from memory, they are checked again for errors. The parity is checked over the same combination of bits, including the parity bit. The 4 check bits are evaluated as follows:

$$C_1 = \text{XOR of bits } (1, 3, 5, 7, 9, 11)$$
$$C_2 = \text{XOR of bits } (2, 3, 6, 7, 10, 11)$$
$$C_4 = \text{XOR of bits } (4, 5, 6, 7, 12)$$
$$C_8 = \text{XOR of bits } (8, 9, 10, 11, 12)$$

ต้องมี 1 เป็นจำนวนคู่แน่ ๆ
C_i ต้องเท่ากับ 0

A 0 check bit designates even parity over the checked bits and a 1 designates odd parity. Since the bits were stored with even parity, the result, $C = C_8C_4C_2C_1 = 0000$, indicates that no error has occurred. However, if $C \neq 0$, then the 4-bit binary number formed by the check bits gives the position of the erroneous bit. For example, consider the following three cases:

Bit position:	1	2	3	4	5	6	7	8	9	10	11	12	
	0	0	1	1	1	0	0	1	0	1	0	0	No error
	1	0	1	1	1	0	0	1	0	1	0	0	Error in bit 1
	0	0	1	1	0	0	0	1	0	1	0	0	Error in bit 5

	C_8	C_4	C_2	C_1
For no error:	0	0	0	0
With error in bit 1:	0	0	0	1
With error in bit 5:	0	1	0	1

อ่านแบบเลขฐานสอง จะบอกว่าบิตใดผิด

Table 7.2
Range of Data Bits for k Check Bits

Number of Check Bits, *k*	Range of Data Bits, *n*
3	2–4
4	5–11
5	12–26
6	27–57
7	58–120

รอม (ROM), พร็อม (PROM), อีพร็อม (EPROM), อีอีพร็อม (EEPROM)

รอม (ROM - Read-Only Memory) แสดงในรูปที่ 7.8 เป็นหน่วยความจำลบเลือนไม่ได้ (non-volatile memory) ที่มีการบันทึกค่าเริ่มต้นมาจากโรงงานผลิต ผู้ใช้ไม่สามารถแก้ไขค่าภายในรอมได้ ในเมนบอร์ดของพีซีจะใช้รอมเพื่อบันทึกโปรแกรม BIOS (Basic Input/Output System) ซึ่งเป็นโปรแกรมแรกที่จะทำงานเมื่อเราเปิดคอมพิวเตอร์ คำสั่งที่บรรจุอยู่ใน BIOS คือให้ค้นหาและโหลดระบบปฏิบัติการ (operating system) ขึ้นมาทำงาน

เพื่อให้การใช้งานรอมมีความยืดหยุ่นมากขึ้น จึงมีรอมชนิดที่สามารถบันทึกค่าใหม่ได้ เช่น

Programmable ROM (PROM)	ผู้ใช้สามารถบันทึกข้อมูลได้เพียงครั้งเดียว
Erasable PROM (EPROM)	ลบข้อมูลด้วยแสงอัลตราไวโอเลตและบันทึกใหม่ได้หลายครั้ง
Electronic EPROM (EEPROM)	ลบข้อมูลด้วยไฟฟ้าและบันทึกใหม่ได้หลายครั้ง

ปัจจุบันใช้ UEFI แทน BIOS

BIOS อาจจะบรรจุอยู่ใน
ROM, PROM, EPROM, EEPROM

EPROM

EPROMs

EEPROM
socket

รูปที่ 7.8 ROM (BIOS), EPROM, EPROM eraser, EEPROM burner

32 เส้น (เขียนย่อ ๆ เป็นเส้นตั้งอันเดียว)

ต้องใช้เกต OR ที่ floating input เป็น 0 (ขอไม่อธิบายว่าสร้างยังไง)

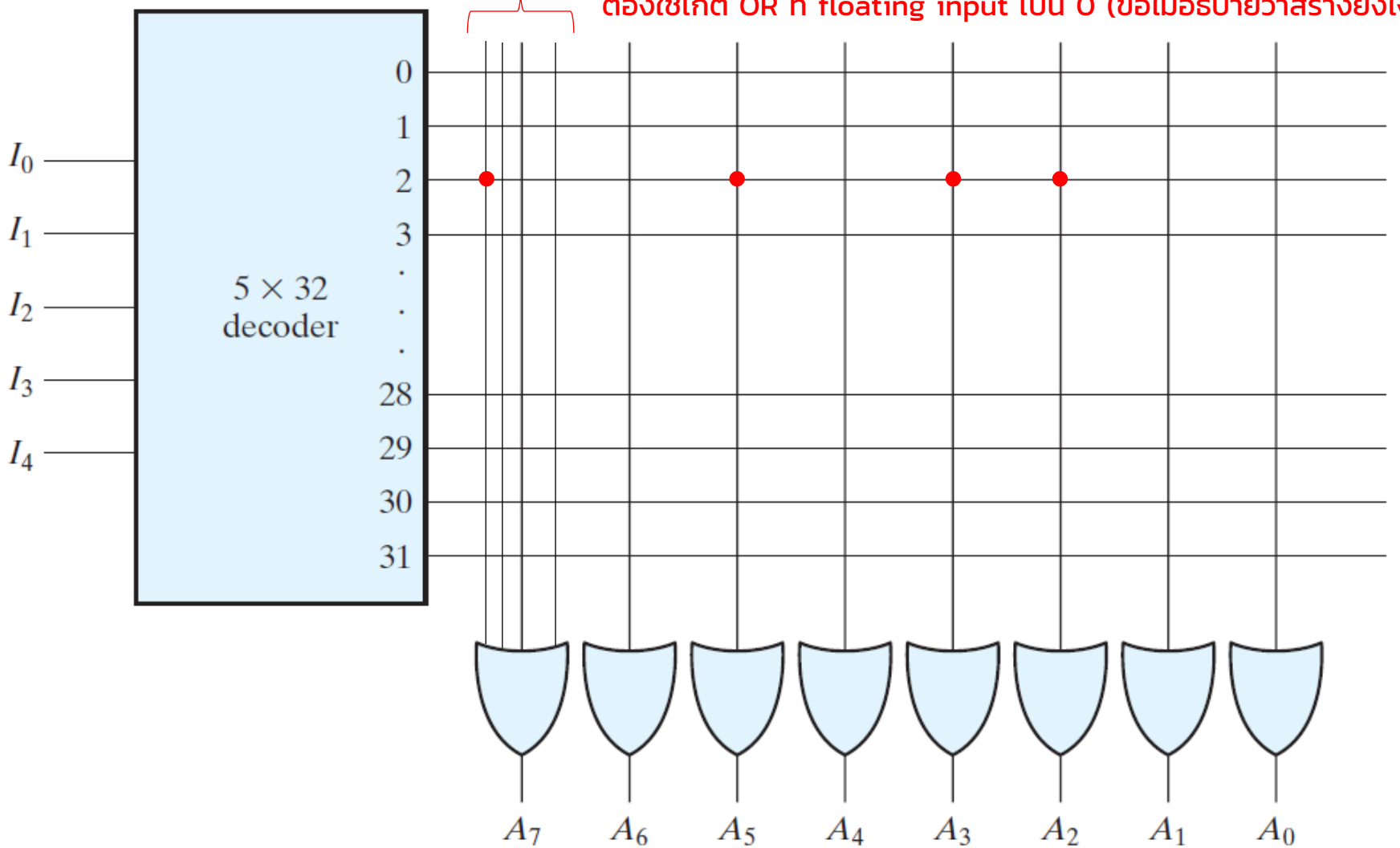


FIGURE 7.10
Internal logic of a 32 × 8 ROM

Table 7.3
ROM Truth Table (Partial)

Inputs					Outputs							
I_4	I_3	I_2	I_1	I_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
		⋮							⋮			
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1

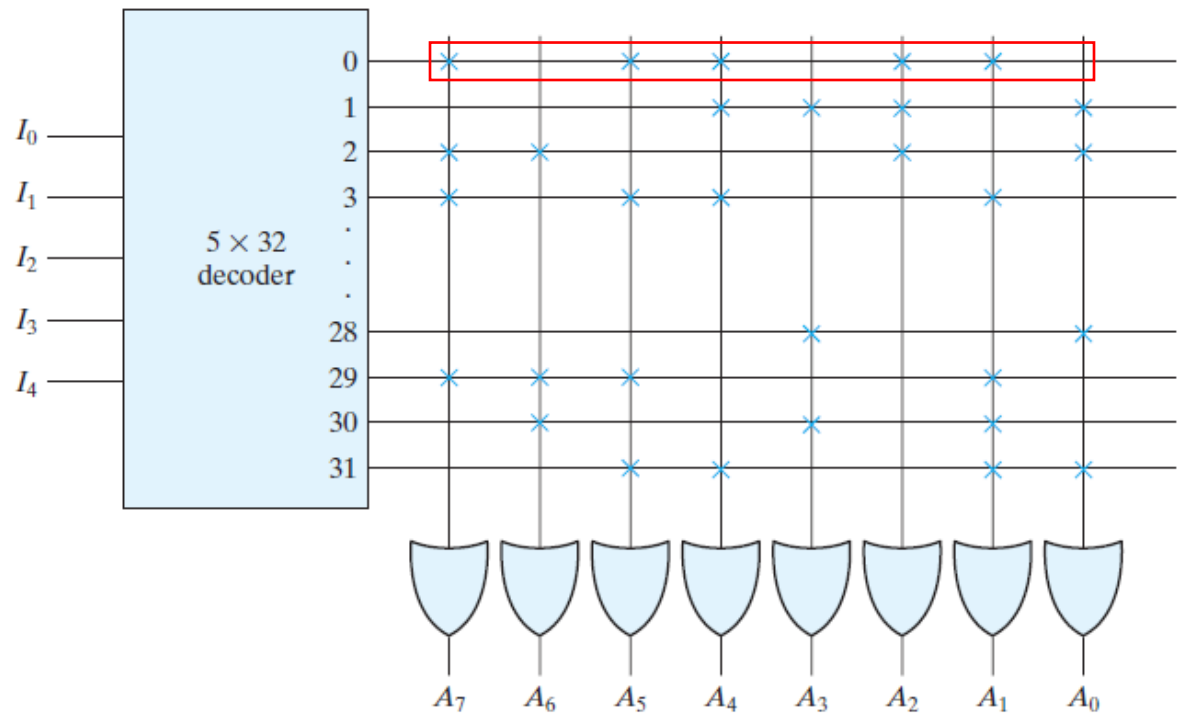
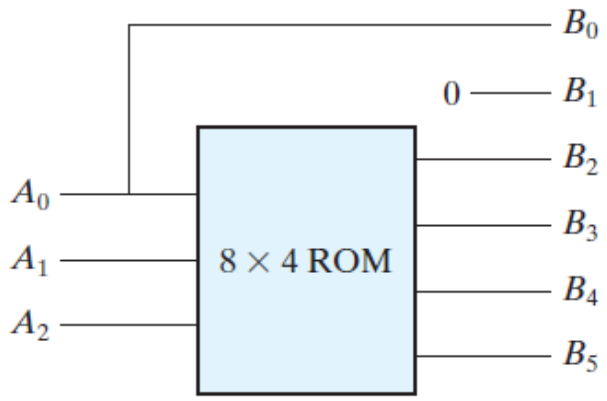


Table 7.4
Truth Table for Circuit of Example 7.1

Inputs			Outputs $f(x) = x^2$						
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	Decimal
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49



(a) Block diagram

A_2	A_1	A_0	B_5	B_4	B_3	B_2
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	0	1
1	1	1	1	1	0	0

(b) ROM truth table

FIGURE 7.12
ROM implementation of Example 7.1

PAL

Programmable Array Logic

ต้องใช้เกต AND ที่ floating input เป็น 1 (ขอไม่อธิบายว่าสร้างยังไง)

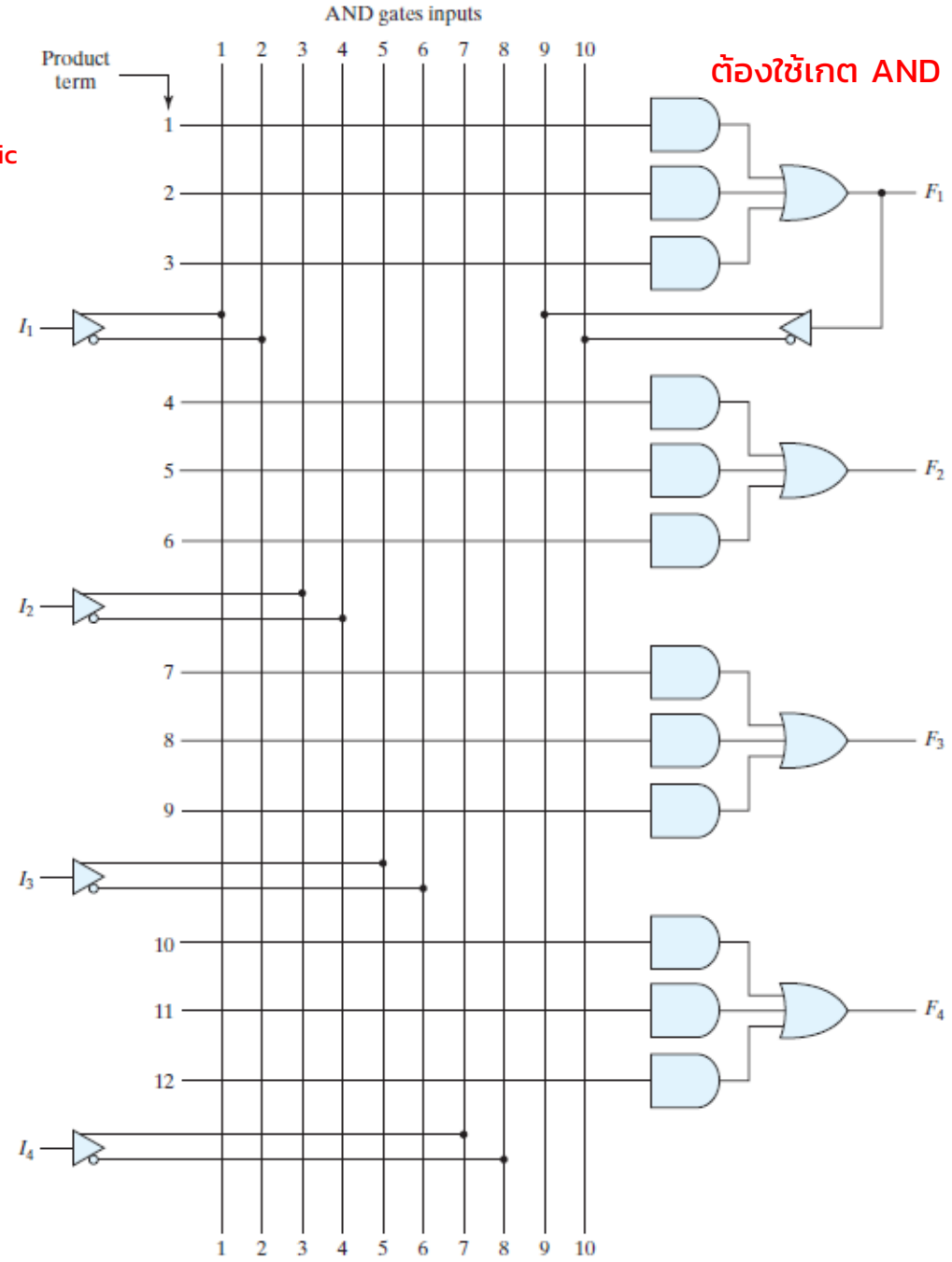
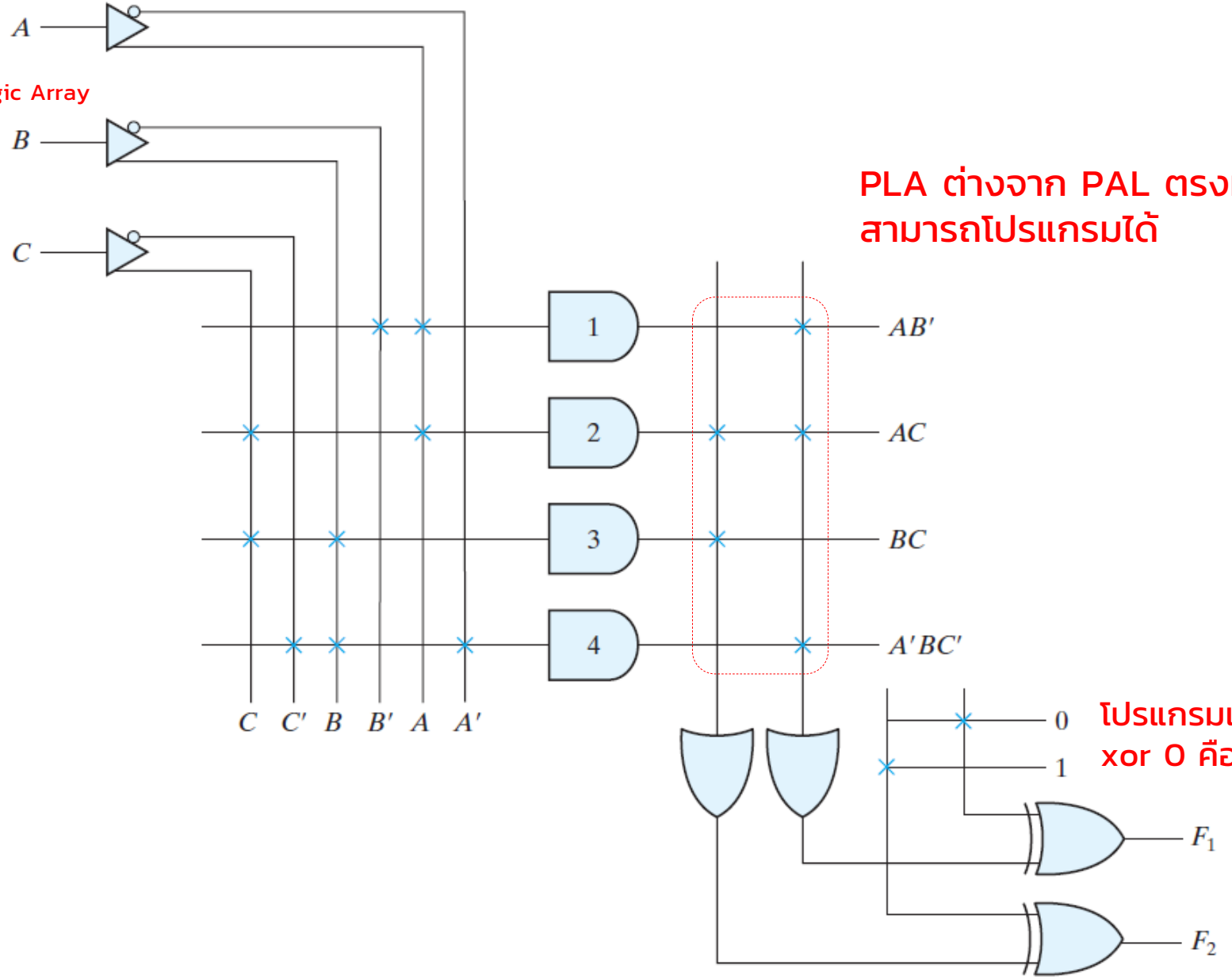


FIGURE 7.16
PAL with four inputs, four outputs, and a three-wide AND–OR structure

PLA

Programmable Logic Array



PLA ต่างจาก PAL ตรงที่จาก AND มา OR สามารถโปรแกรมได้

โปรแกรมเพื่อ complement ฟังก์ชันได้
xor 0 คือไม่ com ถ้า xor 1 คือ complement

$$F_1 = AB' + AC + A'BC'$$
$$F_2 = (AC + BC)'$$

FIGURE 7.14
PLA with three inputs, four product terms, and two outputs

Table 5.3
Second Form of the State Table

Present State		Next State				Output	
		$x = 0$		$x = 1$		$x = 0$	$x = 1$
A	B	A	B	A	B	y	y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

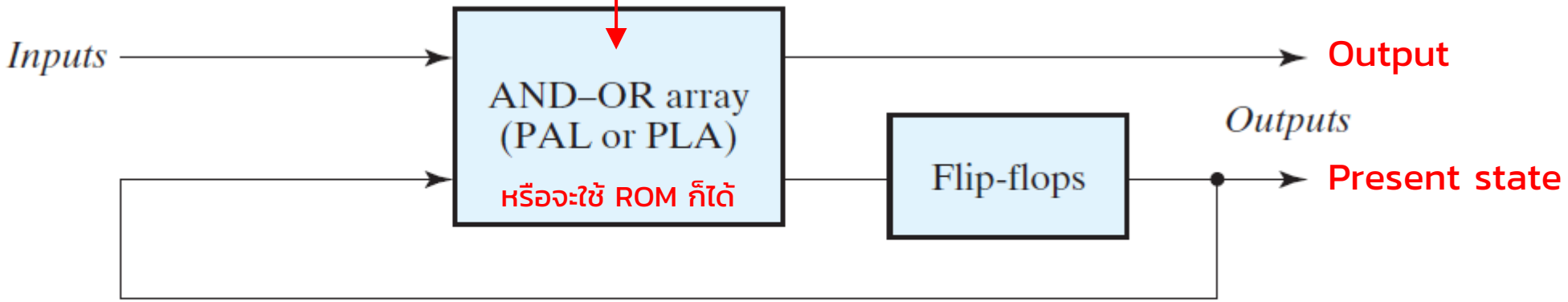


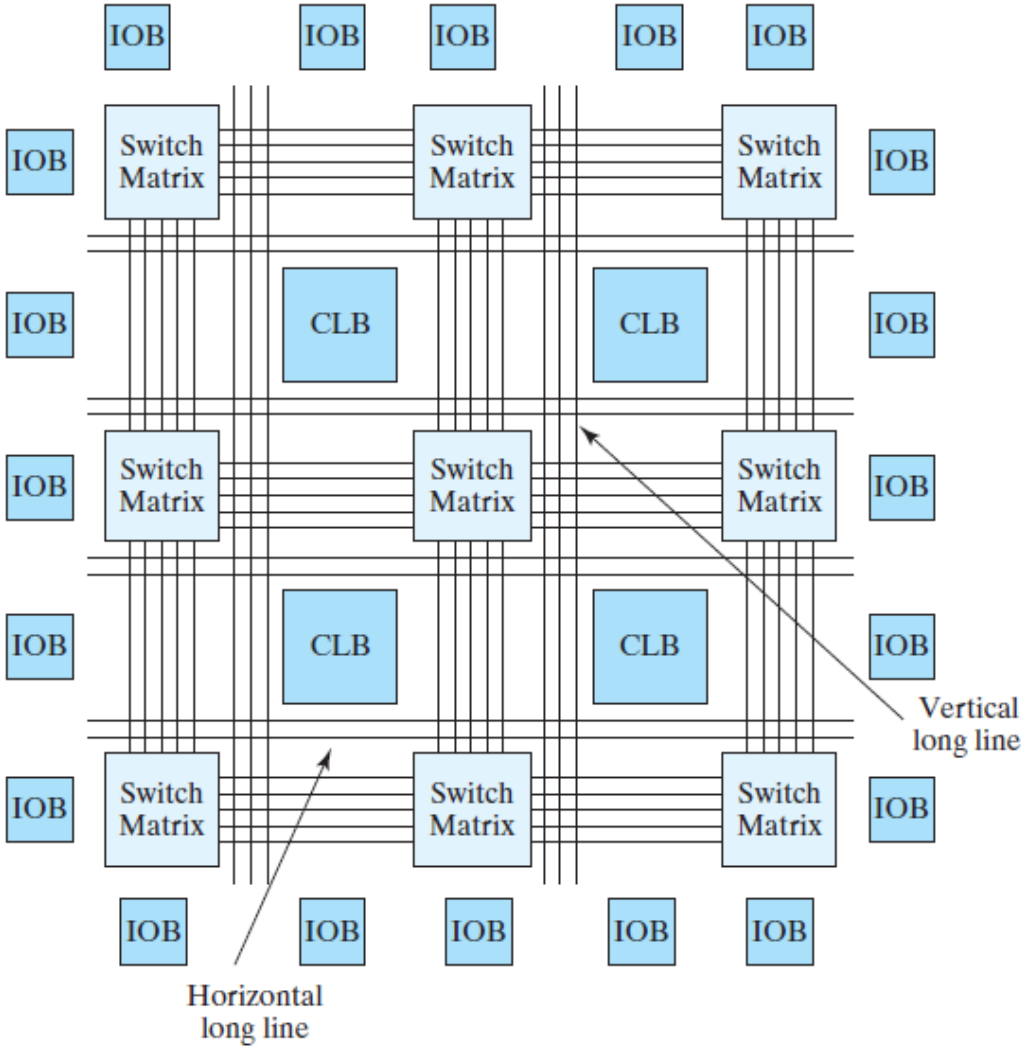
FIGURE 7.18
Sequential programmable logic device

Xilinx FPGAs (FPGA = Field Programmable Gate Array, Xilinx เป็นชื่อยี่ห้อ)

Xilinx launched the world's first commercial FPGA in 1985, with the vintage XC2000 device family.[2] The XC3000 and XC4000 families soon followed, setting the stage for today's Spartan™, and Virtex™ device families. Each evolution of devices brought improvements in density, performance, power consumption, voltage levels, pin counts, and functionality. For example, the Spartan family of devices initially offered a maximum of 40K system gates, but today's Spartan-6 offers 150,000 logic cells plus 4.8Mb block RAM.

Basic Xilinx Architecture

The basic architecture of Spartan and earlier device families consists of an array of configurable logic blocks (CLBs), a variety of local and global routing resources, and input–output (I/O) blocks (IOBs), programmable I/O buffers, and an SRAM-based configuration memory, as shown in Fig. 7.21.



Configuration Bits

0100011010101l....001010

ใช้ทำต้นแบบผลิตภัณฑ์ (product prototype) จำนวนน้อย ๆ ราคาต่อชิ้นแพงกว่า ทำงานที่ความถี่ต่ำกว่าชิปจริง

FIGURE 7.21
Basic architecture of Xilinx Spartan and predecessor devices

Cache (SRAM)

https://courses.cs.washington.edu/courses/cse378/09wi/lectures/lec15.pdf

How does computer (DRAM) memory work?

https://www.youtube.com/watch?v=7J7X7aZvMXQ

How do SSDs work?

https://www.youtube.com/watch?v=E7Up7VuFd8A