Memory

Ch.7

Memory

- *Memory*: a collection of storage cells with the necessary circuits to transfer information to and from them
- *Bit*: a single binary digit (0/1)
- *Byte*: a collection of 8 bits accessed together
- *Word*: a collection of bits whose size is a typical unit of access for the memory (machine dependent)
- For memory, 1K = 2^{10} = 1024, 1M = 2^{20} = 1048576, 1G = 2^{30}

Random-Access Memory (RAM)

- *Random-access memory* takes the same time to access any word regardless of location (c.f. *serial* memories like tapes and disks which takes different amount of time to access words at different locations)
- Allows both read and write operations

Memory Technology

- Random access:
  - Access time same for all locations
  - SRAM: *Static Random Access Memory*
    - Low density, high power, expensive, fast
    - Static: content will last (forever until lose power)
    - Address not divided
    - Use for caches
  - DRAM: *Dynamic Random Access Memory*
    - High density, low power, cheap, slow
    - Dynamic: need to be refreshed regularly
    - Addresses in 2 halves (memory as a 2D matrix):
      - RAS/CAS (Row/Column Access Strobe)
    - Use for main memory
- Magnetic disk

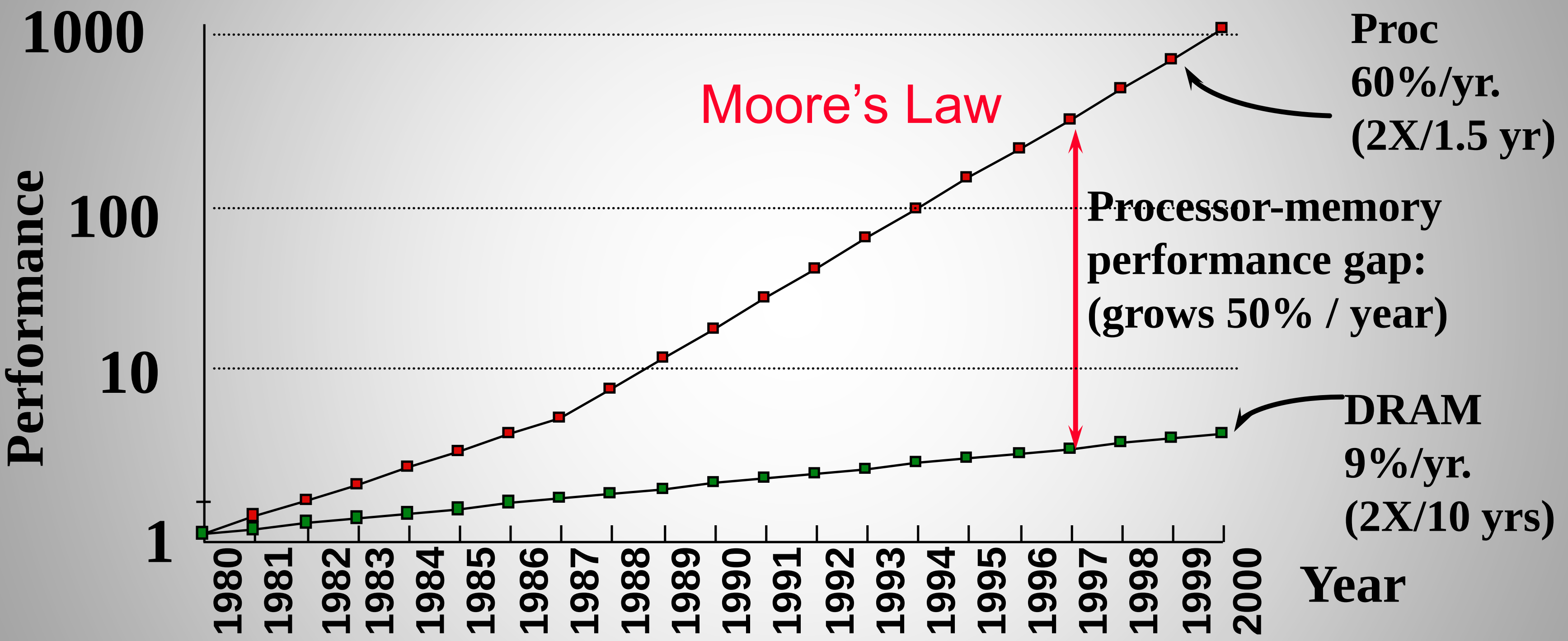
Comparisons of Various Technologies

Memory technology	Typical access time	$ per GB in 2012
SRAM	0.5 – 2.5 ns	$500 – $1,000
DRAM	50 – 70 ns	$10 – $20
Magnetic disk	5,000,000 – 20,000,000 ns	$0.20 – $2
Flash	5,000–50,000 ns	$0.75–$1.00

Ideal memory

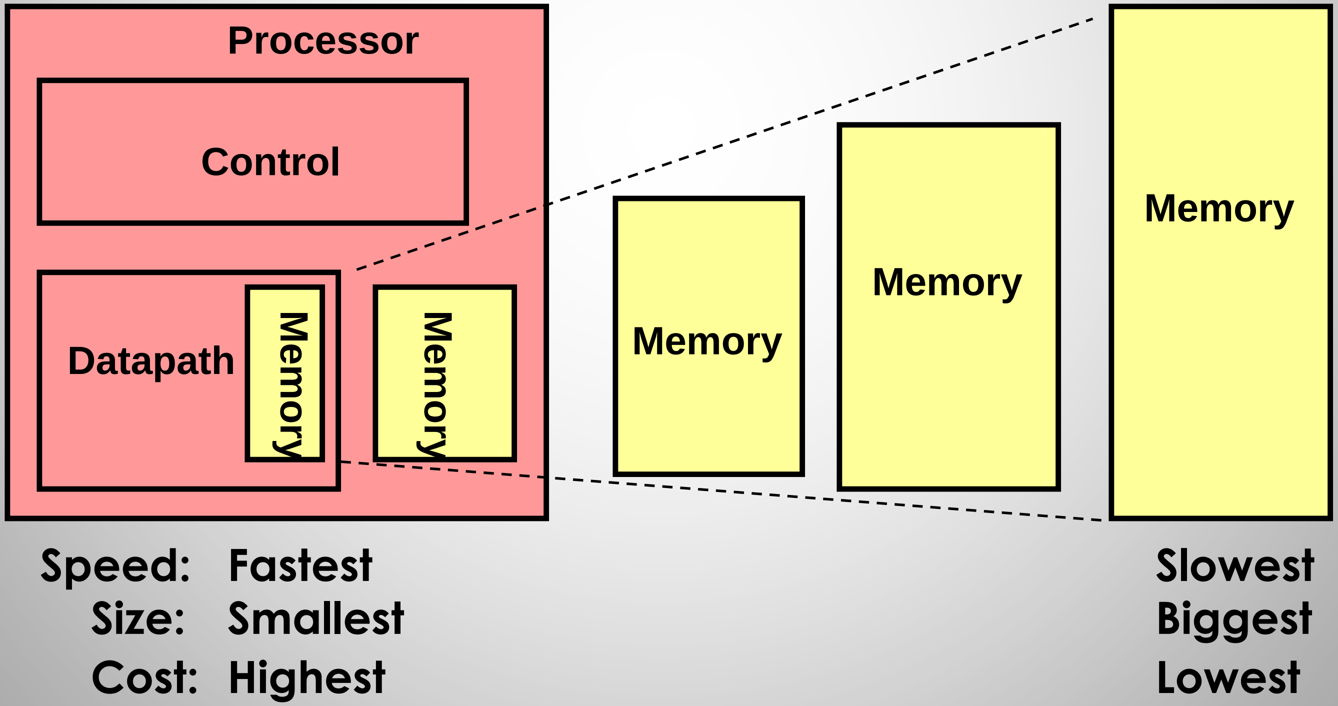
- ♦Access time of SRAM
- ♦Capacity and cost/GB of disk

Processor Memory Latency Gap



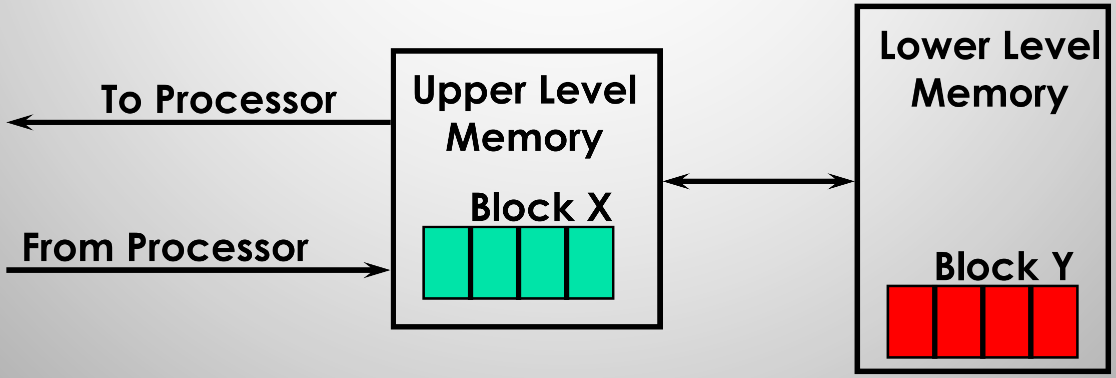
Solution: Memory Hierarchy

- An Illusion of a large, fast, cheap memory
  - Fact: Large memories slow, fast memories small
  - How to achieve: hierarchy, parallelism
- An expanded view of memory system:



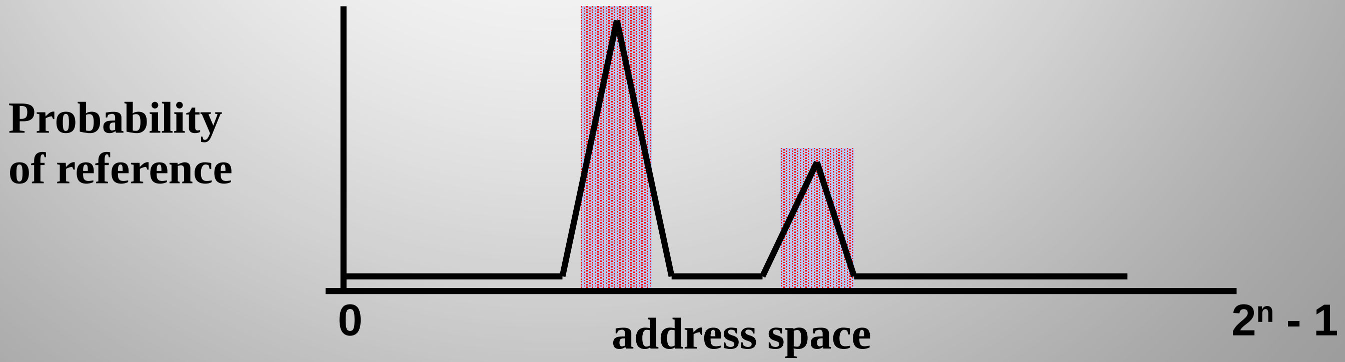
Memory Hierarchy: Principle

- At any given time, data is copied between only two adjacent levels:
  - Upper level: the one closer to the processor
    - Smaller, faster, uses more expensive technology
  - Lower level: the one away from the processor
    - Bigger, slower, uses less expensive technology
- *Block*: basic unit of information transfer
  - Minimum unit of information that can either be present or not present in a level of the hierarchy

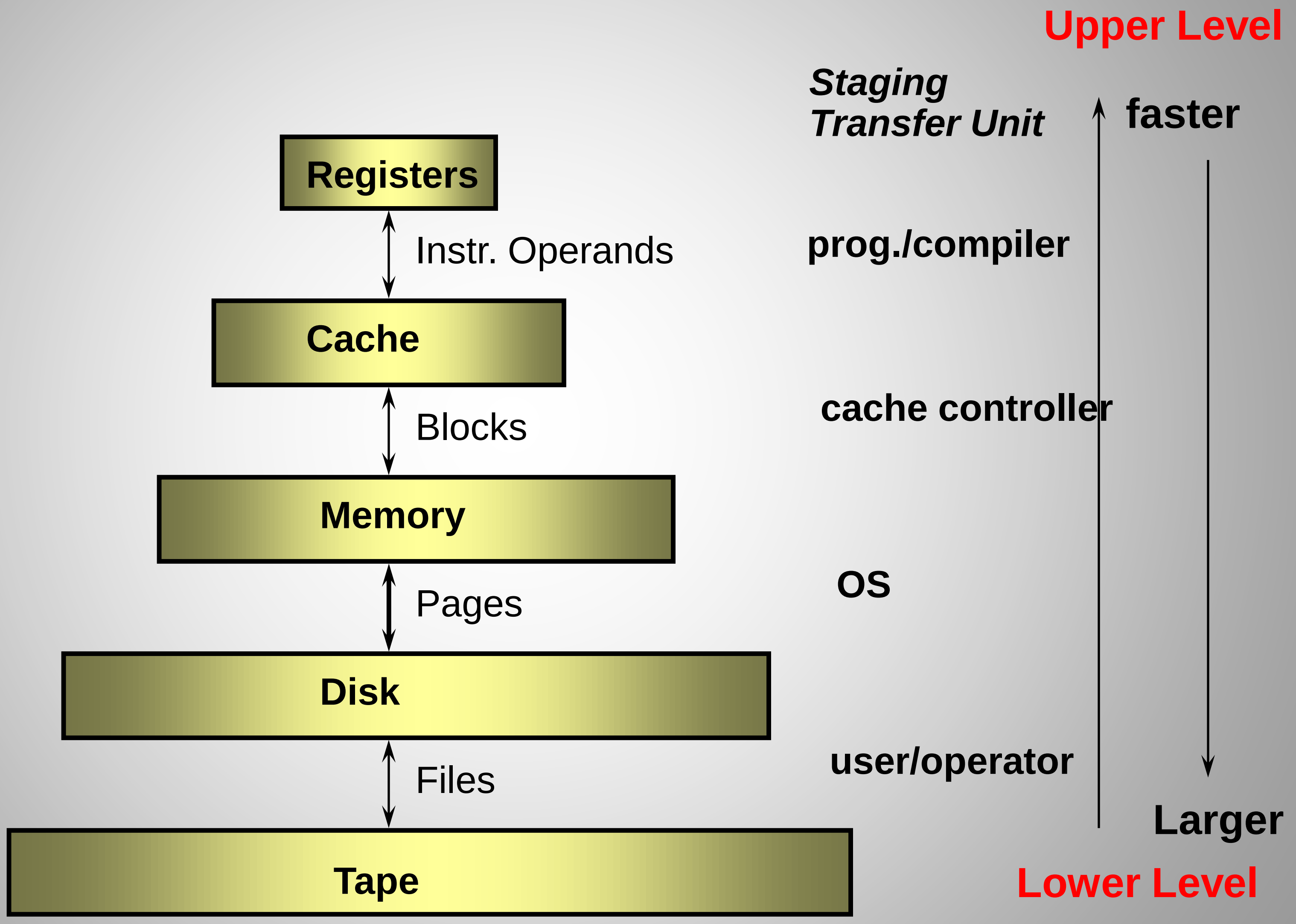


Why Hierarchy Works?

- *Principle of Locality*:
  - Program access a relatively small portion of the address space at any instant of time
  - 90/10 rule: 10% of code executed 90% of time
- Two types of locality:
  - Temporal locality: if an item is referenced, it will tend to be referenced again soon
  - Spatial locality: if an item is referenced, items whose addresses are close by tend to be referenced soon



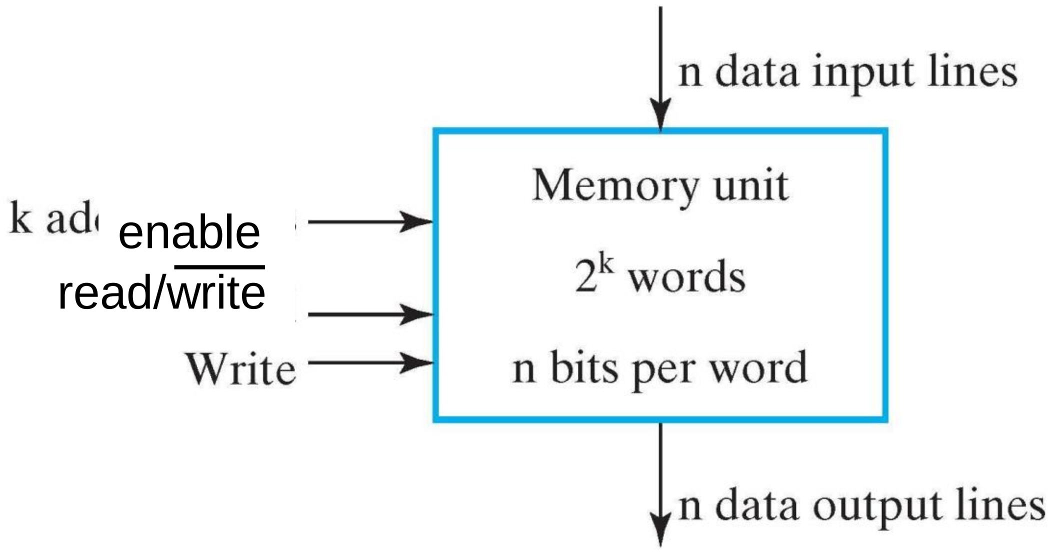
Levels of Memory Hierarchy



How Is the Hierarchy Managed?

- Registers <-> Memory
  - by compiler (programmer?)
- cache <-> memory
  - by the hardware
- memory <-> disks
  - by the hardware and operating system (virtual memory)
  - by the programmer (files)
- Replacement policy

Memory Block Diagram



k address lines => 2^k words

n data input/output lines => n bits per word

Memory Organization Example

- Example memory content
  - A memory with 3 address bits & 8 data bits

Memory Address		Memory
Binary	Decimal	Content
0 0 0	0	1 0 0 0 1 1 1 1
0 0 1	1	1 1 1 1 1 1 1 1
0 1 0	2	1 0 1 1 0 0 0 1
0 1 1	3	0 0 0 0 0 0 0 0
1 0 0	4	1 0 1 1 1 0 0 1
1 0 1	5	1 0 0 0 0 1 1 0
1 1 0	6	0 0 1 1 0 0 1 1
1 1 1	7	1 1 0 0 1 1 0 0

Memory Operations

- *Write*: writing *into memory* a new word

1. Apply the address of the write location to the address lines
2. Place the word to be stored to the data input lines
3. Activate the Write control input

- *Write cycle time* - maximum time from the application of the address to the completion of all internal memory operations required to store a word

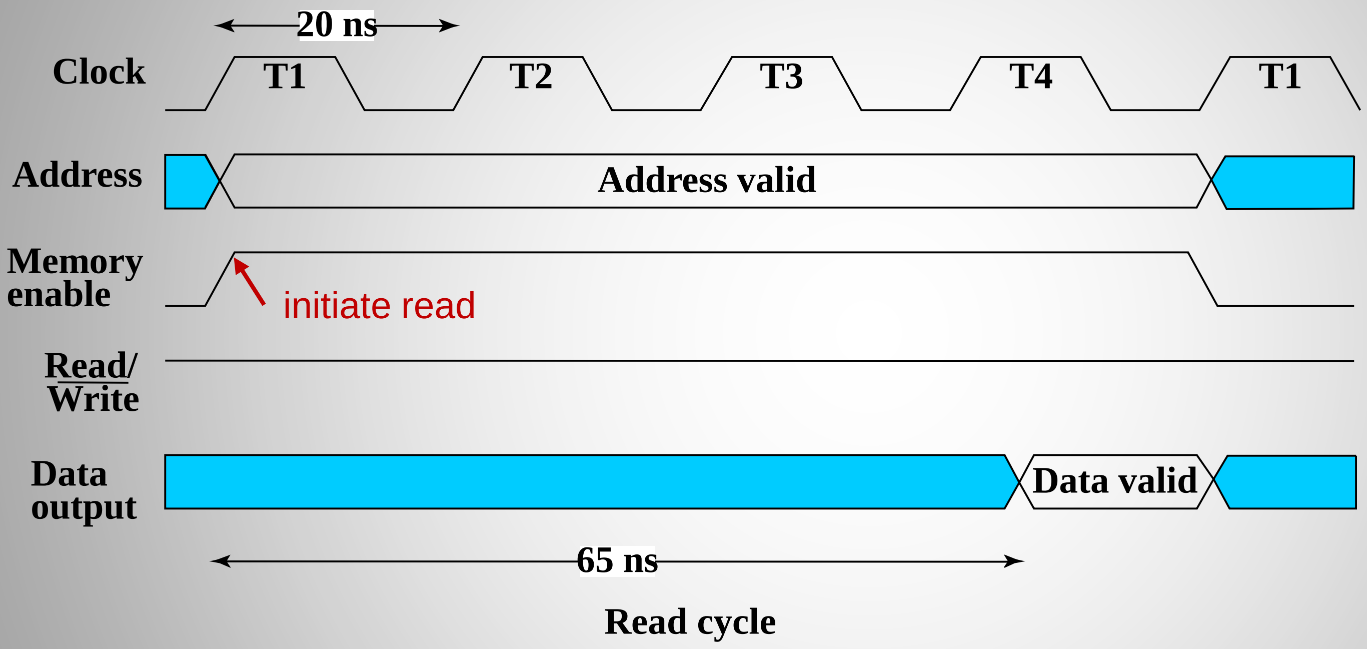
Memory Operations

- *Read*: reading a word *from memory*

1. Apply the address of the desired word to the address lines
2. Activate the Read control input

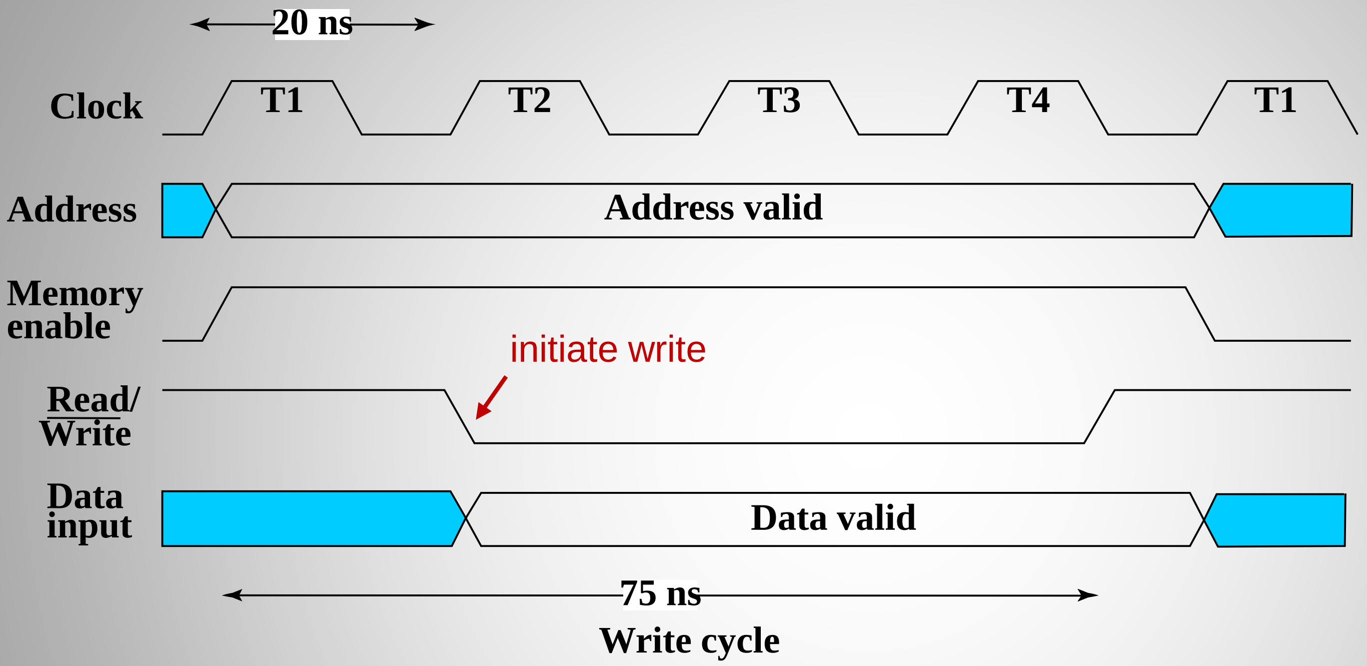
- *Access time* - maximum time from the application of the address to the appearance of the data at the data output lines

Read Cycle Timing Waveform



65ns → 4 clock cycles

Write Cycle Timing Waveform

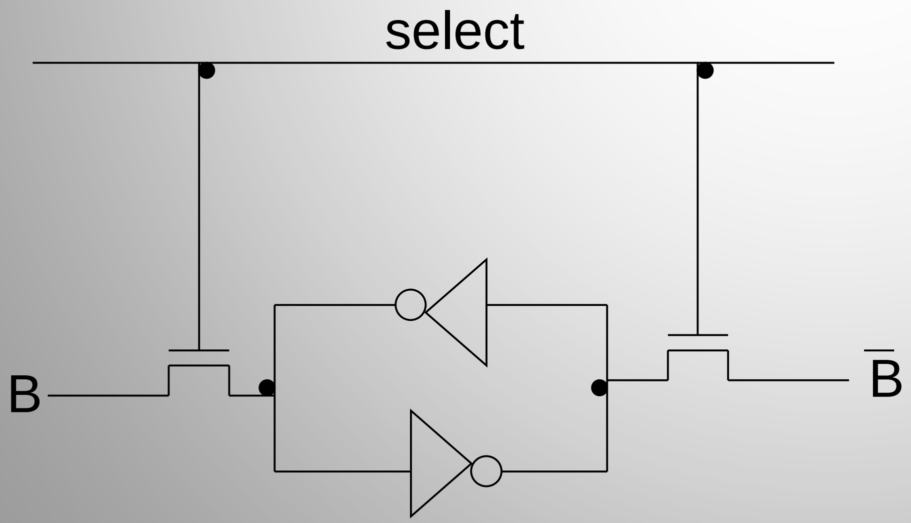


75ns → 4 clock cycles

- Address must already be ready before $Read/\overline{Write}$ changes to 0 to avoid destroying data in other location
- $Read/\overline{Write}$ must stay 0 long enough for write operation to complete
- Address must remain stable a short time after $Read/\overline{Write}$ goes high to avoid destroying data in other location

Static RAM (SRAM)

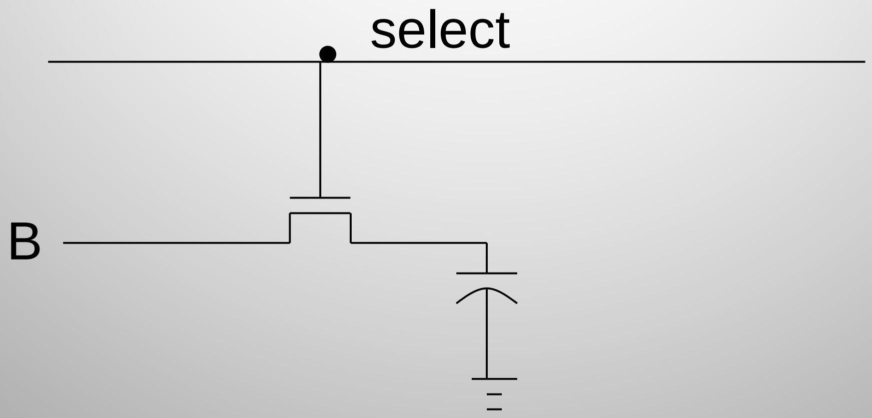
- Stored information remains valid as long as power is applied to the SRAM
- SRAM consists of internal latches that store the binary information
- A typical SRAM cell is made up of two inverters plus two pass transistors (→ a total of 6 transistors)



select	operation
0	hold
1	read/write

Dynamic RAM (DRAM)

- Data stored in DRAM must be refreshed periodically, or else it disappears
- Binary information is stored in the form of electric charges in DRAM
- A 1 bit is stored in a DRAM cell by charging the capacitor while a 0 bit is represented by an uncharged capacitor
- A typical DRAM cell consists of a capacitor plus one pass transistor



DRAM (cont'd)

- *Refresh* operation is needed periodically to restore all lost charges on the charged capacitors (loss due to reading and leakage)
- Refreshing is done by cycling through every word, reading and rewriting each periodically
- Both SRAM and DRAM are *volatile* memories which lose the stored information when power is turned off (c.f. *non-volatile* memories such as ROM, disk and tape which still retain the stored information when power is off)

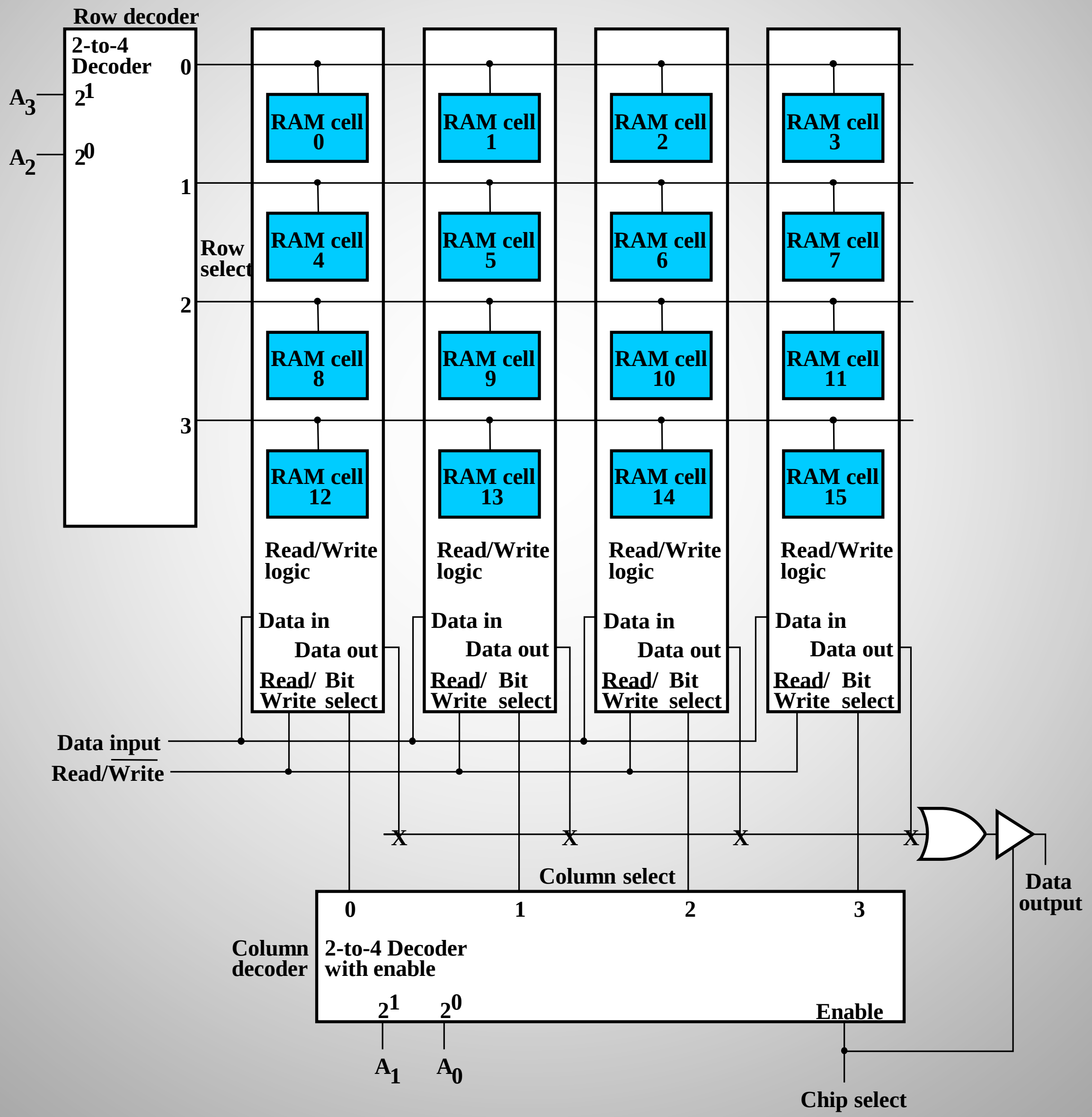
SRAM vs DRAM

- SRAM has shorter read and write cycle times than DRAM
- DRAM offers larger storage capacity in the same area than SRAM
- Typical applications:
  - Cache ↔ SRAM
  - Main memory ↔ DRAM

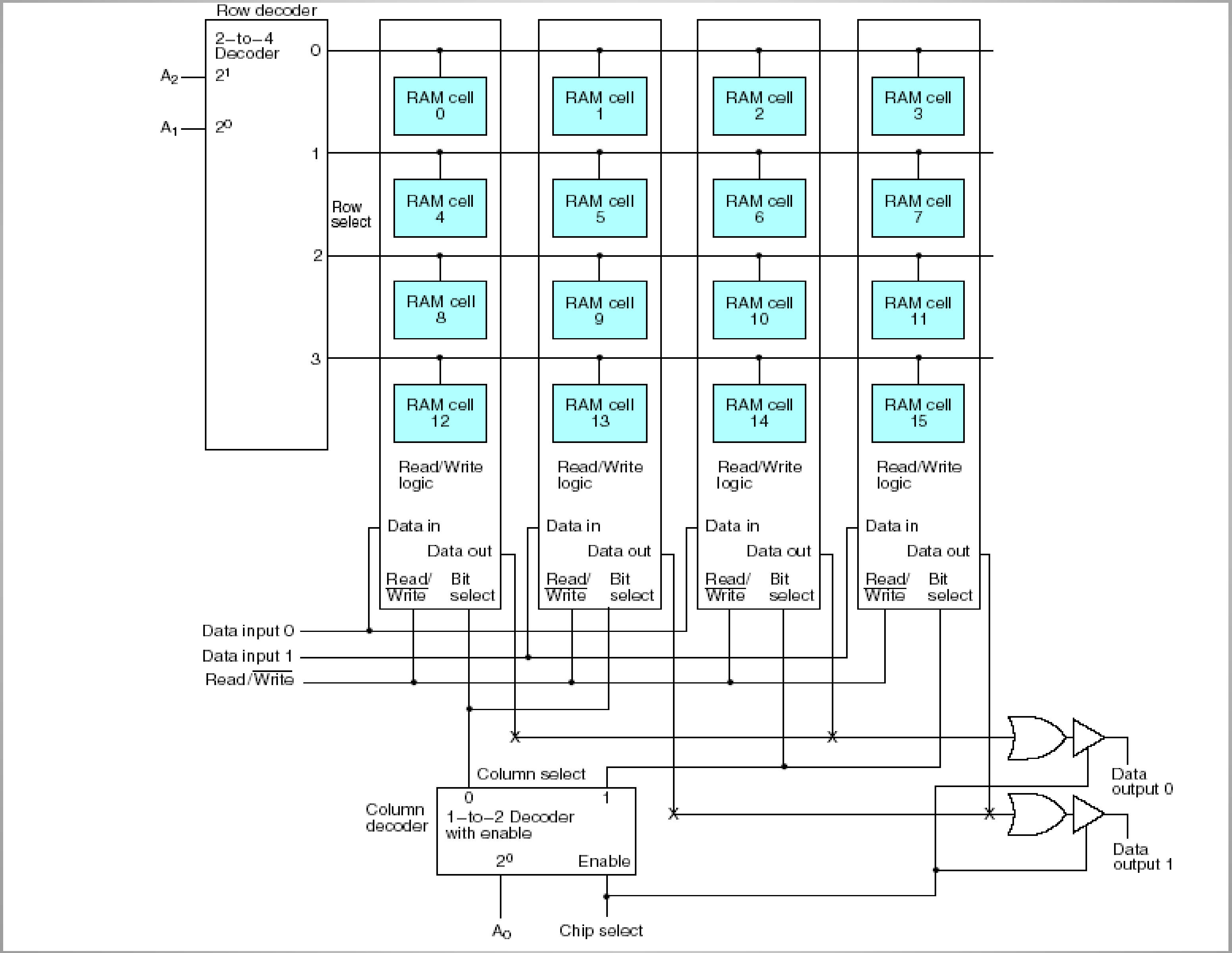
RAM Chip

- *RAM bit slice*: a set of RAM cells with associated read and write circuits
- A typical RAM chip consists of
  - one or more RAM bit slices
  - circuitry for address decoding
  - a three-state buffer for each output line

16x1 SRAM Using a 4x4 RAM Cell Array



8x2 SRAM Using a 4x4 RAM Cell Array



Making Wider Memory from RAM ICs

e.g. Construct a 64Kx16 RAM using 64Kx8 RAM ICs

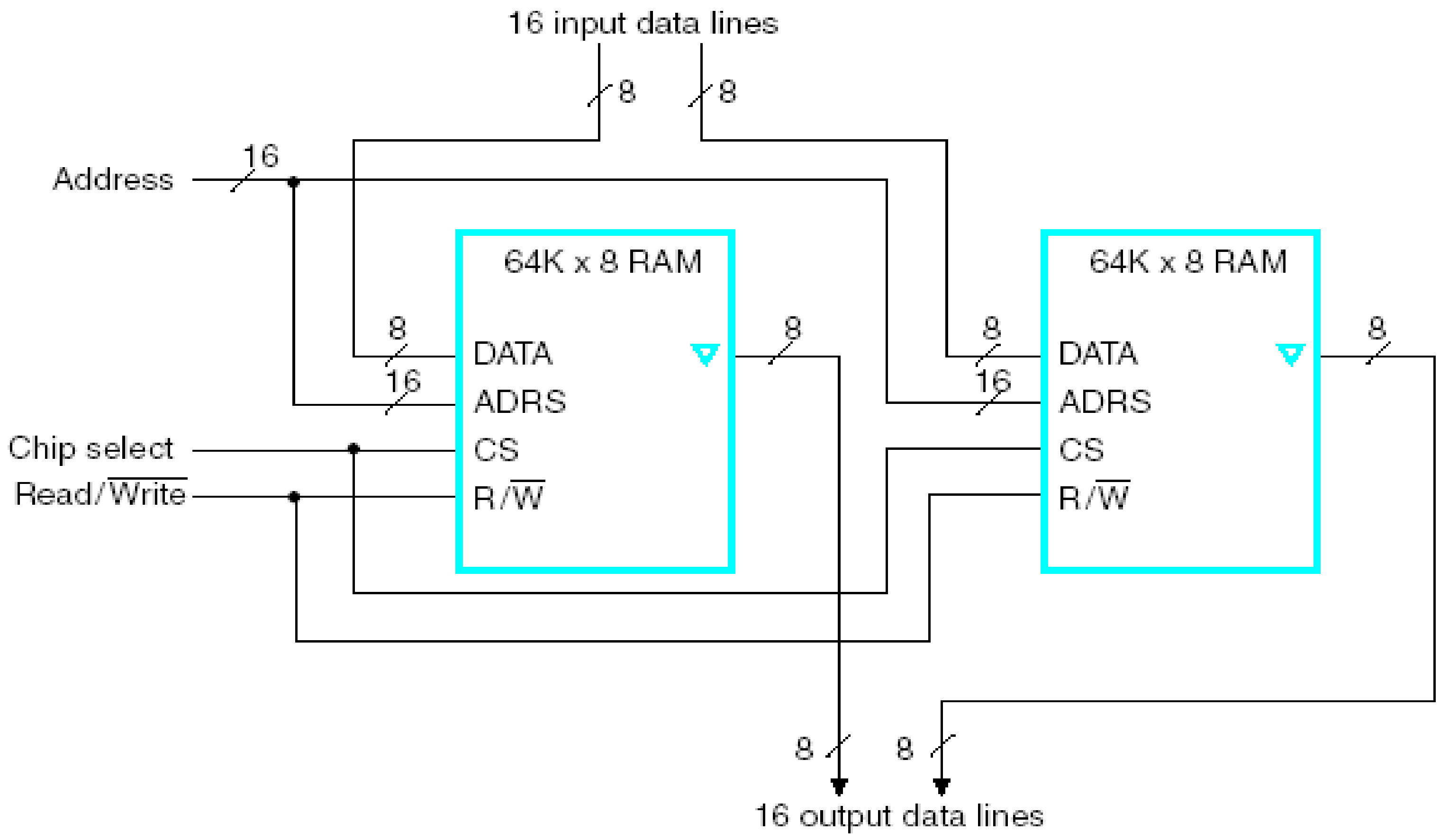
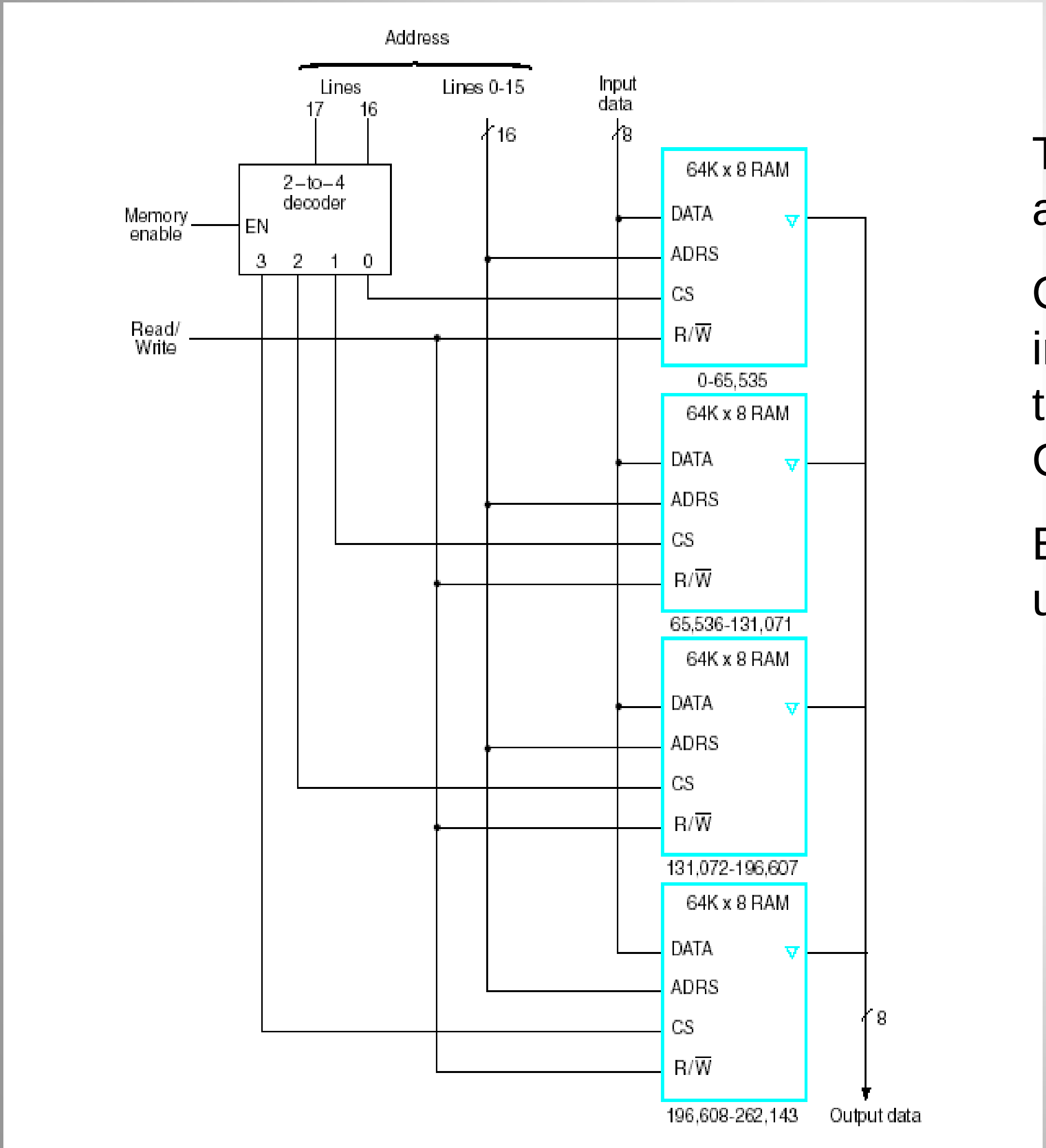


Fig. 6-18 Block Diagram of a 64K × 16 RAM

Making Larger Memory from RAM ICs

e.g. Construct a 256Kx8 RAM using 64Kx8 RAM ICs



The output lines of the 4 RAM ICs are connected to a common bus.

Q. Can you explain why the high-impedance output state provided by a tri-state buffer at an output line when CS=0 is important?

Exercise. Construct a 2Kx8 RAM using 1Kx4 RAM ICs.

補充教材

Error Detection And Correction

- Improve the reliability of a memory unit
- A simple error detection scheme
  - a parity bit
  - a single bit error can be detected, but cannot be corrected
- An error-correction code
  - generates multiple parity check bits
  - the check bits generate a unique pattern, called a syndrome
  - the specific bit in error can be identified

Hamming Code

- *k* parity bits are added to an *n*-bit data word
  - **($2^k - 1 \geq n + k$)**
  - 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

Example: 8-bit data word 11000100

- Include 4 parity bits and the 8-bit word ⇒ 12 bits

  $2^k - 1 \geq n + k$, $n = 8 \Rightarrow k = 4$

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

- Calculate the parity bits: even parity — assumption

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

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

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

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

- Store the 12-bit composite word in memory.

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

- When the 12 bits are read from the memory
  - Check bits are calculated

    C_1 = XOR of bits (1, 3, 5, 7, 9, 11)

    C_2 = XOR of bits (2, 3, 6, 7, 10, 11)

    C_4 = XOR of bits (4, 5, 6, 7, 12)

    C_8 = XOR of bits (8, 9, 10, 11, 12)

  - If no error has occurred

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

$\Rightarrow \mathbf{C} = C_8C_4C_2C_1 = 0000$

- One-bit error
  - error in bit 1
    - C_1 = XOR of bits (1, 3, 5, 7, 9, 11) = 1
    - C_2 = XOR of bits (2, 3, 6, 7, 10, 11) = 0
    - C_4 = XOR of bits (4, 5, 6, 7, 12) = 0
    - C_8 = XOR of bits (8, 9, 10, 11, 12) = 0
    - $C_8C_4C_2C_1$ = 0001
  - error in bit 5
    - $C_8C_4C_2C_1$ = 0101
- Two-bit error
  - errors in bits 1 and 5
    - $C_8C_4C_2C_1$ = 0100

7.5 Read-Only Memory

- Store permanent binary information
- 2^k x n ROM
  - k address input lines
  - enable input(s)
  - three-state outputs

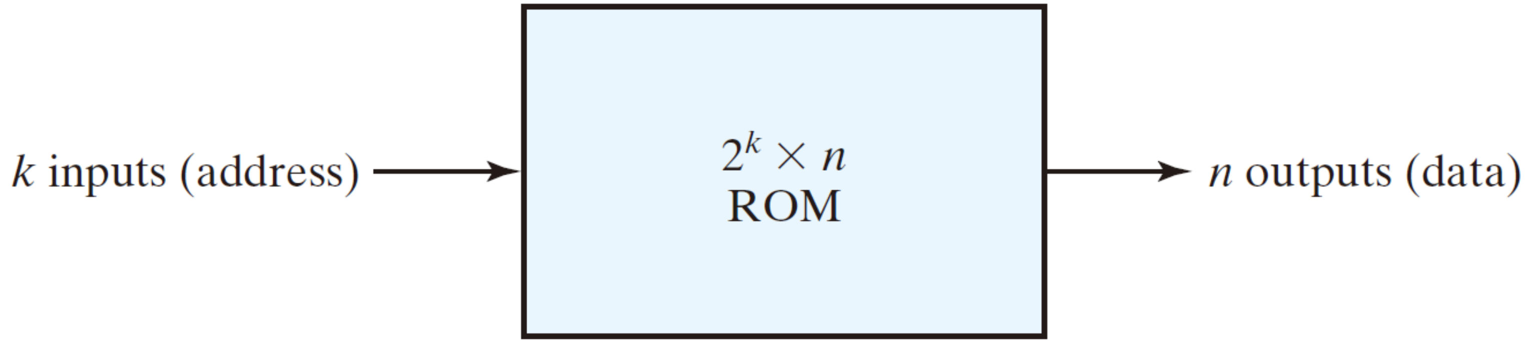


FIGURE 7.9
ROM block diagram

- 32 x 8 ROM
  - 5-to-32 decoder
  - 8 OR gates
    - each has 32 inputs
  - 32x8 internal programmable connections

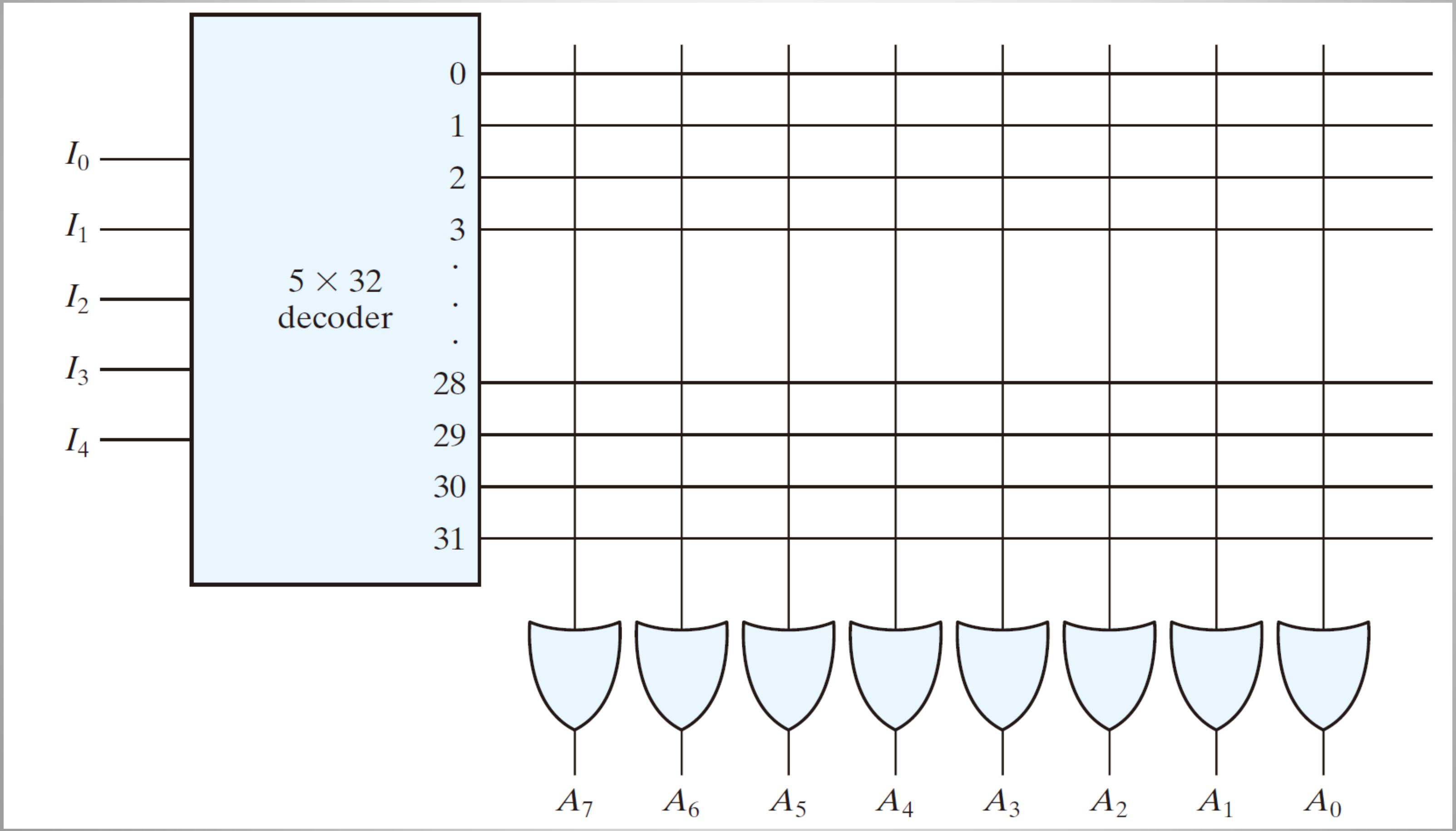


FIGURE 7.10
Internal logic of a 32 × 8 ROM

- programmable interconnections
  - close (two lines are connected)
  - or open
  - A fuse that can be blown by applying a high voltage pulse

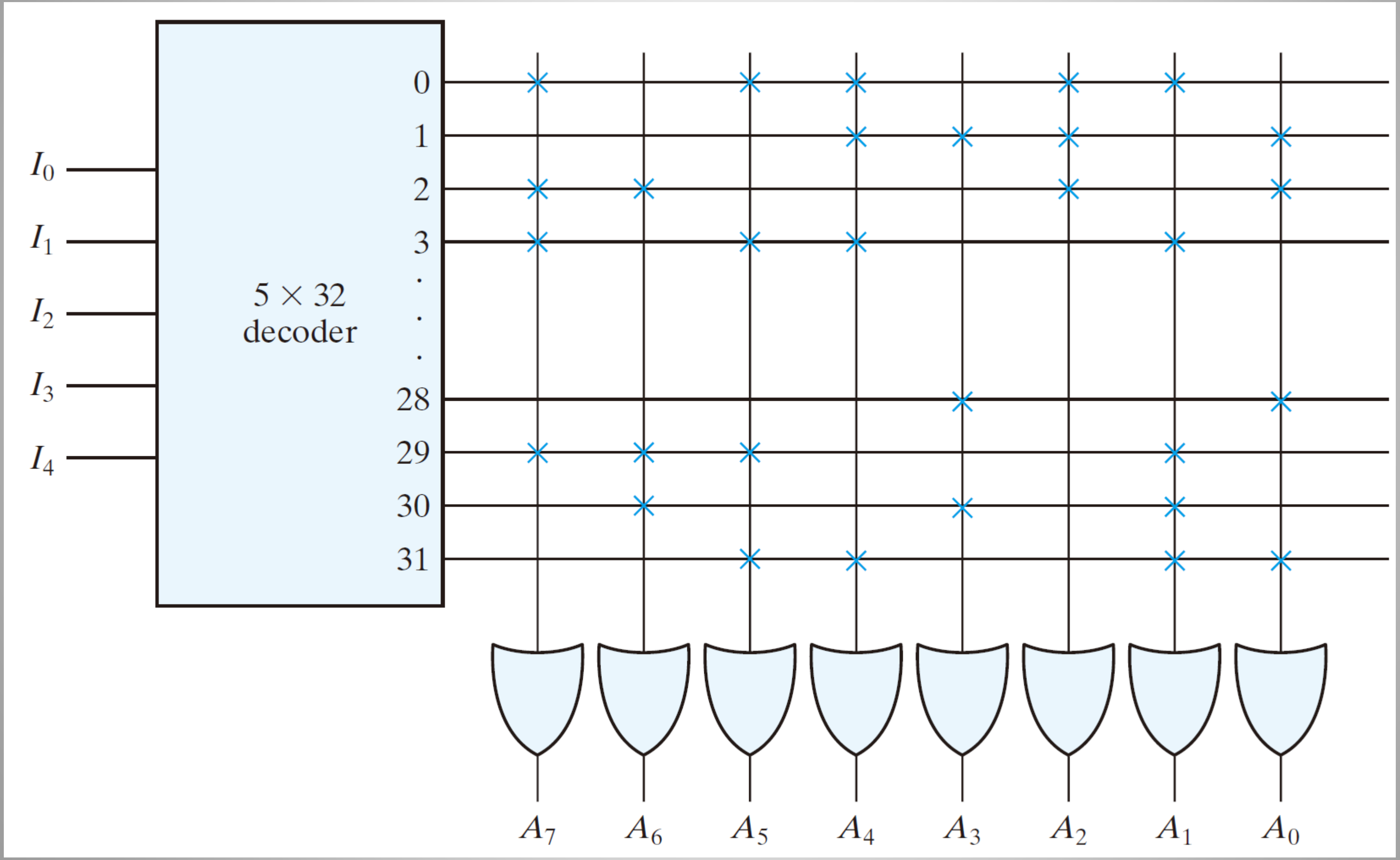


FIGURE 7.11
Programming the ROM according to Table 7.3

• ROM truth table (partial)

– an example

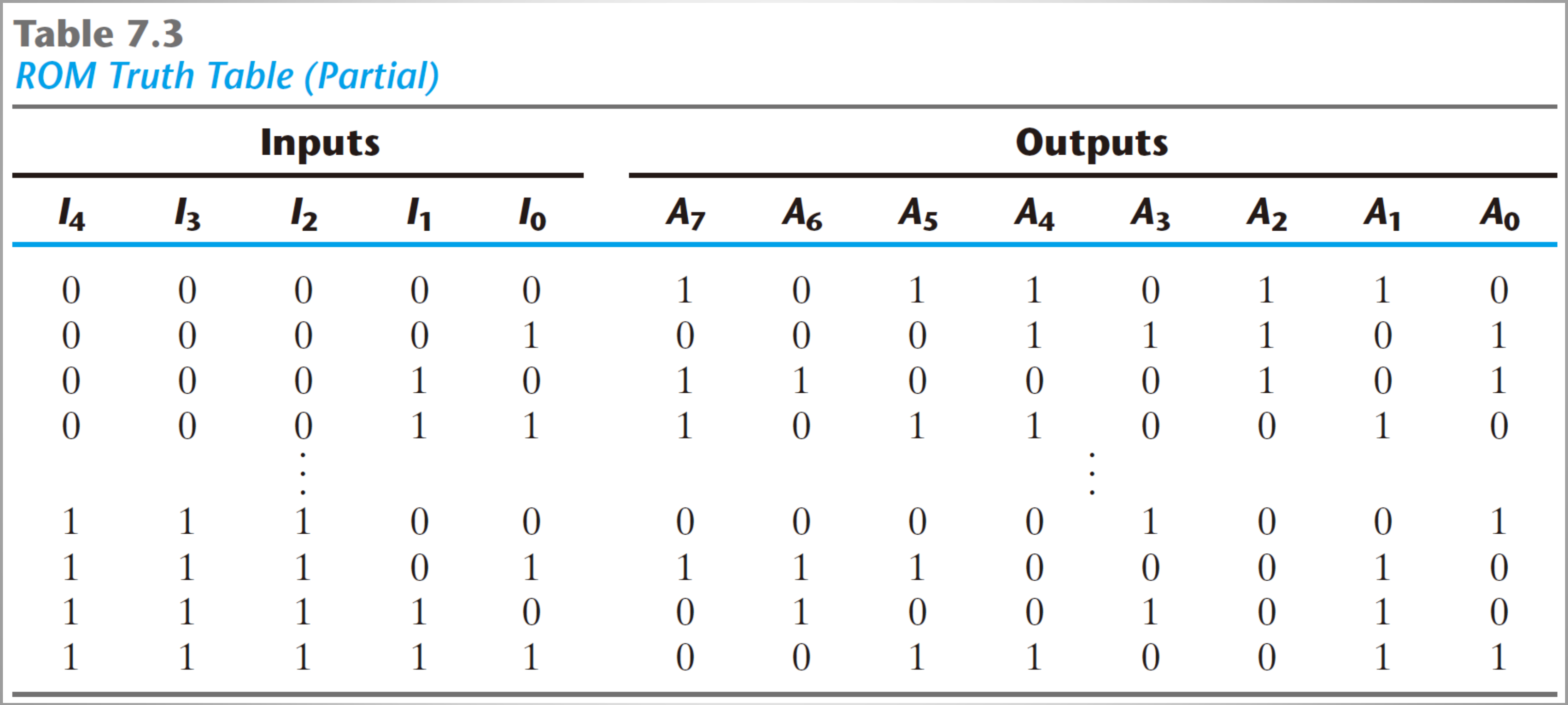


Table 7.3

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

Combinational Circuit Implementation

- ROM: a decoder + OR gates
  - sum of minterms
  - a Boolean function = sum of minterms
  - For an *n*-input, *m*-output combinational ckt
    $\Rightarrow 2^n \times m$ ROM
- Design procedure:

1. Determine the size of ROM
2. Obtain the programming truth table of the ROM
3. The truth table = the fuse pattern

• Example 7-1

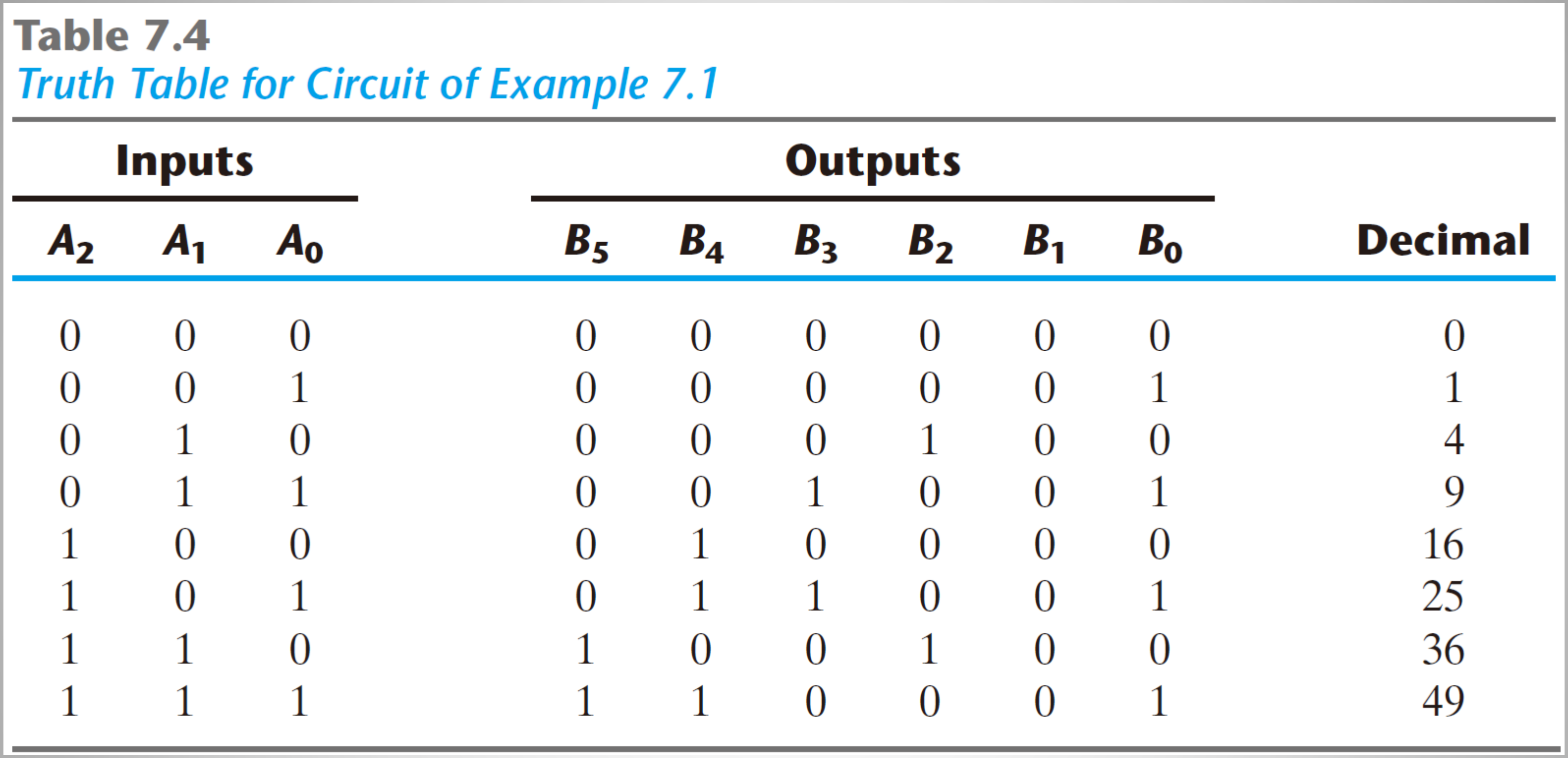
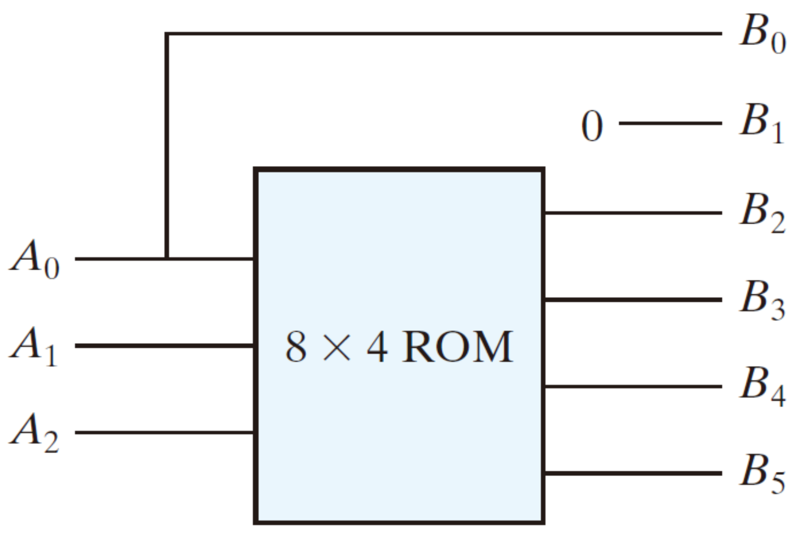


Table 7.4
Truth Table for Circuit of Example 7.1

Inputs			Outputs						
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

- 3 inputs, 6 outputs
- B_1=0
- B_0=A_0
- 8x4 ROM

- ROM implementation
  - Truth table



(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

Types of ROM

- Types of ROM
  - mask programming ROM
    - IC manufacturers
    - is economical only if large quantities
  - PROM: Programmable ROM
    - fuses
    - universal programmer
  - EPROM: erasable PROM
    - floating gate
    - ultraviolet light erasable
  - EEPROM: electrically erasable PROM
    - longer time is needed to write
    - flash ROM
    - limited times of write operations

Combinational PLDs

- Programmable two-level logic
  - an AND array and an OR array

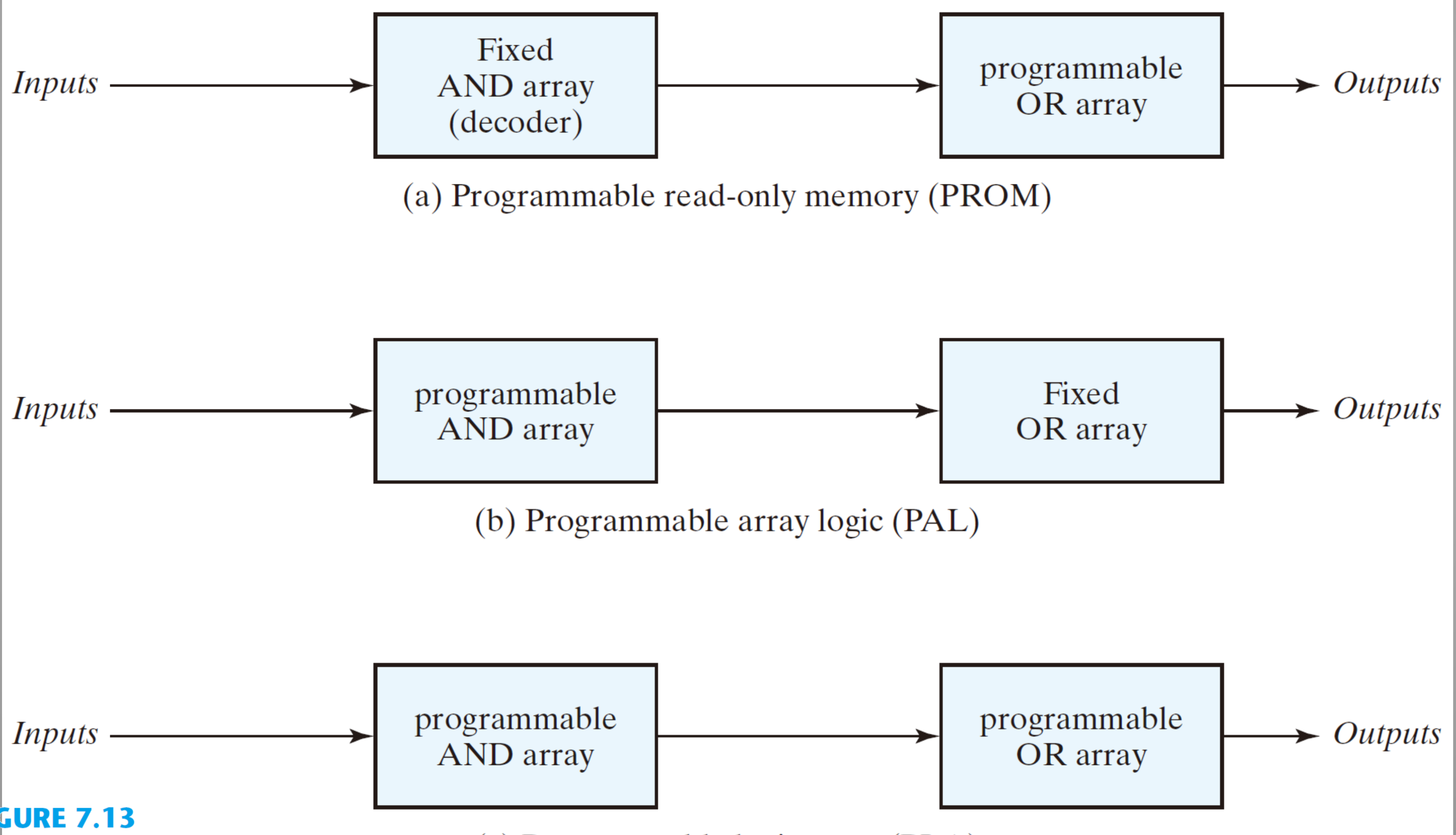
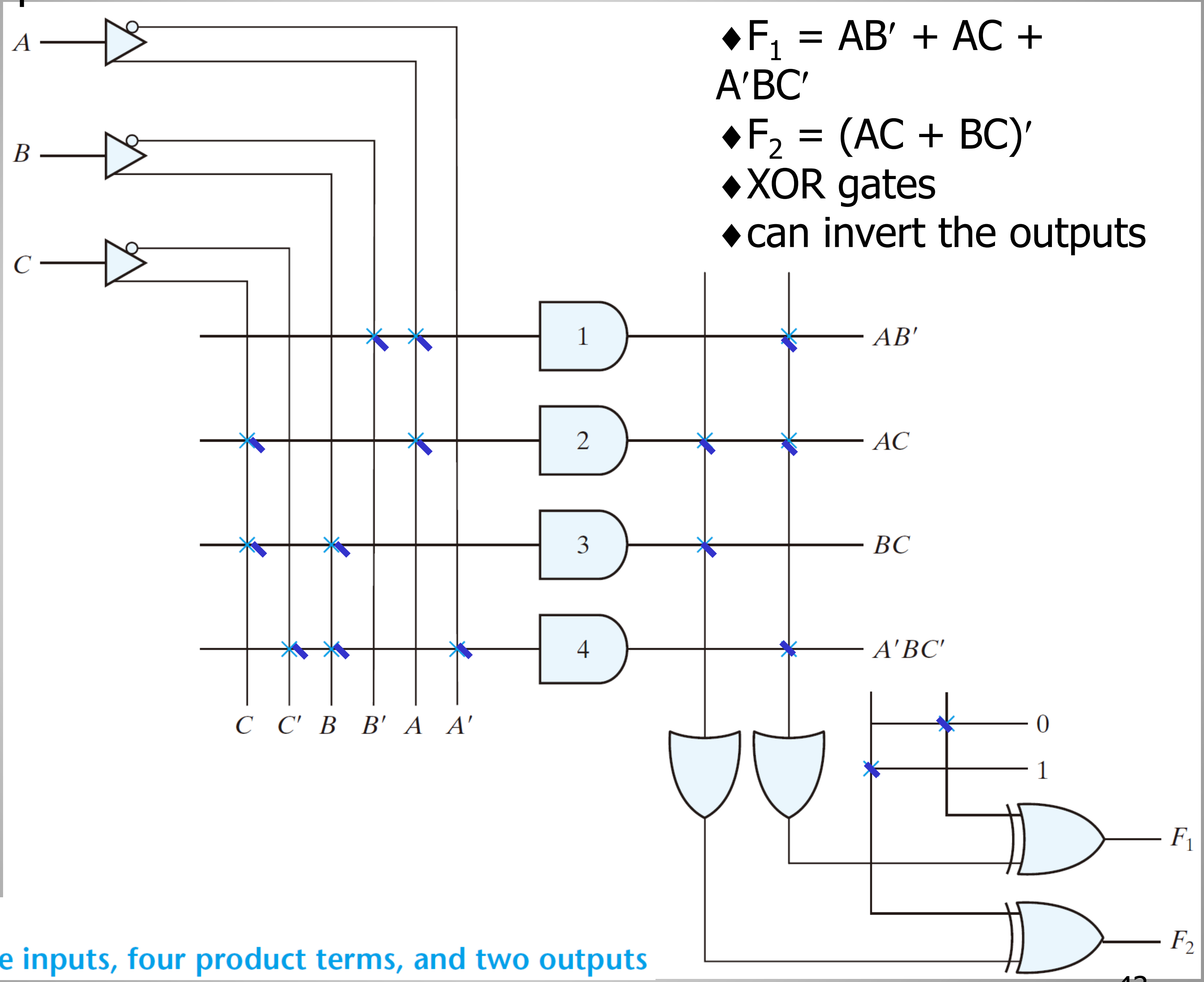


FIGURE 7.13
Basic configuration of three PLDs

7.6 Programmable Logic Array

- PLA
  - an array of programmable AND gates
    - can generate any product terms of the inputs
  - an array of programmable OR gates
    - can generate the sums of the products
  - more flexible than ROM
  - use less circuits than ROM
    - only the needed product terms are generated

- An example



- ♦ $F_1 = AB' + AC + A'BC'$
- ♦ $F_2 = (AC + BC)'$
- ♦ XOR gates
- ♦ can invert the outputs

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

- PLA programming table
  - specify the fuse map

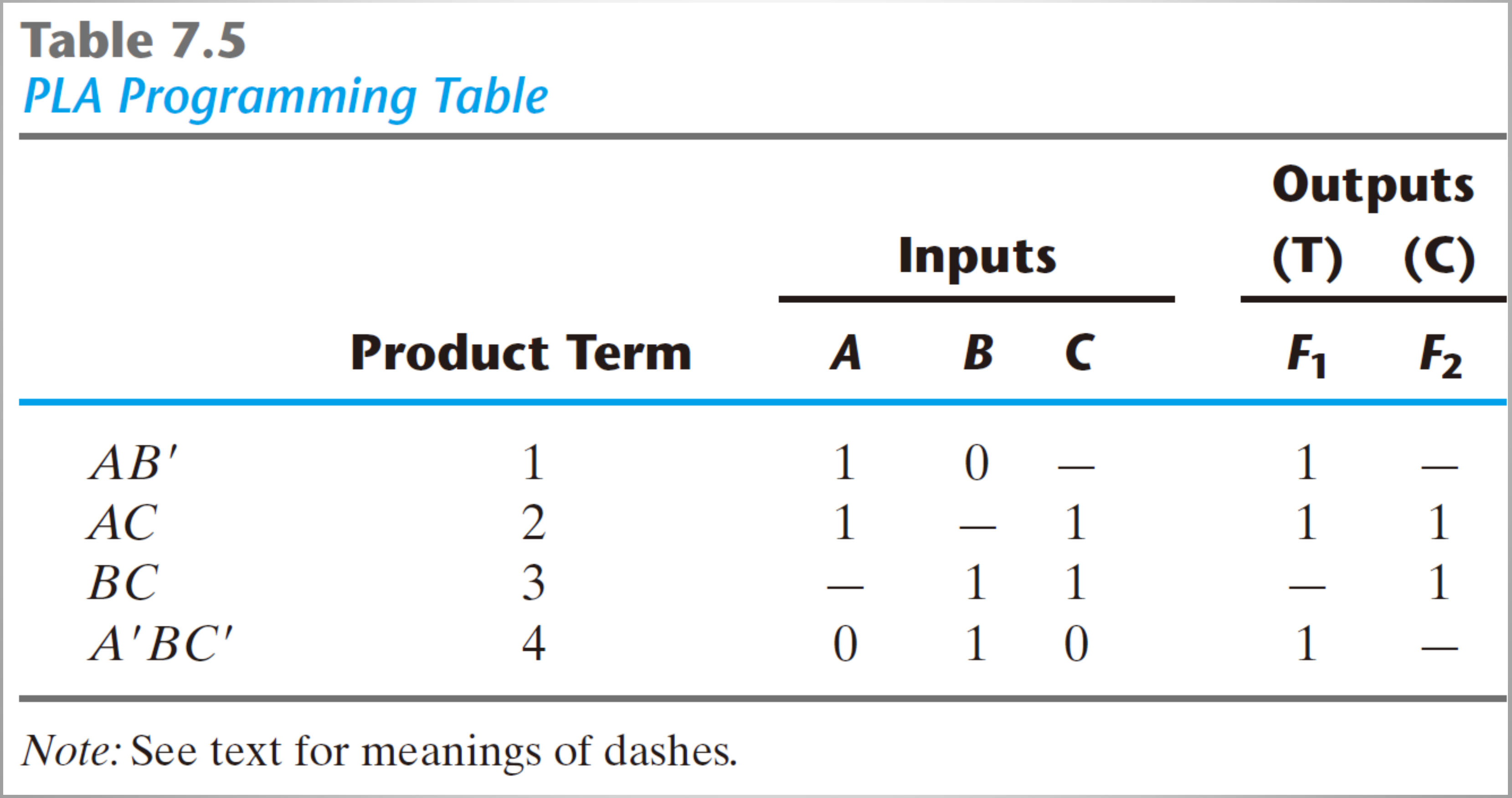


Table 7.5
PLA Programming Table

		Inputs			Outputs	
					(T)	(C)
	Product Term	***A***	***B***	***C***	F_1	F_2
AB'	1	1	0	—	1	—
AC	2	1	—	1	1	1
BC	3	—	1	1	—	1
$A'BC'$	4	0	1	0	1	—

Note: See text for meanings of dashes.

- The size of a PLA
  - The number of inputs
  - The number of product terms (AND gates)
  - The number of outputs (OR gates)
- When implementing with a PLA
  - reduce the number of distinct product terms
  - the number of terms in a product is not important

Examples 7-2

- $F_1(A, B, C) = \Sigma\ (0, 1, 2, 4)$; $F_2(A, B, C) = \Sigma\ (0, 5, 6, 7)$
- both the true value and the complement of the function should be simplified to check

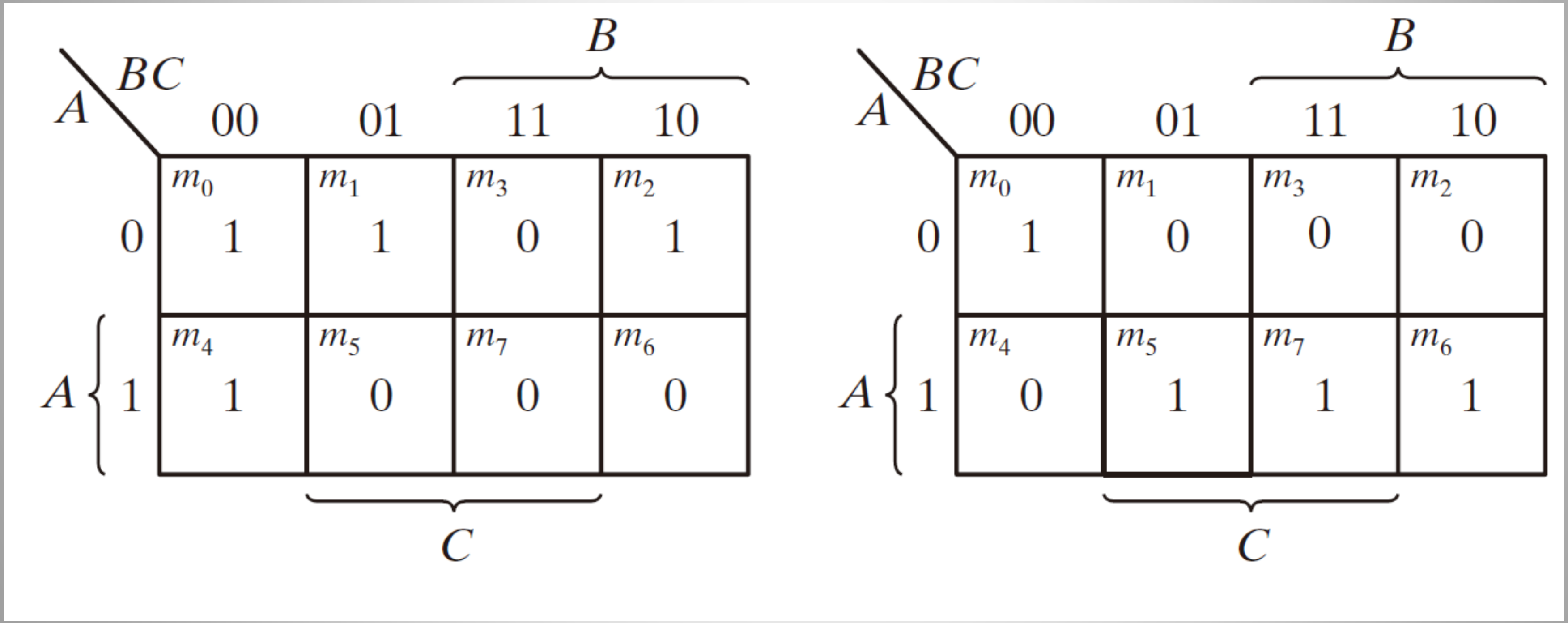
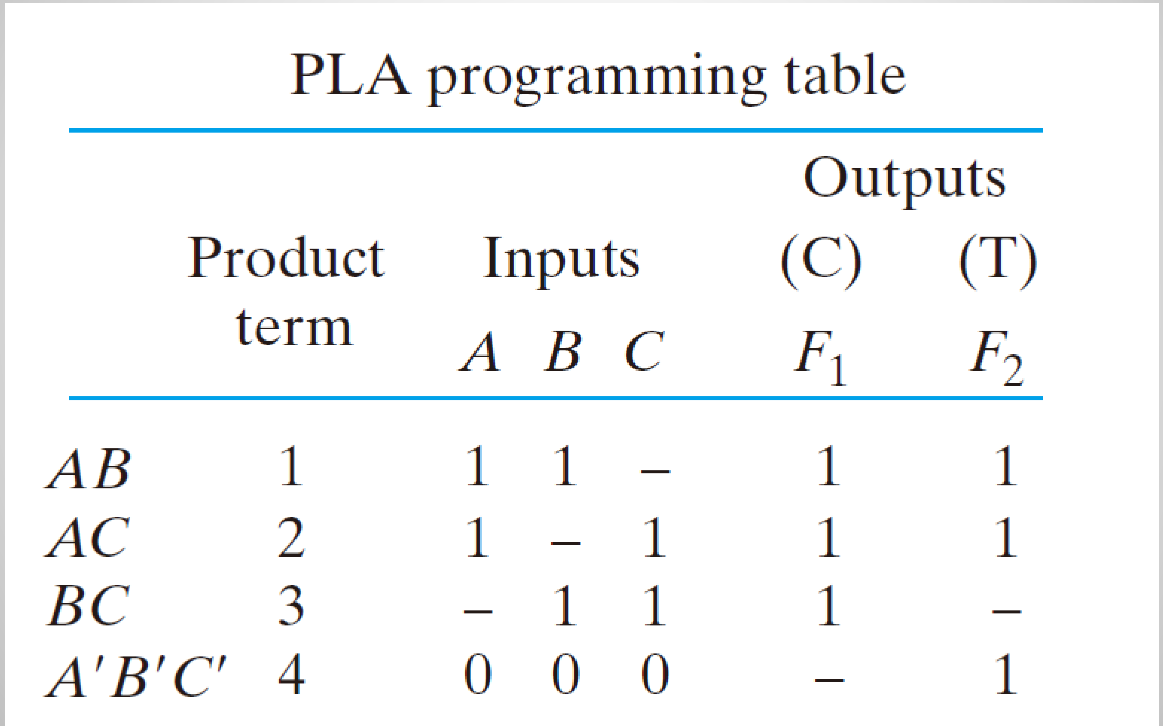


FIGURE 7.15
Solution to Example 7.1

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



PLA programming table

Product term		Inputs			Outputs (C)	Outputs (T)
		A	B	C	F_1	F_2
AB	1	1	1	–	1	1
AC	2	1	–	1	1	1
BC	3	–	1	1	1	–
$A'B'C'$	4	0	0	0	–	1

FIGURE 7.15
Solution to Example 7.1

Basic Xilinx Architecture

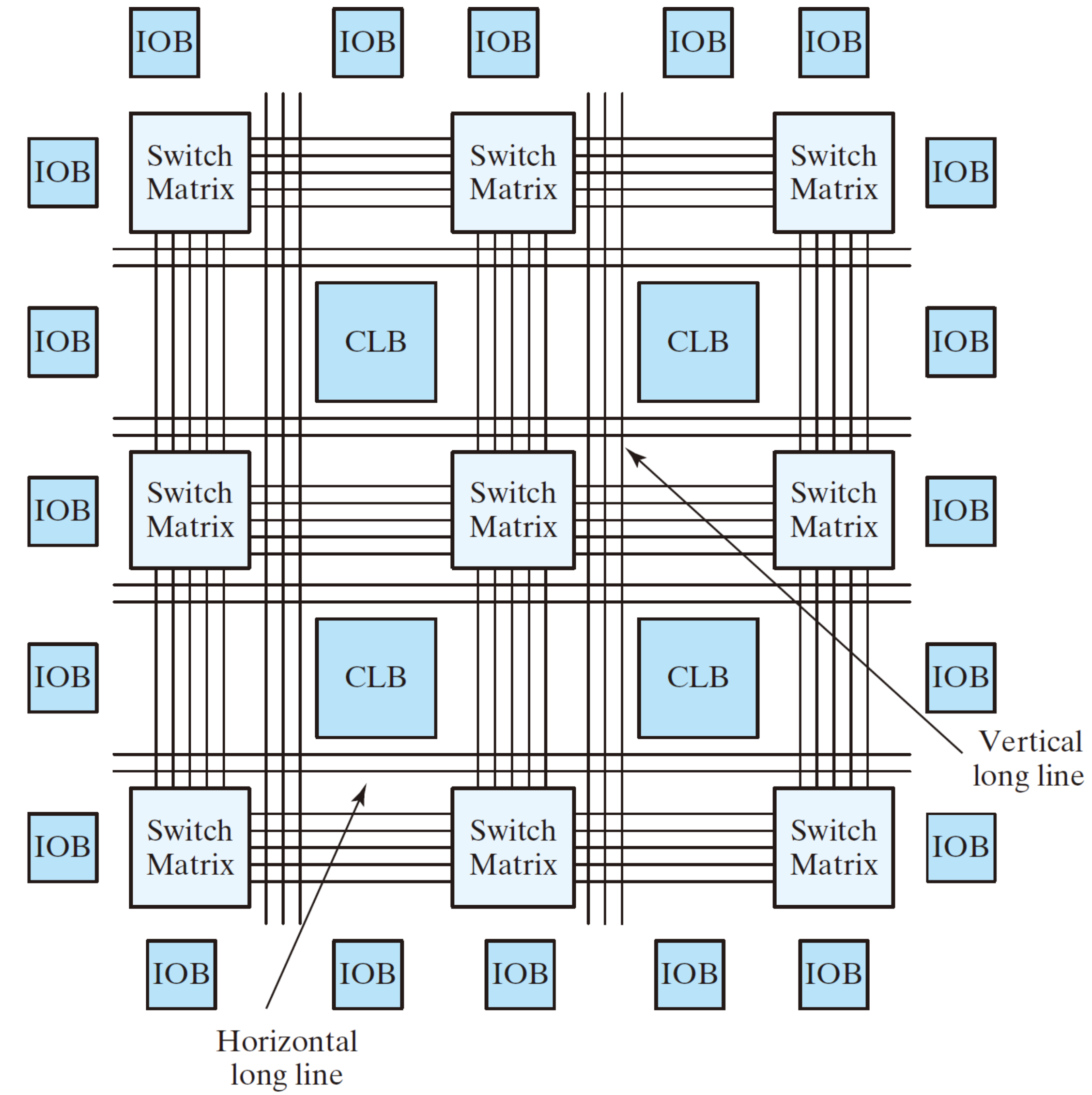


FIGURE 7.21
Basic architecture of Xilinx Spartan and predecessor devices

Xilinx Spartan II FPGAs

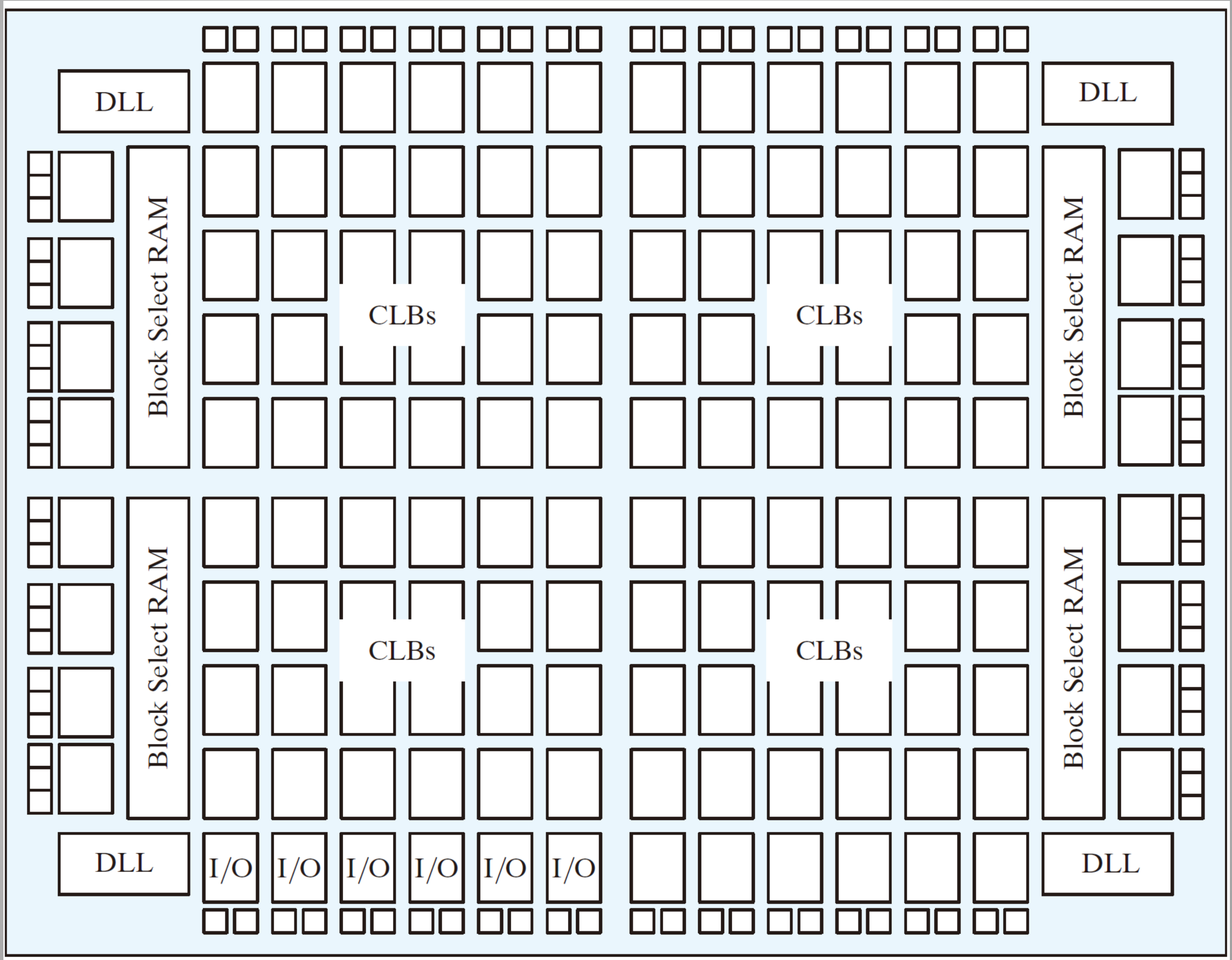


FIGURE 7.28
Spartan II architecture