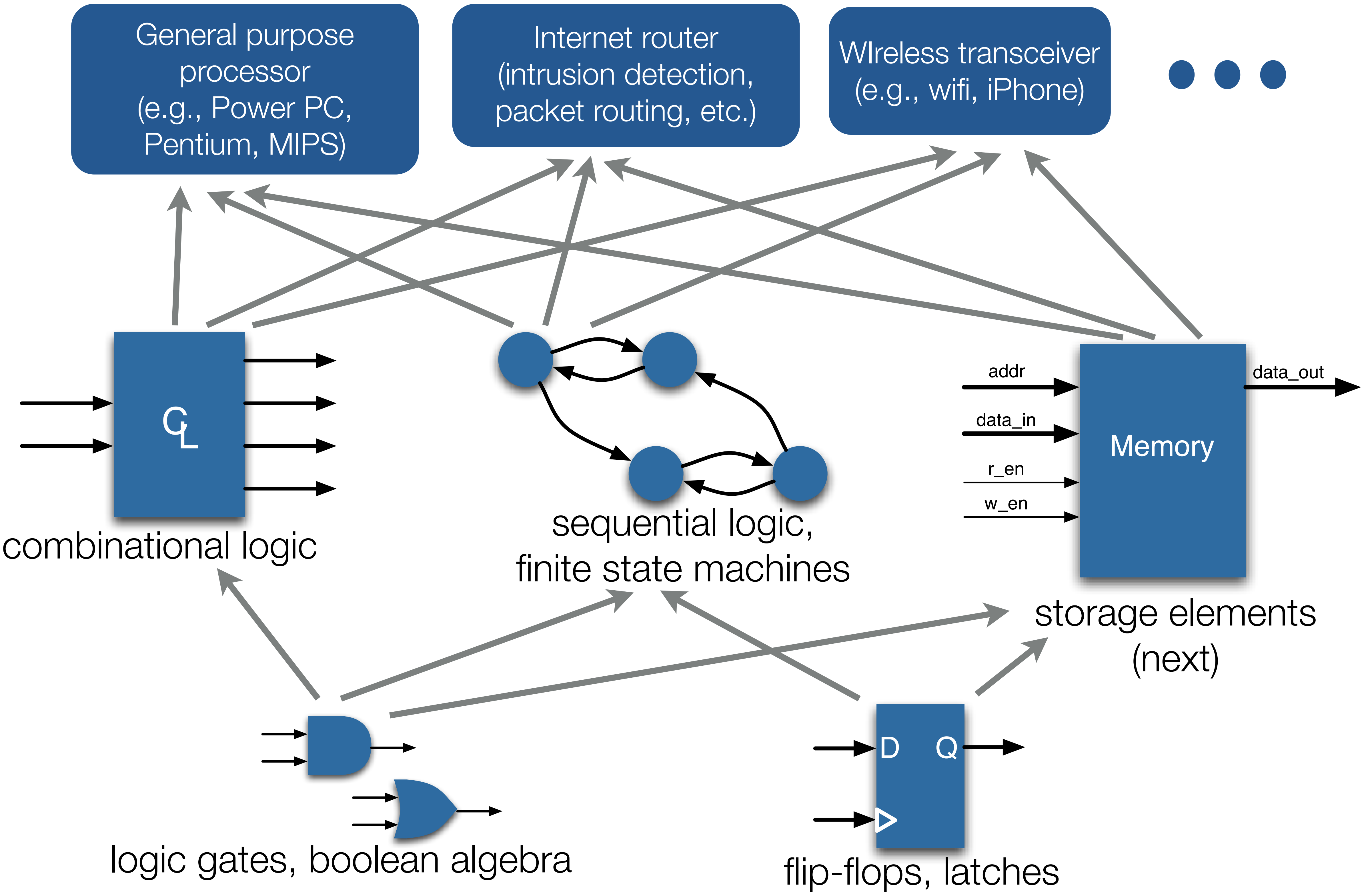
CSEE 3827: Fundamentals of Computer Systems

Storage

The big picture



General purpose processor (e.g., Power PC, Pentium, MIPS)

Internet router (intrusion detection, packet routing, etc.)

WIreless transceiver (e.g., wifi, iPhone)

combinational logic

sequential logic, finite state machines

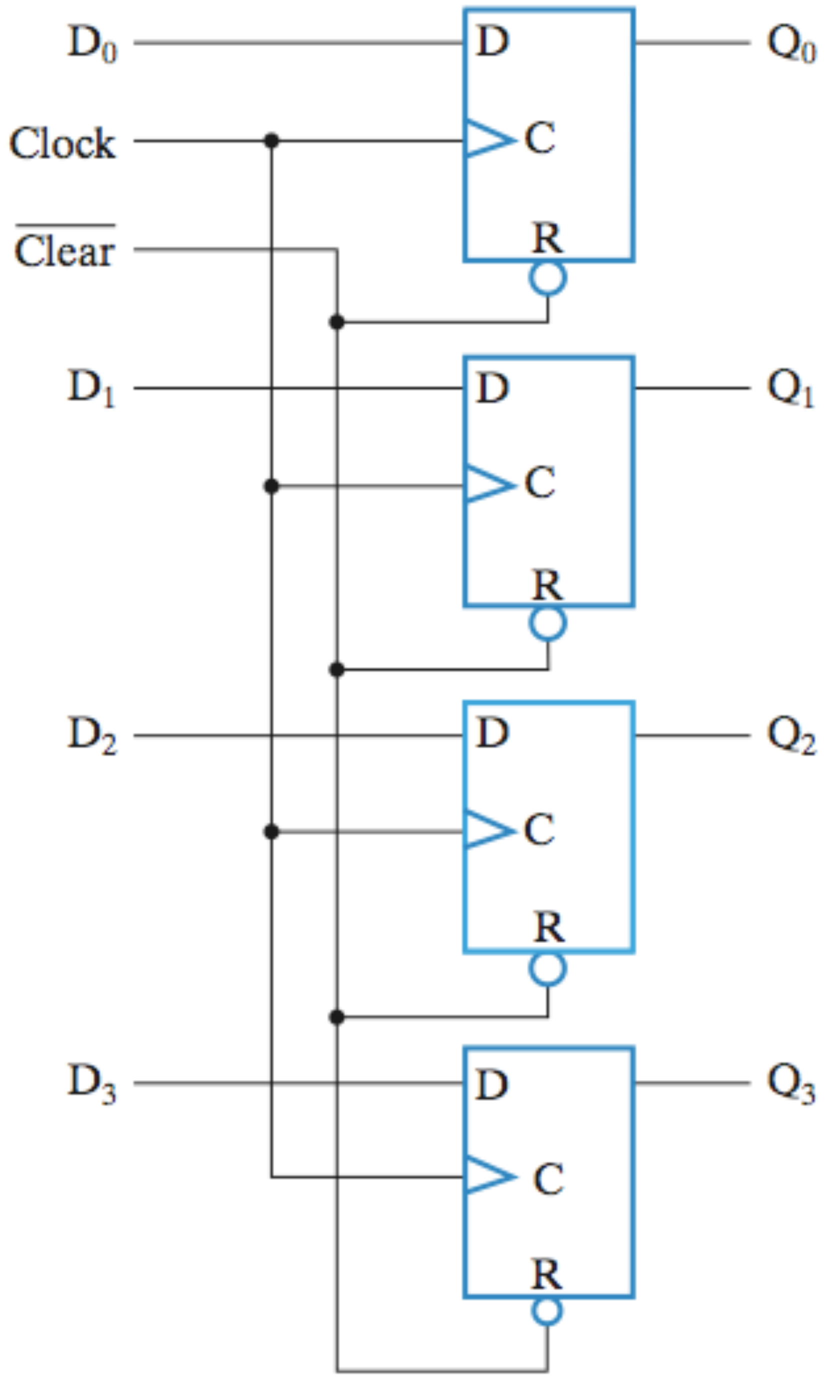
storage elements (next)

logic gates, boolean algebra

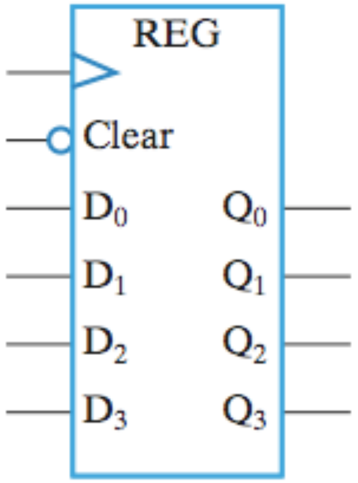
flip-flops, latches

Registers

A flip-flop can store 1 bit. A register is a set of n flip flops that stores n bits.



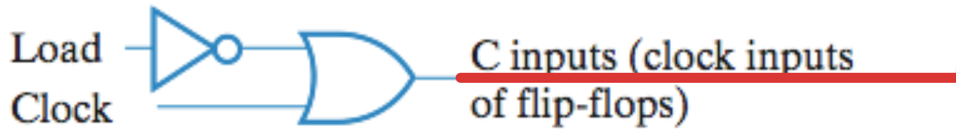
(a) Logic diagram



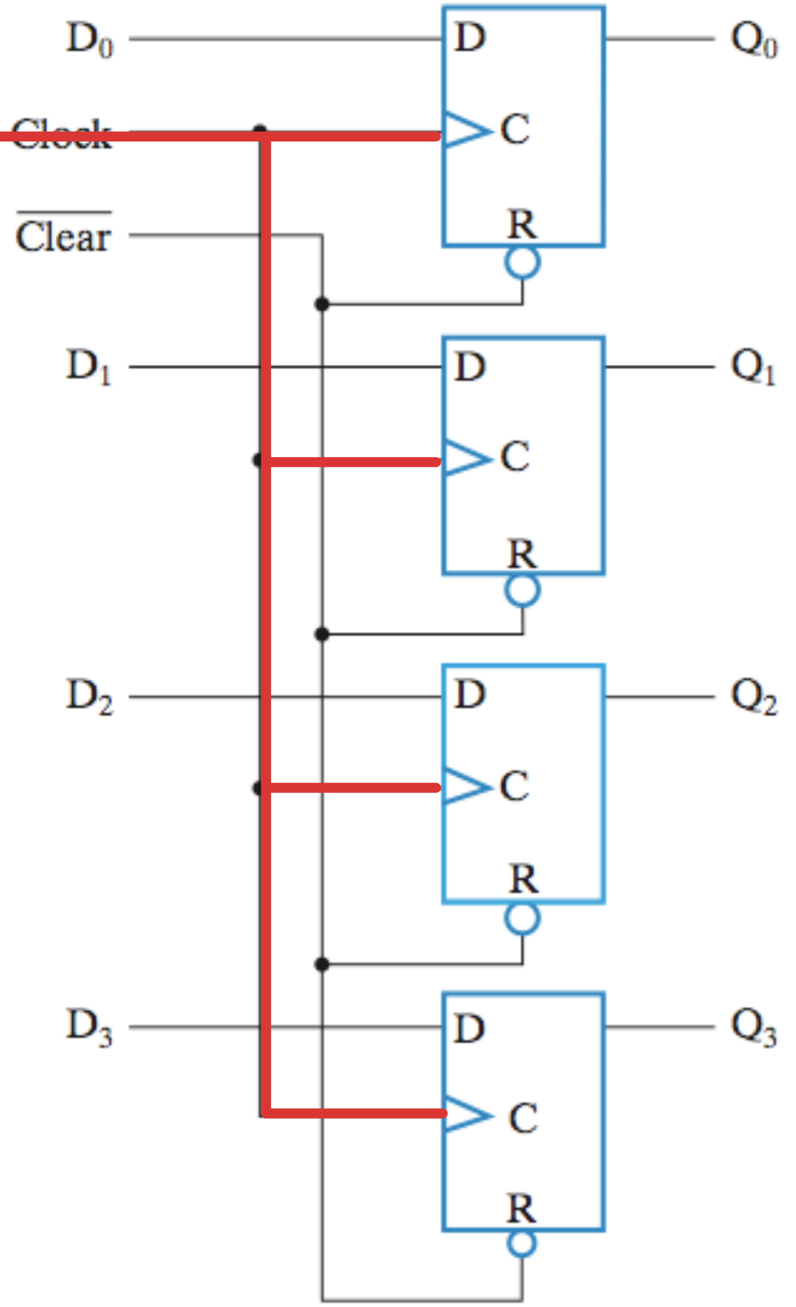
(b) Symbol

M. Morris Mano & Charles R. Kime
LOGIC AND COMPUTER DESIGN FUNDAMENTALS, 4e

Register w. load control input (v1)



(c) Load control input



(a) Logic diagram

Encapsulate logic inside register

Send clock signal to flip-flops only when Load=1

This technique is called clock gating

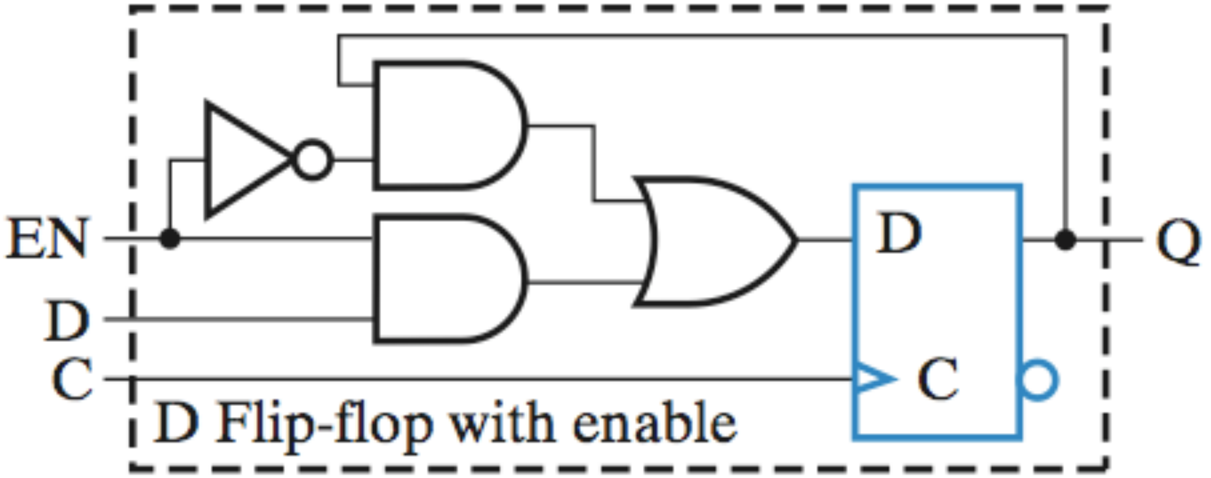
M. Morris Mano & Charles R. Kime
LOGIC AND COMPUTER DESIGN FUNDAMENTALS, 4e

Register w. load control input (v2)

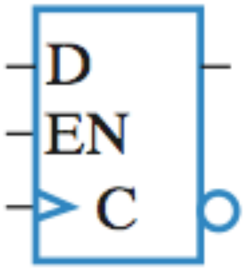
Encapsulate logic inside flip-flop

EN signal selects between current value of register (Q) or new value (D)

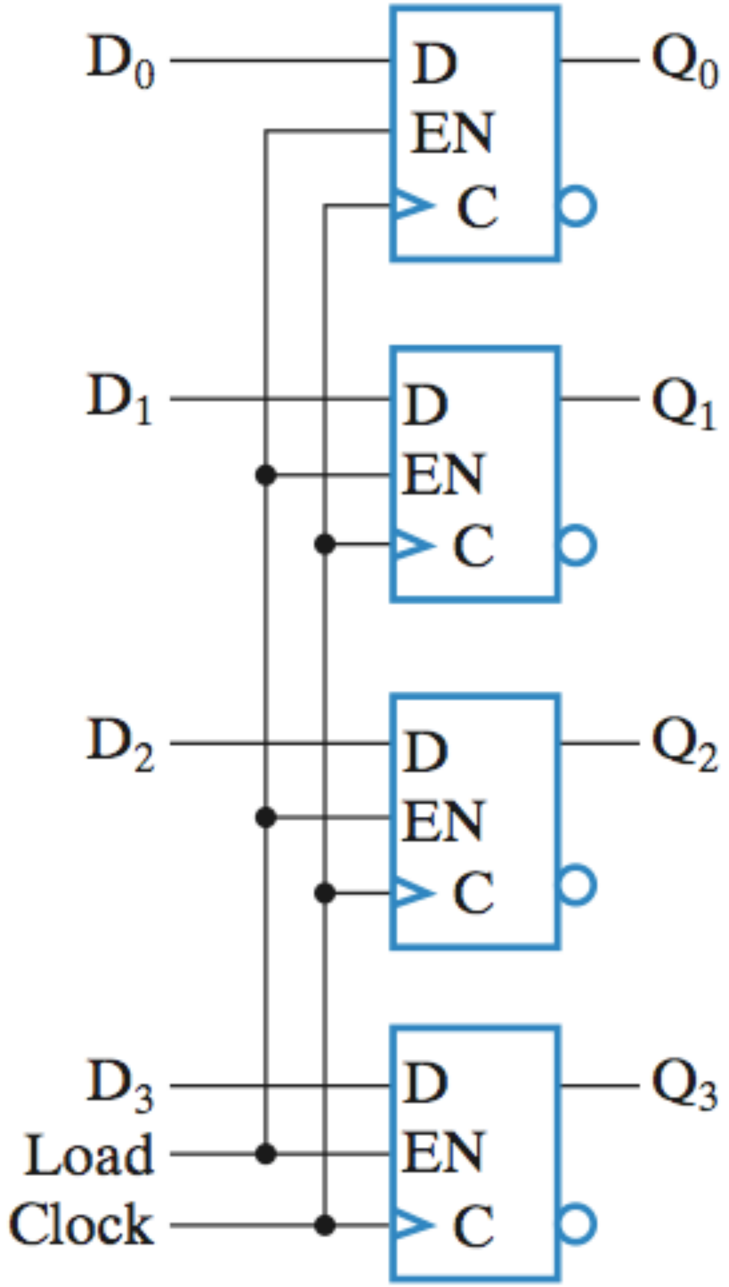
Preferable to v1 as it leaves clock signal unadultered.



(a)



(b)

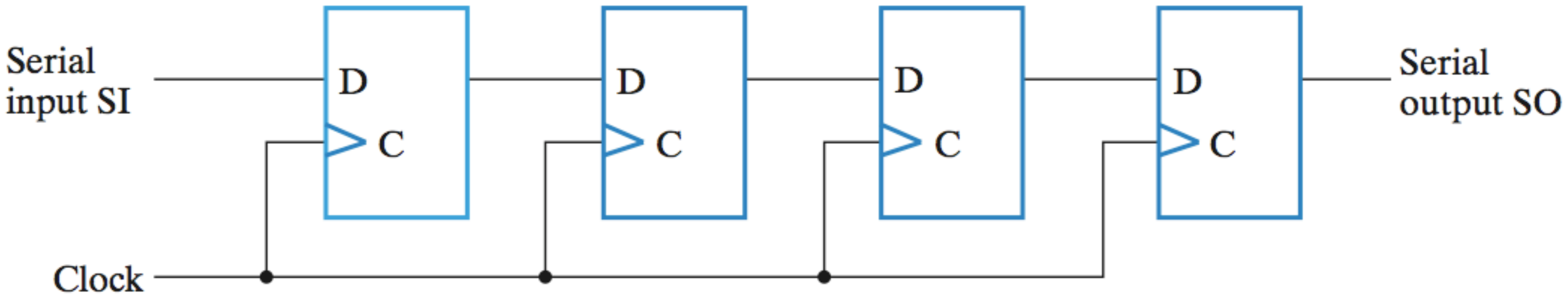


(c)

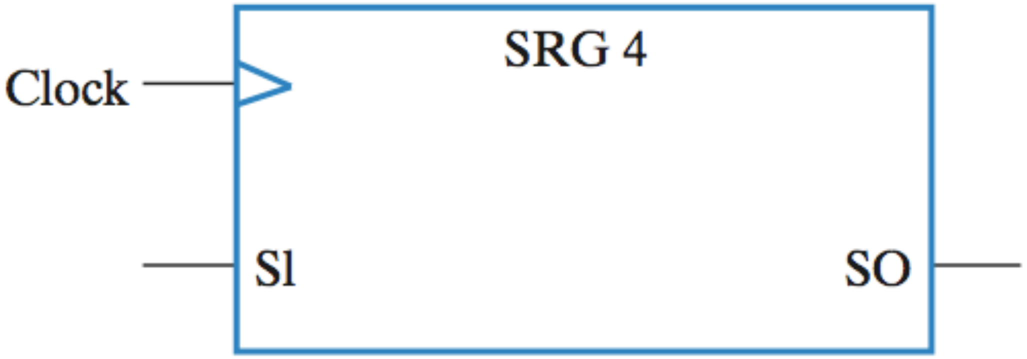
M. Morris Mano & Charles R. Kime
LOGIC AND COMPUTER DESIGN FUNDAMENTALS, 4e

Shift register

A register capable of shifting bits laterally in one or both directions

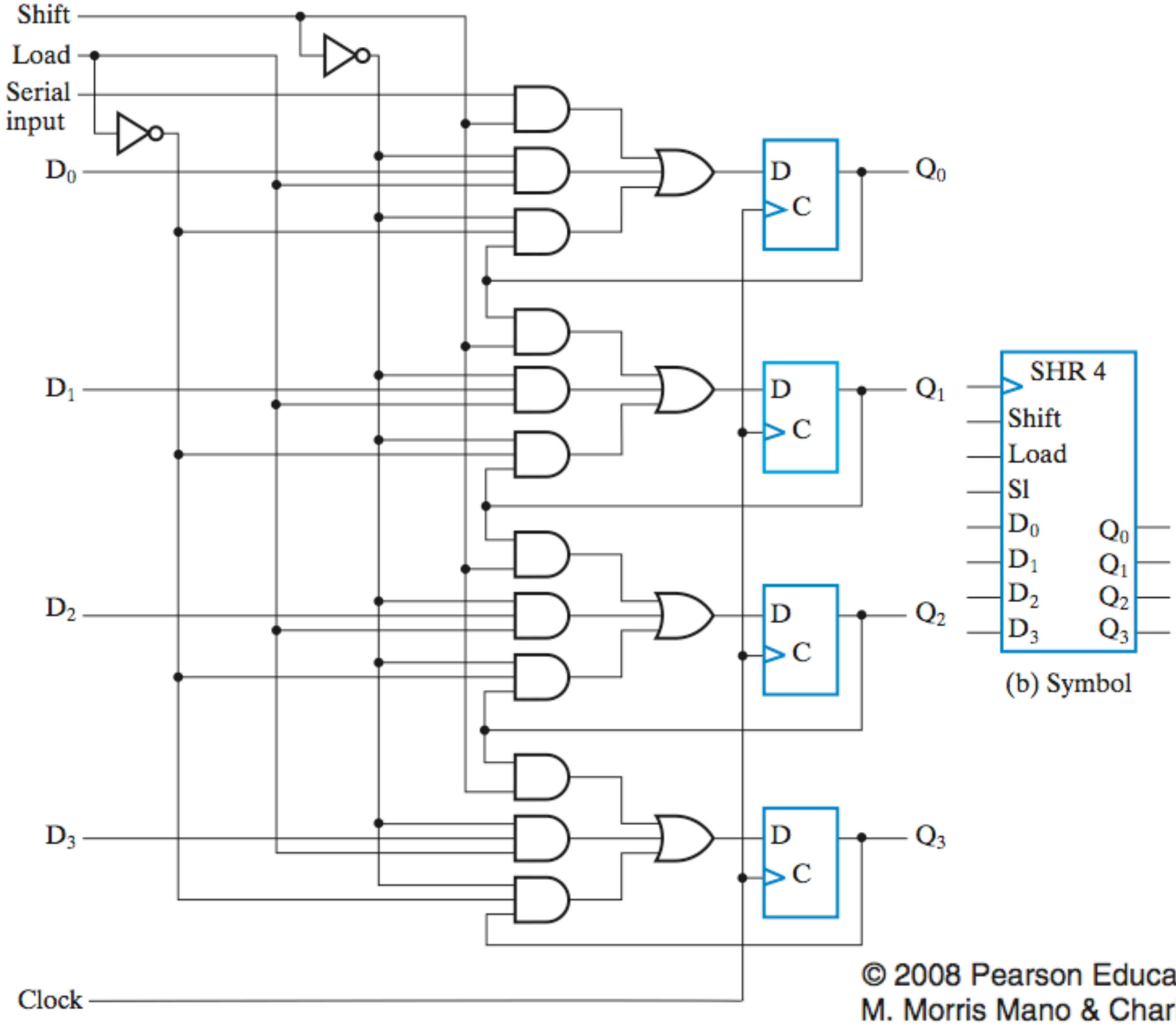


(a) Logic diagram



(b) Symbol

Shift register w. parallel load



(b) Symbol

Three modes:

Serial input

Parallel input

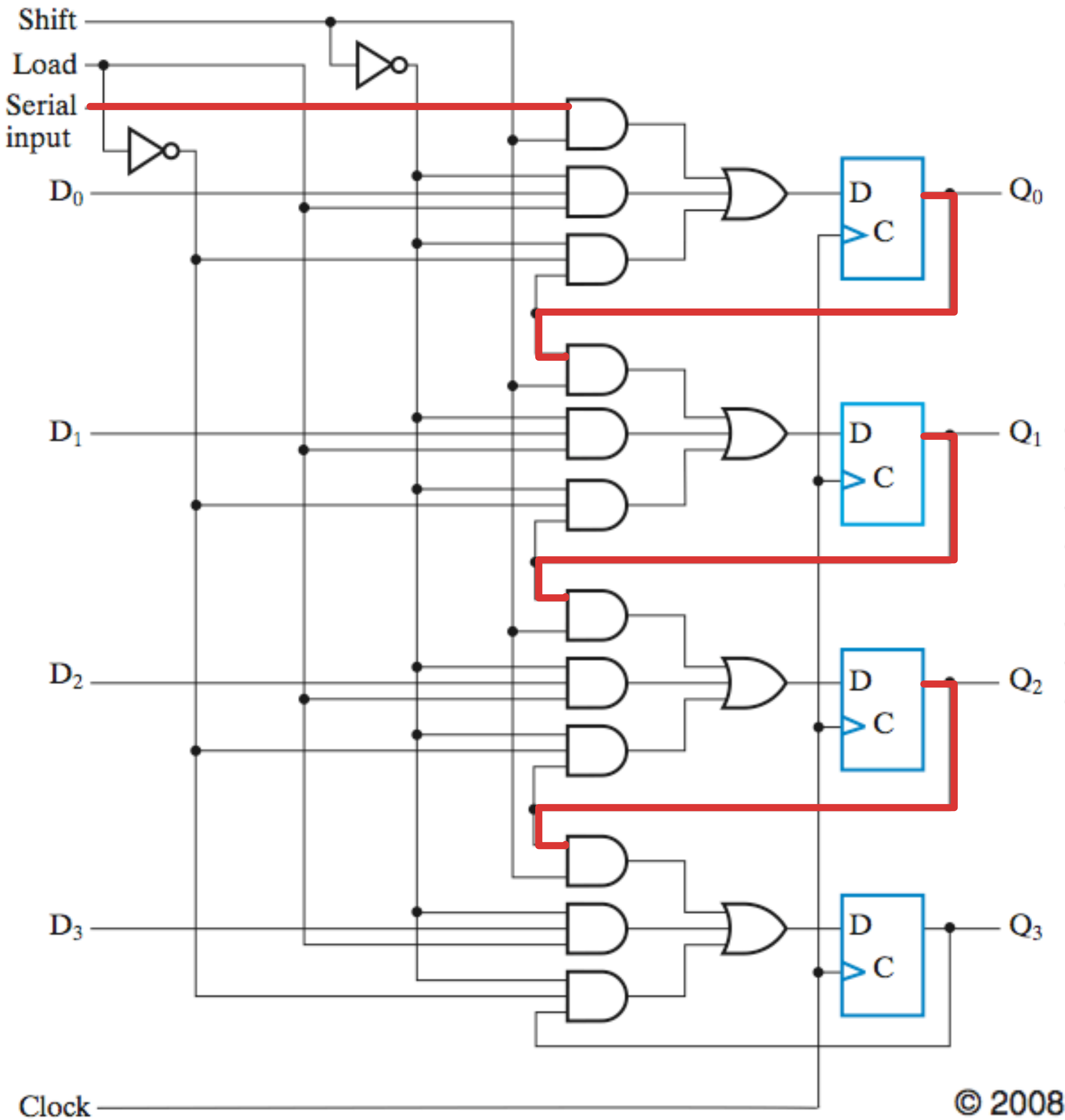
No input

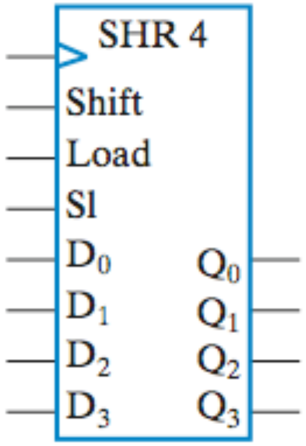
M. Morris Mano & Charles R. Kime

LOGIC AND COMPUTER DESIGN FUNDAMENTALS, 4e

Shift register w. parallel load

Serial input operation
(enabled by Shift)

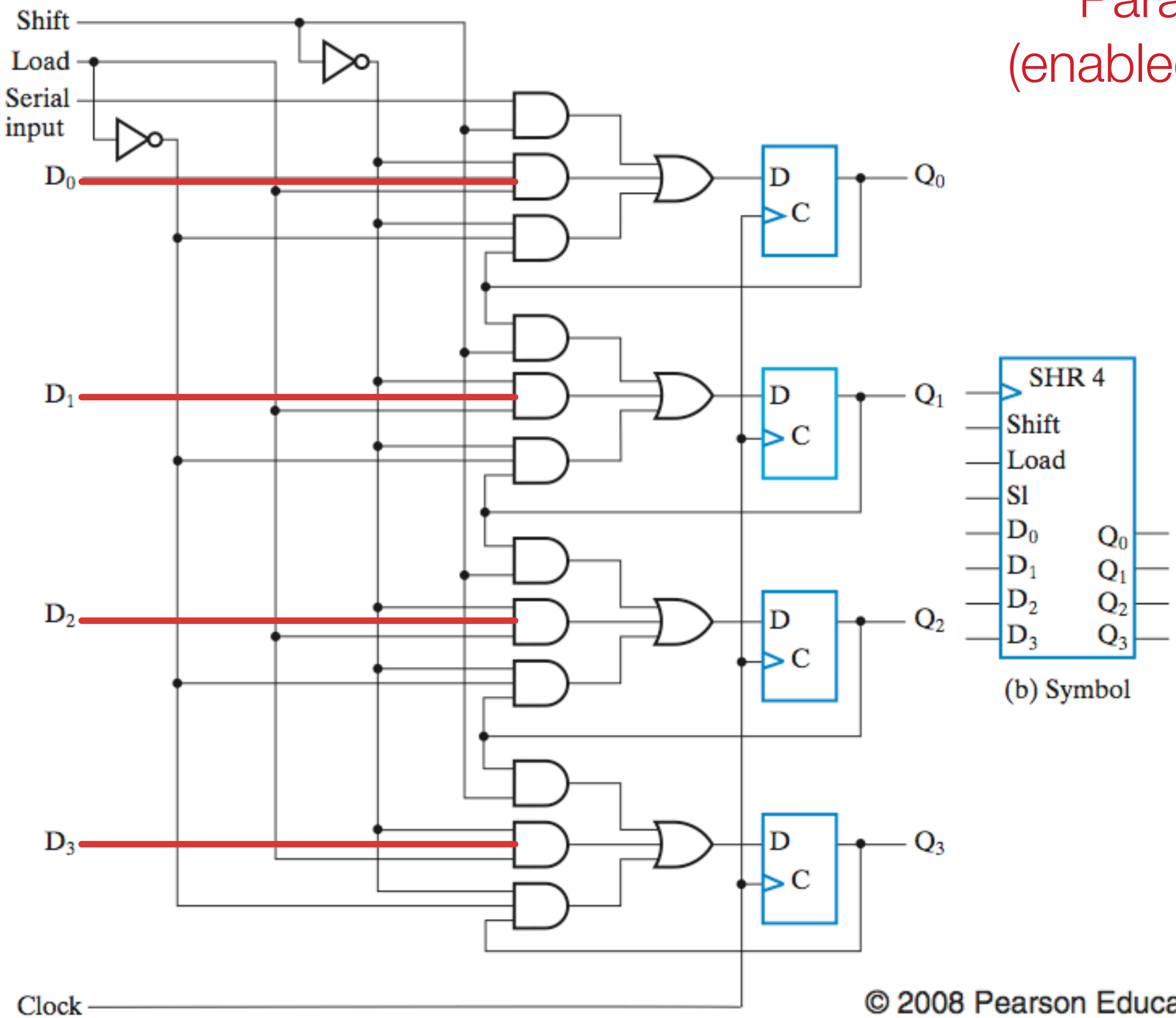




(b) Symbol

Shift register w. parallel load

Parallel load operation
(enabled by Load AND $\overline{\text{Shift}}$)

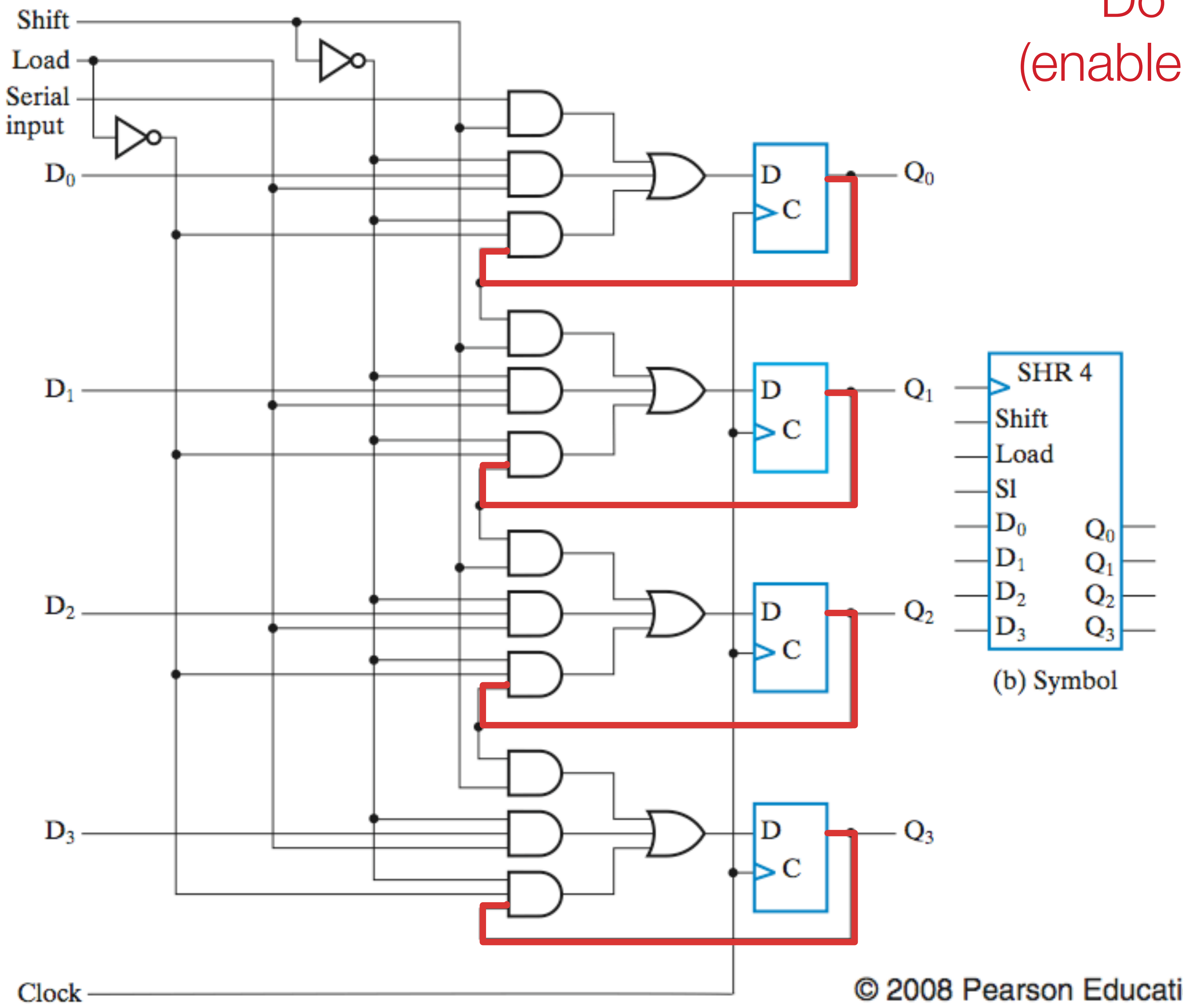


(b) Symbol

M. Morris Mano & Charles R. Kime
LOGIC AND COMPUTER DESIGN FUNDAMENTALS, 4e

Shift register w. parallel load

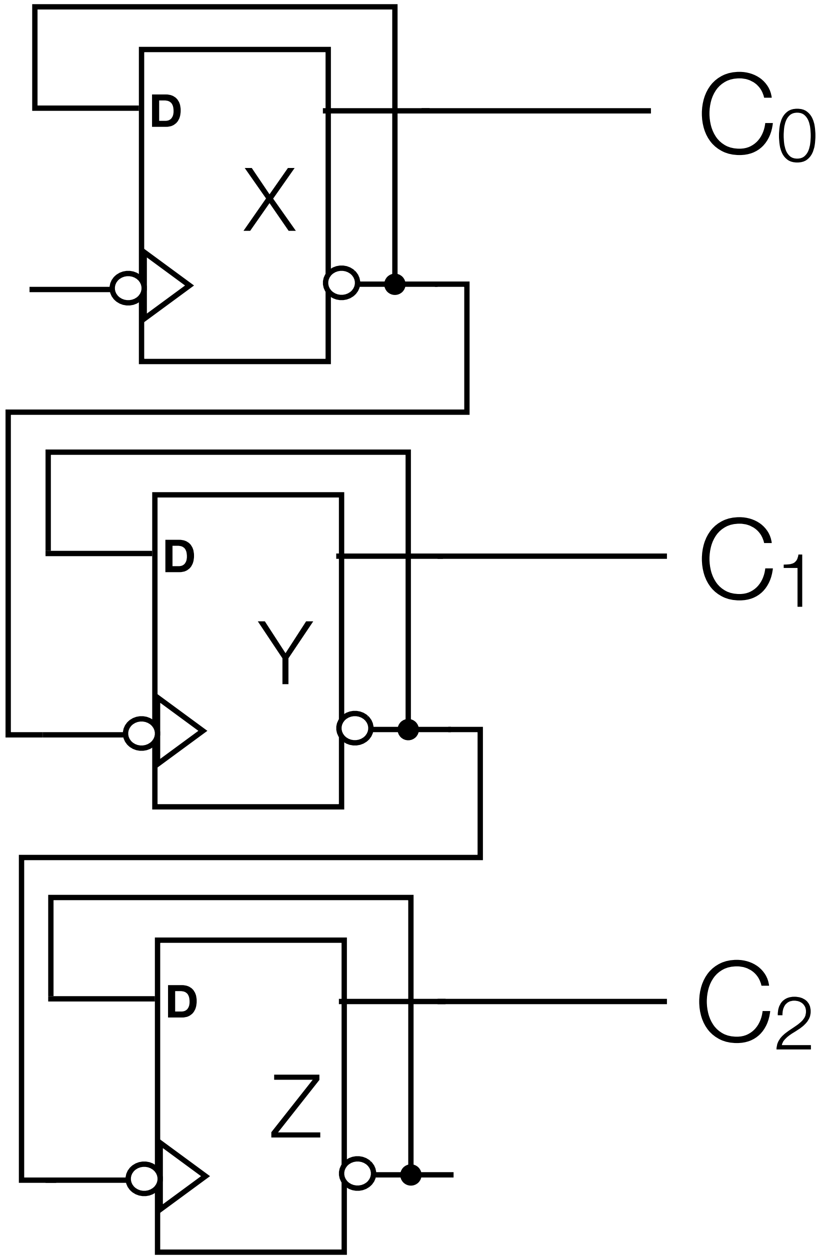
Do nothing operation
(enabled by $\overline{\text{Load}}$ AND $\overline{\text{Shift}}$)

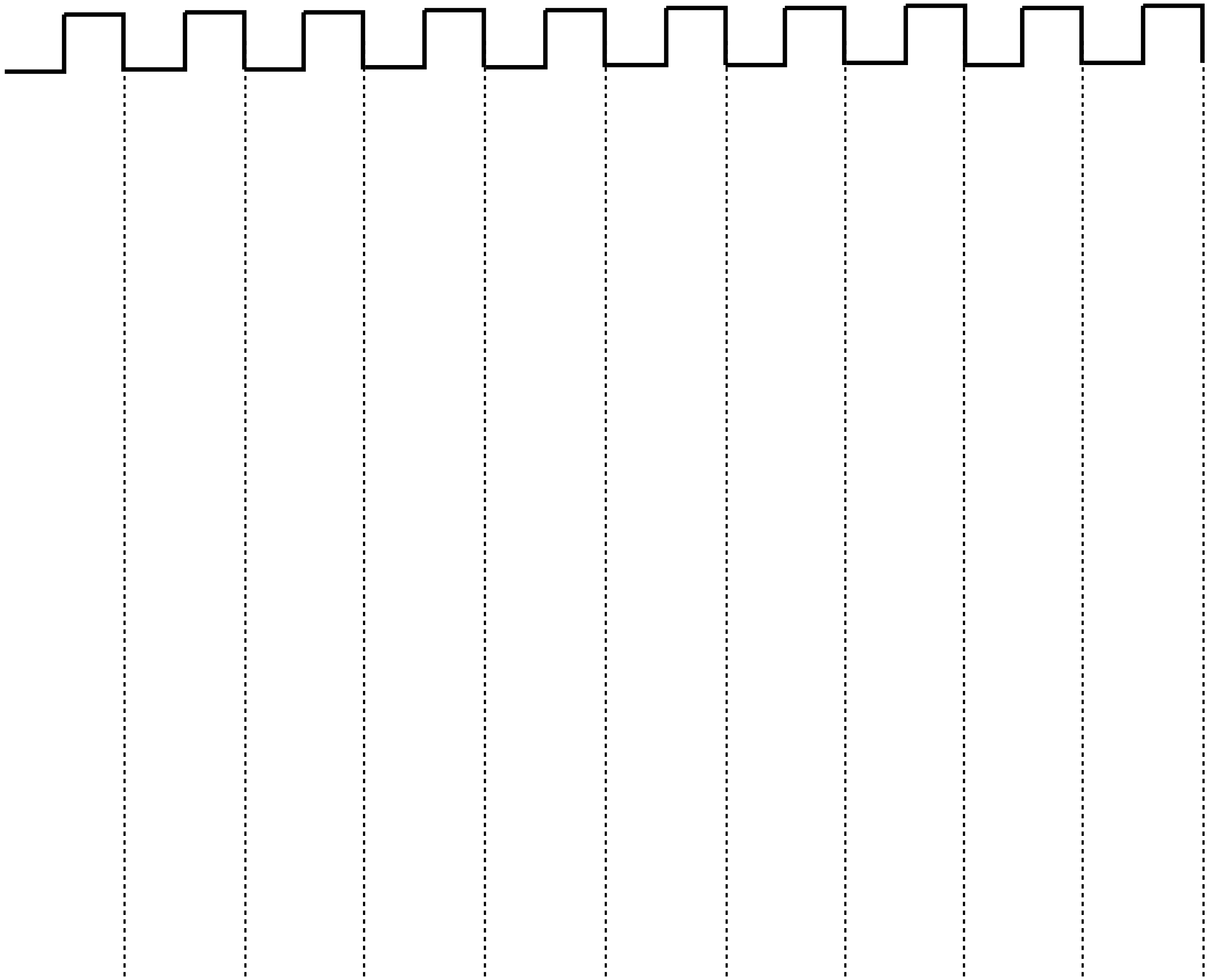


(b) Symbol

M. Morris Mano & Charles R. Kime
LOGIC AND COMPUTER DESIGN FUNDAMENTALS, 4e

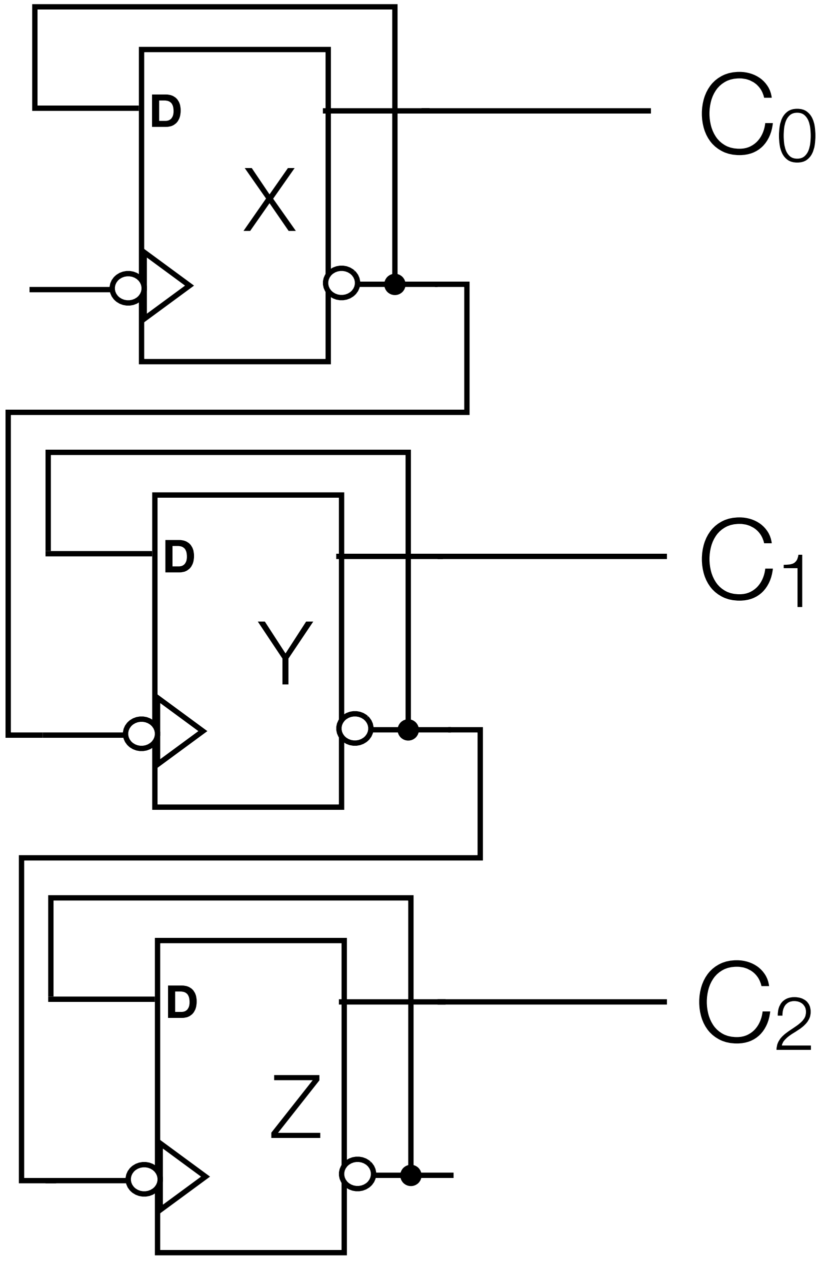
Ripple Counter

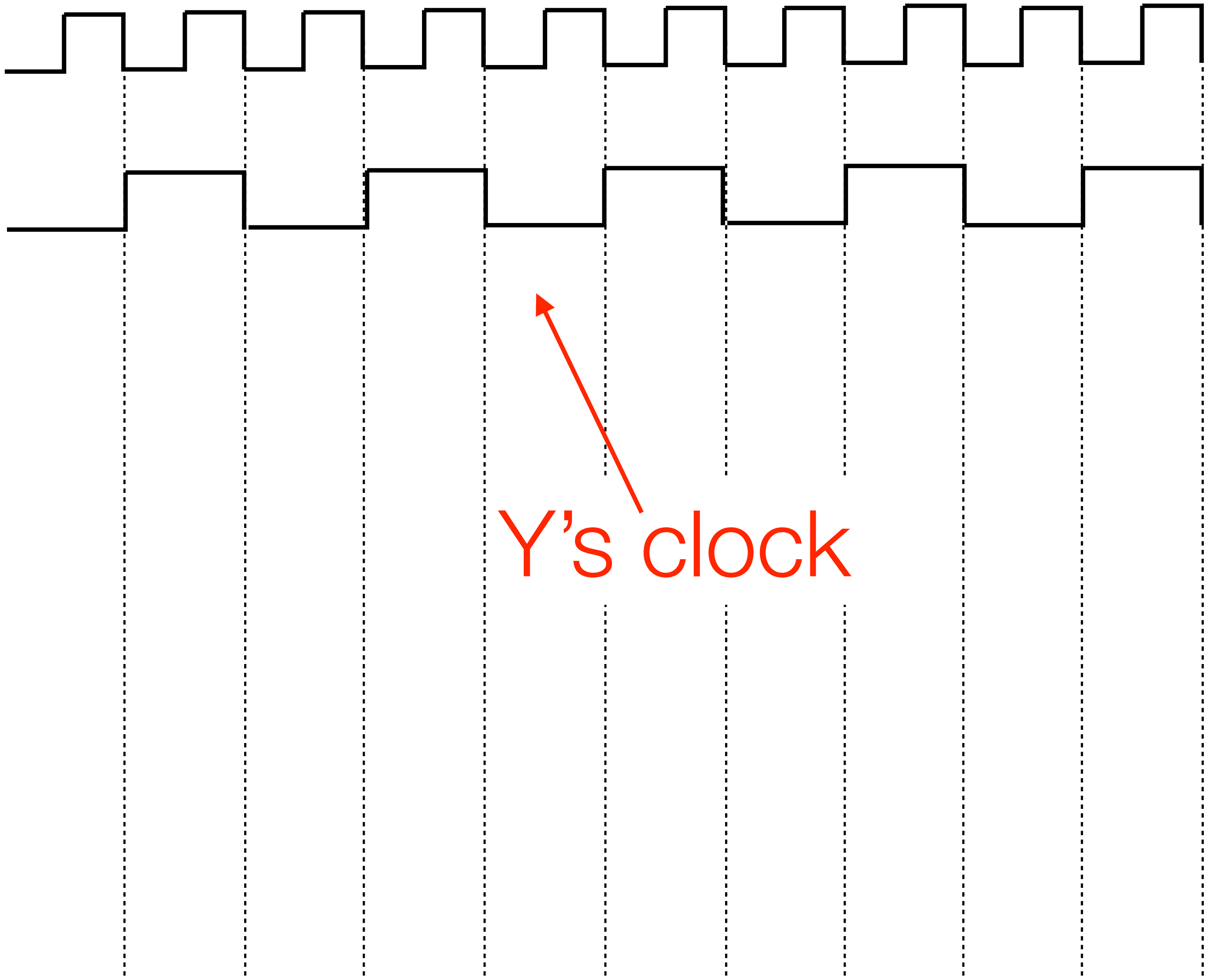




- Each flip-flop feeds Q into D: complements its value
- X’s Q feeds into Y’s clock, Y’s Q feeds into Z’s clock - why?

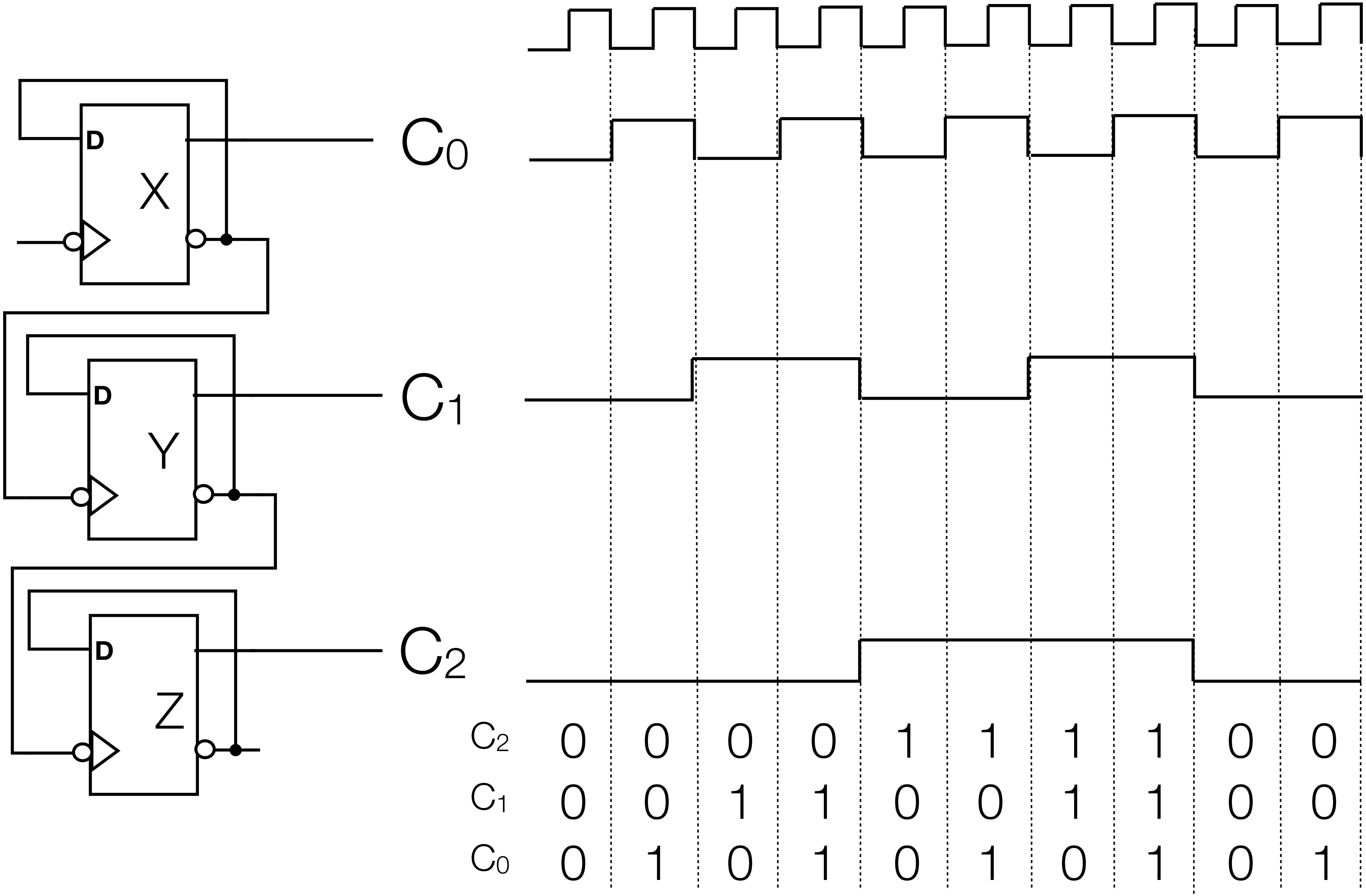
Ripple Counter





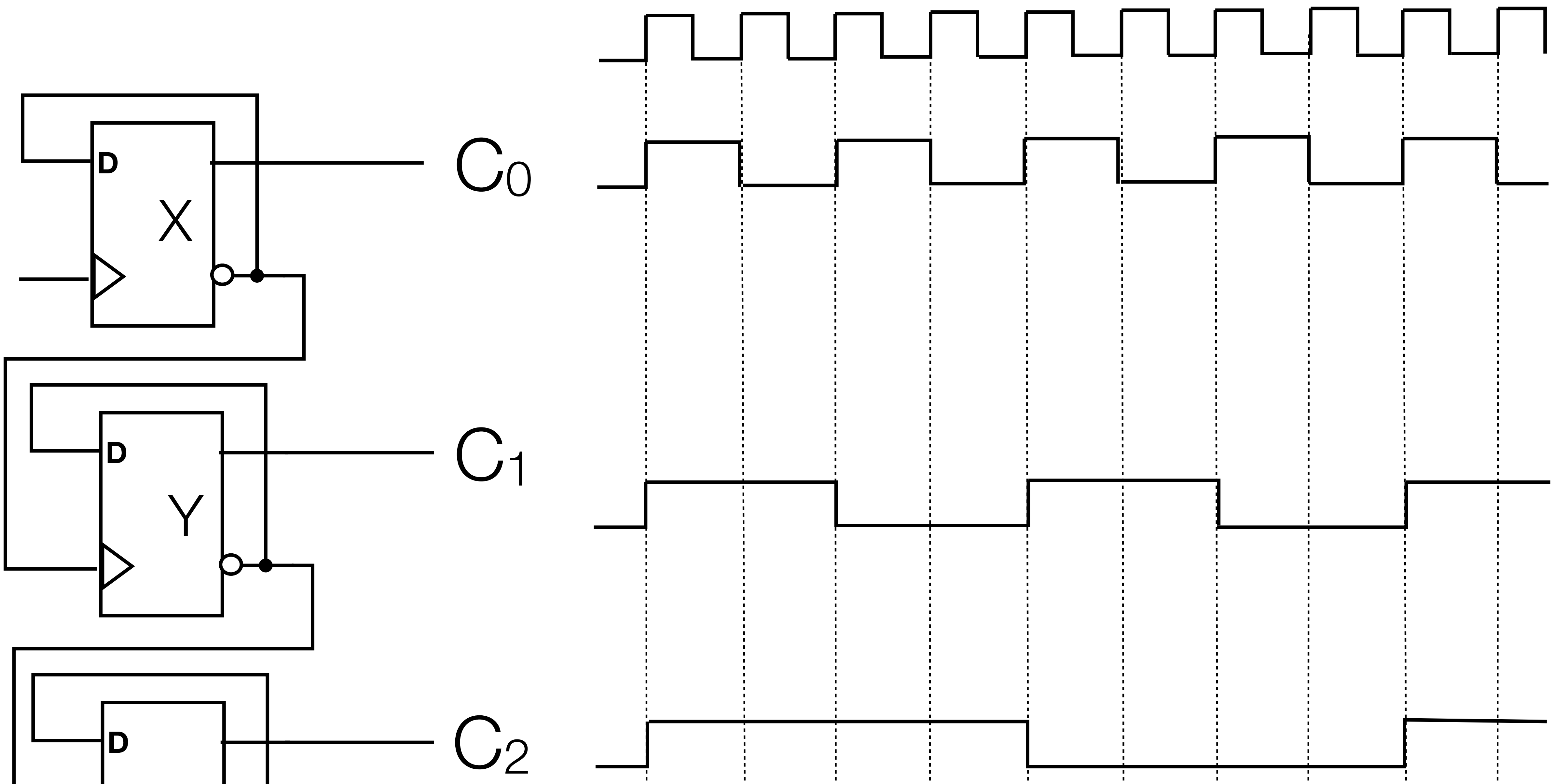
- Note: each FF is negative-edge triggered

Ripple Counter



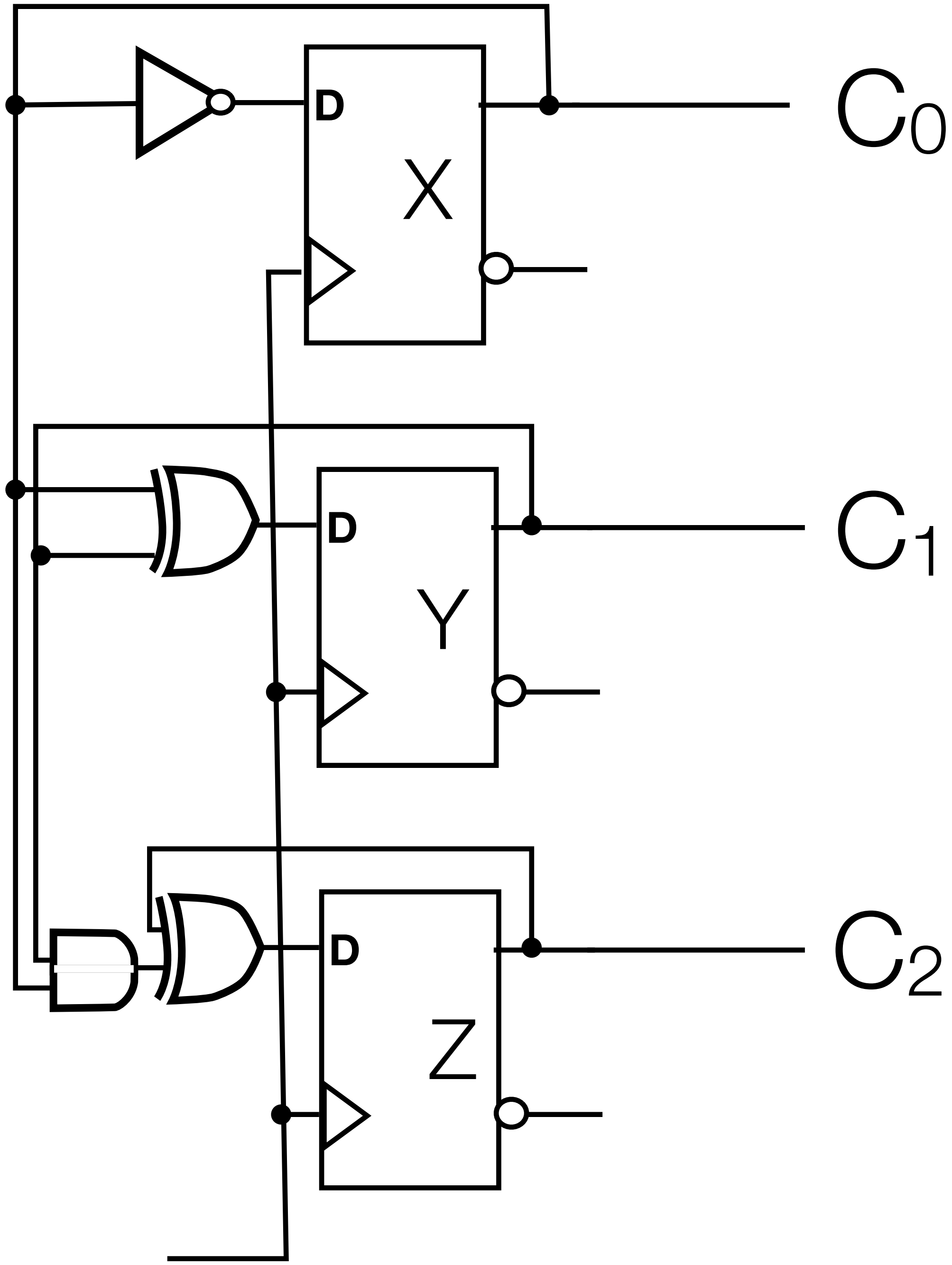
C_2	0	0	0	0	1	1	1	1	0	0
C_1	0	0	1	1	0	0	1	1	0	0
C_0	0	1	0	1	0	1	0	1	0	1

Ripple Counter with Positive Edge Triggering



C_2	0	1	1	1	1	0	0	0	0	1	1
C_1	0	1	1	0	0	1	1	0	0	1	1
C_0	0	1	0	1	0	1	0	1	0	1	0

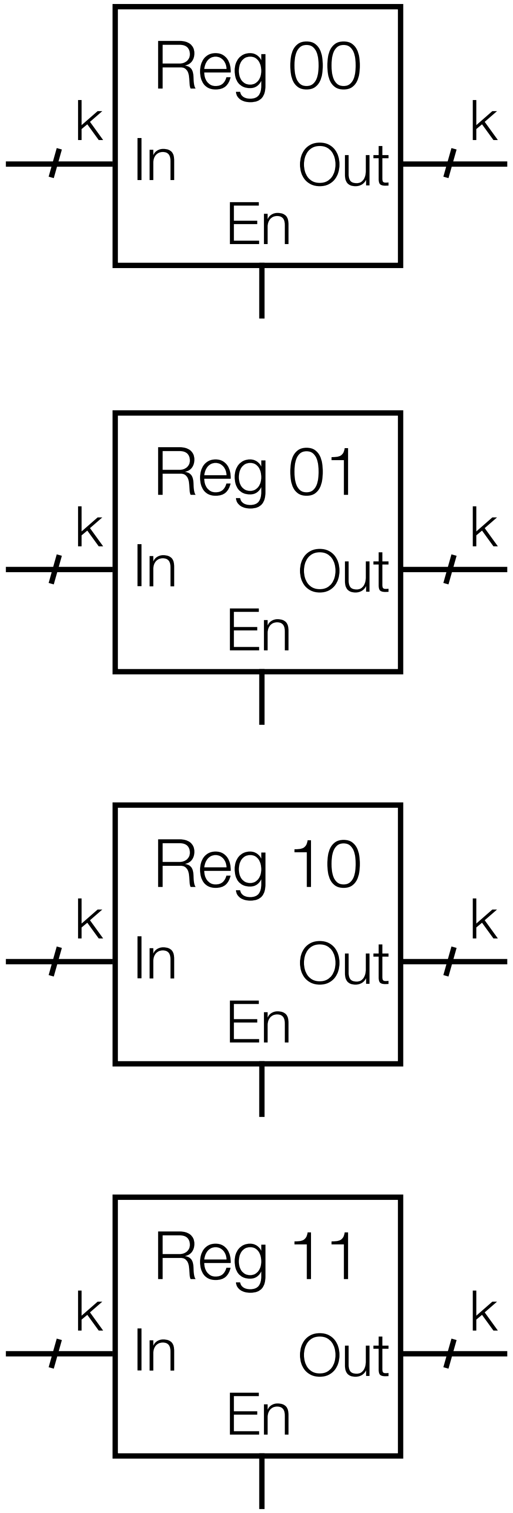
Serial Counter (counting upward)

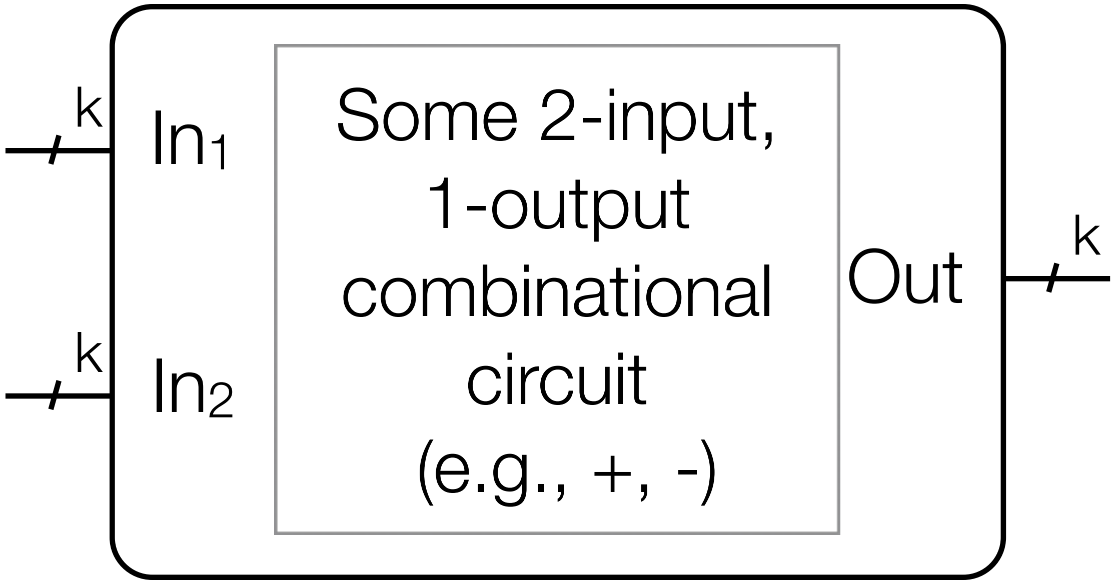


C_2	0	0	0	0	1	1	1	1	0	0
C_1	0	0	1	1	0	0	1	1	0	0
C_0	0	1	0	1	0	1	0	1	0	1

- Y(t+1) differs from Y(t) only when X(t)=1
- Z(t+1) differs from Z(t) when both X(t)=1 and Y(t)=1
- so Y changes when X (evaluates to TRUE), Z changes when XY (evaluates to TRUE)
- $X = \bar{X}$, $Y = Y \oplus X$, $Z = Z \oplus XY$
- for the jth bit (in a j+1 bit counter), $B_j = B_j \oplus Bj_{-1}B_{j-2}...B_1B_0$

Register MUXing



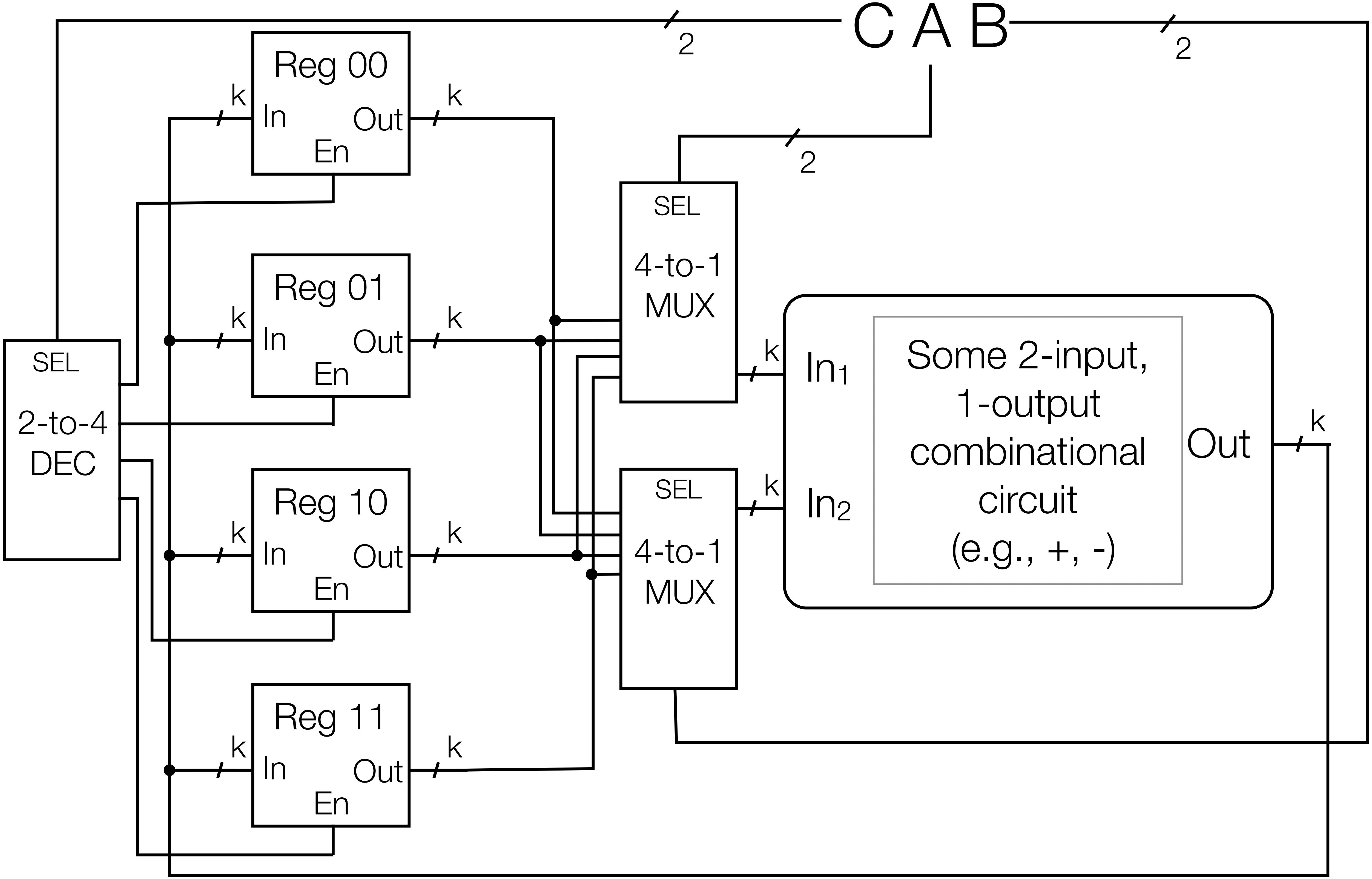


- Suppose have
  - 4 k-bit registers with Enable (for writing)
  - combinational logic to perform some operation OP
- Goal: Allow system to select registers
  - 2 registers selected for inputs (A & B)
  - 1 register for output (C)
  - At end of cycle: C = A OP B
- Note: A&B can be same register, C can also be same as A or B

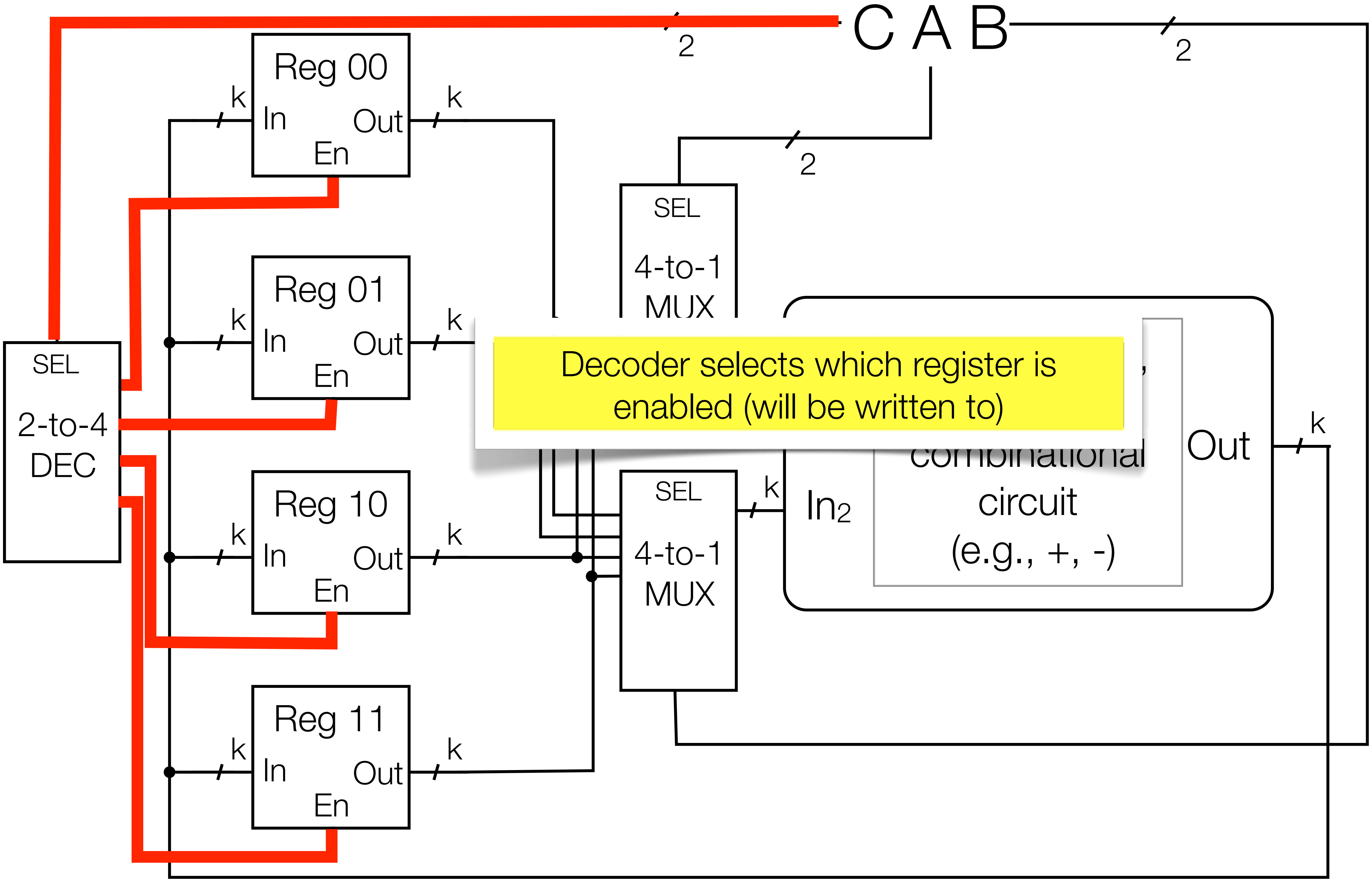
Register MUXing example

- e.g.,
  - Reg01 = Reg01 OP Reg10
  - Reg10 = Reg10 OP Reg10 (if OP were +, this would do Reg10 *= 2)
- OP selecting: be able to choose from different OPs, e.g.
  - Reg01 = Reg01 + Reg10
  - Reg10 = Reg01 * Reg00

Register MUXing

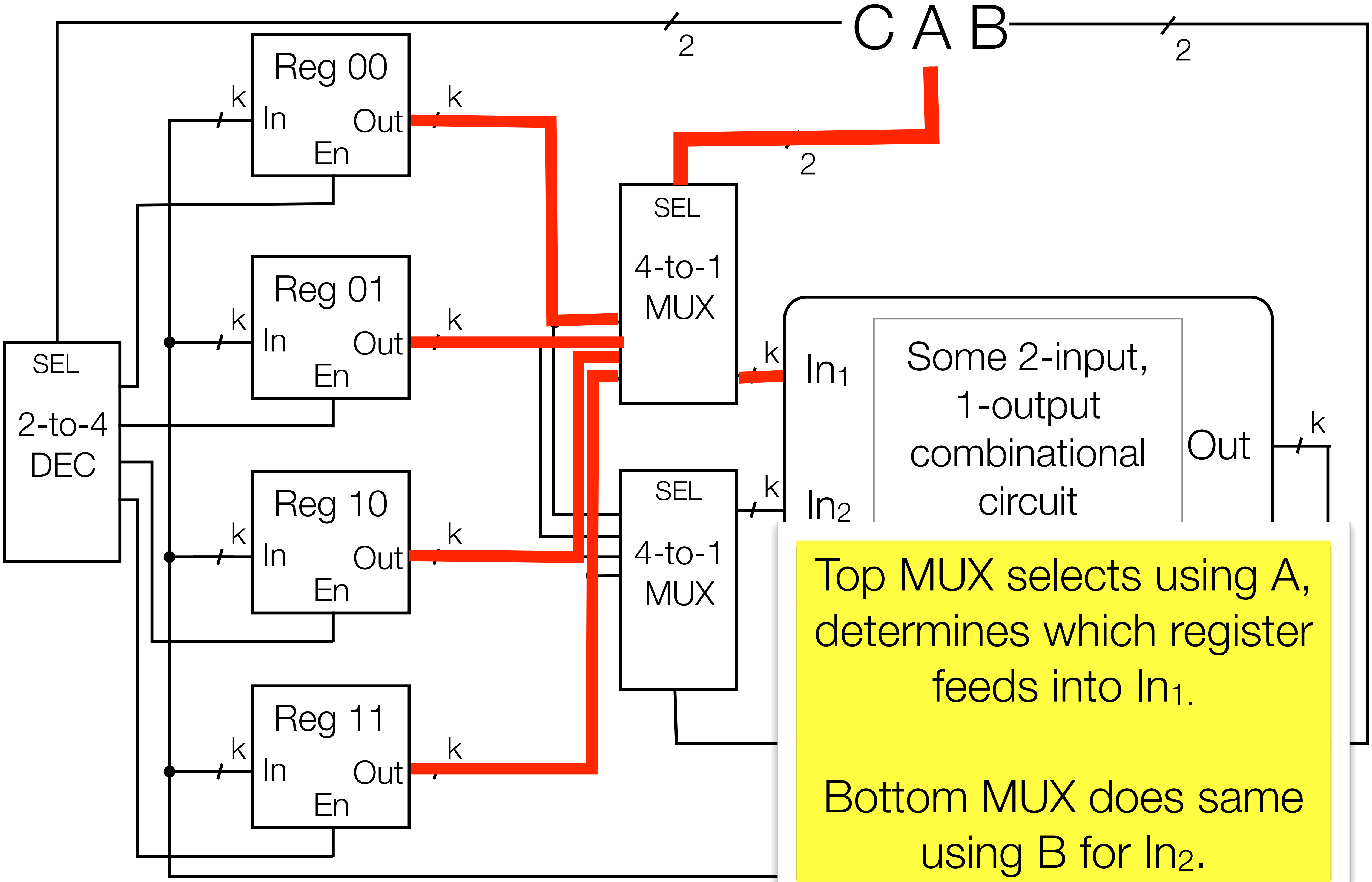


Register MUXing



Decoder selects which register is enabled (will be written to)

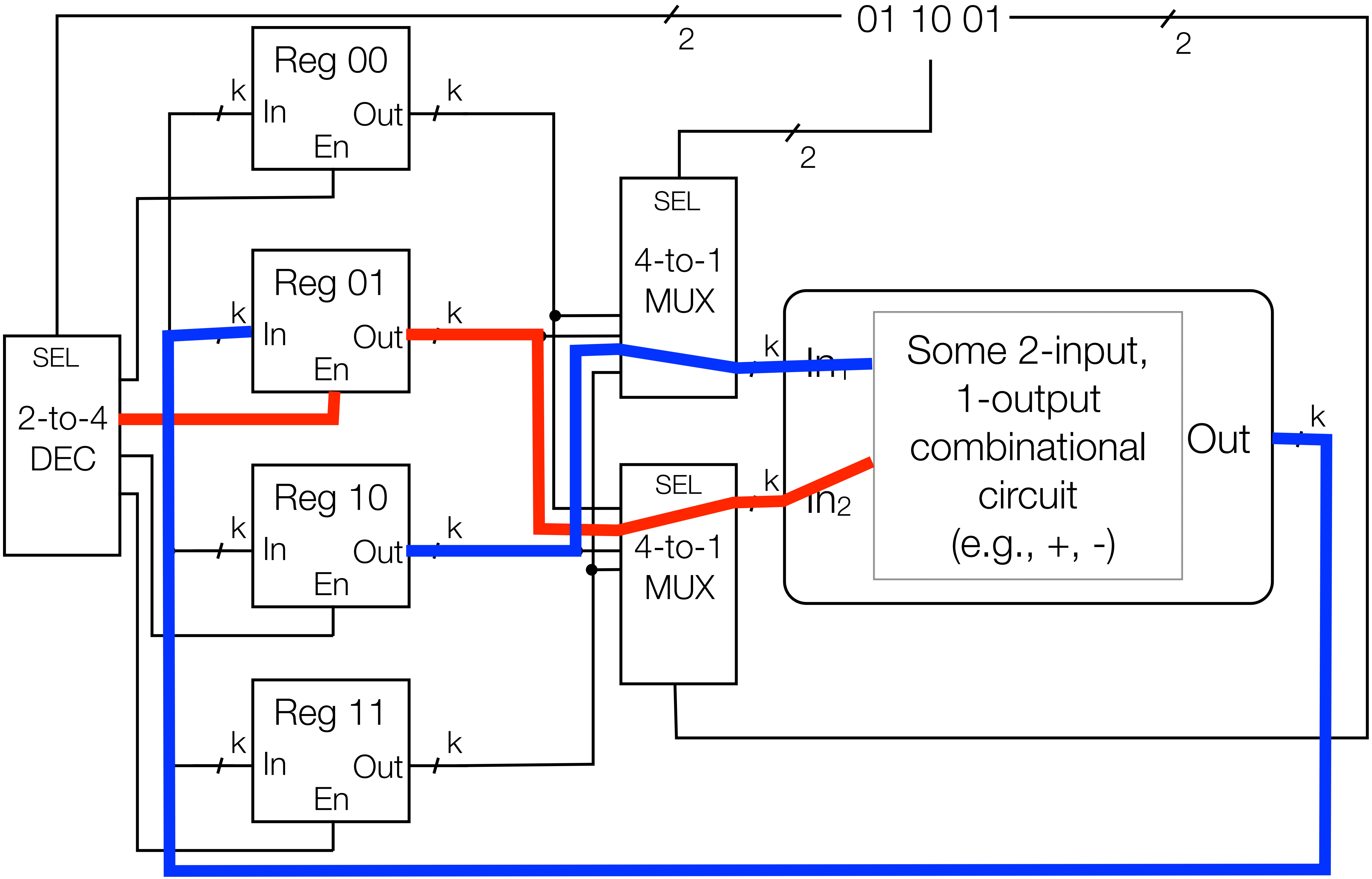
Register MUXing



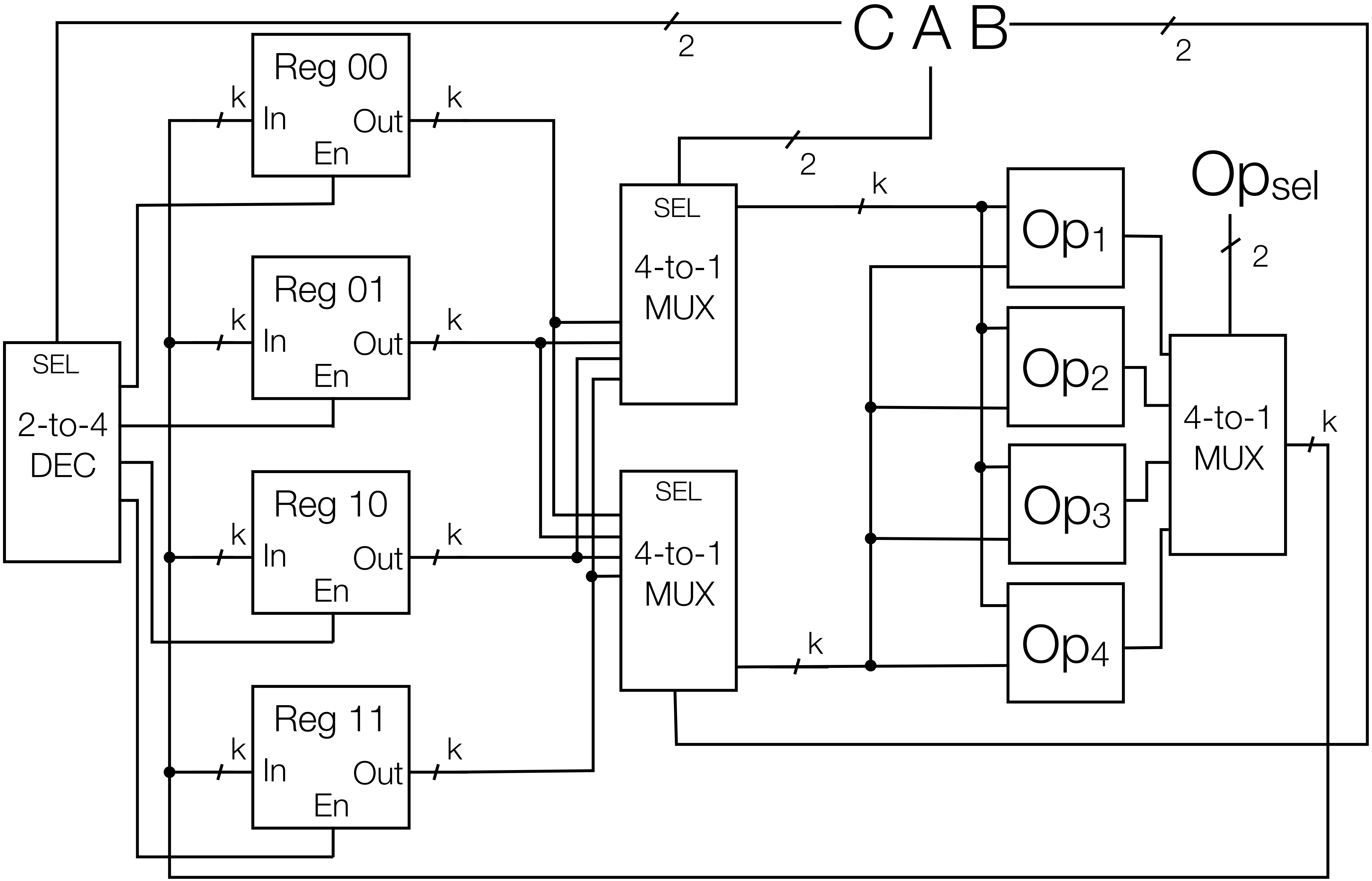
Top MUX selects using A, determines which register feeds into In$_1$.

Bottom MUX does same using B for In$_2$.

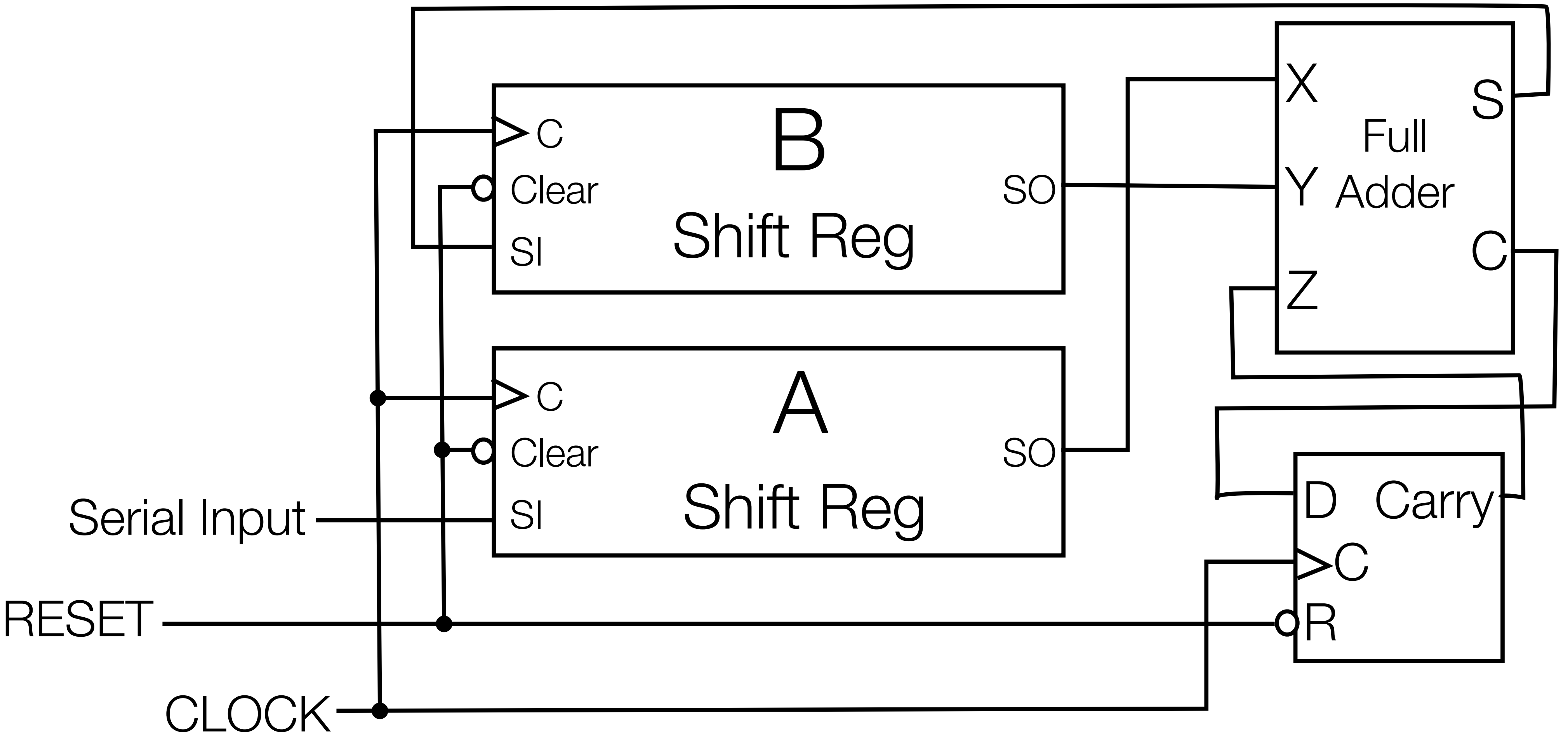
Example: Reg01 = Reg10 OP Reg01



Register MUXing and OP MUXing

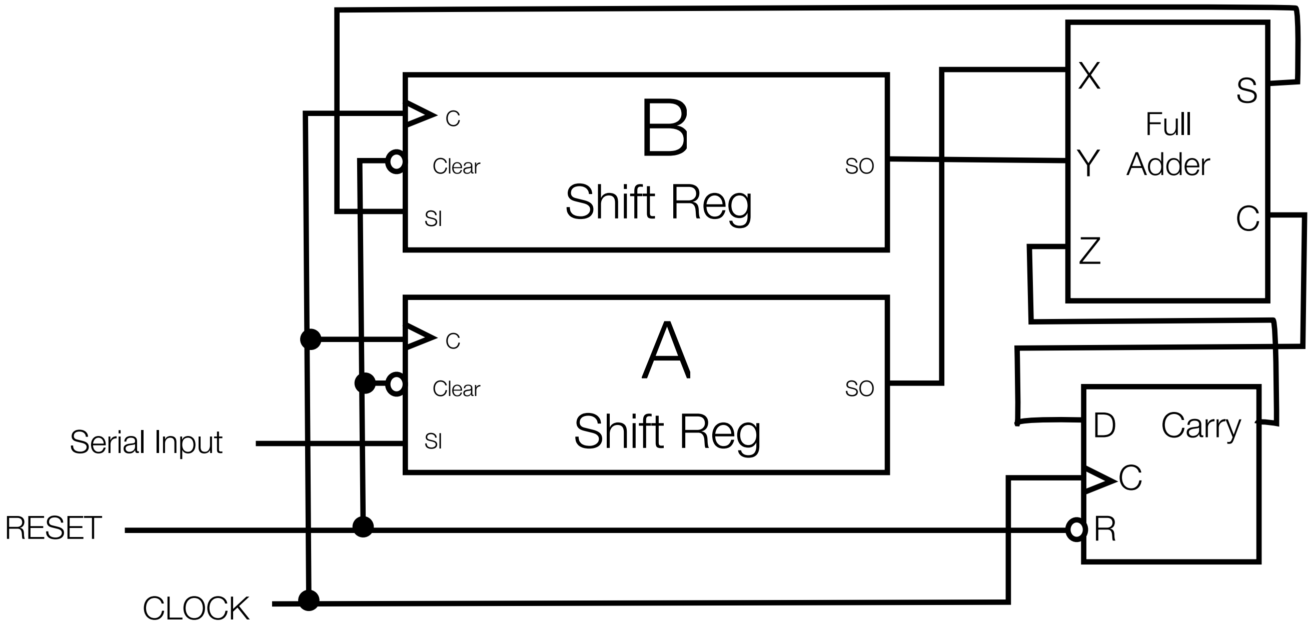


Serial Adder Circuit



- New Value pushed into A register
- Result pushed into B register
- If Shift Reg holds N bits, SUM starts calculating during N+1st cycle
- Done after 2N cycles

Serial Adder Circuit Example: 0101 + 0011 = 1100



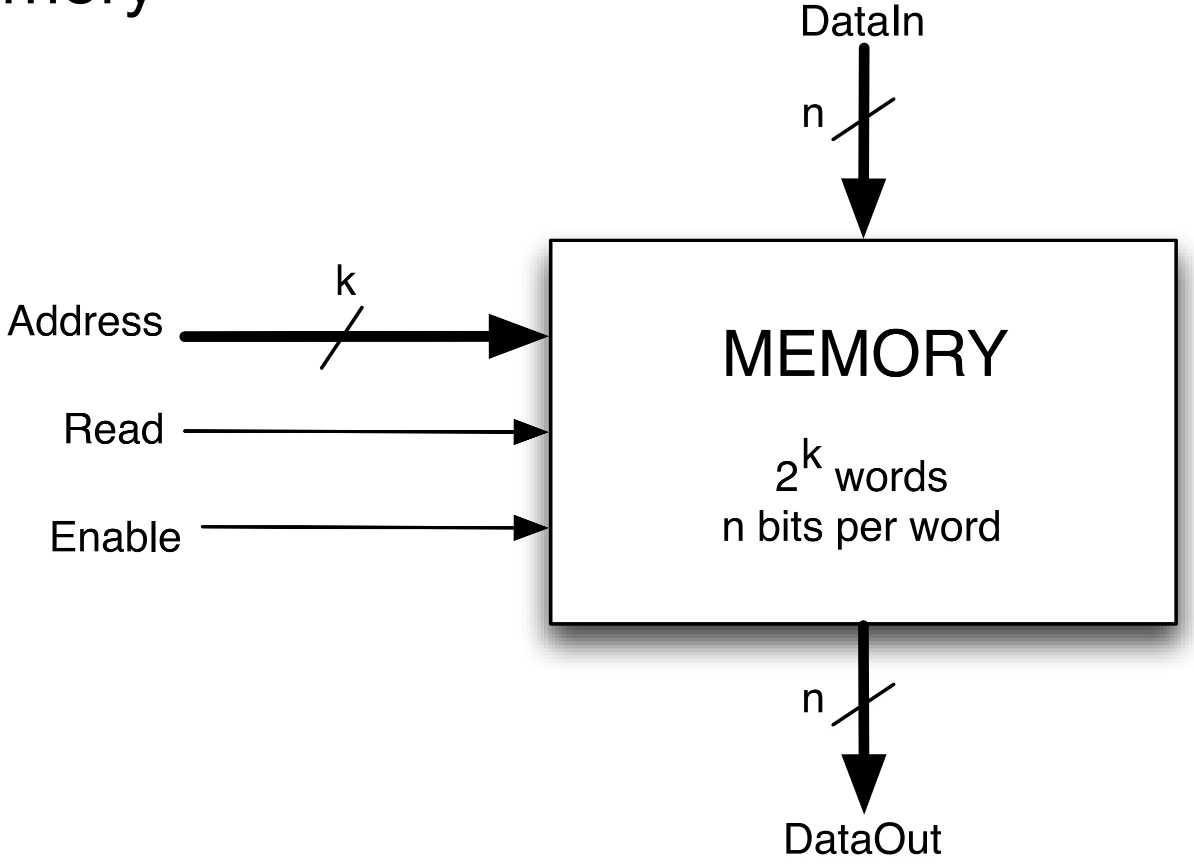
To time 7, just rolling values into registers by adding with 0’s (B output)

Adding starts time 8 with least significant bits

Time	In	A	B	S	C
0	1	“0000”	“0000”	0	0
1	0	“1000”	“0000”	0	0
2	1	“0100”	“0000”	0	0
3	0	“1010”	“0000”	0	0
4	0	“0101”	“0000”	1	0
5	0	“1010”	“1000”	0	0
6	1	“1101”	“0100”	1	0
7	1	“0110”	“1010”	0	0
8	X	“0011”	“0101”	0	1
9	X	“X011”	“0010”	0	1
10	X	“XX01”	“0001”	1	1
11	X	“XXX0”	“1000”	1	0
12	X	“XXXX”	“1100”	0	0

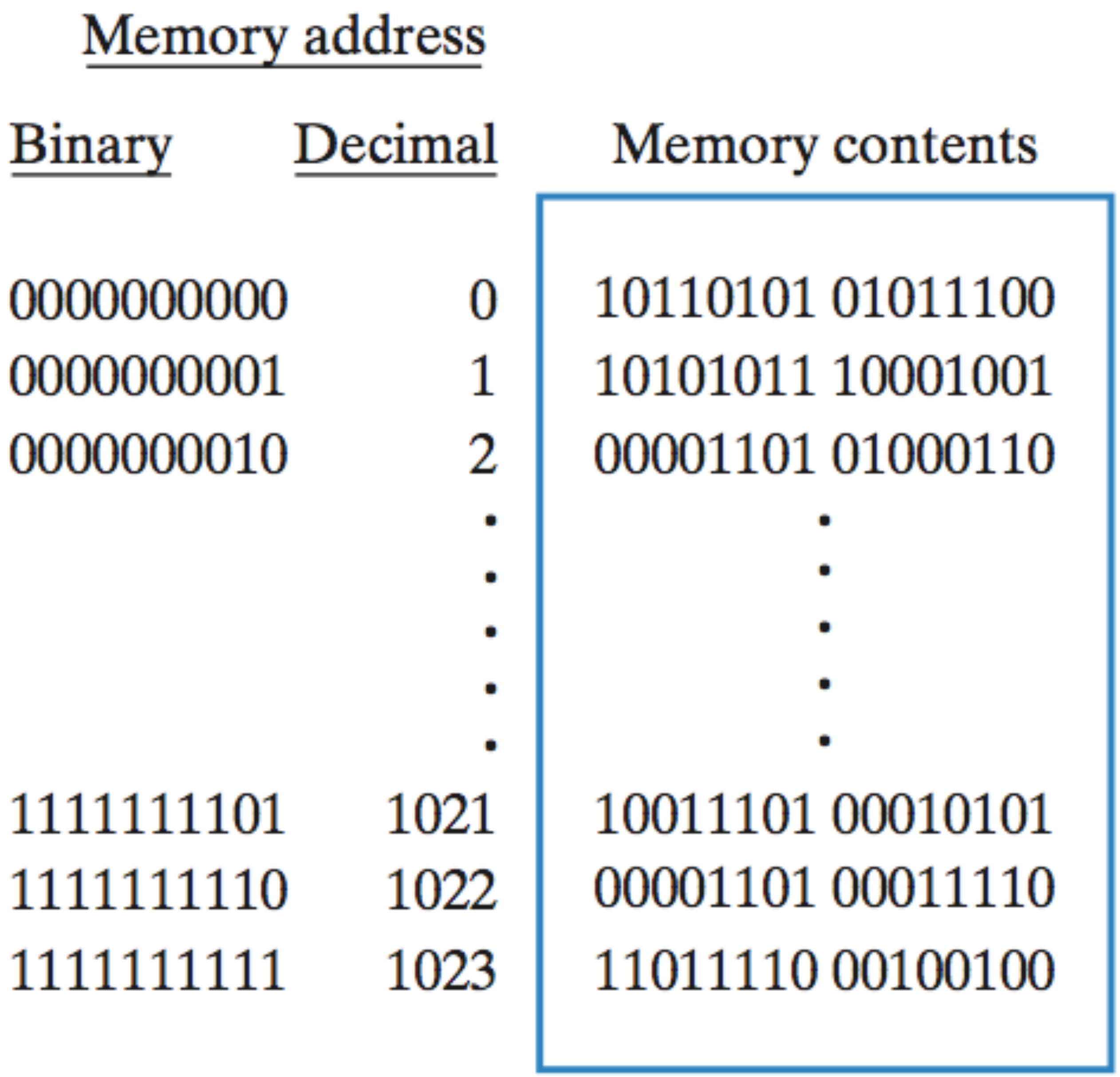
Memory interface

- Stores data in *word* units
- A *word* is several bytes (16-, 32-, or 64-bit words are typical)
- *write* operations store data to memory
- *read* operations retrieve data from memory



An n-bit value can be read from or written to each k-bit address

Conceptual view of memory

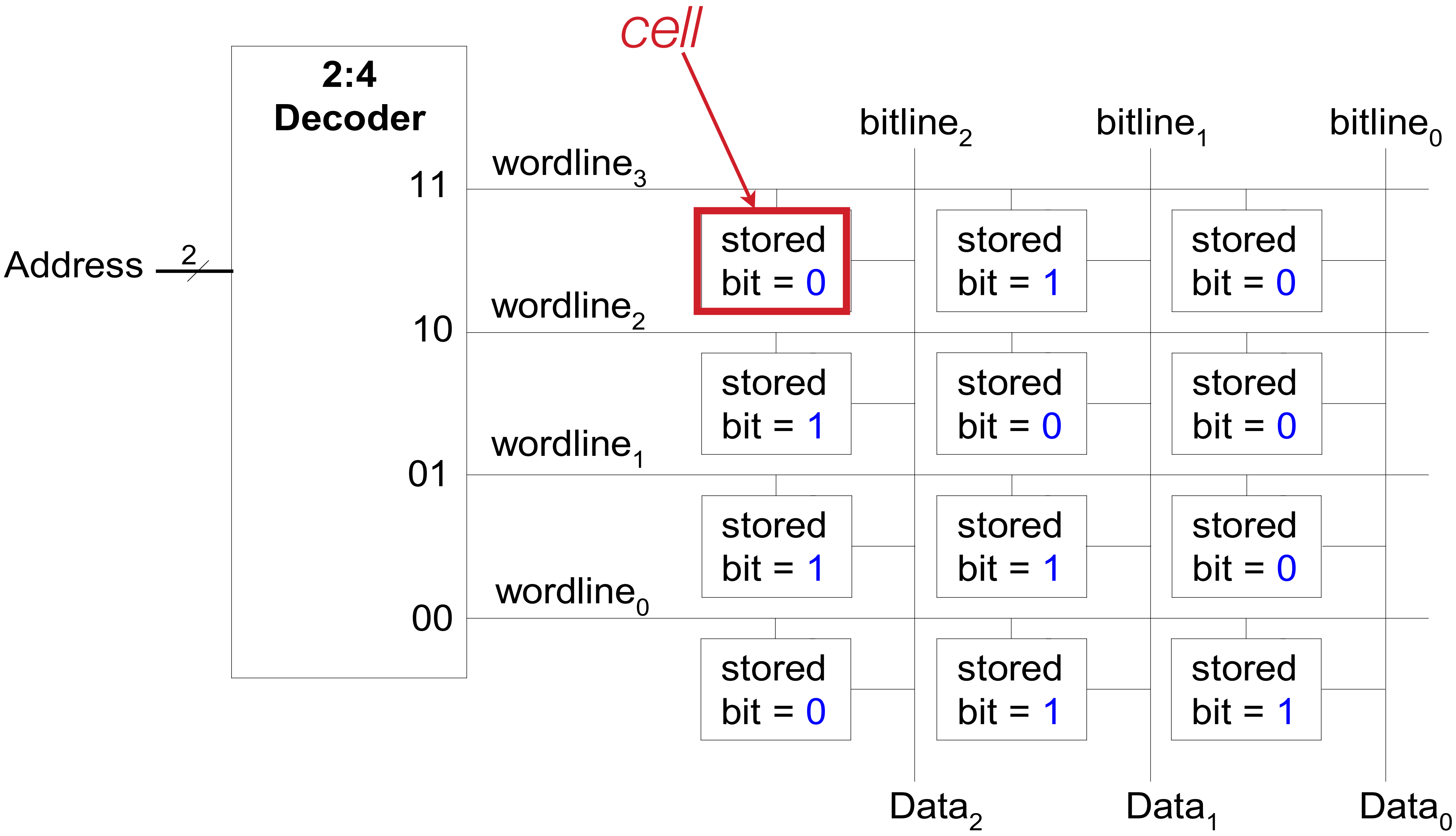


Memory address		
Binary	Decimal	Memory contents
0000000000	0	10110101 01011100
0000000001	1	10101011 10001001
0000000010	2	00001101 01000110
	⋮	⋮
1111111101	1021	10011101 00010101
1111111110	1022	00001101 00011110
1111111111	1023	11011110 00100100

M. Morris Mano & Charles R. Kime
LOGIC AND COMPUTER DESIGN FUNDAMENTALS, 4e

Memory array architecture (1)

Memory is a 2D array of bits. Each bit stored in a *cell*.



cell

2:4 Decoder

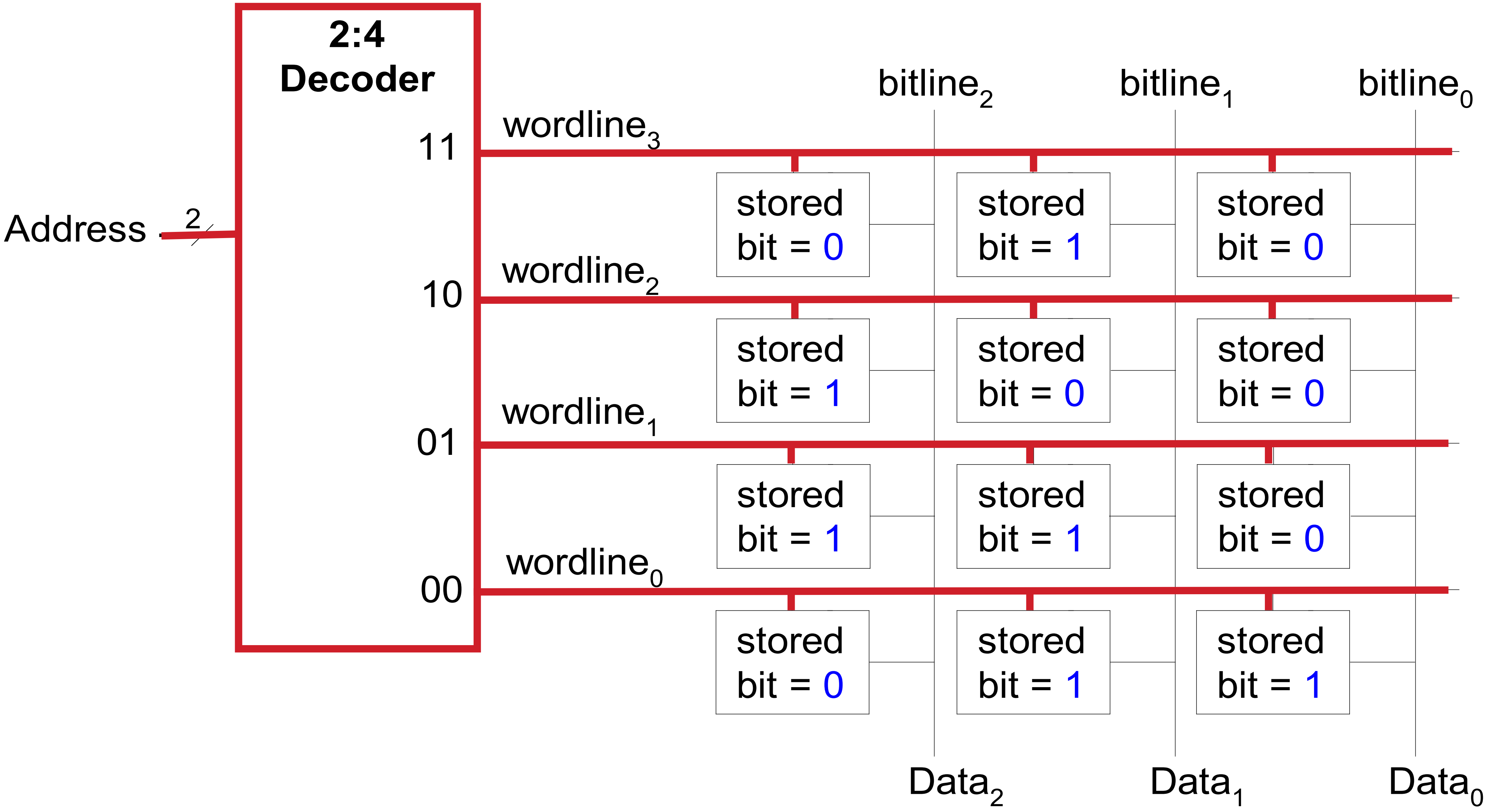
Address 2

		bitline$_2$	bitline$_1$	bitline$_0$
11	wordline$_3$	stored bit = 0	stored bit = 1	stored bit = 0
10	wordline$_2$	stored bit = 1	stored bit = 0	stored bit = 0
01	wordline$_1$	stored bit = 1	stored bit = 1	stored bit = 0
00	wordline$_0$	stored bit = 0	stored bit = 1	stored bit = 1

Data$_2$ Data$_1$ Data$_0$

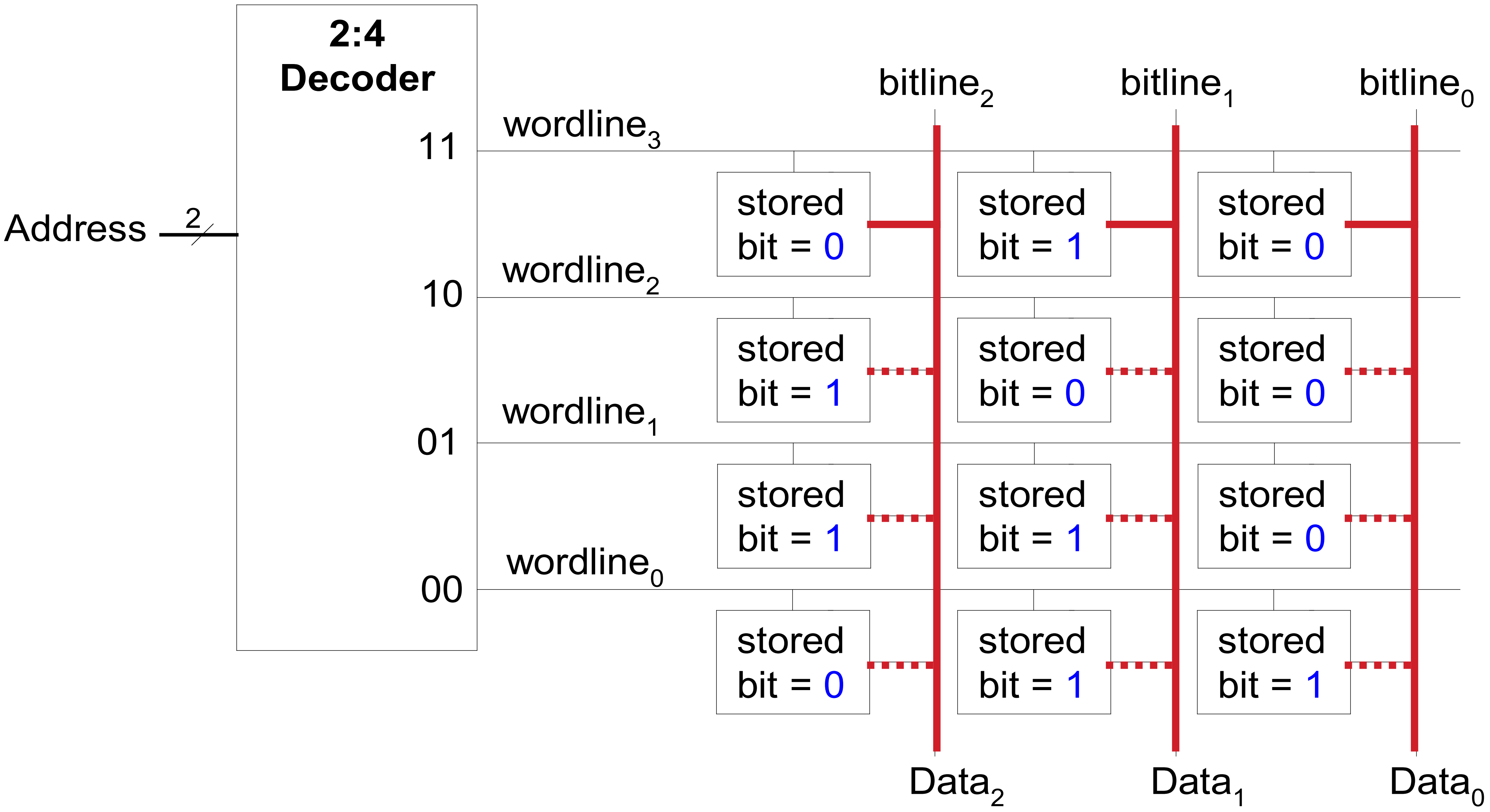
Memory array architecture (2)

Address is decoded into set of *wordlines*.
Wordlines select row to be read/written.
Only one wordline=1 at a time.



Memory array architecture (3)

When writing, contents of word written to *bitlines.*



2:4 Decoder

Address 2

11 wordline$_3$

10 wordline$_2$

01 wordline$_1$

00 wordline$_0$

bitline$_2$ bitline$_1$ bitline$_0$

stored bit = 0 stored bit = 1 stored bit = 0

stored bit = 1 stored bit = 0 stored bit = 0

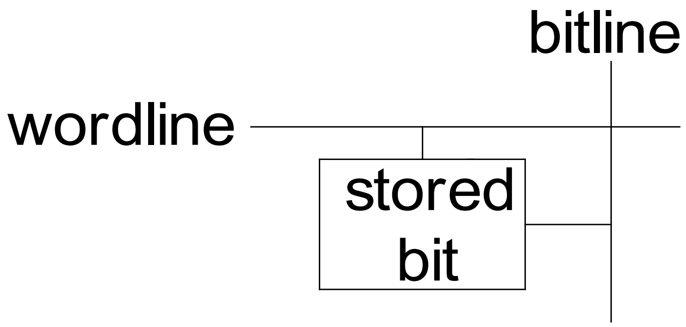
stored bit = 1 stored bit = 1 stored bit = 0

stored bit = 0 stored bit = 1 stored bit = 1

Data$_2$ Data$_1$ Data$_0$

Memory cell abstraction

Cell is base element of memory that stores a single bit



wordline	stored bit	bitline
0	x	Z
1	0	0
1	1	1

Implemented with a tristate buffer. Value "Z" does not drive output wire to either a 0 or 1.

Implementation of cell varies with type of memory.

Types of memory

Random access memory (RAM)

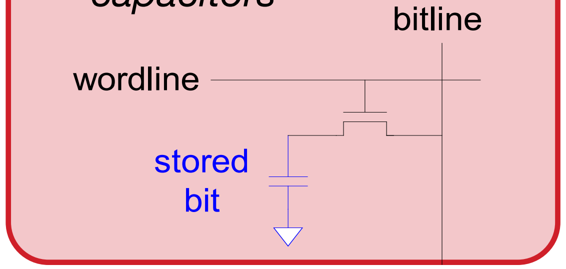
Volatile (no storage when power off)

Fast reads and writes

Historically called RAM because equal time to read/write all addresses (in contrast to serial-access devices such as a hard disk or tape). Somewhat misleading as ROM also can have uniform access time.

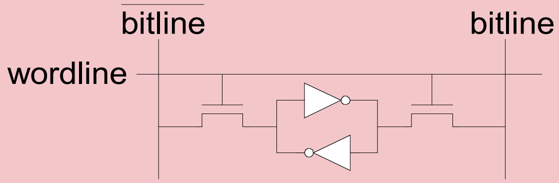
Dynamic RAM (DRAM)

Cell stores data w. capacitors



Static RAM (SRAM)

Cell stores data w. cross-coupled inverters



Flip-flop

Register

Read-only memory (ROM)

Non-volatile (retains data when powered off)

Fast reads, writing is impossible or slow (again, misleading name)

Historically called ROMs because written by permanently blowing fuses (so rewriting was impossible). Modern ROMs, such as flash memory in iPod are rewritable, just slowly.

ROM

Mask-programmed (at chip fab)

PROM

Fuse- or antifuse-programmed

FLASH

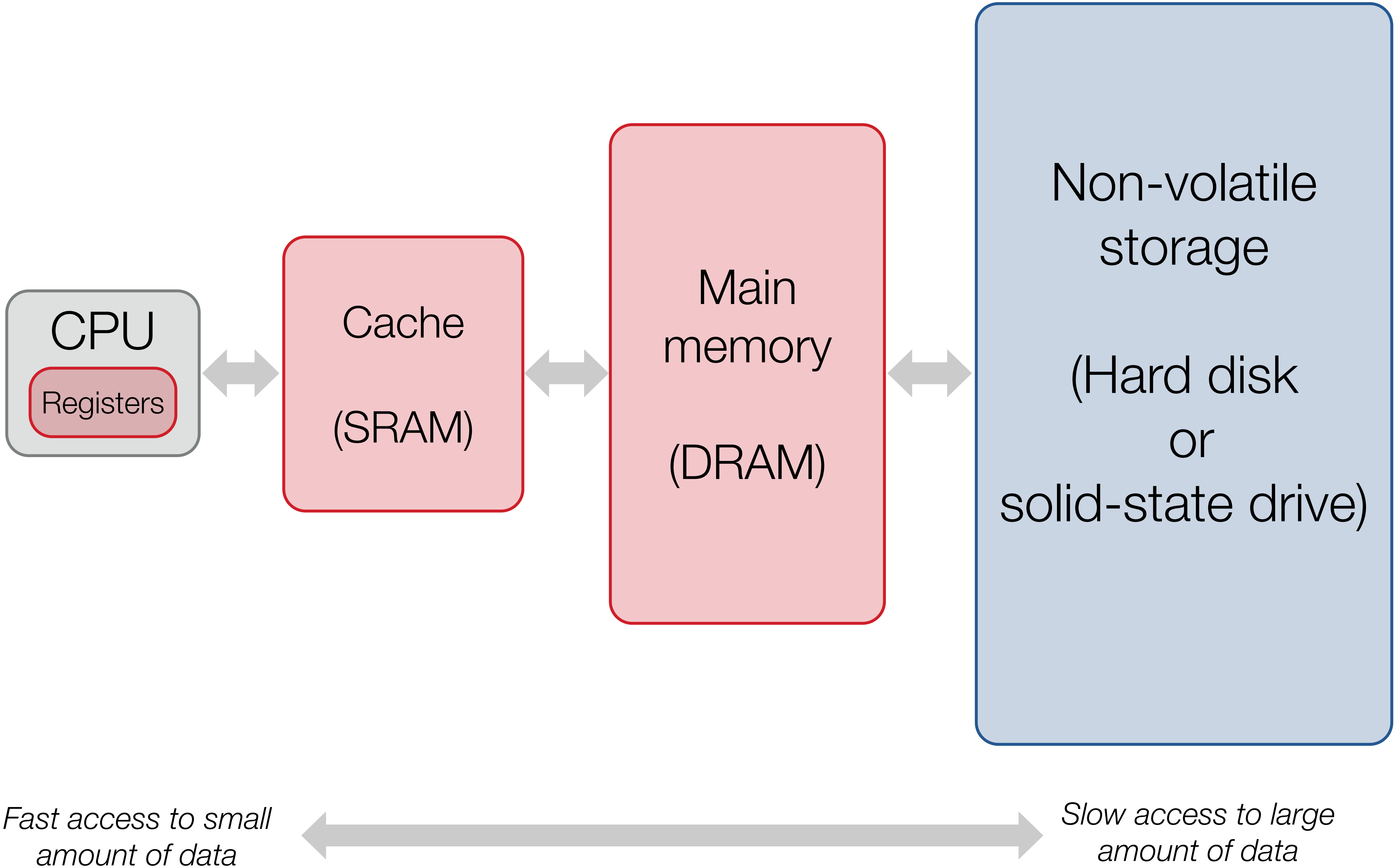
Electrically erasable floating gate with multiple erasure and programming modes

Hard Disk

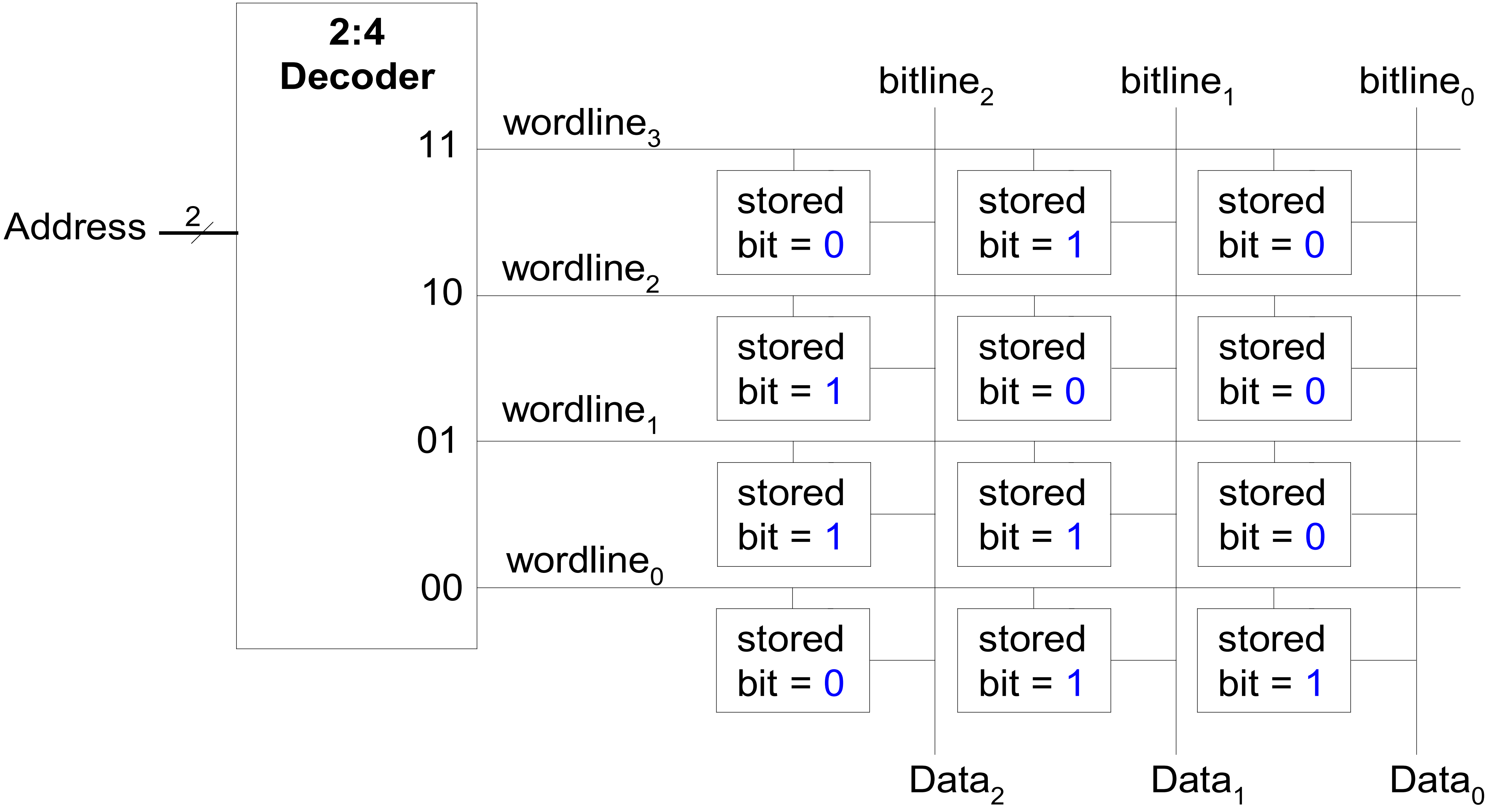
Volatile storage (RAM) comparisons

	Flip-flop	SRAM	DRAM
Transistors / bit	~20	6	1
Density	Low	Medium	High
Access time	Fast	Medium	Slow
Destructive read?	No	No	Yes (refresh required)
Power consumption	High	Medium	Low

Storage hierarchy


CPU
Registers
Cache
(SRAM)
Main memory
(DRAM)
Non-volatile storage
(Hard disk or solid-state drive)
Fast access to small amount of data
Slow access to large amount of data

Bottom-up examination of SRAM circuits



2:4 Decoder

Address 2

11 wordline$_3$

10 wordline$_2$

01 wordline$_1$

00 wordline$_0$

bitline$_2$ bitline$_1$ bitline$_0$

stored bit = 0 | stored bit = 1 | stored bit = 0

stored bit = 1 | stored bit = 0 | stored bit = 0

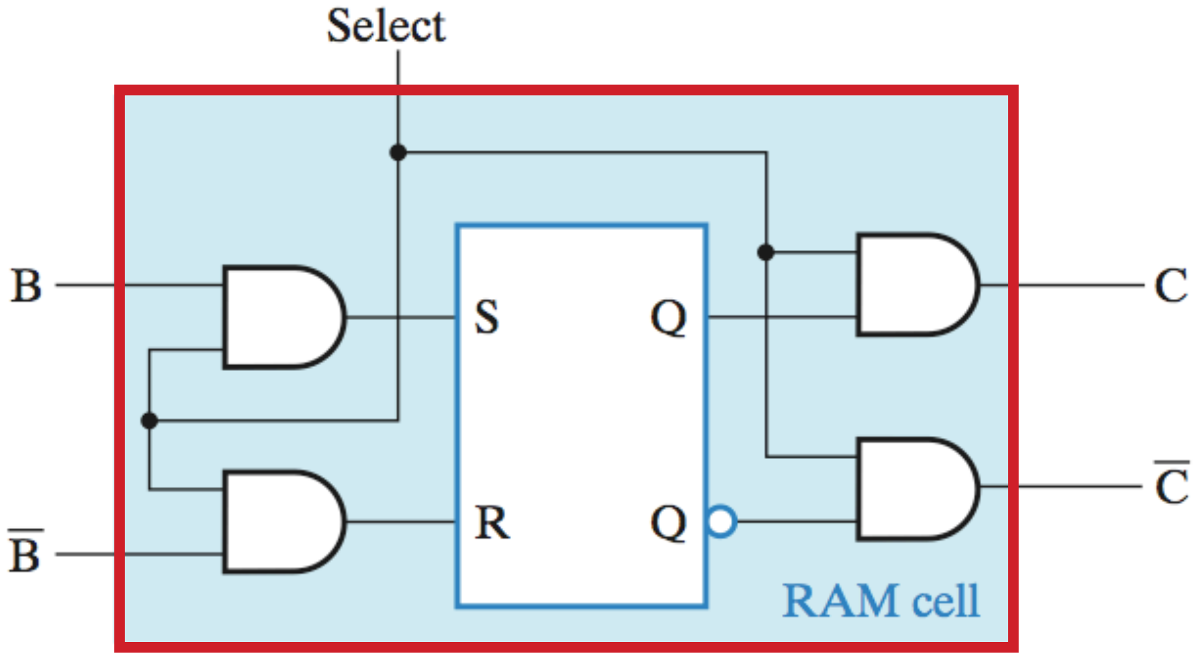
stored bit = 1 | stored bit = 1 | stored bit = 0

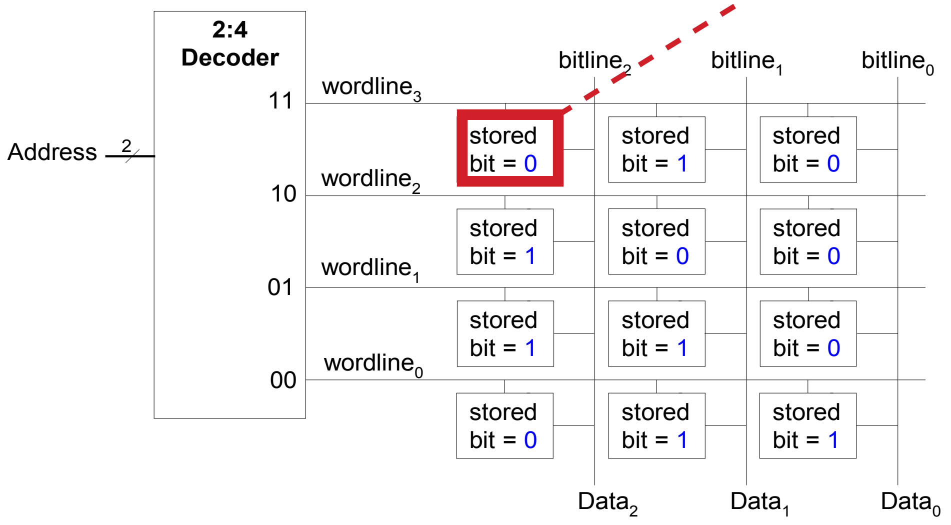
stored bit = 0 | stored bit = 1 | stored bit = 1

Data$_2$ Data$_1$ Data$_0$

Bottom-up examination of SRAM circuits (2)

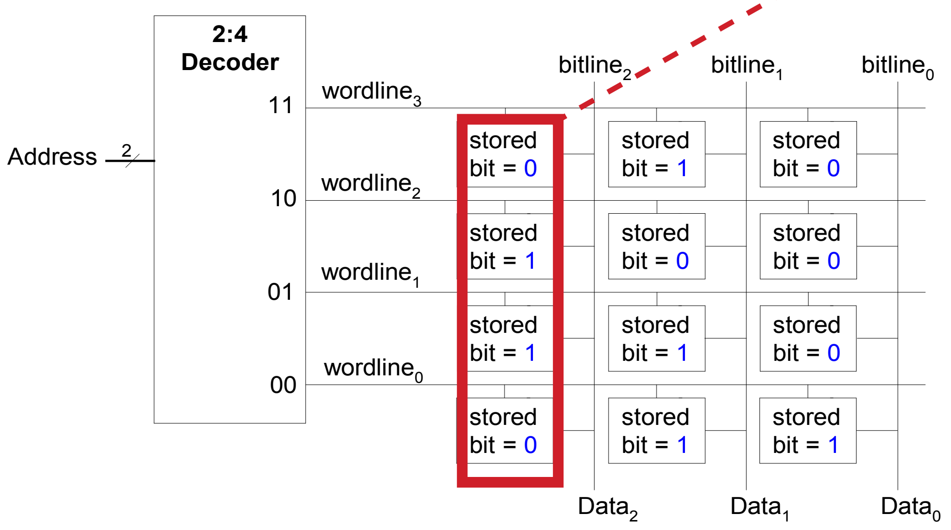
Bits stored in cells
(modeled by an SR latch)



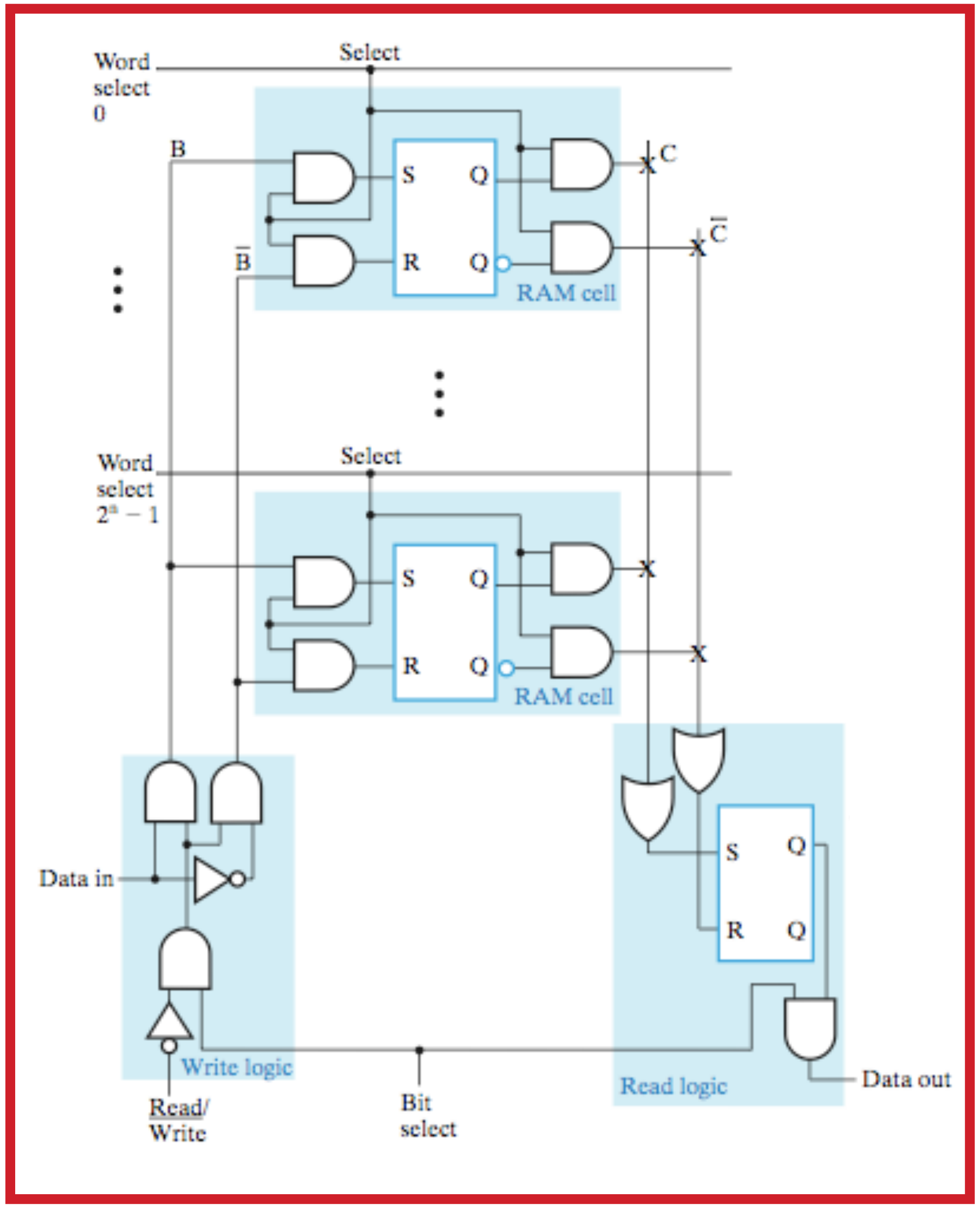


Bottom-up examination of SRAM circuits (3)

Cells wired into bitslices.

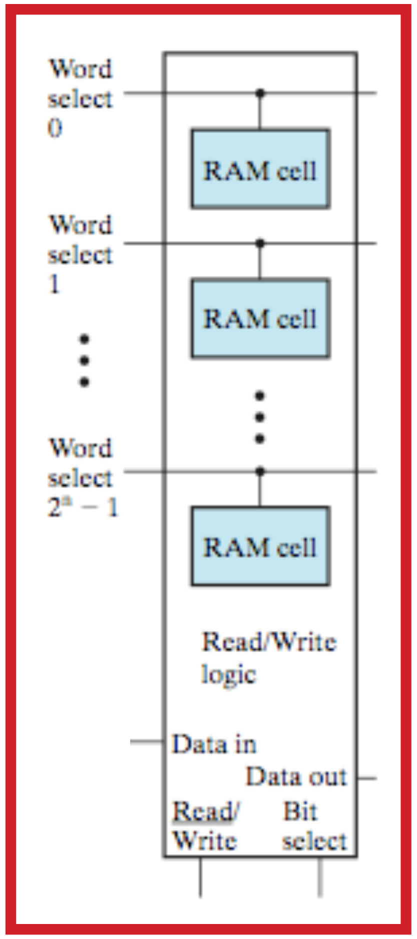


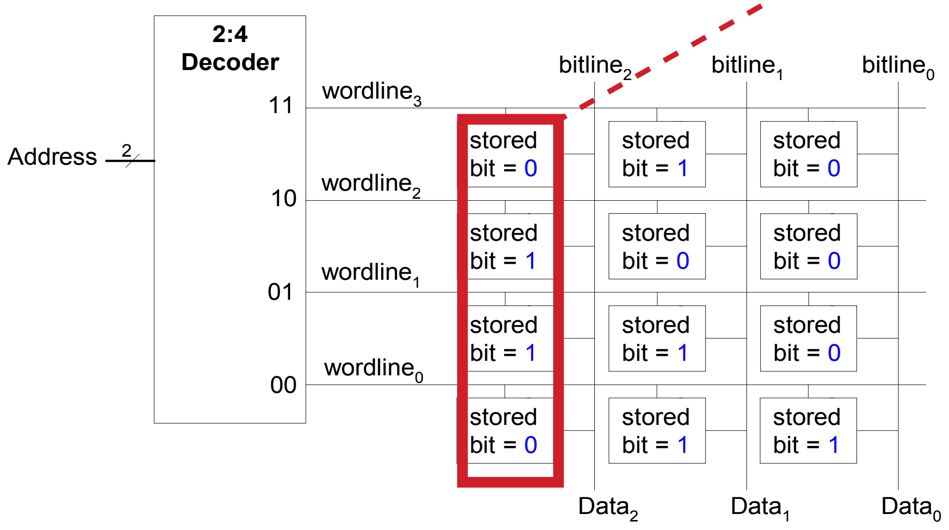
	bitline$_2$	bitline$_1$	bitline$_0$
wordline$_3$	stored bit = 0	stored bit = 1	stored bit = 0
wordline$_2$	stored bit = 1	stored bit = 0	stored bit = 0
wordline$_1$	stored bit = 1	stored bit = 1	stored bit = 0
wordline$_0$	stored bit = 0	stored bit = 1	stored bit = 1



Bottom-up examination of SRAM circuits (4)

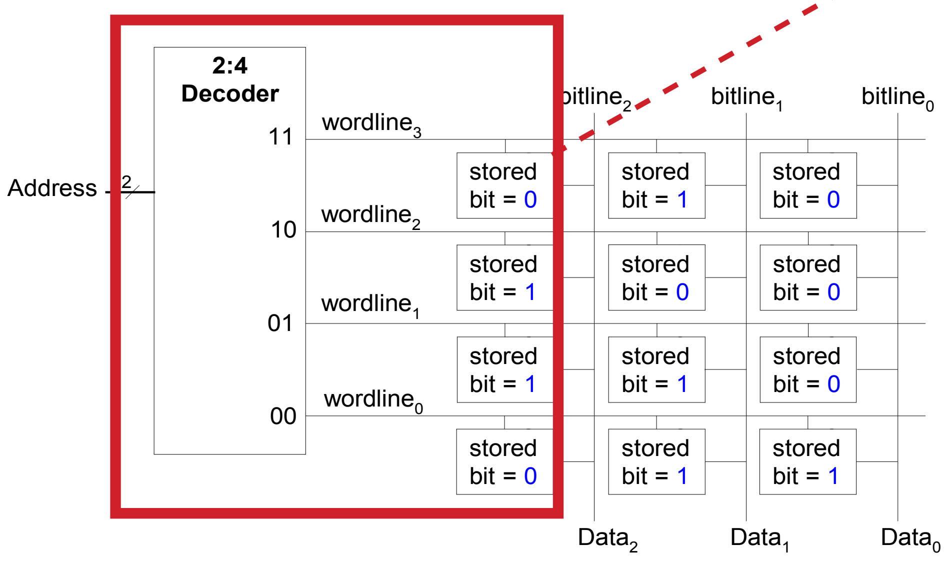
Block diagram of a bitslice

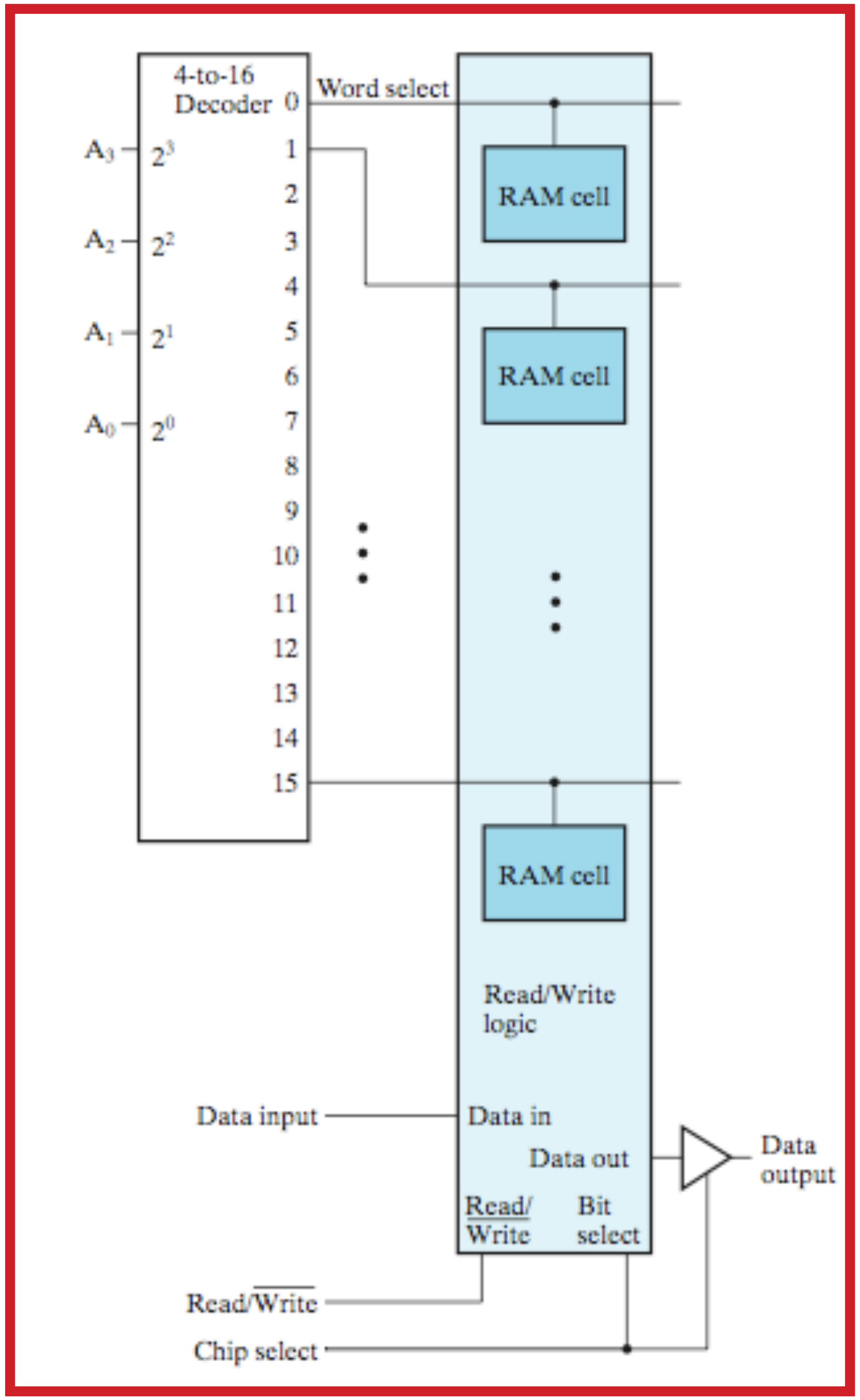




Bottom-up examination of SRAM circuits (5)

Address decoded to select one row of bitslice for read/write



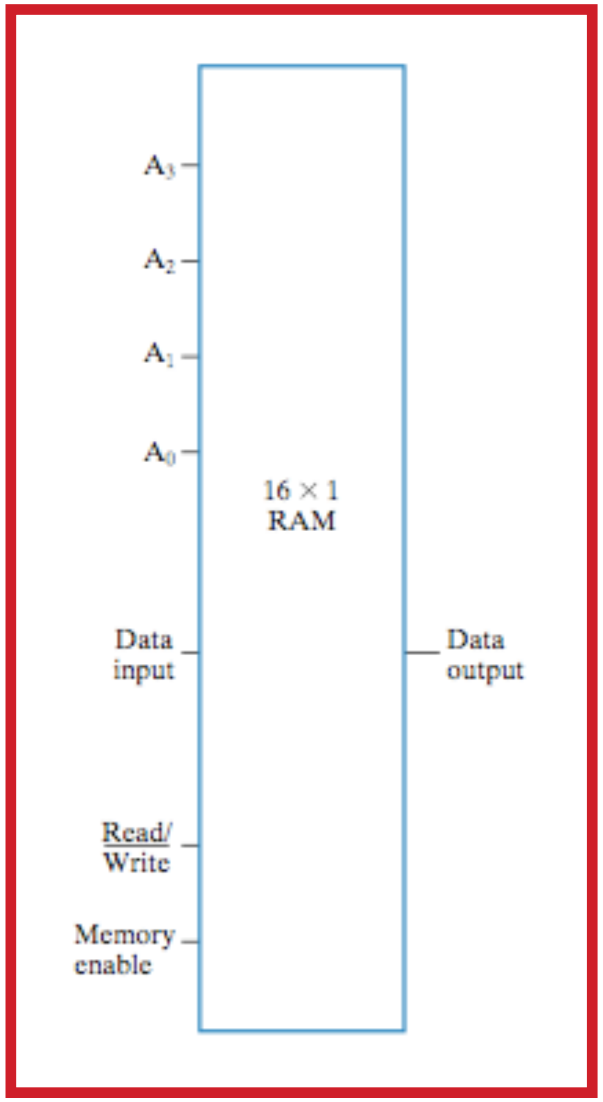


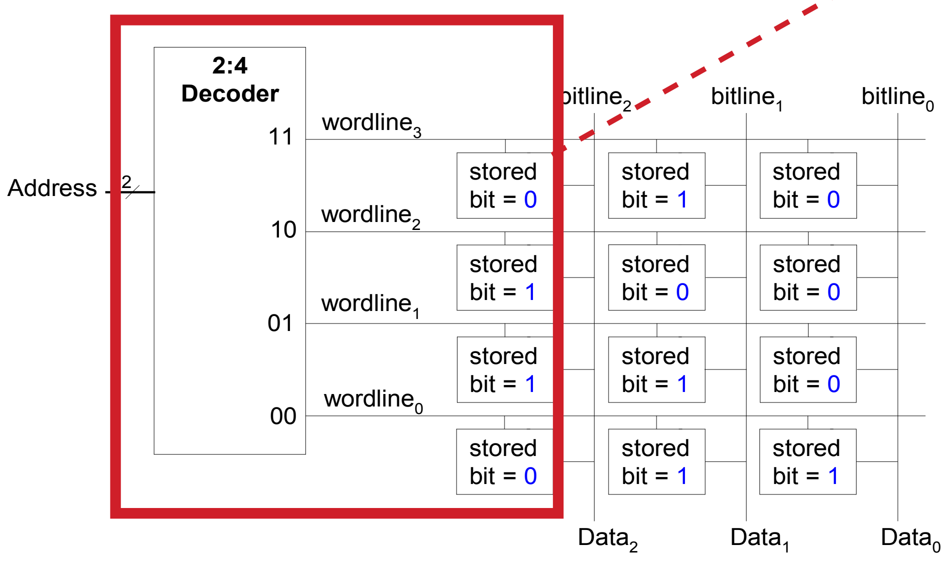
Copyright © 2007 Elsevier

© 2008 Pearson Education, Inc.
M. Morris Mano & Charles R. Kime
LOGIC AND COMPUTER DESIGN FUNDAMENTALS, 4e

Bottom-up examination of SRAM circuits (6)

To increase word size,
add bitslices.

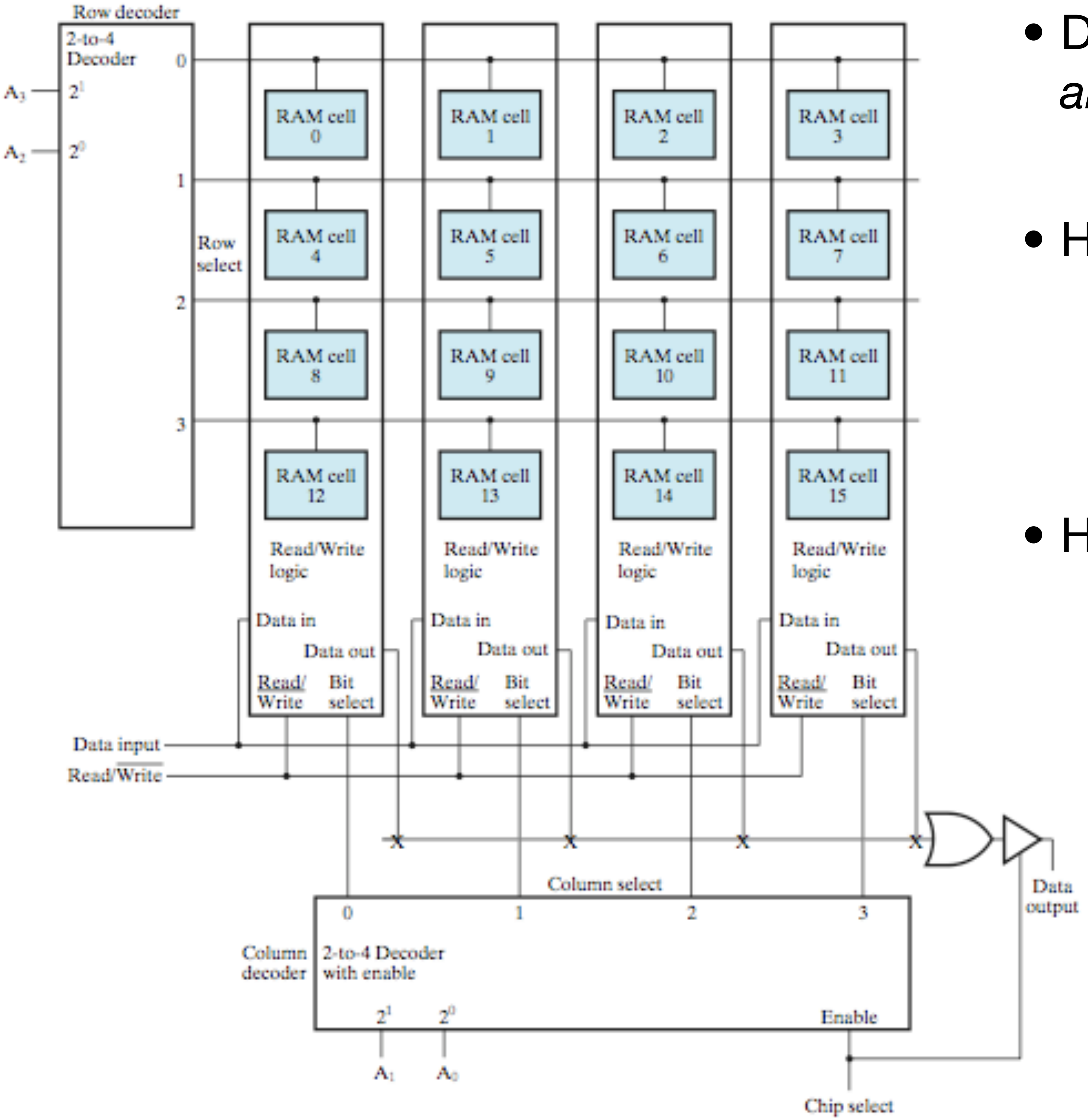




Copyright © 2007 Elsevier

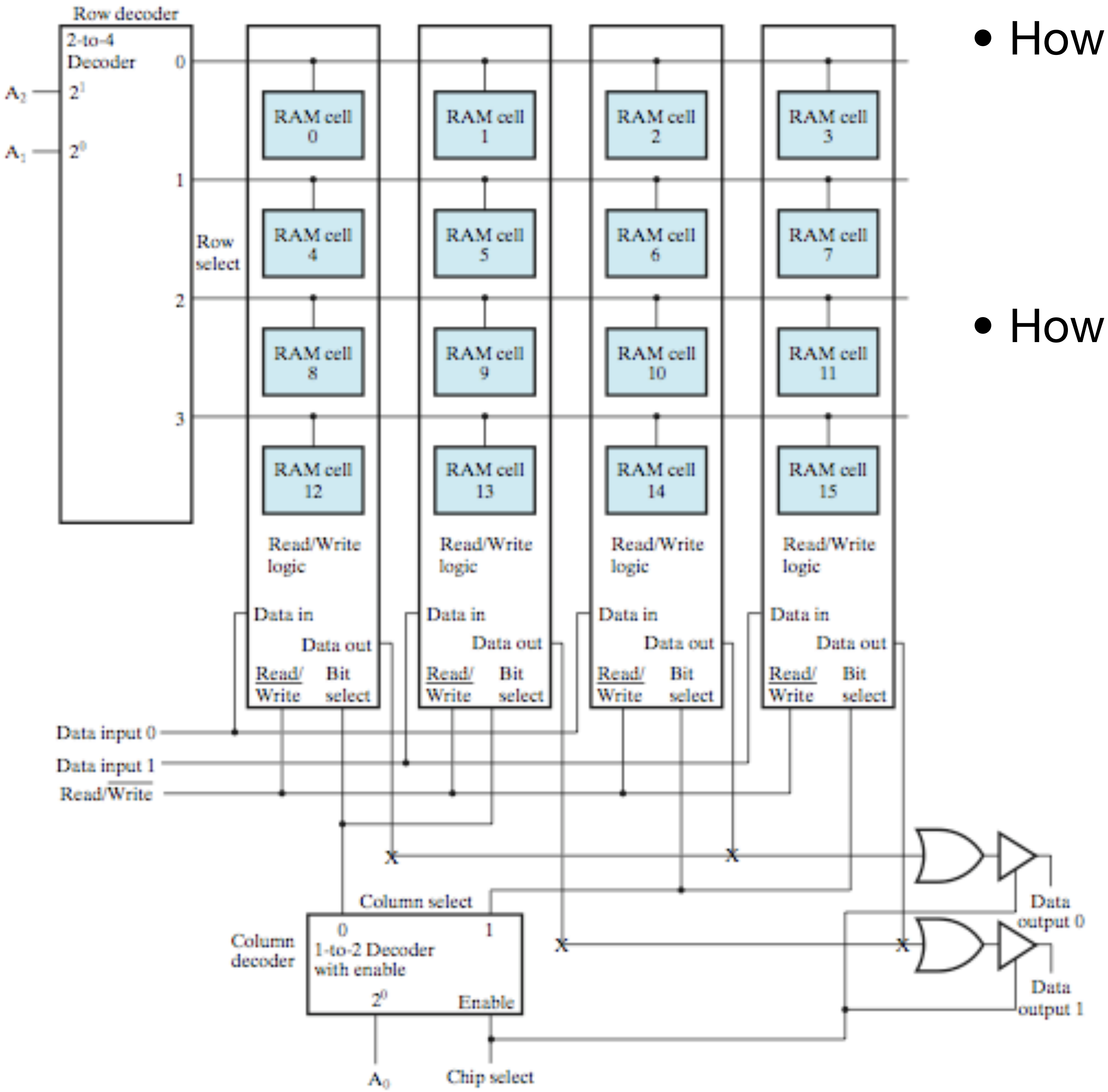
© 2008 Pearson Education, Inc.
M. Morris Mano & Charles R. Kime
LOGIC AND COMPUTER DESIGN FUNDAMENTALS, 4e

Coincident cell selection



- Decode address into *both* row *and* column select signals
- How many words in this RAM?
- How many bits per word?

Coincident cell selection w. larger words

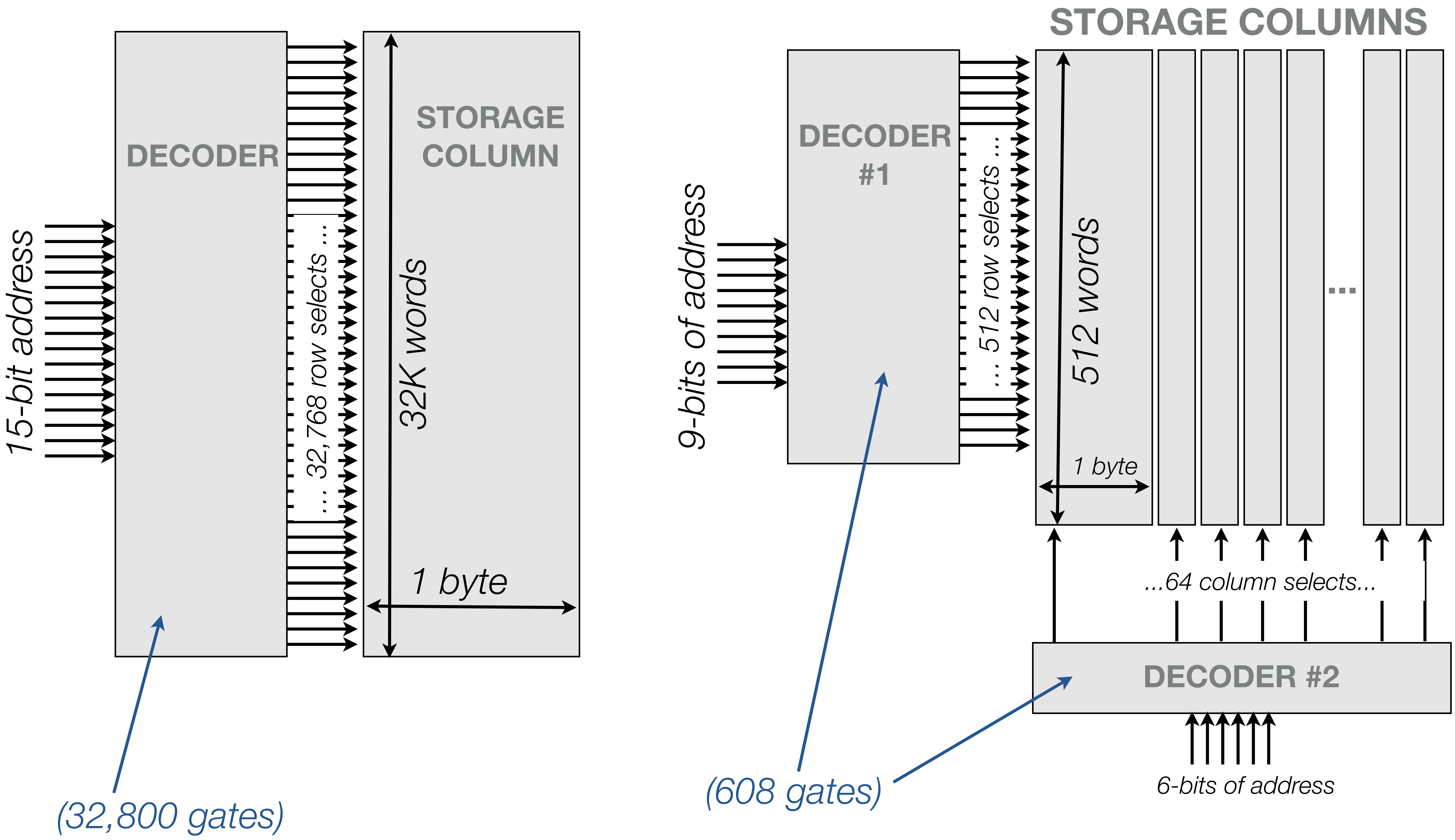


- How many words in this RAM?

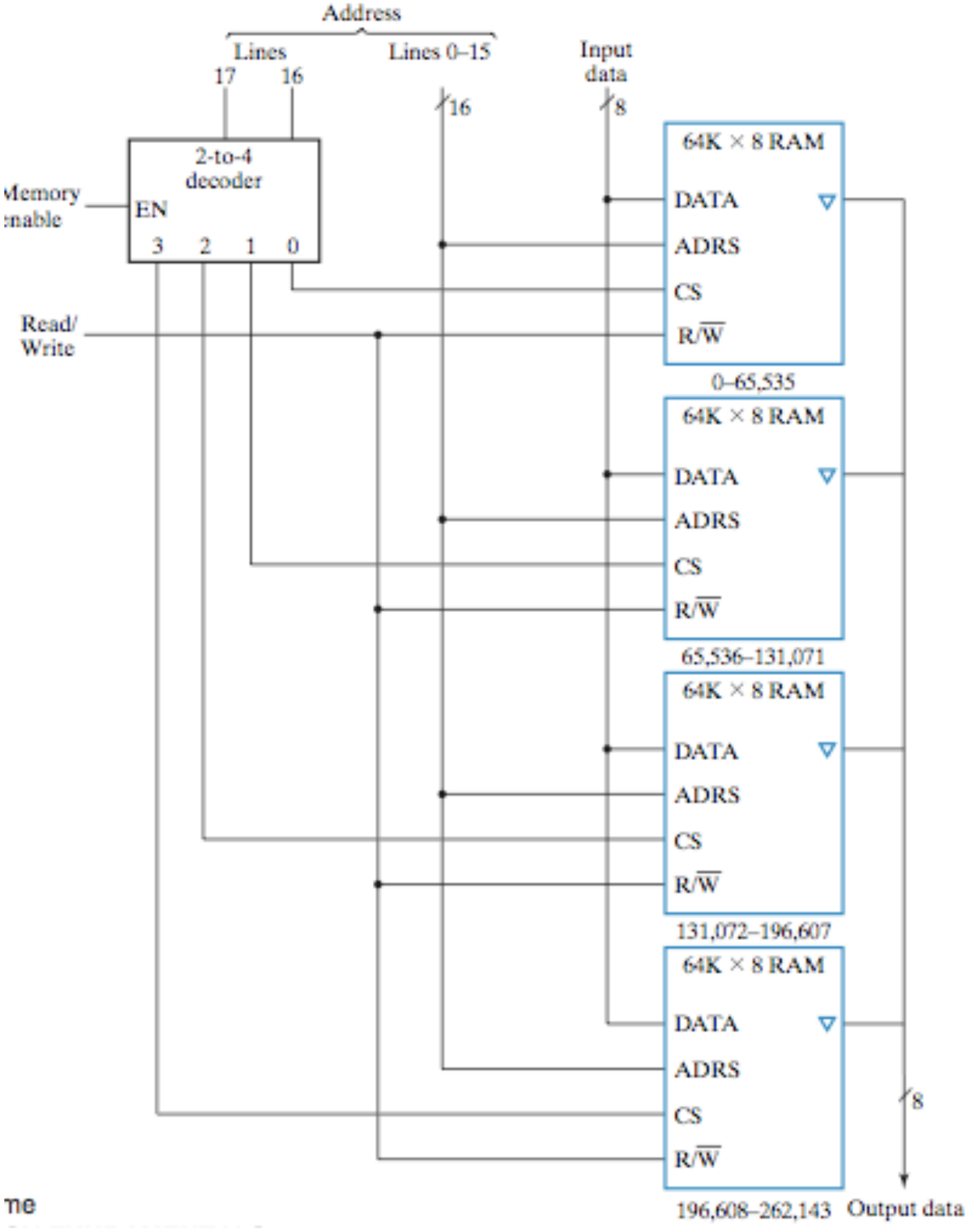
- How many bits per word?

M. Morris Mano & Charles R. Kime
LOGIC AND COMPUTER DESIGN FUNDAMENTALS, 4e

Coincident cell selection saves decode logic



Multi-chip memories

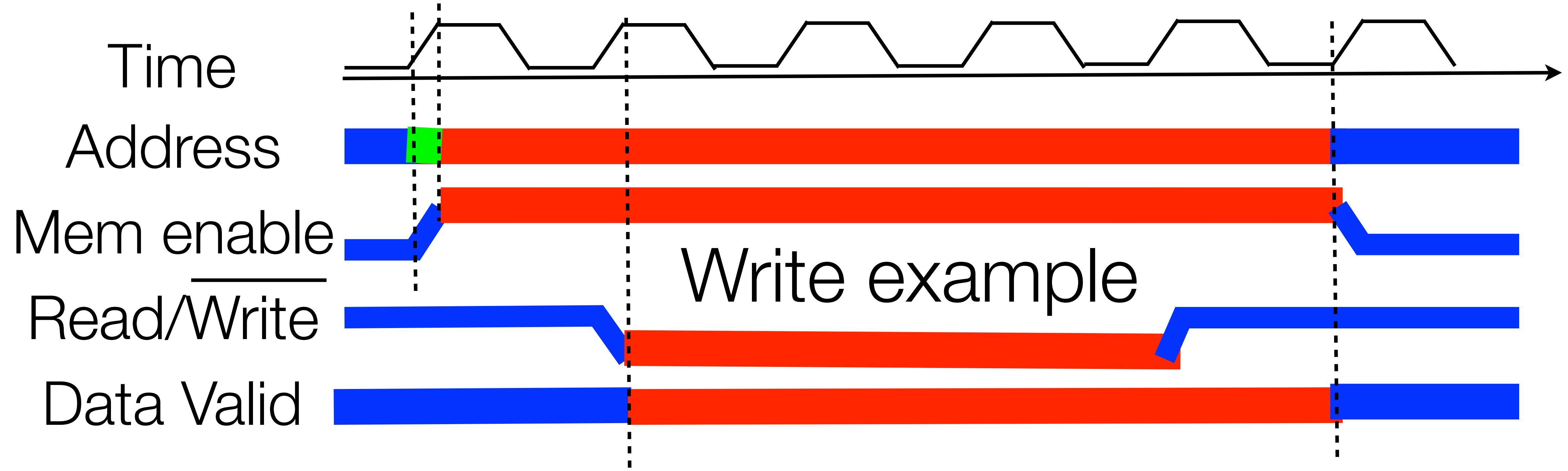


- If you need a larger memory than any available chip

- Wire multiple RAM chips together to work in concert as one large memory

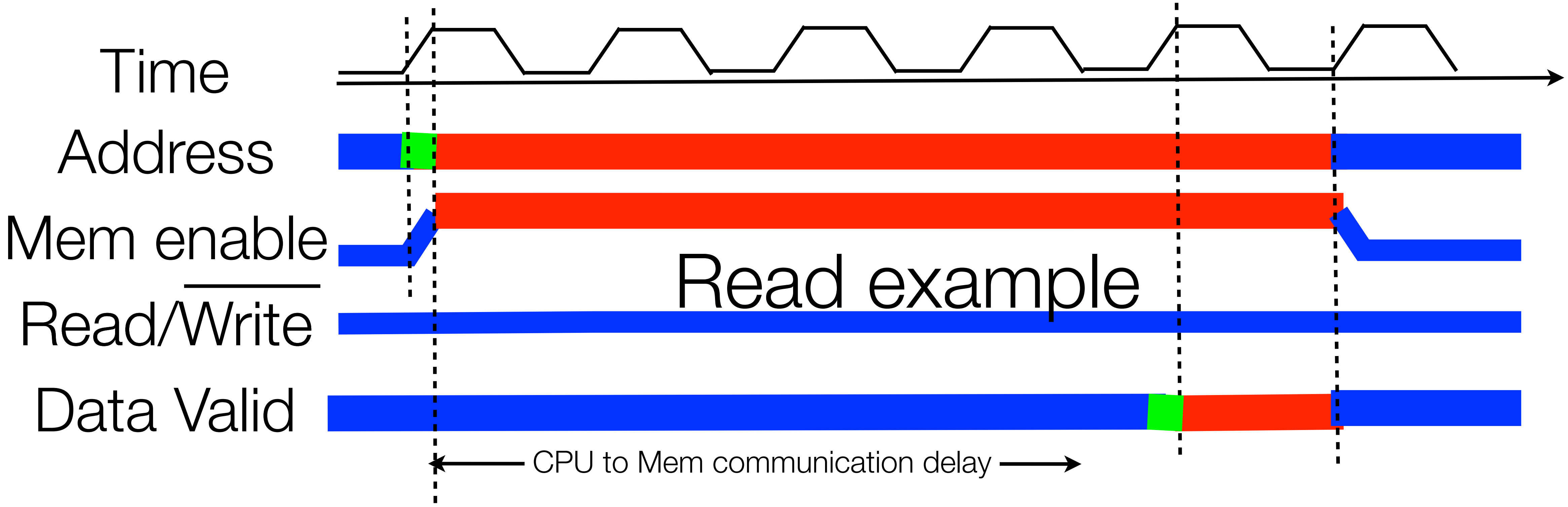
Memory Timing: Write example

- Even though memory not “on the clock”, timing still an issue:
  - inputs are “on the clock”
  - must first be properly enabled for reading or writing before data is transferred
    - Appropriate address chosen
    - Appropriate segment enabled
    - Appropriate read/write configuration set
    - Data valid: period during which writing must be performed



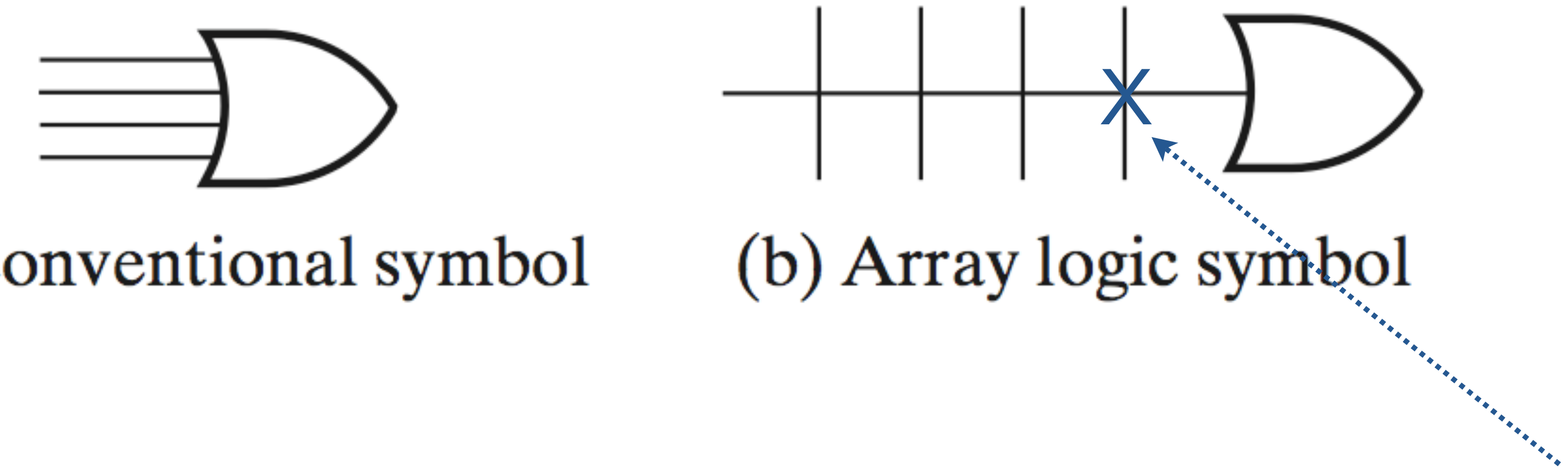
Memory Timing: Read example

- Even though memory not "on the clock", timing still an issue:
  - inputs are "on the clock"
  - must first be properly enabled for reading or writing before data is transferred
    - Appropriate address chosen
    - Appropriate segment enabled
    - Appropriate read/write configuration set
    - Data valid: period during which writing must be performed



Programmable Logic Devices

- Programmable logic devices (PLDs)
  - Structured like memories
  - Used to implement combinational logic



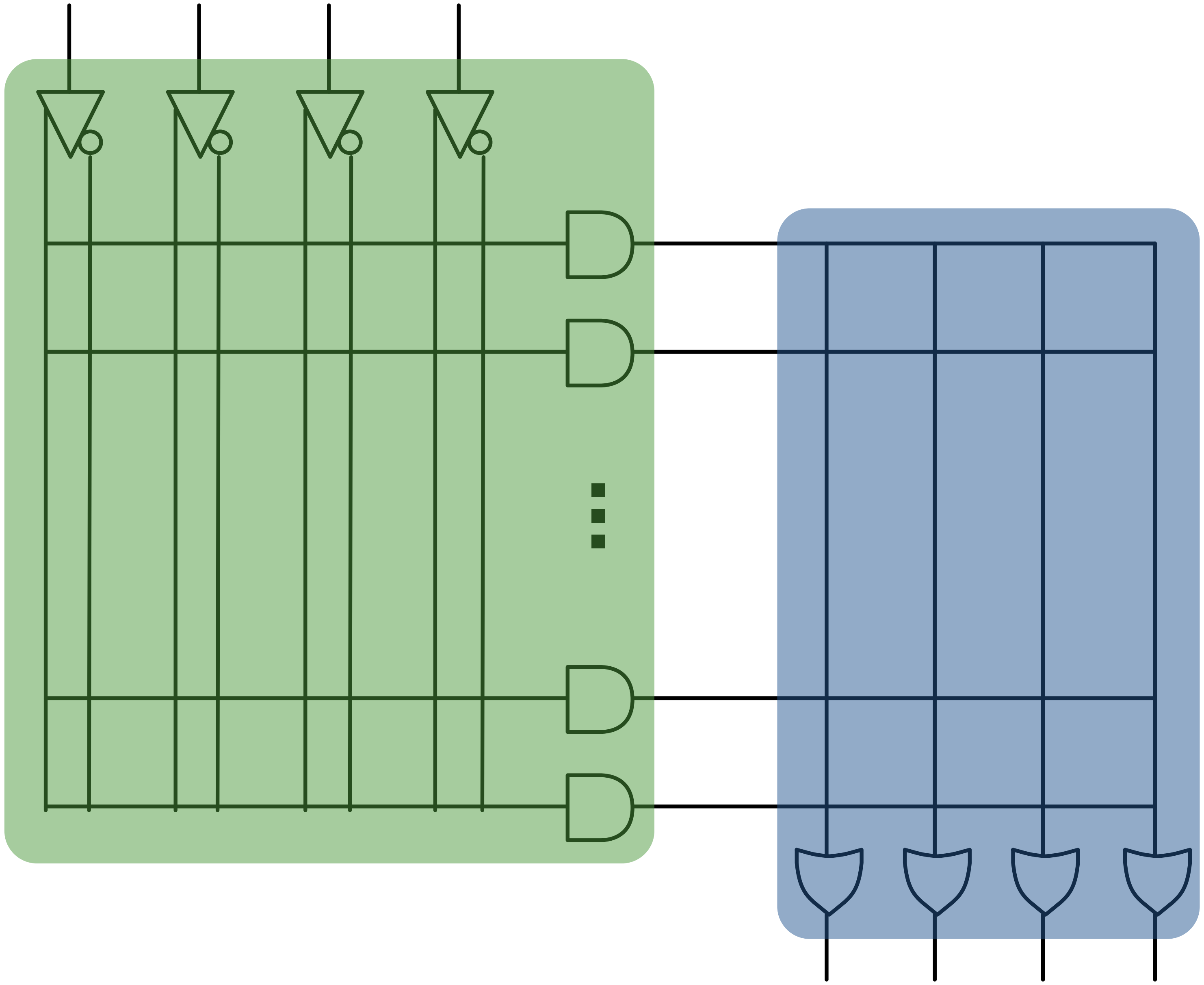
(a) Conventional symbol

(b) Array logic symbol

  - "X" on array logic means wire connected to logic gate (e.g. above)
  - Connections can be either permanent (e.g., fuse, mask) or not (e.g., Flash)

M. Morris Mano & Charles R. Kime
LOGIC AND COMPUTER DESIGN FUNDAMENTALS, 4e

General PLD architecture

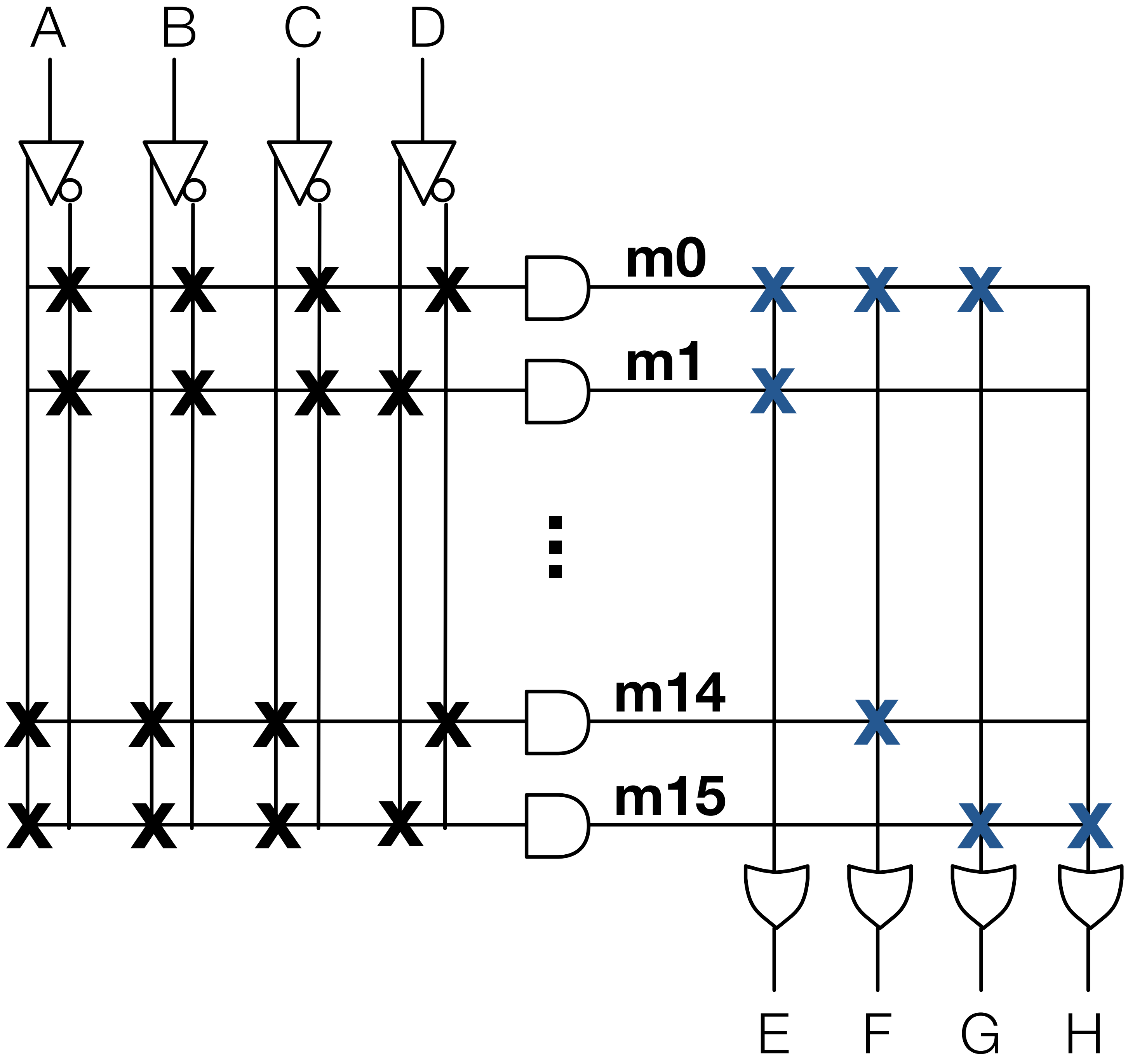


Fixed AND, programmable OR = Programmable ROM (PROM)

Programmable AND, fixed OR = Programmable Array Logic (PAL)

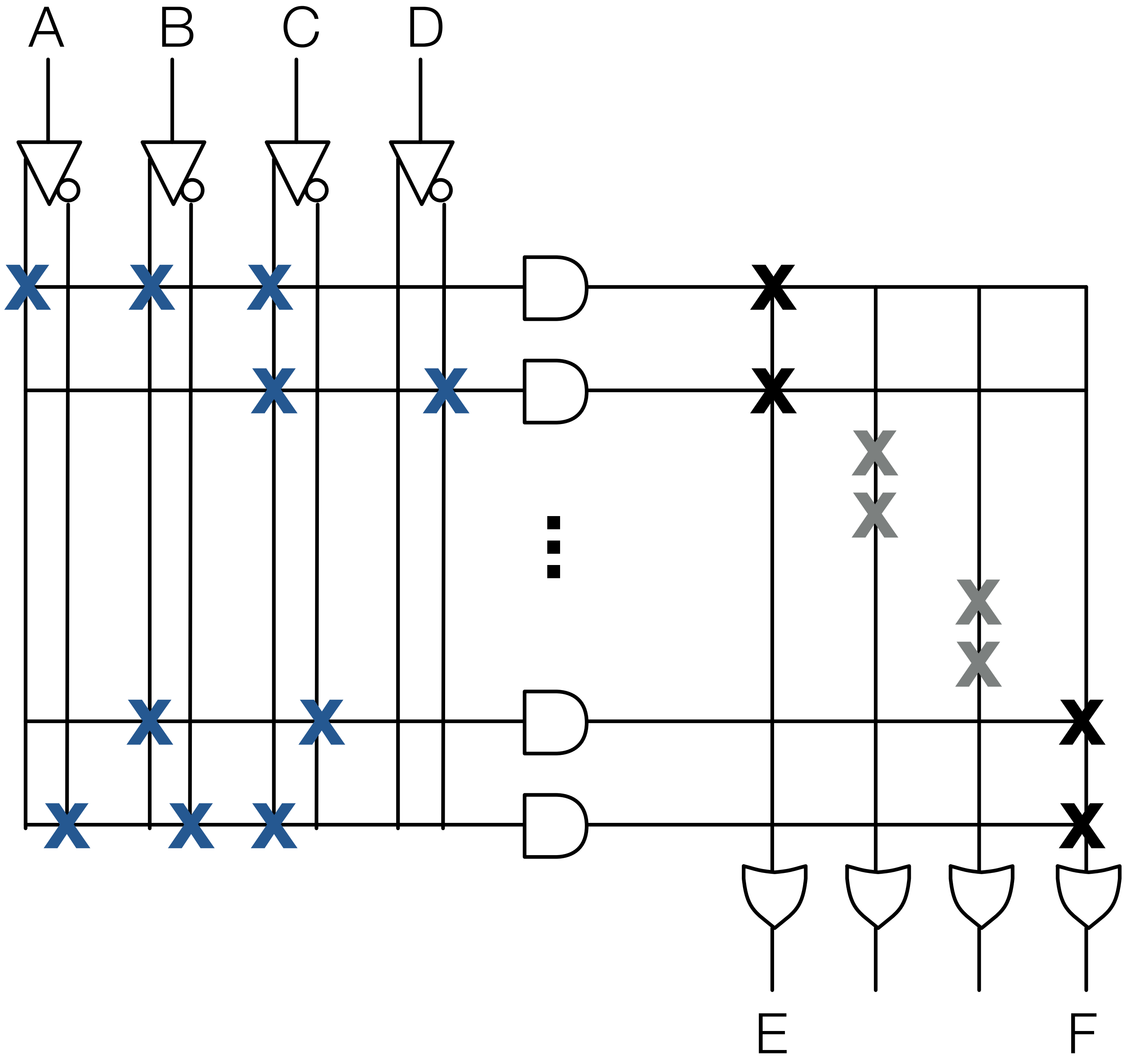
Programmable AND, programmable OR = Programmable Logic Array (PLA)

Programmable ROM (PROM)



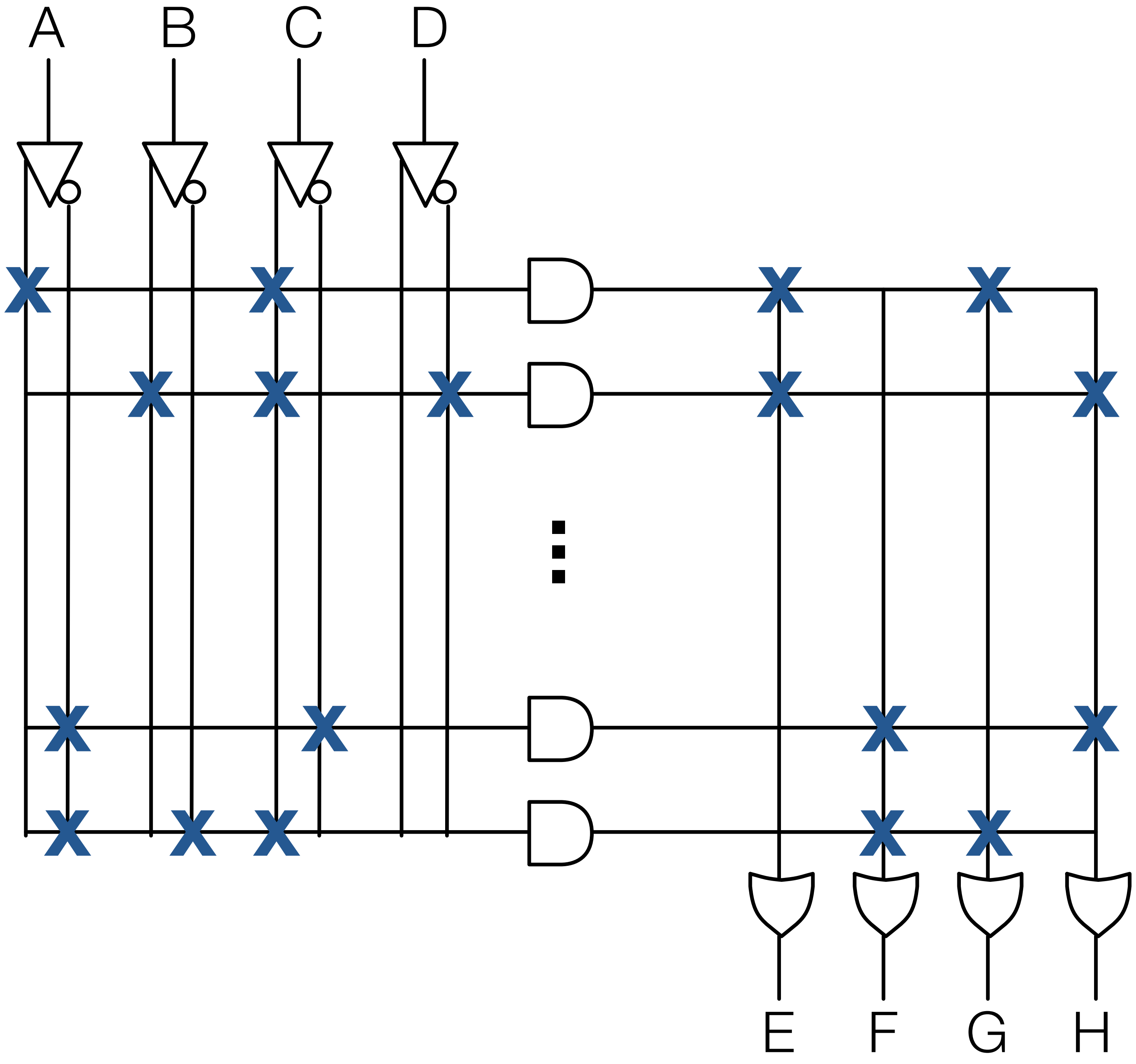
Fixed (**X**) AND, programmable (**X**) OR

Programmable Array Logic (PAL)



Programmable (X) AND, fixed (X) OR

Programmable Logic Array (PLA)



Programmable (X) AND, programmable (X) OR