Computer Organization and Architecture, Pt. 1

Charles Abzug, Ph.D.
Department of Computer Science
James Madison University
Harrisonburg, VA 22807

Voice Phone: 540-568-8746, E-mail: CharlesAbzug@ACM.org
Home Page: http://www.cs.jmu.edu/users/abzugcx

Carpinelli Table 1.1: BASIC BOOLEAN FUNCTIONS I

Carpinelli Table 1.2: BASIC BOOLEAN FUNCTIONS II

Notation for Boolean Logical Operations

A’ ≡ Ā ≡ ~A ≡ !A ≡ NOT A

$X \vee Y$ ≡ X+Y ≡ X OR Y

$X \wedge Y$ ≡ X•Y ≡ XY ≡ X AND Y

X NAND Y

X NOR Y

$X \oplus Y$ ≡ X XOR Y

X XNOR Y

Carpinelli Table 1.3: ALL POSSIBLE BOOLEAN FUNCTIONS of TWO INPUT VARIABLES

$\boxed{x \wedge y}$ $\boxed{x \oplus y}$ $\boxed{x \vee y}$

x	y	0	$\wedge$	xy'	x	$x'y$	y	$\oplus$	$\vee$	NOR	XNOR	y'	$x + y'$	x'	$x' + y$	NAND	1
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Carpinelli Table 1.4: TRUTH TABLE for a COMPOSITE BOOLEAN FUNCTION

$F(x,y) = x \bullet y' + y \bullet z$

Carpinelli Table 1.5: TRUTH TABLE for another COMPOSITE BOOLEAN FUNCTION

$$F(x,y) = (x + y') \bullet (y + z)$$

Carpinelli Table 1.6: TRUTH TABLES DEMONSTRATING ALGEBRAIC EQUIVALENCES RESULTING from DeMORGAN'S THEOREMS

$$F(x,y) = (x \bullet y' + y \bullet z)' = (x' + y) \bullet (y' + z) = x' \bullet y' + x' \bullet z' + y \bullet z'$$

Carpinelli Figure 1.1: BASIC KARNAUGH MAPS (3-variable and 4-variable)

Carpinelli Figure 1.2: GENERATION of GRAY CODE

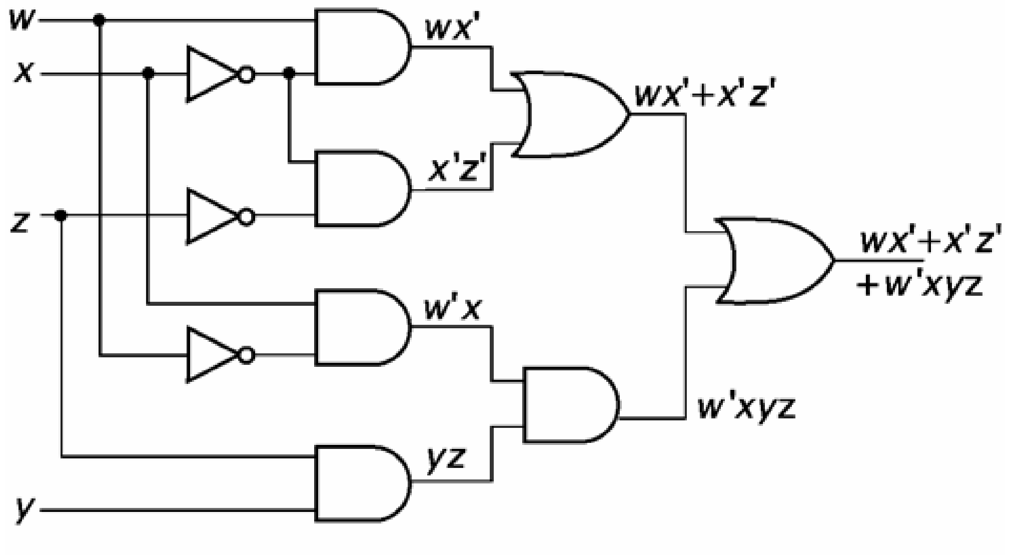
Carpinelli Figure 1.3: SAMPLE 3-VARIABLE KARNAUGH MAP SHOWING ALGEBRAIC SIMPLIFICATION

Carpinelli Figure 1.4: SAMPLE 4-VARIABLE KARNAUGH MAP SHOWING ALGEBRAIC SIMPLIFICATION

Carpinelli Figure 1.5 : ANOTHER 4-VARIABLE KARNAUGH MAP SHOWING ALGEBRAIC SIMPLIFICATION

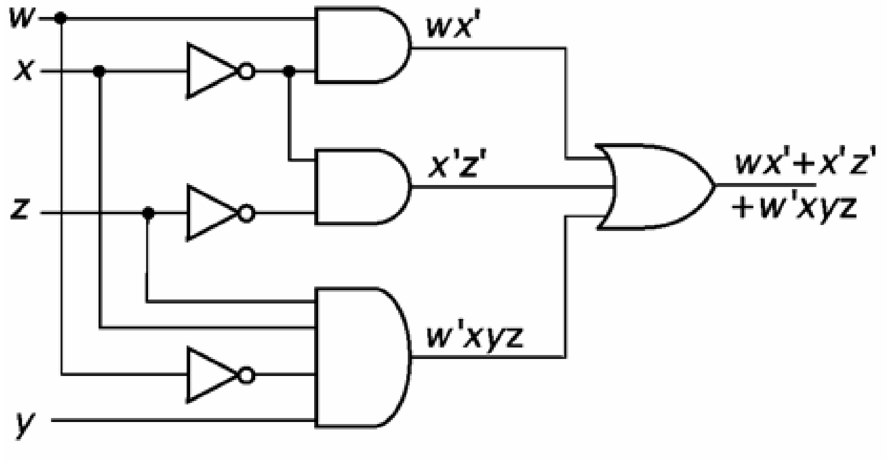
Carpinelli Figure 1.7: LOGIC CIRCUITS IMPLEMENTING a BOOLEAN FUNCTION

Carpinelli Figure 1.7: LOGIC CIRCUITS IMPLEMENTING a BOOLEAN FUNCTION



(a)

NOTE: This is a three-stage circuit — first stage AND gates, second stage one OR and one AND gate, third stage an OR gate. This is not how we implement a Boolean function normally.



(b)

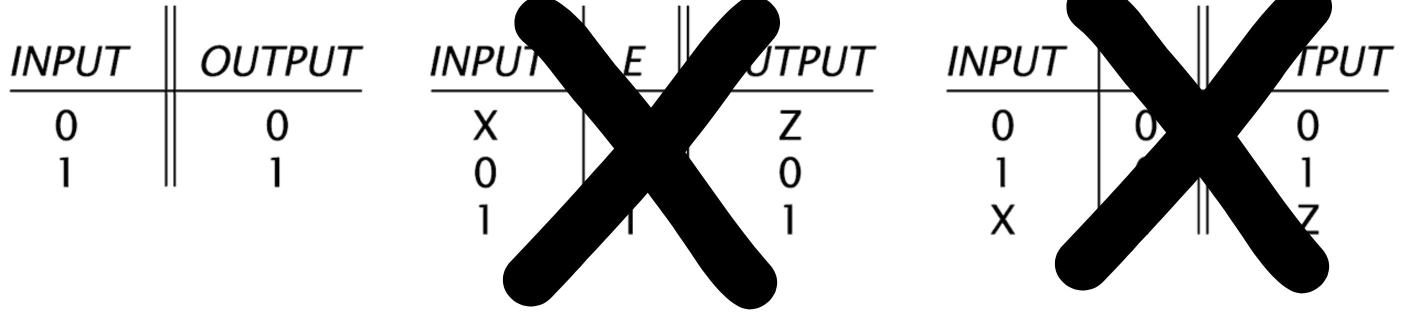
This is a normal logic-circuit implementation of a Boolean function.

Carpinelli Figure 1.8:
SIMPLE BUFFER GATE and TRI-STATE GATES

Carpinelli Table 1.7, as it appears in the text: TRUTH TABLES for BUFFER GATES

Simple Buffer Gate	Tri-State Buffer (Active-High Enable)	Tri-State Buffer (Active-Low Enable)

Carpinelli Table 1.7, TRUTH TABLES FOR BUFFER GATES *CORRECTED* to COUNTING ORDER:



Simple Buffer Gate

INPUT	*OUTPUT*
0	0
1	1

Tri-State Buffer (Active-High Enable)

INPU	*E*	*UTPUT*
X	[illegible]	Z
0	[illegible]	0
1	1	1

Tri-State Buffer (Active-Low Enable)

INPUT	[illegible]	*TPUT*
0	0	0
1	[illegible]	1
X	[illegible]	Z

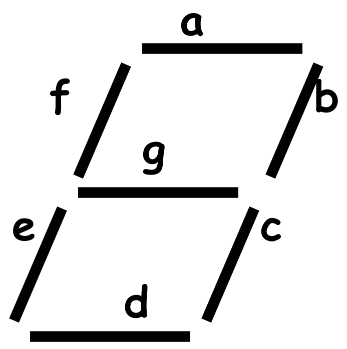
Original figure or table © 2001 by Addison Wesley Longman, Inc

Truth Tables in Counting Order:

E	*In*	*Out*
0	X	Z
1	0	0
1	1	1

E	*In*	*Out*
0	0	0
0	1	1
1	X	Z

BCD-to-Seven-Segment Decoder: SEGMENT LABELS



Carpinelli Figure 1.22:
BCD-to-SEVEN-SEGMENT DECODERS:
TRUTH TABLES for all SEVEN BOOLEAN FUNCTIONS

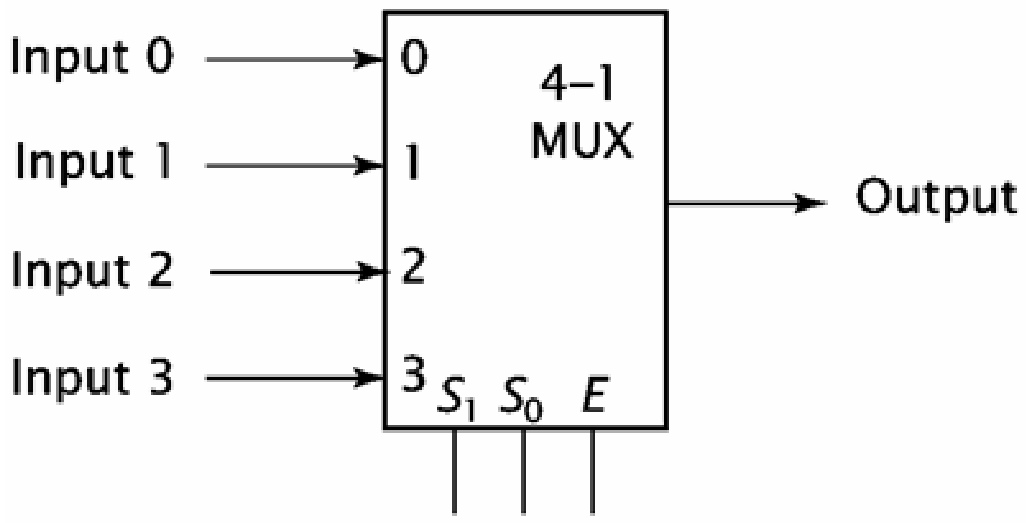
Carpinelli Figure 1.23 (a): KARNAUGH MAP SIMPLIFICATION of TWO BOOLEAN FUNCTIONS

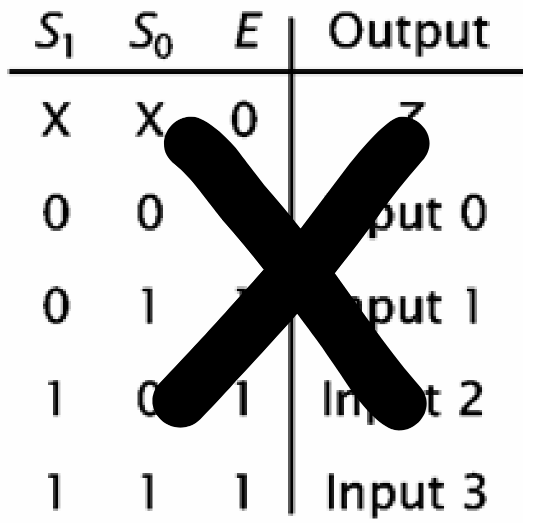
Carpinelli Figure 1.23 (b): LOGICAL CIRCUITS IMPLEMENTING TWO BOOLEAN FUNCTIONS

Carpinelli Figure 1.9 (a): MULTIPLEXOR

Carpinelli Figure 1.9 (b), as it appears in the text:

Carpinelli Figure 1.9 (b), *CORRECTED:*





S_1	S_0	E	Output
~~X~~	~~X~~	~~0~~	~~Z~~
~~0~~	~~0~~	[illegible]	~~Input 0~~
~~0~~	~~1~~	~~1~~	~~Input 1~~
~~1~~	~~0~~	~~1~~	~~Input 2~~
~~1~~	~~1~~	~~1~~	~~Input 3~~

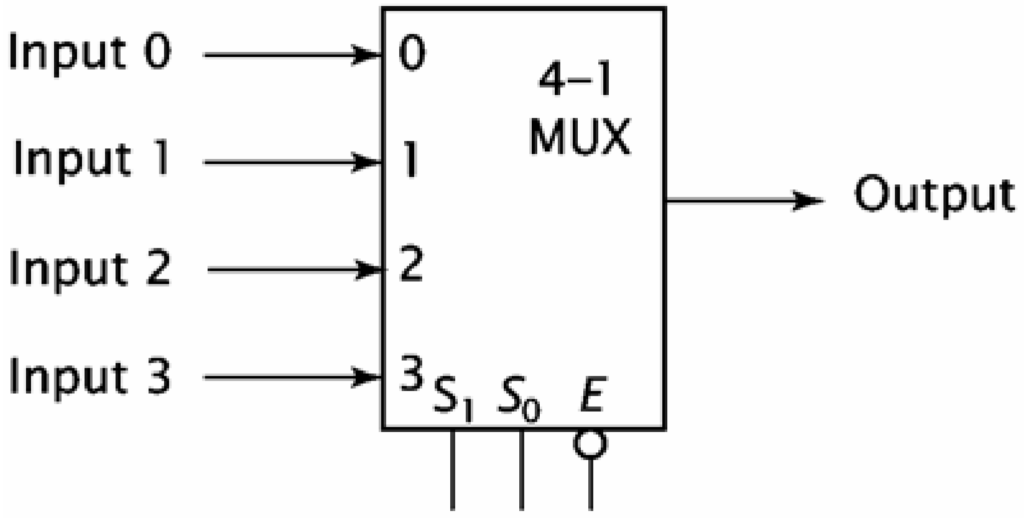
Original figure or table © 2001 by Addison Wesley Longman, Inc

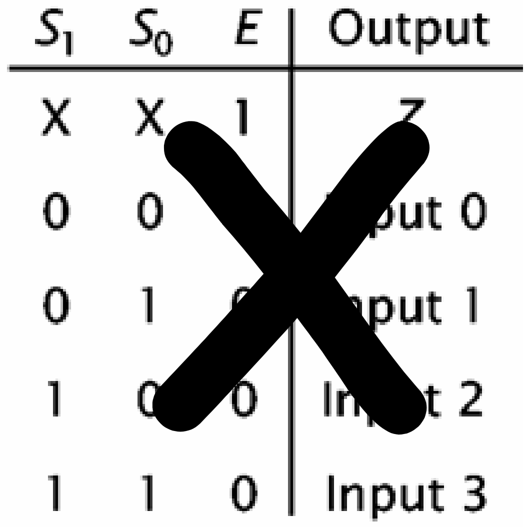
Truth Table is now in <u>Counting Order</u>:

E	*S_1*	*S_0*	*Out*
0	X	X	Z
1	0	0	Input 0
1	0	1	Input 1
1	1	0	Input 2
1	1	1	Input 3

Carpinelli Figure 1.9 (c), as it appears in the text:

Carpinelli Figure 1.9 (c), *CORRECTED:*





~~S_1~~	~~S_0~~	~~E~~	~~Output~~
~~X~~	~~X~~	~~1~~	~~Z~~
~~0~~	~~0~~	~~0~~	~~Input 0~~
~~0~~	~~1~~	~~0~~	~~Input 1~~
~~1~~	~~0~~	~~0~~	~~Input 2~~
~~1~~	~~1~~	~~0~~	~~Input 3~~

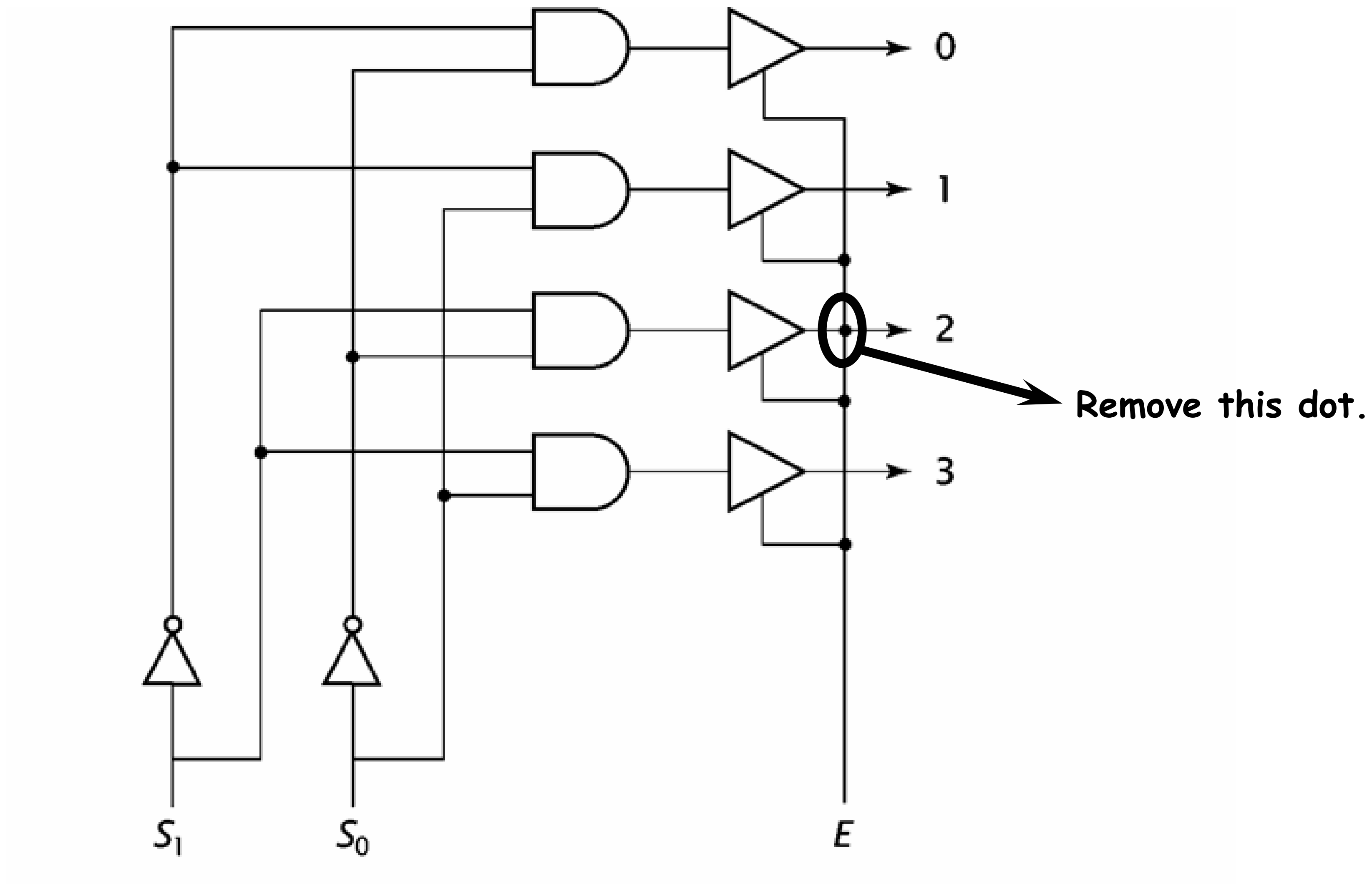
Truth Table is now in Counting Order:

E	*S_1*	*S_0*	*Out*
0	0	0	Input 0
0	0	1	Input 1
0	1	0	Input 2
0	1	1	Input 3
1	X	X	Z

Carpinelli Figure 1.10: CONSTRUCTION of a 4-1 MULTIPLEXOR from THREE 2-1 MULTIPLEXORS

Carpinelli Figure 1.11 (a), as it appears in the text: DECODER

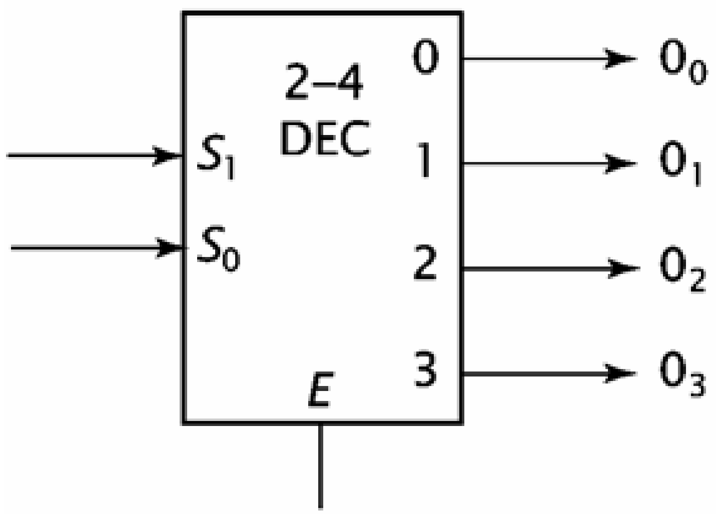
Carpinelli Figure 1.11 (a), showing CORRECTION:

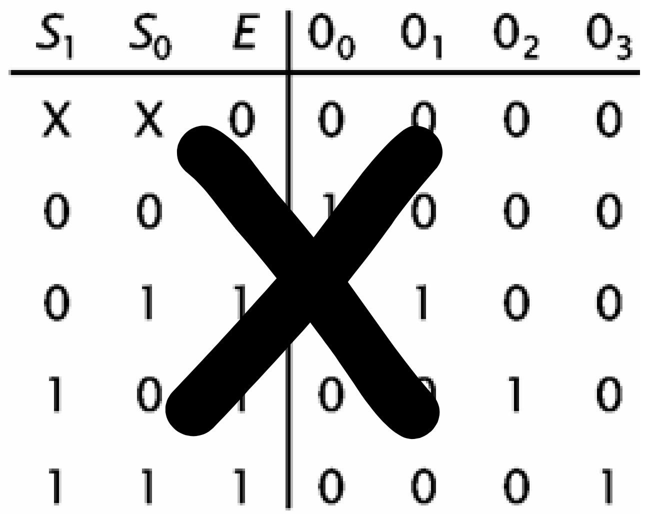


Original figure or table © 2001 by Addison Wesley Longman, Inc

Carpinelli Figure 1.11 (b), as it appears in the text:

Carpinelli Figure 1.11 (b), *CORRRECTED:*





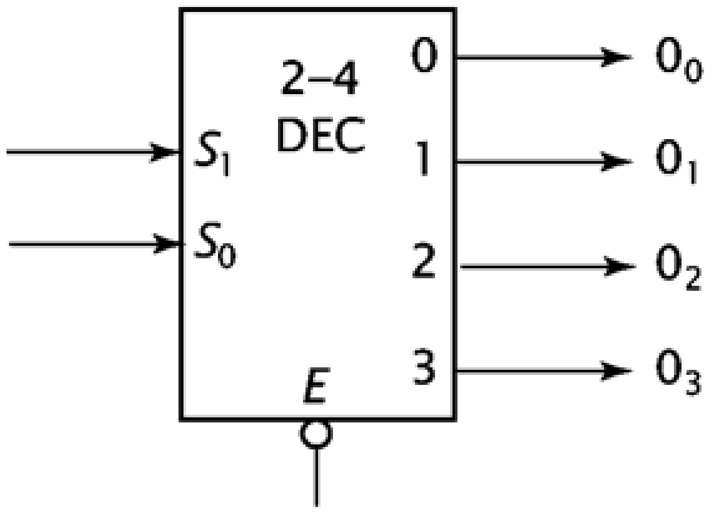
S_1	S_0	E	O_0	O_1	O_2	O_3
X	X	0	0	0	0	0
0	0	1	1	0	0	0
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	1

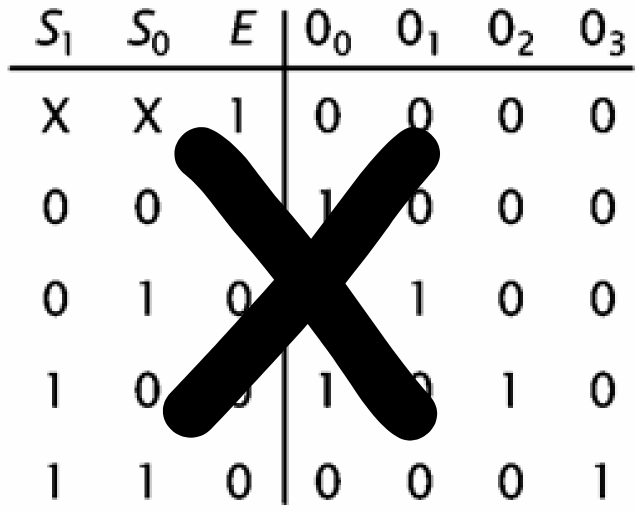
Truth Table with several corrections, and in Counting Order:

E	*S_1*	*S_0*	*Out_0*	*Out_1*	*Out_2*	*Out_3*
0	X	X	Z	Z	Z	Z
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	1	1

Carpinelli Figure 1.11 (c), as it appears in the text:

Carpinelli Figure 1.11 (c), *CORRECTED:*





~~S_1~~	~~S_0~~	~~E~~	~~O_0~~	~~O_1~~	~~O_2~~	~~O_3~~
~~X~~	~~X~~	~~1~~	~~0~~	~~0~~	~~0~~	~~0~~
~~0~~	~~0~~	~~0~~	~~1~~	~~0~~	~~0~~	~~0~~
~~0~~	~~1~~	~~0~~	~~0~~	~~1~~	~~0~~	~~0~~
~~1~~	~~0~~	~~0~~	~~1~~	~~0~~	~~1~~	~~0~~
~~1~~	~~1~~	~~0~~	~~0~~	~~0~~	~~0~~	~~1~~

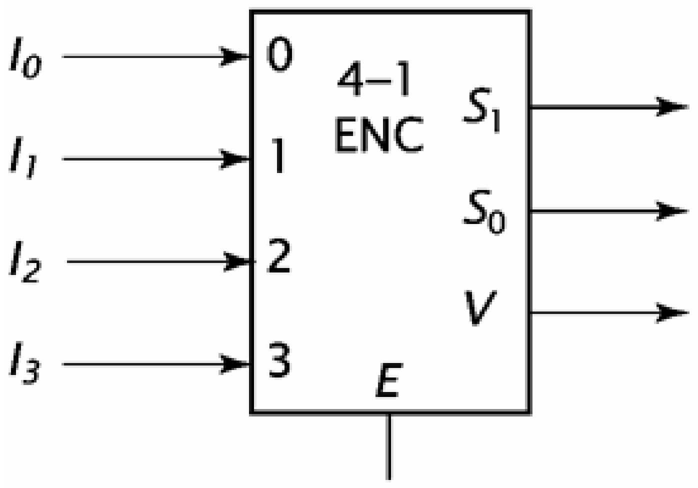
Truth Table with several corrections, and in Counting Order:

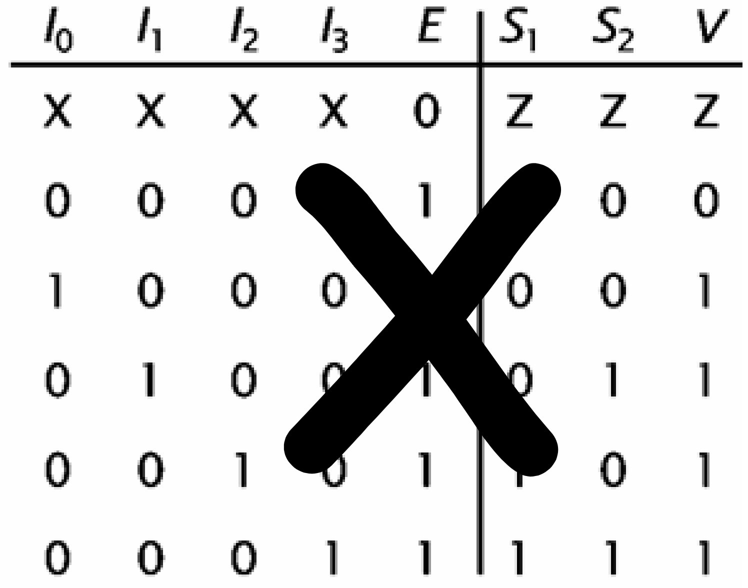
E	*S_1*	*S_0*	*Out_0*	*Out_1*	*Out_2*	*Out_3*
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	1	1
1	X	X	Z	Z	Z	Z

Carpinelli Figure 1.12 (a): SIMPLE ENCODER

Carpinelli Figure 1.12 (b), as it appears in the text:

Carpinelli Figure 1.12 (b), *CORRECTED:*





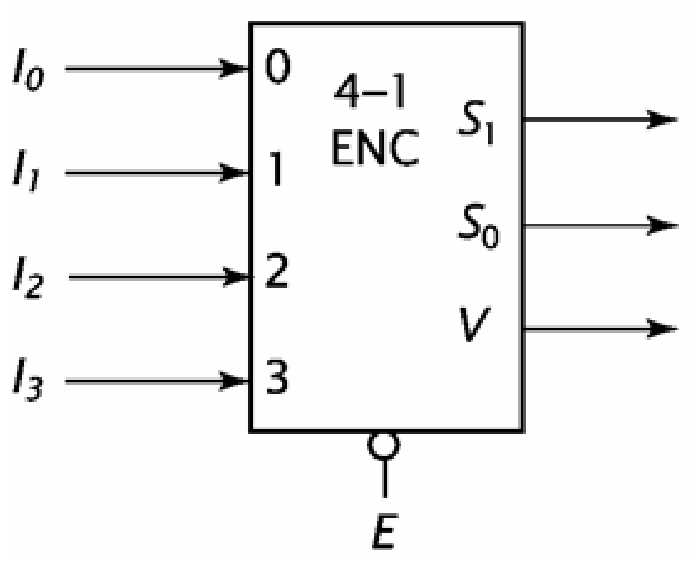
I_0	I_1	I_2	I_3	E	S_1	S_2	V
~~X~~	~~X~~	~~X~~	~~X~~	~~0~~	~~Z~~	~~Z~~	~~Z~~
~~0~~	~~0~~	~~0~~	[illegible]	~~1~~	[illegible]	~~0~~	~~0~~
~~1~~	~~0~~	~~0~~	~~0~~	[illegible]	~~0~~	~~0~~	~~1~~
~~0~~	~~1~~	~~0~~	~~0~~	~~1~~	~~0~~	~~1~~	~~1~~
~~0~~	~~0~~	~~1~~	~~0~~	~~1~~	~~1~~	~~0~~	~~1~~
~~0~~	~~0~~	~~0~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~

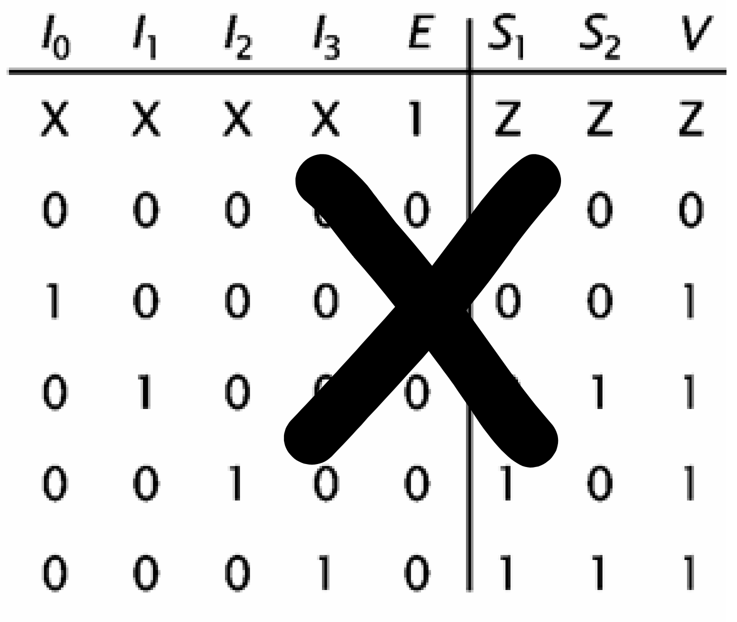
Abbreviated Truth Table
in Counting Order
(not all input combinations are
shown, since external constraints
allow at most one input line
to have a value of 1):

E	I_3	I_2	I_1	I_0	S_1	S_0	V
0	X	X	X	X	Z	Z	Z
1	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	1	1
1	0	1	0	0	1	0	1
1	1	0	0	0	1	1	1

Carpinelli Figure 1.12 (c), as it appears in the text:

Carpinelli Figure 1.12 (c), *CORRECTED:*





I_0	I_1	I_2	I_3	E	S_1	S_2	V
X	X	X	X	1	Z	Z	Z
0	0	0	0	0	[illegible]	0	0
1	0	0	0	[illegible]	0	0	1
0	1	0	0	0	[illegible]	1	1
0	0	1	0	0	1	0	1
0	0	0	1	0	1	1	1

Abbreviated Truth Table
in Counting Order
(not all input combinations are
shown, since external constraints
allow at most one input line
to have a value of 1):

E	I_3	I_2	I_1	I_0	S_1	S_0	*V*
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1
0	0	0	1	0	0	1	1
0	0	1	0	0	1	0	1
0	1	0	0	0	1	1	1
1	X	X	X	X	Z	Z	Z

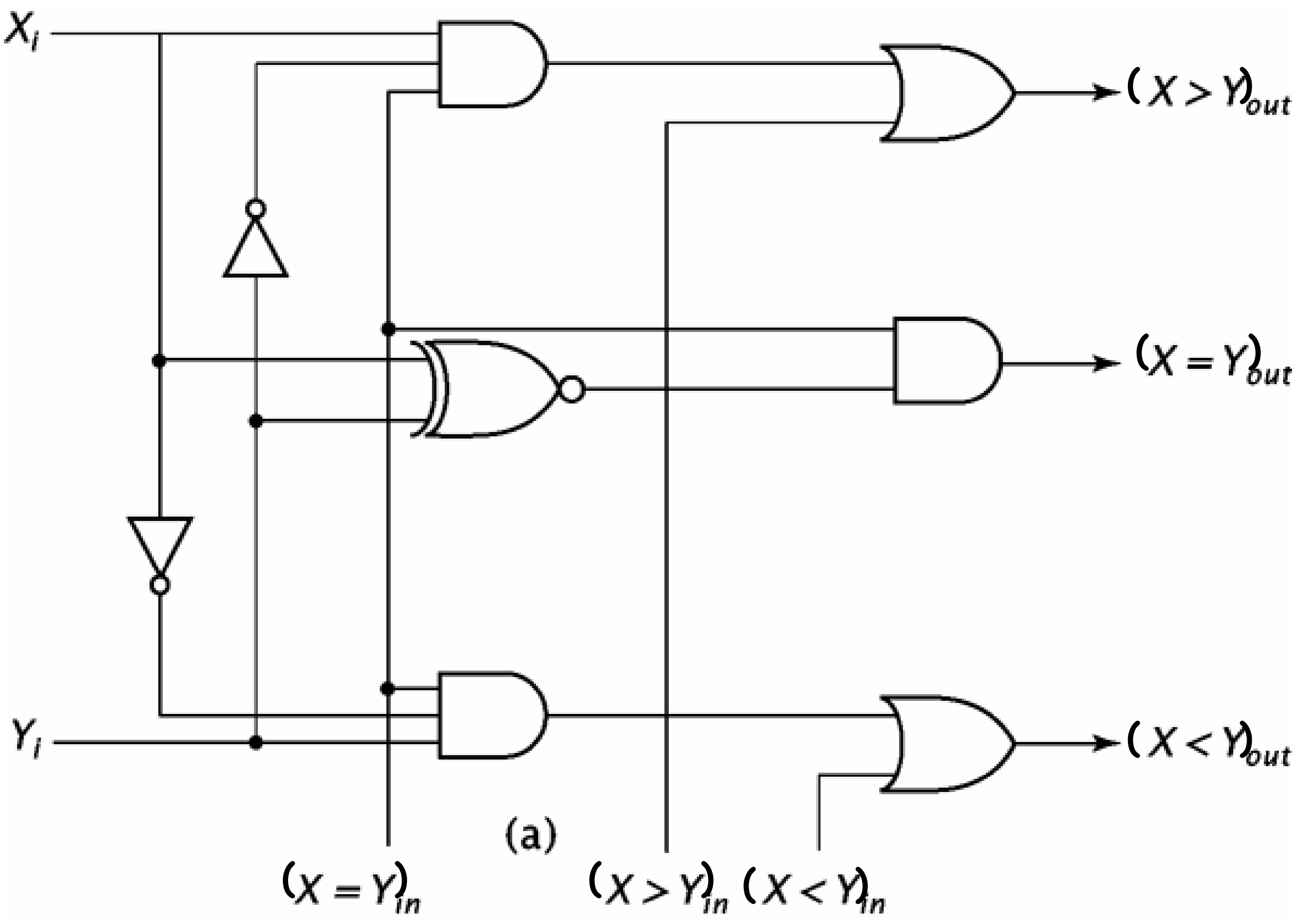
Carpinelli Figure 1.13 (a): PRIORITY ENCODER

Carpinelli Figure 1.13 (b) and (c) PRIORITY ENCODER (alternative version)

Carpinelli Figure 1.14: ONE-BIT COMPARATOR

Carpinelli Figure 1.15, as it appears in the text: SINGLE-BIT STAGE of a MULTI-BIT COMPARATOR

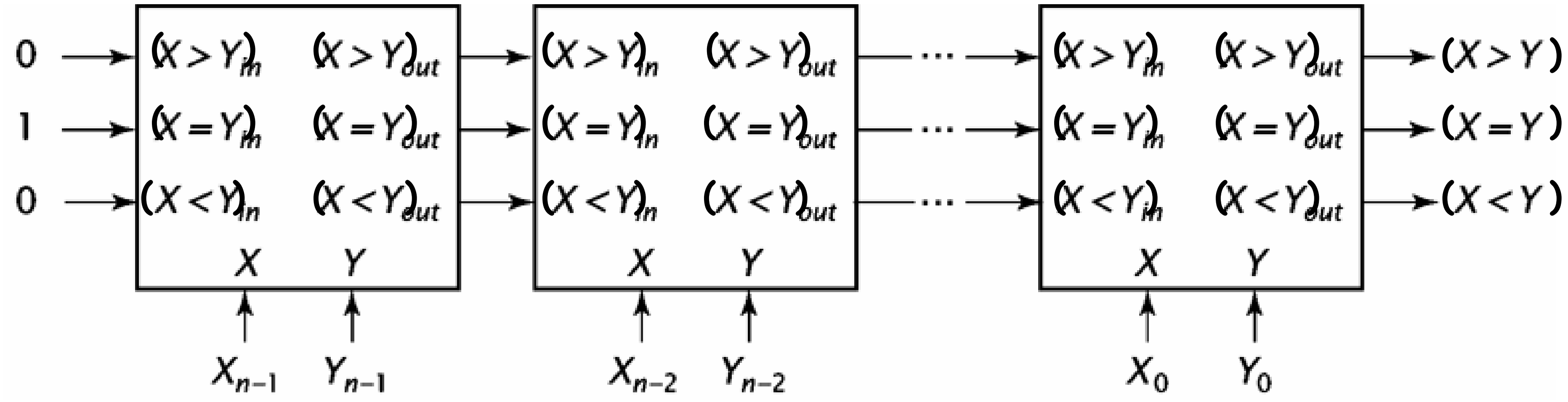
Carpinelli Figure 1.15, *CORRECTED*: SINGLE-BIT STAGE of a MULTI-BIT COMPARATOR



Original figure or table © 2001 by Addison Wesley Longman, Inc

Carpinelli Figure 1.16, as it appears in the text:

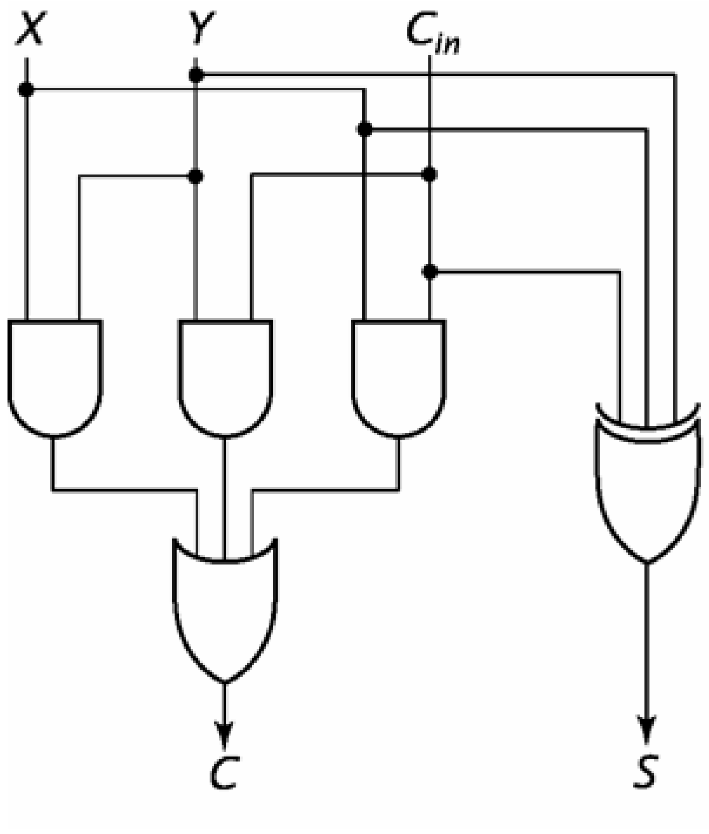
Carpinelli Figure 1.16, *CORRECTED:*

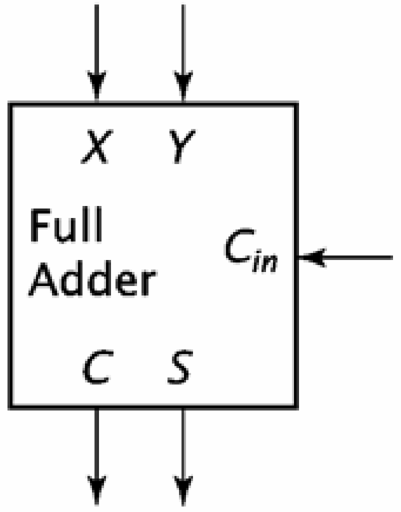


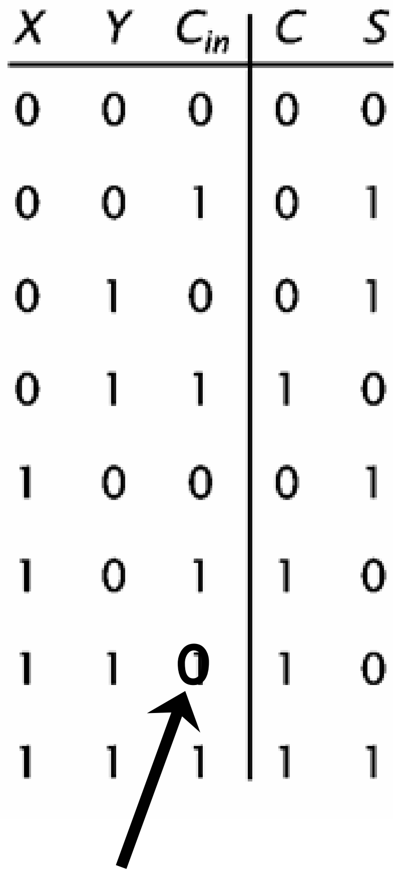
Carpinelli Figure 1.17: HALF ADDER

Carpinelli Figure 1.18, as it appears in the text: FULL ADDER

Carpinelli Figure 1.18, *CORRECTED*: FULL ADDER



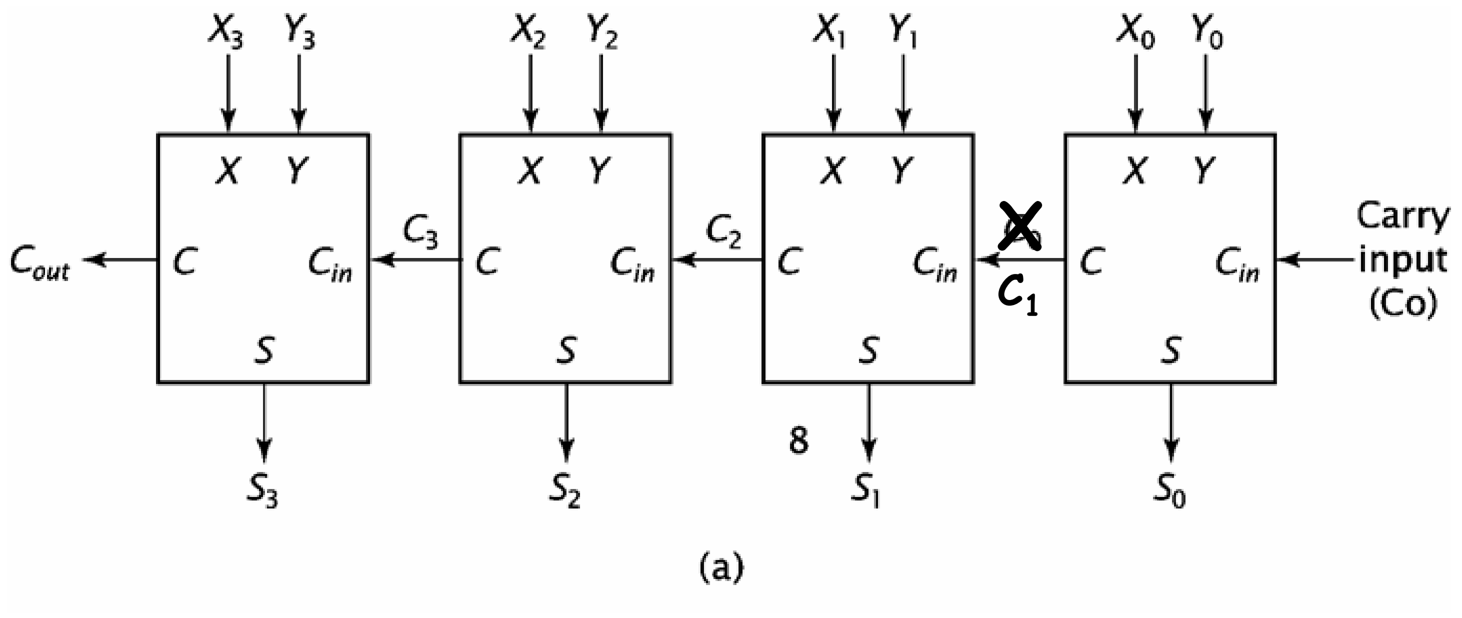




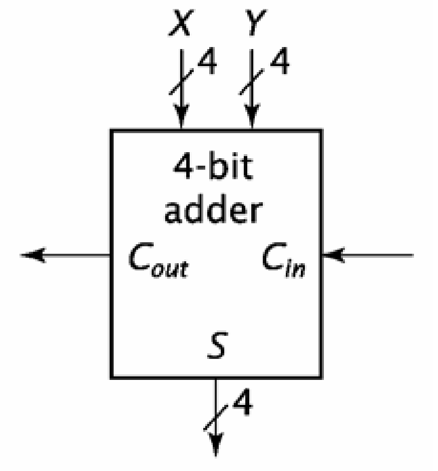
X	Y	C_{in}	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Carpinelli Figure 1.19, as it appears in the text: FOUR-BIT ADDER

Carpinelli Figure 1.19, CORRECTED: FOUR-BIT ADDER



(a)



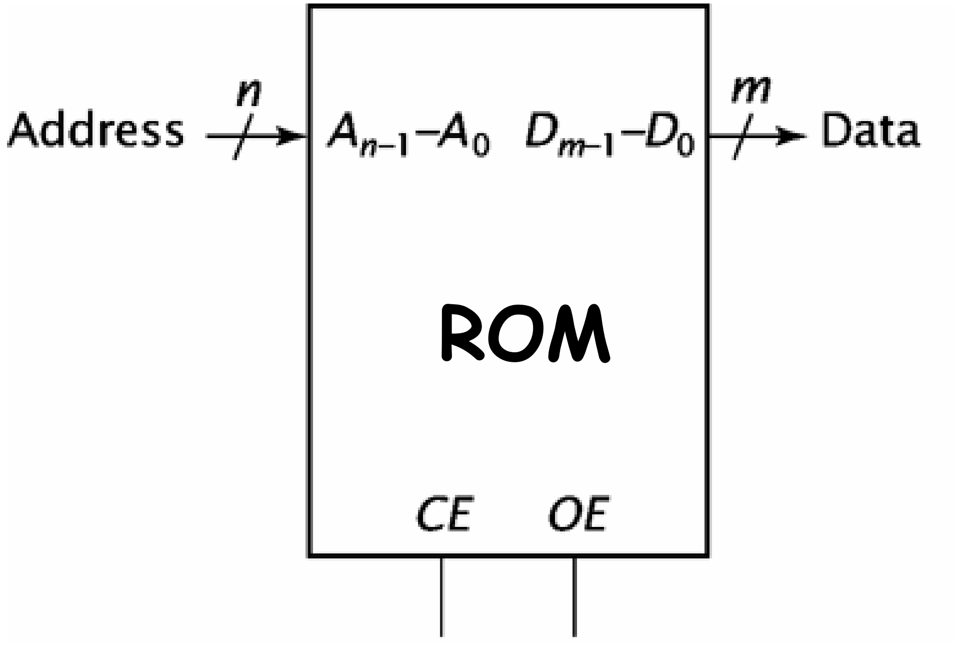
(b)

Original figure or table © 2001 by Addison Wesley Longman, Inc

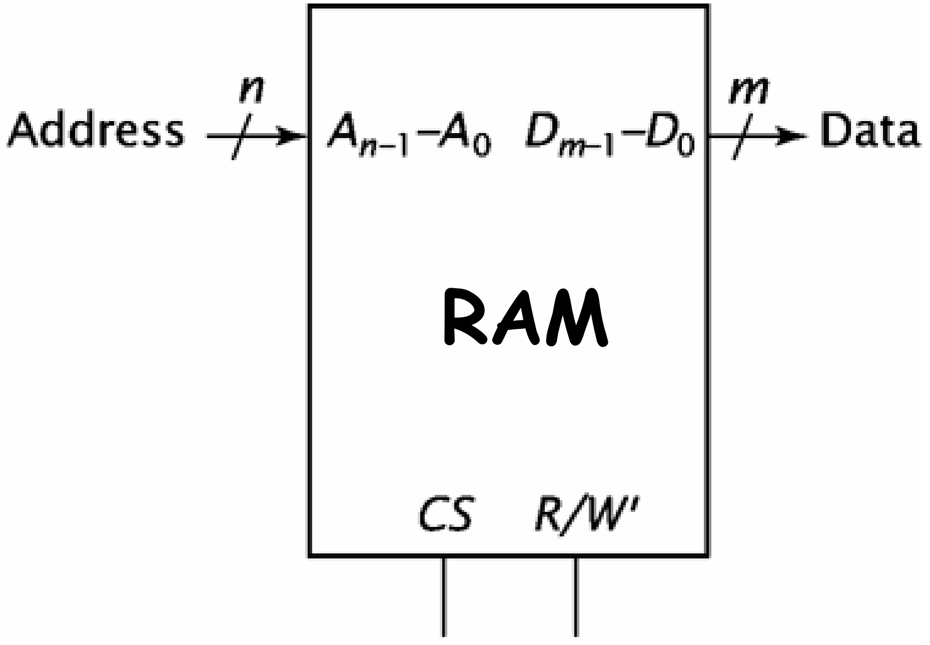
Carpinelli Figure 1.20: SUBTRACTOR

Carpinelli Figure 1.21, as it appears in the text: MEMORY MODULES

Carpinelli Figure 1.21, *ENHANCED:* MEMORY MODULES



(a)



(b)

USE of MULTIPLE MEMORY MODULES

1. In general, a MEMORY consists of some number of addressable units, each of which is called a WORD.

2. The contents of a single word of memory are described as a string of bits. The words in a computer memory are all of identical width, *w*, and can be described as :

$$D_{(w-1)}\ D_{(w-2)}\ D_{(w-3)}\ .\ .\ .\ D_0.$$

3. For a memory containing a total of *T* words, it is necessary to specify a numeric address in order to be able to uniquely identify each word. For this purpose, some number of address bits is necessary. The required number of address bits is given by:

$$A \geq \lceil \log_2(T) \rceil$$

4. We can construct the memory from a number of memory modules of equal size. If each module contains *N* words, then the individual module requires *n* address bits to specify which word within the module is to be accessed. Since *N* is always an exact power of 2,

$$n = \log_2(N)$$

USE of MULTIPLE MEMORY MODULES (continued)

5. For a total of T words, the number of modules (memory chips) required is:

$$M = T / N$$

$$\lceil \log_2 (M) \rceil = \lceil \log_2 (T) \rceil - \log_2 N$$

6. The total address lines are thus divided into two groups: m chip-select lines, and n within-chip word-select lines:

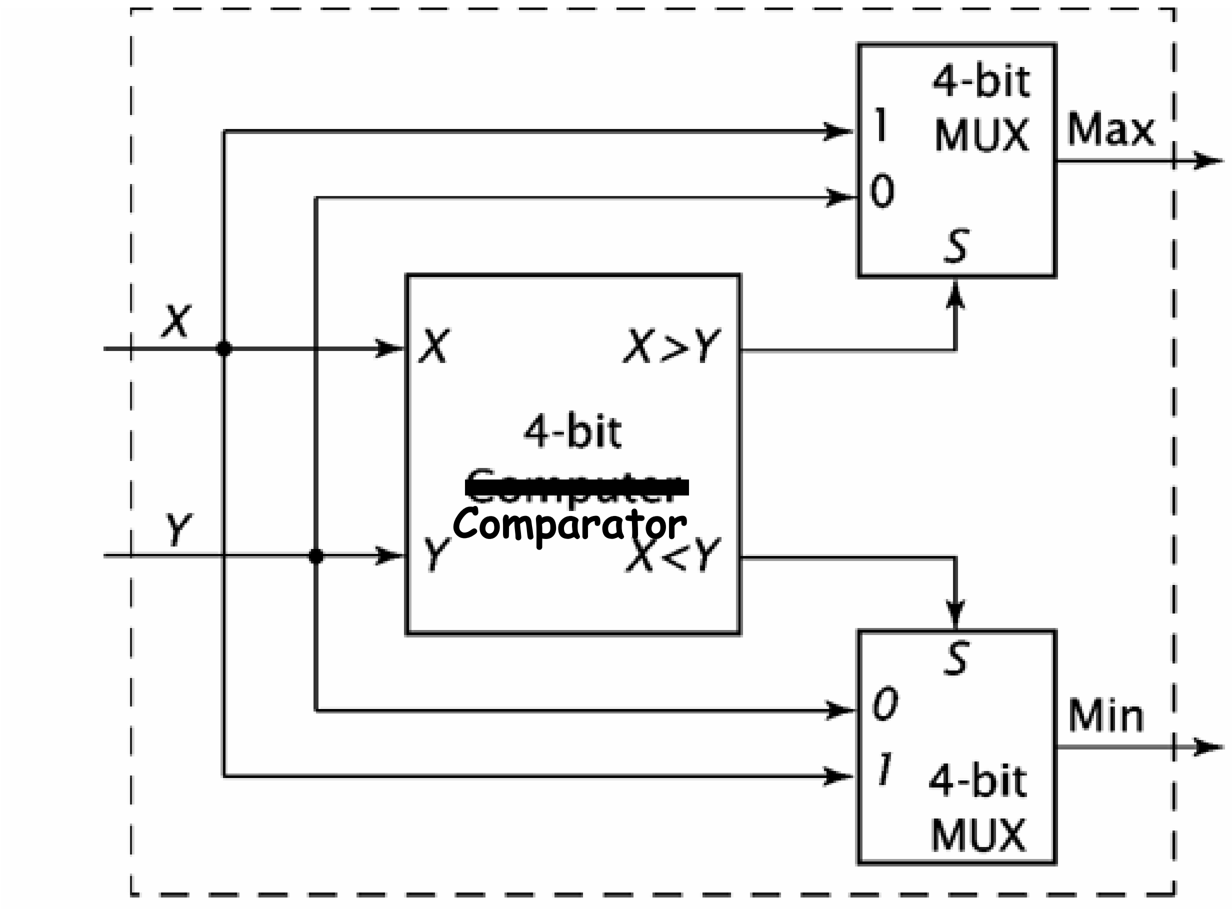
$$A = m + n$$

7. The n within-chip address lines are connected in parallel to the address-select input terminals of all of the memory modules.

8. The m chip-select lines go to an (m-x-2^m) decoder. Each output line from the decoder connects to the Chip-Enable input line of one memory chip.

Carpinelli Figure 1.24 (a), as it appears in the text: TWO-NUMBER SORTER

Carpinelli Figure 1.24 (a), *CORRECTED*: TWO-NUMBER SORTER



Original figure or table © 2001 by Addison Wesley Longman, Inc

Carpinelli Figure 1.24 (b): FOUR-NUMBER SORTER

TWO PRINCIPAL TYPES of LOGIC CIRCUITS:

1. COMBINATIONAL

2. SEQUENTIAL

TWO PRINCIPAL TYPES of LOGIC CIRCUITS:

1. COMBINATIONAL

   Current output (*0*s and *1*s), after the passage of sufficient time to allow for circuit stabilization, is dependent solely upon current input (*0*s and *1*s),

2. SEQUENTIAL

   Current output depends <u>not only</u> on current input, but also on the prior history of the circuit state.

Carpinelli Figure 1.25

Carpinelli Figure 1.26 (a): POSITIVE-EDGE-TRIGGERED D FLIP-FLOP

Carpinelli Figure 1.26 (b): POSITIVE-GATED D LATCH

Carpinelli Figure 1.27: POSITIVE-GATED D LATCH, with Asynchronous *SET* and *CLEAR*

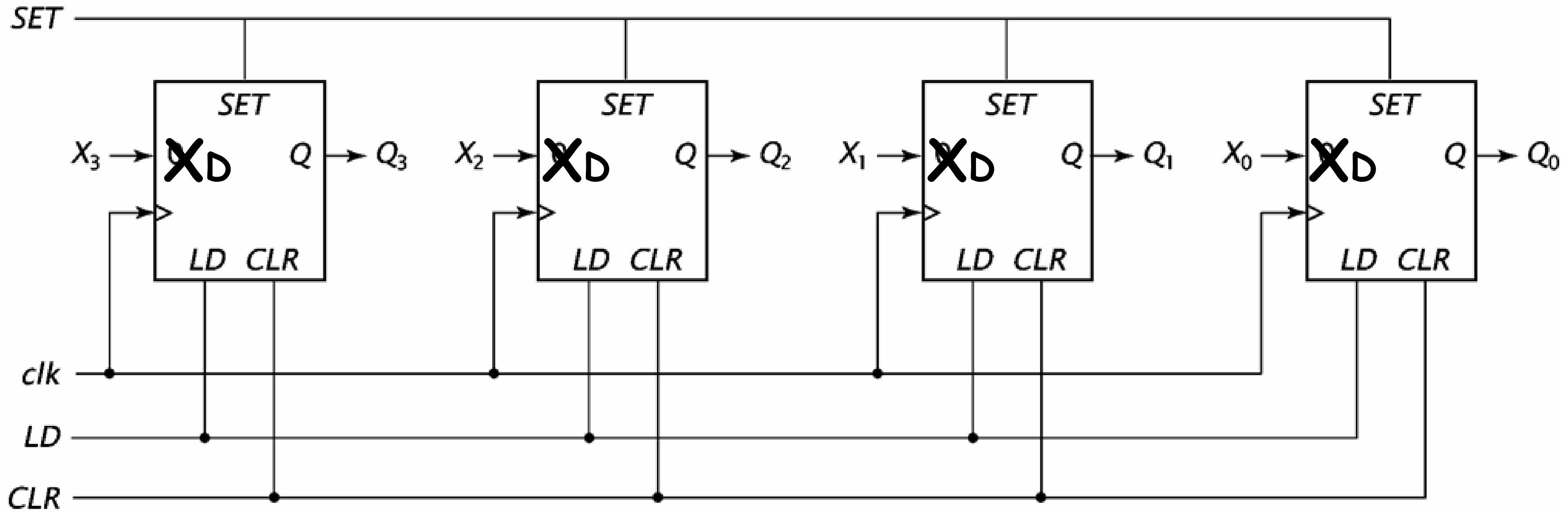
Carpinelli Figure 1.28: <u>*SR*</u> LATCH

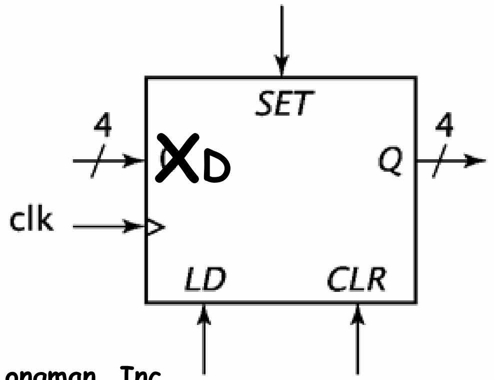
Carpinelli Figure 1.25: POSITIVE-EDGE-TRIGGERED *JK* FLIP-FLOP

Carpinelli Figure 1.30: POSITIVE-EDGE-TRIGGERED *T* FLIP-FLOP

Carpinelli Figure 1.31, as it appears in the text: FOUR-BIT REGISTER (*not* a 4-Bit Flip-Flop)

Carpinelli Figure 1.31, *CORRECTED*: FOUR-BIT REGISTER (*not* a 4-Bit Flip-Flop)

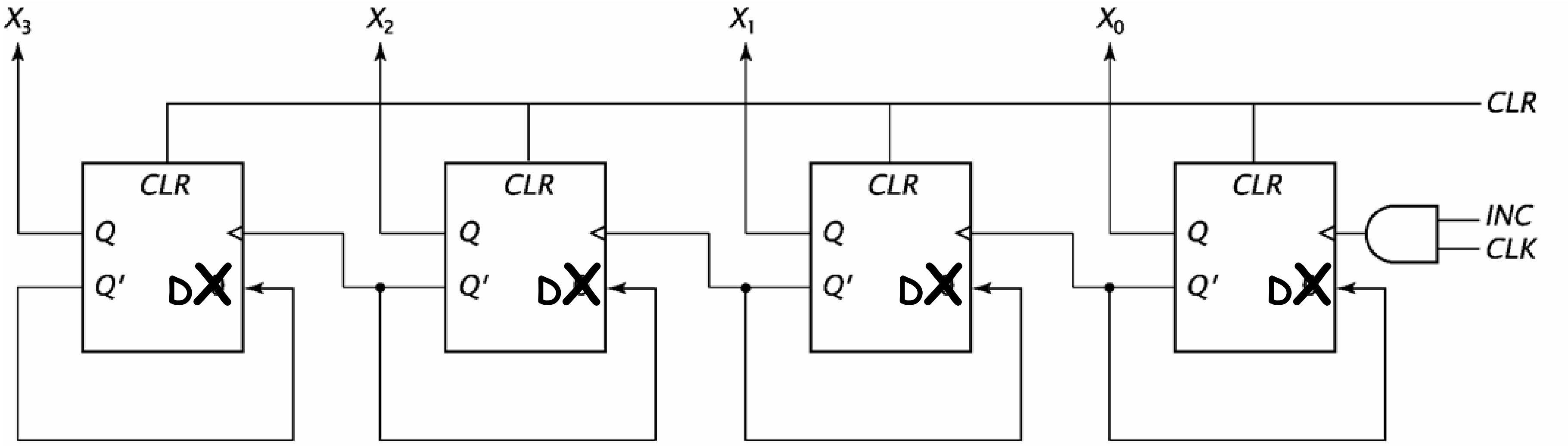




Original figure or table © 2001 by Addison Wesley Longman, Inc

Carpinelli Figure 1.32, as it appears in the text: Four-Bit Incrementing (“Up”) Counter

Carpinelli Figure 1.32, *CORRECTED*: Four-Bit Incrementing ("Up") Counter



CLR	INC	CLK	X_{3-0}
1	X	X	0
0	0	X	X_{3-0}
0	1	not↑	X_{3-0}
0	1	↑	$X_{3-0}+1$

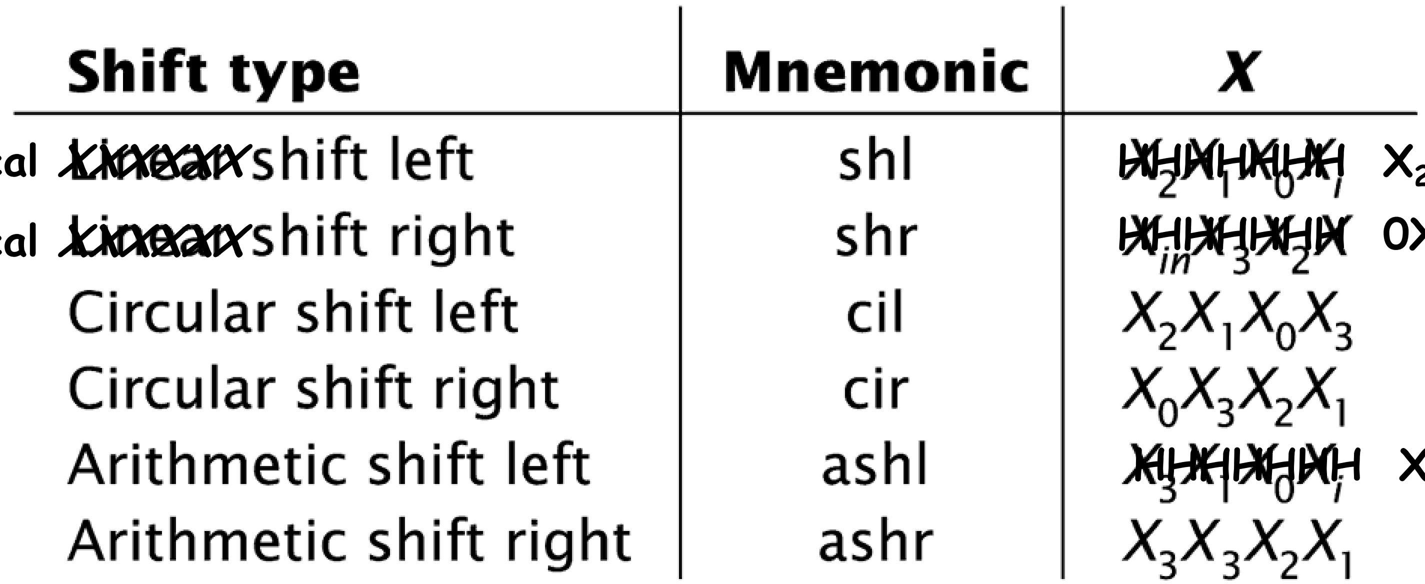
Carpinelli Figure 1.33: FOUR-BIT INCREMENTING/DECREMENTING COUNTER (“Up/Down” Counter)

Carpinelli Figure 1.34: FOUR-BIT LEFT-SHIFT REGISTER

Carpinelli Table 1.8, as it appears in the text: SHIFT OPERATIONS

Carpinelli Table 1.8, *CORRECTED*: SHIFT OPERATIONS

The corrected version corresponds to the definitions of the various types of shift operations typically implemented in the Arithmetic Logic Unit (ALU) portion of Central Processing Units (CPUs).



Shift type	Mnemonic	*X*
Logical ~~Linear~~ shift left	shl	~~$X_2X_1X_0X_i$~~ $X_2X_1X_00$
Logical ~~Linear~~ shift right	shr	~~$X_{in}X_3X_2X$~~ $0X_3X_2X_1$
Circular shift left	cil	$X_2X_1X_0X_3$
Circular shift right	cir	$X_0X_3X_2X_1$
Arithmetic shift left	ashl	~~$X_3X_1X_0X_i$~~ $X_2X_1X_00$
Arithmetic shift right	ashr	$X_3X_3X_2X_1$

Original figure or table © 2001 by Addison Wesley Longman, Inc

Carpinelli Figure 1.35: PROGRAMMABLE LOGIC ARRAY (PLA)

Carpinelli Figure 1.36: PROGRAMMABLE ARRAY of LOGIC (PAL)

Carpinelli Problem 1.9:

$$F = WXY' + WXZ + W'XY + XYZ'$$
$$= (WXY'Z + WXY'Z') + (WXYZ + WXY'Z) + (W'XYZ + W'XYZ') + (WXYZ' + W'XYZ')$$

CARRY-LOOKAHEAD ADDER

$g_i = X_i \wedge Y_i$

$p_i = X_i + Y_i$ ($\underline{\text{not}}$ $X_i \oplus Y_i$)

FINITE-STATE MACHINES:

SEQUENTIAL and COMBINATIONAL LOGIC

Carpinelli Figure 2.1 (a): GENERIC STATE-MACHINE MODEL

Carpinelli Figure 2.3: STATE DIAGRAMS for the JK FLIP-FLOP

Carpinelli Figure 2.4: *JK* FLIP-FLOP

State Table:

State Diagram:

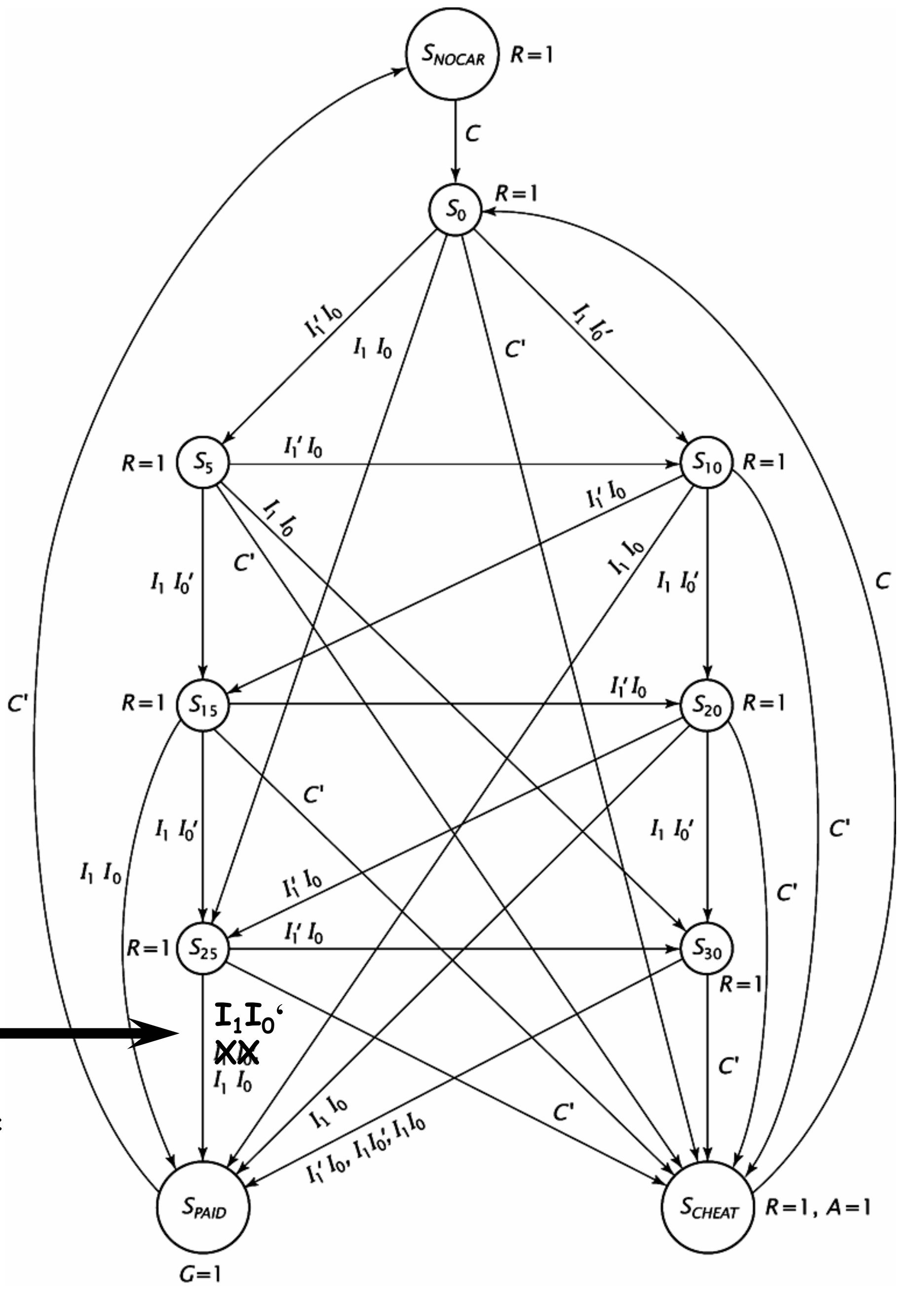
Carpinelli Figure 2.5 (b): STATE DIAGRAM for the MODULO-6 COUNTER: MOORE MACHINE

Carpinelli Figure 2.6 (b): STATE DIAGRAM for the STRING-CHECKER: MOORE MACHINE

Carpinelli Figure 2.A (b): STATE DIAGRAM for the STRING-CHECKER with REVISED STATE ASSIGNMENTS: MOORE MACHINE

Carpinelli Figure 2.7, as it appears in the text: STATE DIAGRAM for the TOLL-BOOTH CONTROLLER: MOORE MACHINE

Carpinelli Figure 2.7, *CORRECTED*: STATE DIAGRAM for the TOLL-BOOTH CONTROLLER: MOORE MACHINE



Original figure or table © 2001 by Addison Wesley Longman, Inc

Carpinelli Table 2.5: STATES for the TOLL-BOOTH CONTROLLER: MOORE MACHINE

Carpinelli Table 2.6: STATE TABLE for the TOOL-BOOTH CONTROLLER: MOORE MACHINE

Carpinelli Figure 2.8 (b): STATE DIAGRAM for the MODULO-6 COUNTER: MOORE MACHINE (Assigned State Values Included

Carpinelli Figure 2.10 (a): GENERIC MOORE MACHINE

Carpinelli Figure 2.10 (b): MOORE MACHINE IMPLEMENTATION of the MODULO-6 COUNTER

Carpinelli Figure 2.11: KARNAUGH MAPS for the NEXT STATE of the MODULO-6 COUNTER

END