Combinational Logic

Supplemental Notes

Yul Williams, D.Sc.

Purpose of Course Notes

These course notes are provided to increase your understanding of the course materials presented to you in the Course Content area of the WebTycho online classroom. They will generally follow the outline of the online modules and the course textbook.

The course notes offer insight into the various topics presented in the topics covered online, but, they are not to be considered a complete coverage of each topic area.

Week 2 Goals

- At the conclusion of this session, the student should be able to:
  - Understand the basic complex logic circuits
  - Understand and use a tri-state buffer
  - Understand how to perform basic arithmetic operations on binary data
  - Perform addition and subtraction using two's complement
  - Explain how a carry lookahead adder works
  - Explain the differences in volatile and non-volatile memories
  - Describe the operation of a flip-flop

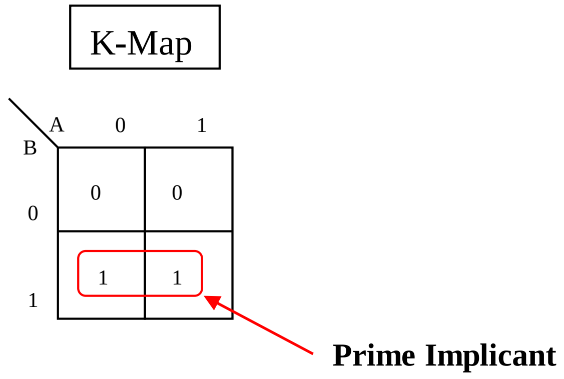
Week 2: Outline

- Completing Karnaugh Maps
- K-map Essentials
- Buffers
- More Complex Combinational Components
- Addition, Subtraction and Twos Complement
- Arithmetic Circuits
- Memory

Completing Karnaugh Maps

Prime Implicants

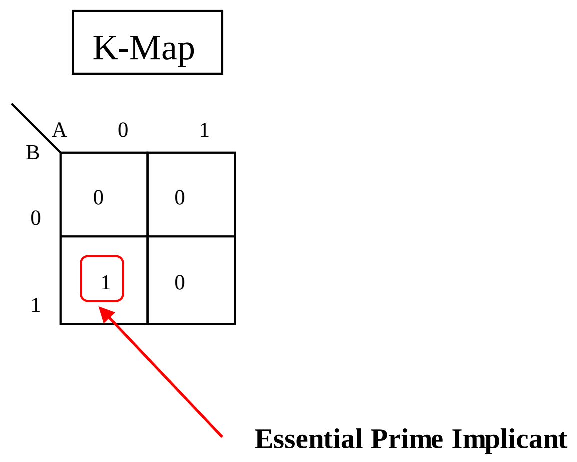
- The final groupings of cells in the K-maps are called prime implicants
- The final size of the groupings is irrelevant



Essential Prime Implicants

- As a goal, we must always look to form the fewest groups. This guarantees that we get the minimal final expression
- Sometimes we find cells in the K-map that cannot be combined with any other cells. These are referred to a essential prime implicants. They must be included in the final expression.

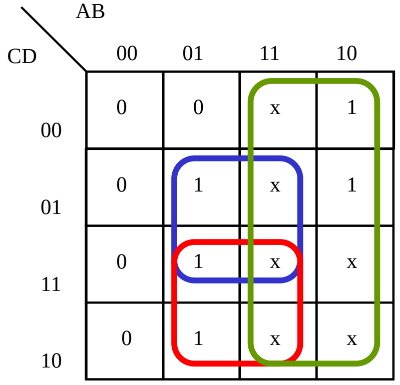
Essential Prime Implicant Example



Don’t Care Condition

- In some cases, you will have expressions where some of the possible inputs are unspecified and the output values will not be used.
- We call this a don’t care condition
- The don’t care condition is represented in the K-map using the symbol X
- X’s may be combined with the 1’s or 0’s to yield larger prime implicant groupings
  - The same rules (groupings in powers of 2) still apply

Forming Prime Implicants with Don’t Cares



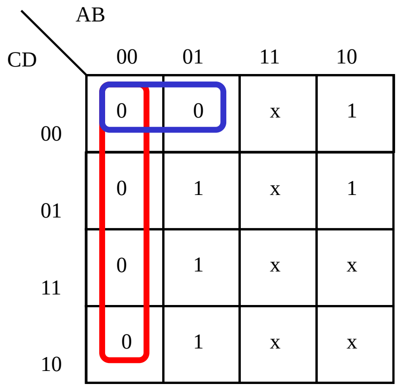
CD \ AB	00	01	11	10
00	0	0	x	1
01	0	1	x	1
11	0	1	x	x
10	0	1	x	x

F = A + BC + BD

Compliment of a Minimized Expression

- When given an expression F, then a minimized sum of products may be obtained by forming prime implicants using the 1's and don't cares.
- A minimal sum of products expression F' may be obtained by grouping the 0's and don't cares

Minimal SOP form of F’



CD \ AB	00	01	11	10
00	0	0	x	1
01	0	1	x	1
11	0	1	x	x
10	0	1	x	x

F’ = A’C’D’ + A’B’

K-map Essentials

Combining Don’t Cares

- Don’t cares may be combined with 1’s when attempting to minimize an expression F
- Don’t cares may be combined with 0’s when attempting to minimize an expression F’

K-map Size

- The size of the K-map is determined by the number of different variables in the expression F.
  - An expression of 2 variables requires a K-map of size 2^2 or 4
  - An expression of 3 variables requires a K-map of size 2^3 or 8
  - An expression of 4 variables requires a K-map of size 2^4 or 16

Importance of Gray Code

- Each time we form a K-map, the ordering of the minterms must be done according to the Gray code.
- This ensures that only one digit differs from a specific number to the numbers in adjacent positions

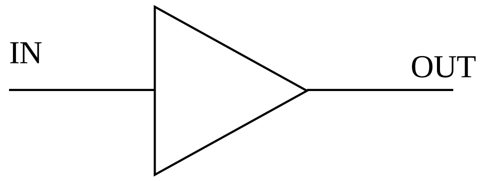
Buffers

What they are and what they do

The Role of the Buffer

- Buffers perform no “logic” operation on its input value to produce another output
- Its output value is the same as its input value
- We will discuss two primary types of buffers:
  - The regular buffer
  - The tri-state buffer

The Regular Buffer



IN	OUT
0	0
1	1

The Regular Buffer passes its input directly to its output as indicated by its truth table. When a logic value has to traverse long wires, its signal strength becomes degraded. The buffer regenerates the signal to a higher value to maintain its integrity.

The Tri-state Buffer

- The tri-state buffer performs the same function as the regular buffer
  - It boosts the signal of the input
- It also extends the functionality of the regular buffer by implementing an enabling function E.
  - When E is active, the input value is passed from the input terminal to the output terminal of the buffer.
  - When E is inactive, it disconnects the path from the input terminal to the output terminal by placing the buffer into a high impedance state Z.

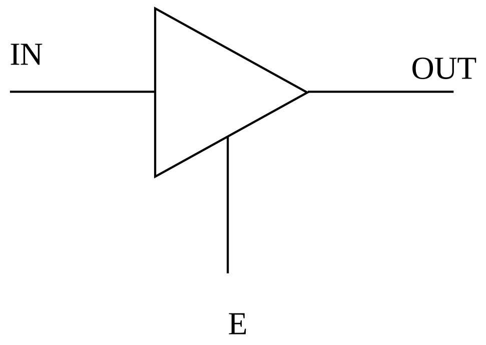
The High Impedance State

- When the buffer enters the Z state, it has the effect of disconnecting itself from the rest of the logic circuit
- It does this by limiting the current flow in the buffer so low, that the buffer acts as though it is not connected to anything

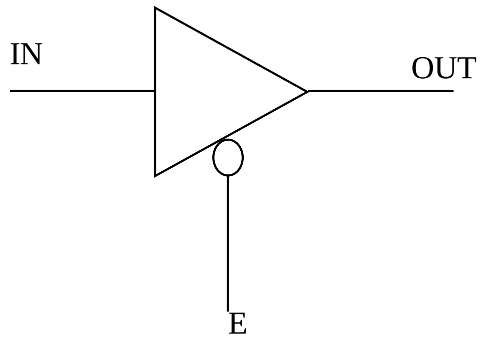
Enabling Function for the Buffer

- The tri-state buffer is enabled using the E signal input.
- The E signal determines whether a signal is passed from the input to the output terminal or if the buffer enters the Z state
- The enable input may be active HI or active LO
  – Active HI implies that E is true when its input value is a logic ‘1’
  – Active LO implies that E is true when its input value is a logic ‘0’

Tri-state Buffer Symbols



IN	E	OUT
X	0	Z
0	1	0
1	1	1



IN	E	OUT
0	0	0
1	0	1
X	1	Z

More Complex Combinational Components

Multiplexers, Decoders, Encoders
and Comparators

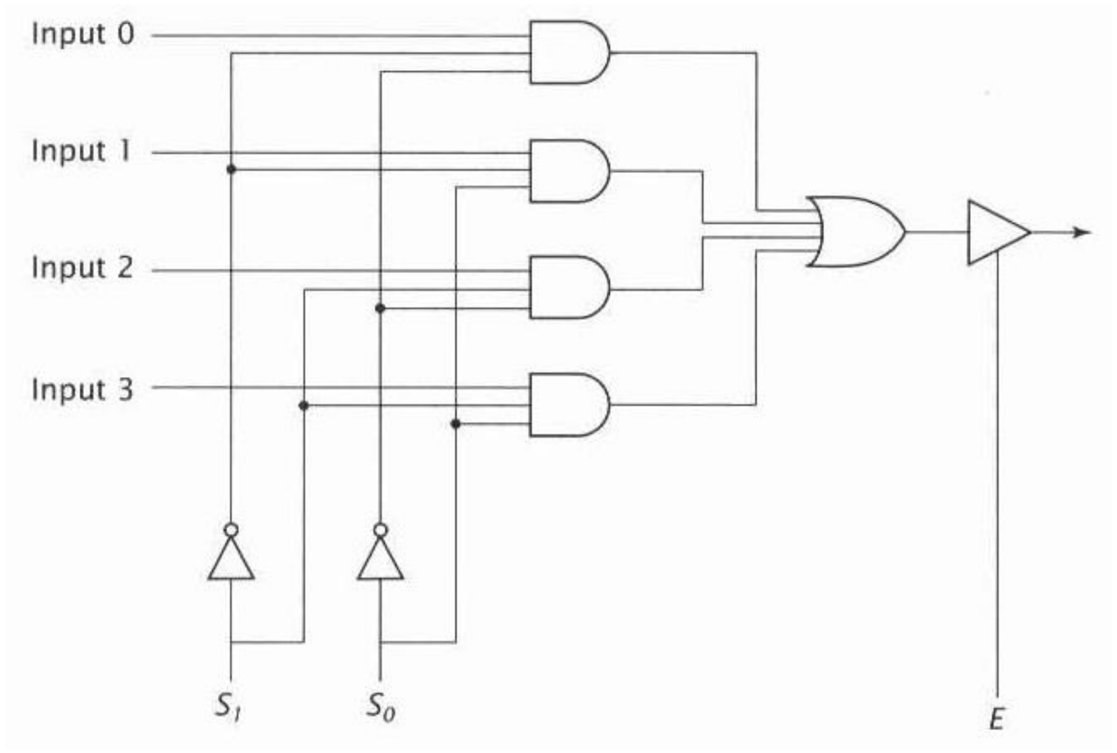
Macrofunctions

- Now that we have a basic understanding of digital logic, we can construct any possible logic circuit
- At some point, however, this process will get extremely time consuming.
- What if we constructed commonly used blocks of logic circuits to in order to simplify the design process
- These blocks are designed as higher-level integrated circuits (ICs) called macrofunctions

The Multiplexer

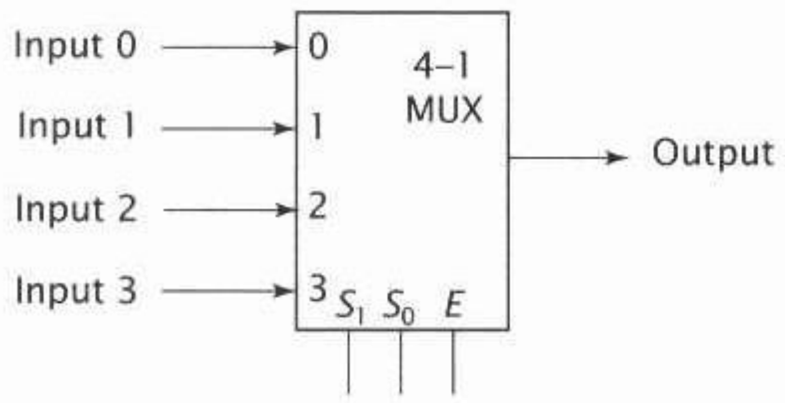
- One commonly used macrofunction is the multiplexer (MUX)
- The mux is a device that selects between a number of input signals and transfers the selected signal to the output terminal
- Let’s take a look at a simple mux…

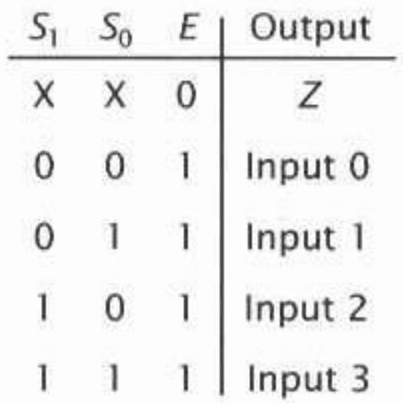
Multiplexer



Gate level representation

Macro-level Mux Representation

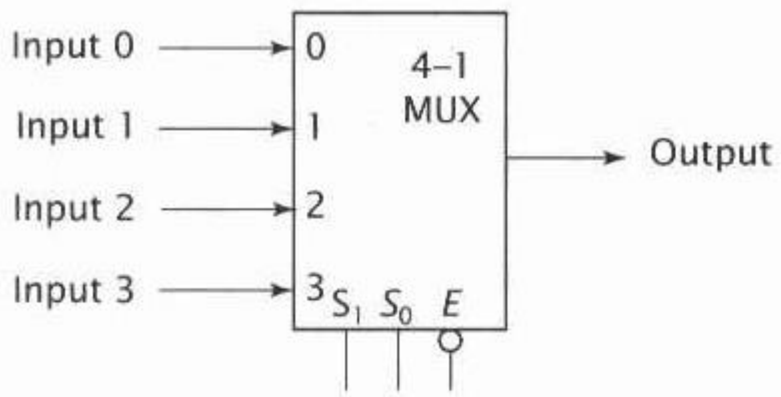


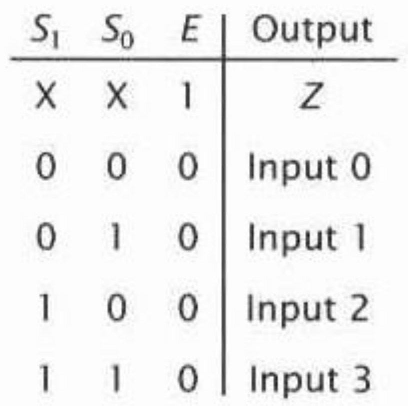


S_1	S_0	E	Output
X	X	0	Z
0	0	1	Input 0
0	1	1	Input 1
1	0	1	Input 2
1	1	1	Input 3

Notice that the macro level hides the detail of the internal logic. It exposes only the interface terminals. The assumption here is that the function is understood by the logic designer. This particular view is the active HI version of the mux.

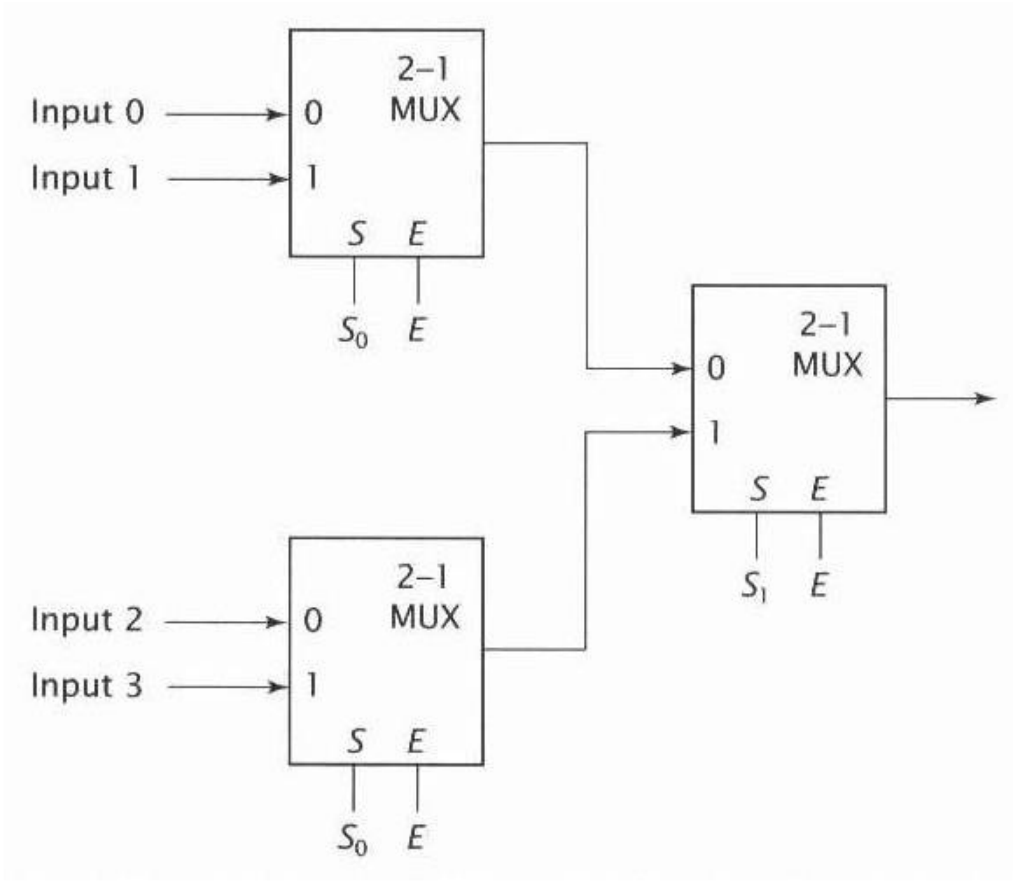
Active LO Version of the MUX





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

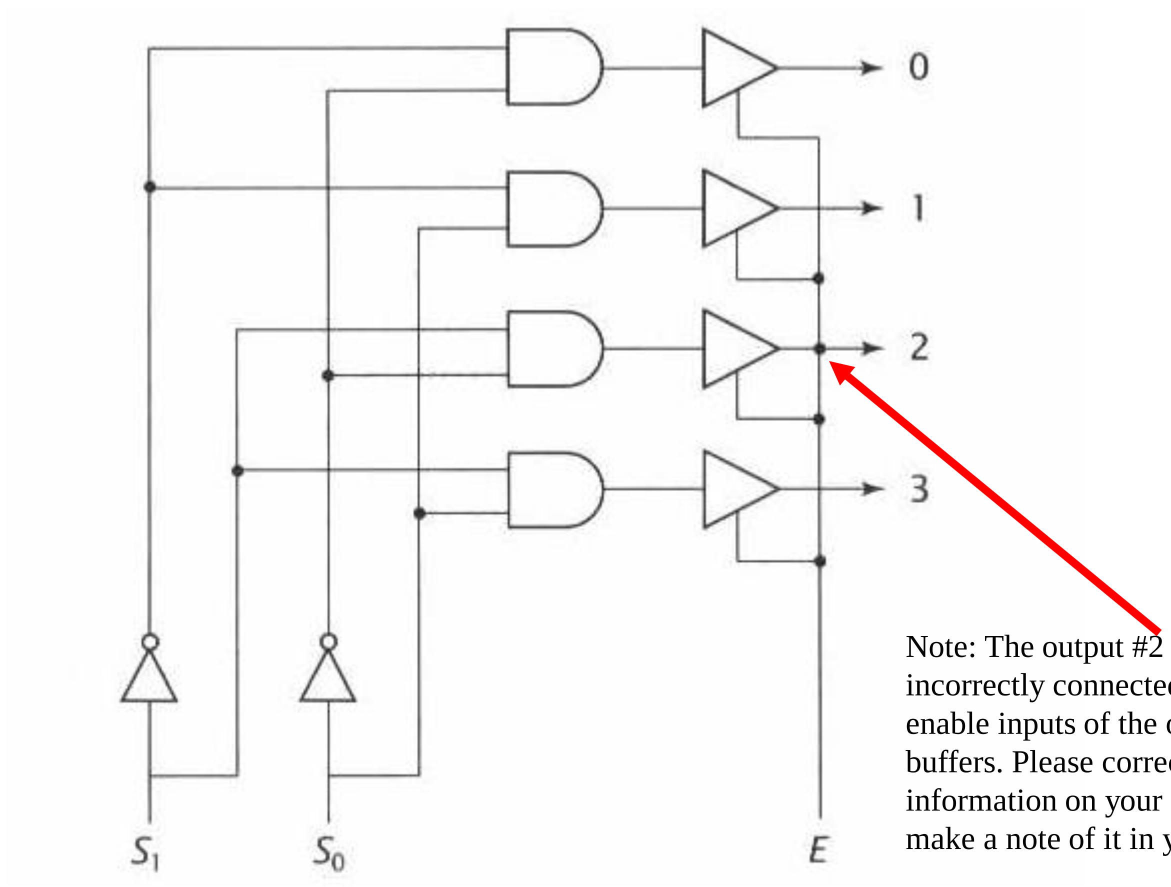
Building a 4-1 Mux using 2-1 Mux Components



The Decoder

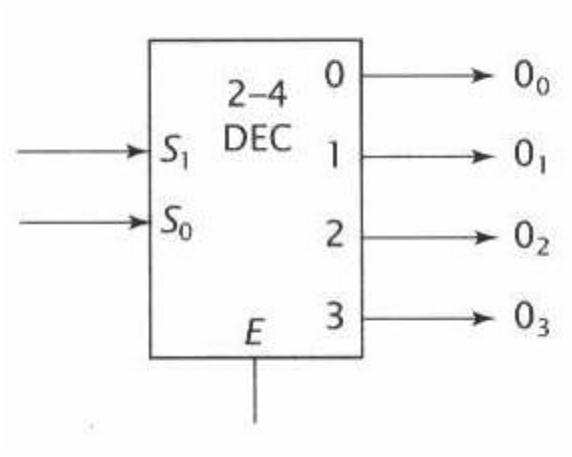
- A decoder accepts a value and routes the value to an appropriate terminal.
- A decoder is a 1 to many device
  - A decoder with n inputs, and 2^n outputs

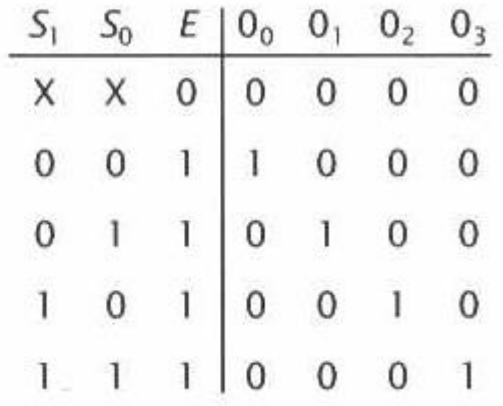
Decoder



Note: The output #2 is incorrectly connected to the enable inputs of the other tri-state buffers. Please correct this information on your papers and make a note of it in your books.

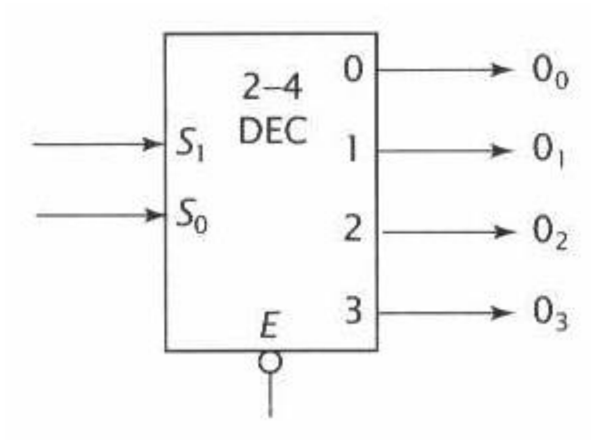
Decoder- Active Hi Enable

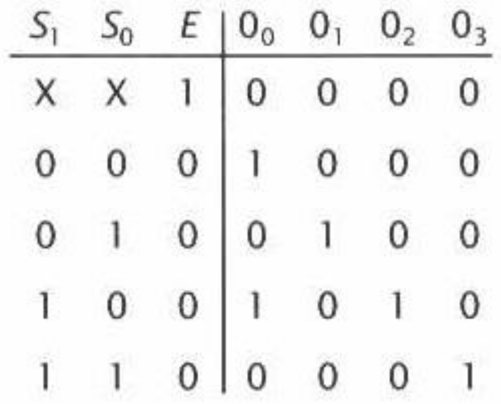




S_1	S_0	E	0_0	0_1	0_2	0_3
X	X	0	0	0	0	0
0	0	1	1	0	0	0
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	1

Decoder Active LO



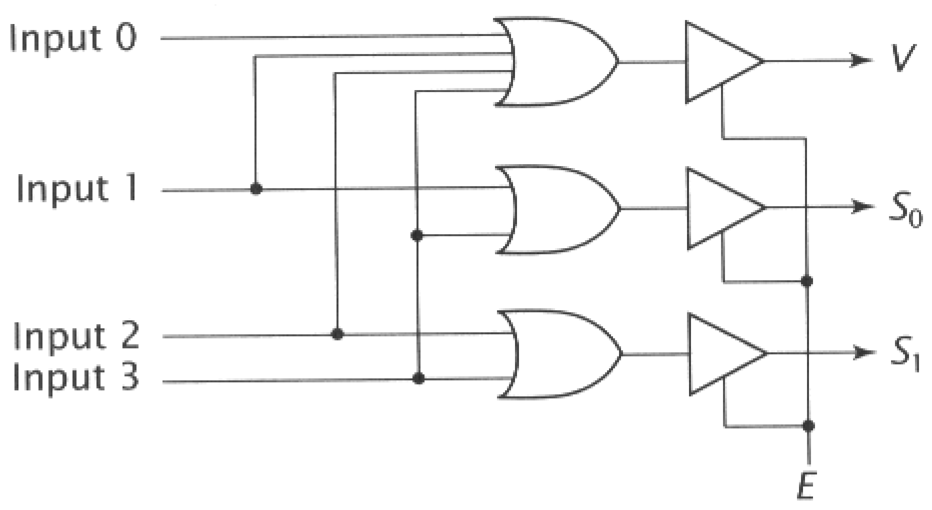


S_1	S_0	E	0_0	0_1	0_2	0_3
X	X	1	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	1	0	0
1	0	0	1	0	1	0
1	1	0	0	0	0	1

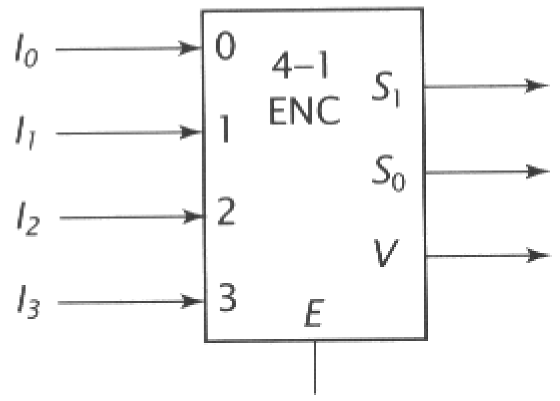
The Encoder

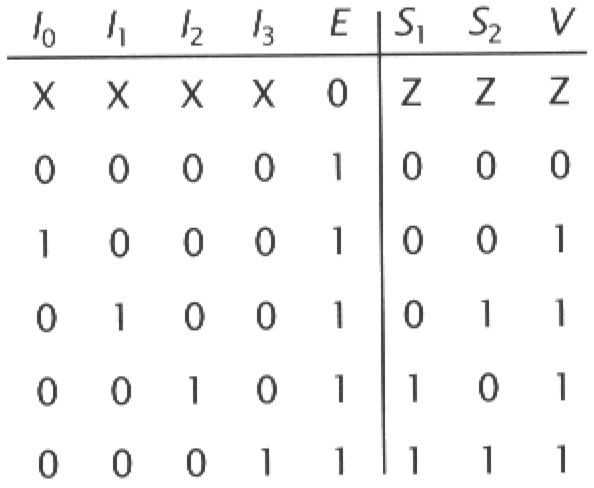
- The encoder does the exact opposite of the decoder
  - It has 2^n inputs and n outputs

Logic Representation of the Encoder



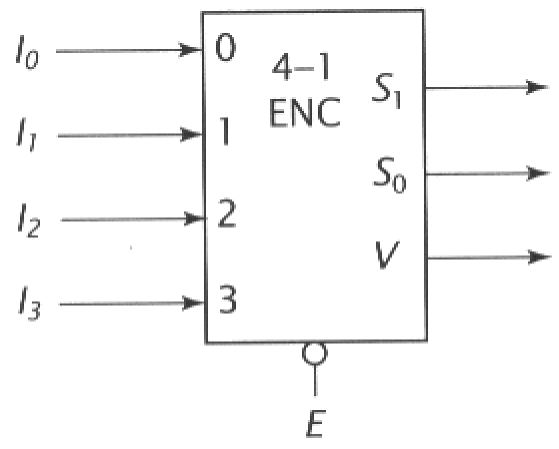
Active HI Encoder





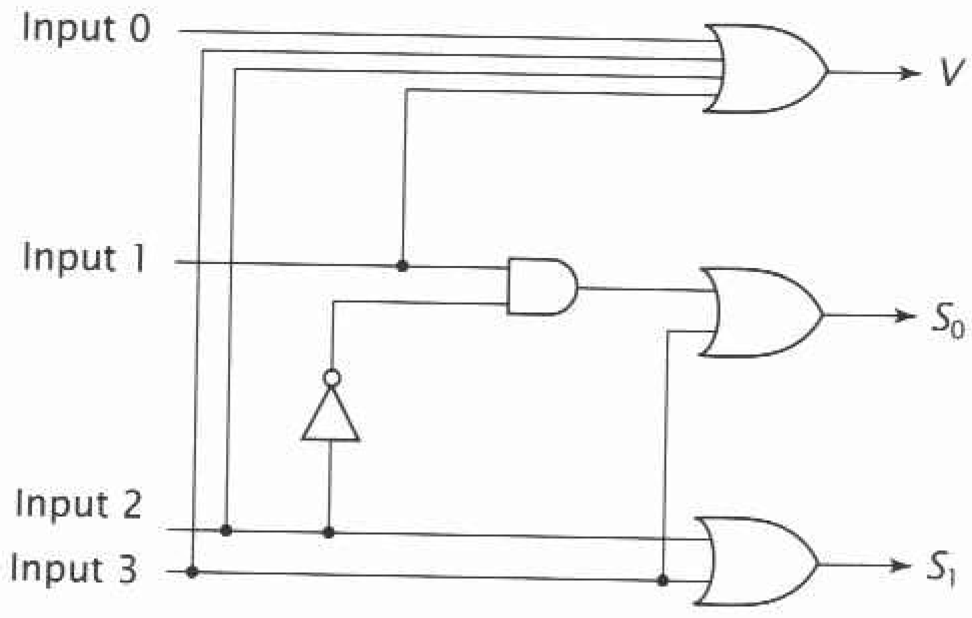
I_0	I_1	I_2	I_3	E	S_1	S_2	V
X	X	X	X	0	Z	Z	Z
0	0	0	0	1	0	0	0
1	0	0	0	1	0	0	1
0	1	0	0	1	0	1	1
0	0	1	0	1	1	0	1
0	0	0	1	1	1	1	1

Active LO Encoder



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	0	0	0
1	0	0	0	0	0	0	1
0	1	0	0	0	0	1	1
0	0	1	0	0	1	0	1
0	0	0	1	0	1	1	1

The Priority Encoder

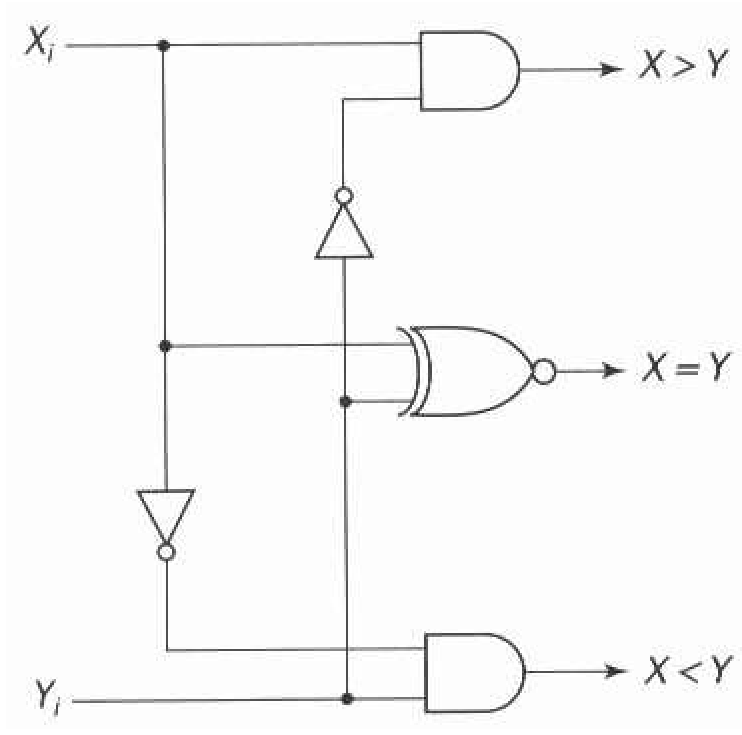


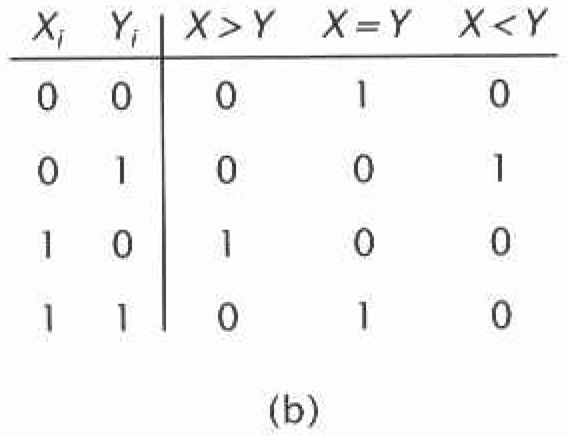
Input						
0	1	2	3	S_1	S_0	V
0	0	0	0	0	0	0
1	0	0	0	0	0	1
0	1	0	0	0	1	1
0	0	1	0	1	0	1
0	0	0	1	1	1	1

The Comparator

- A comparator compares two n-bit binary values to determine which one is greater or if they are equal in value

Comparator – Logic Level

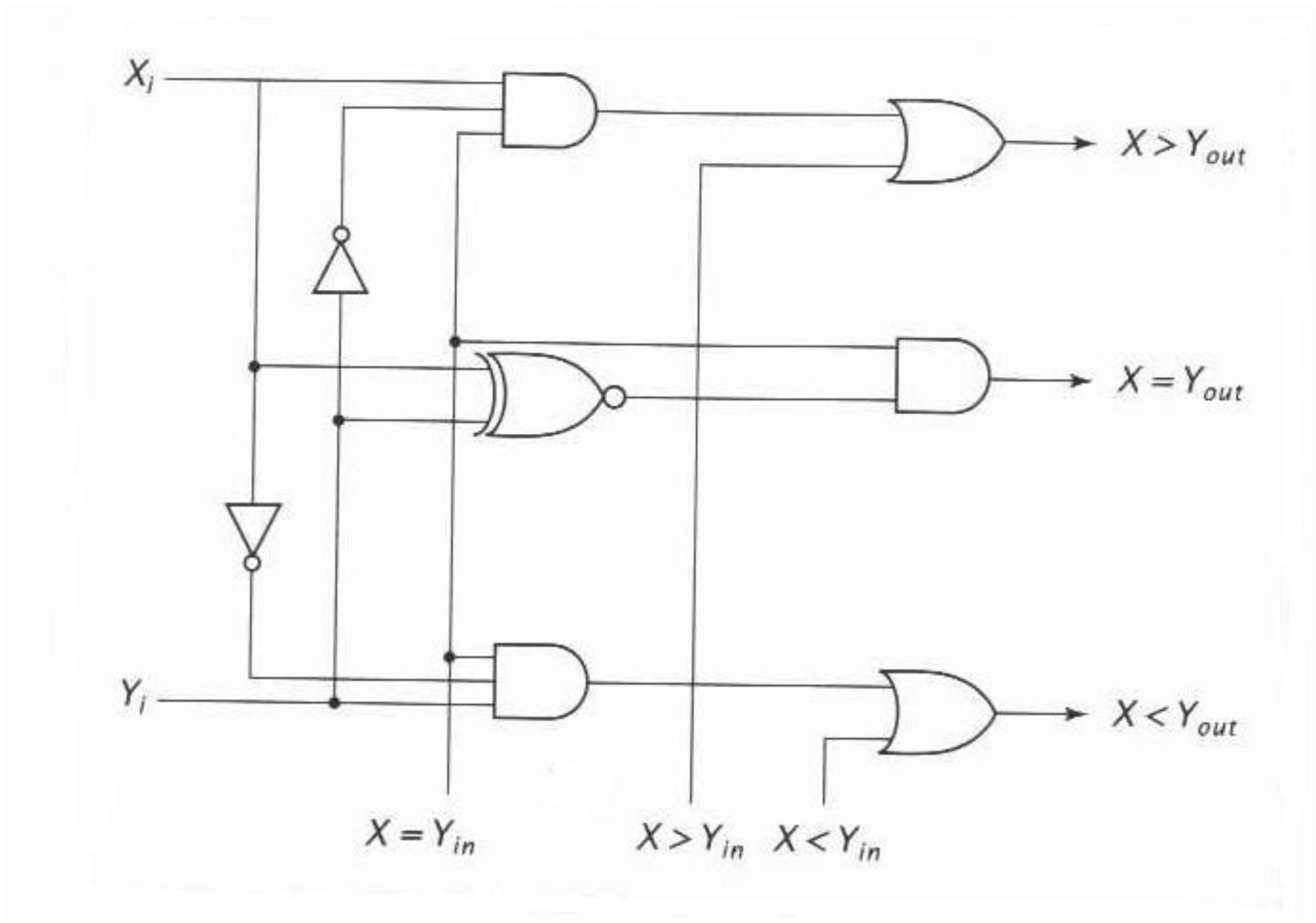




X_i	Y_i	$X > Y$	$X = Y$	$X < Y$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

(b)

Comparator with Propagated Inputs



Addition, Subtraction and Twos Complement

Binary Addition

```
        1100 (12)          1101 (13)
  +     0011 (3)     +     0111 (7)
        ------             ------
        1111 (15)         10100  (20)
```

Binary Subtraction

```
  1100 (12)        1101 (13)
- 0011 (3)       - 0111 (7)
  ------           ------
  1001 (9)         0110  (6)
```

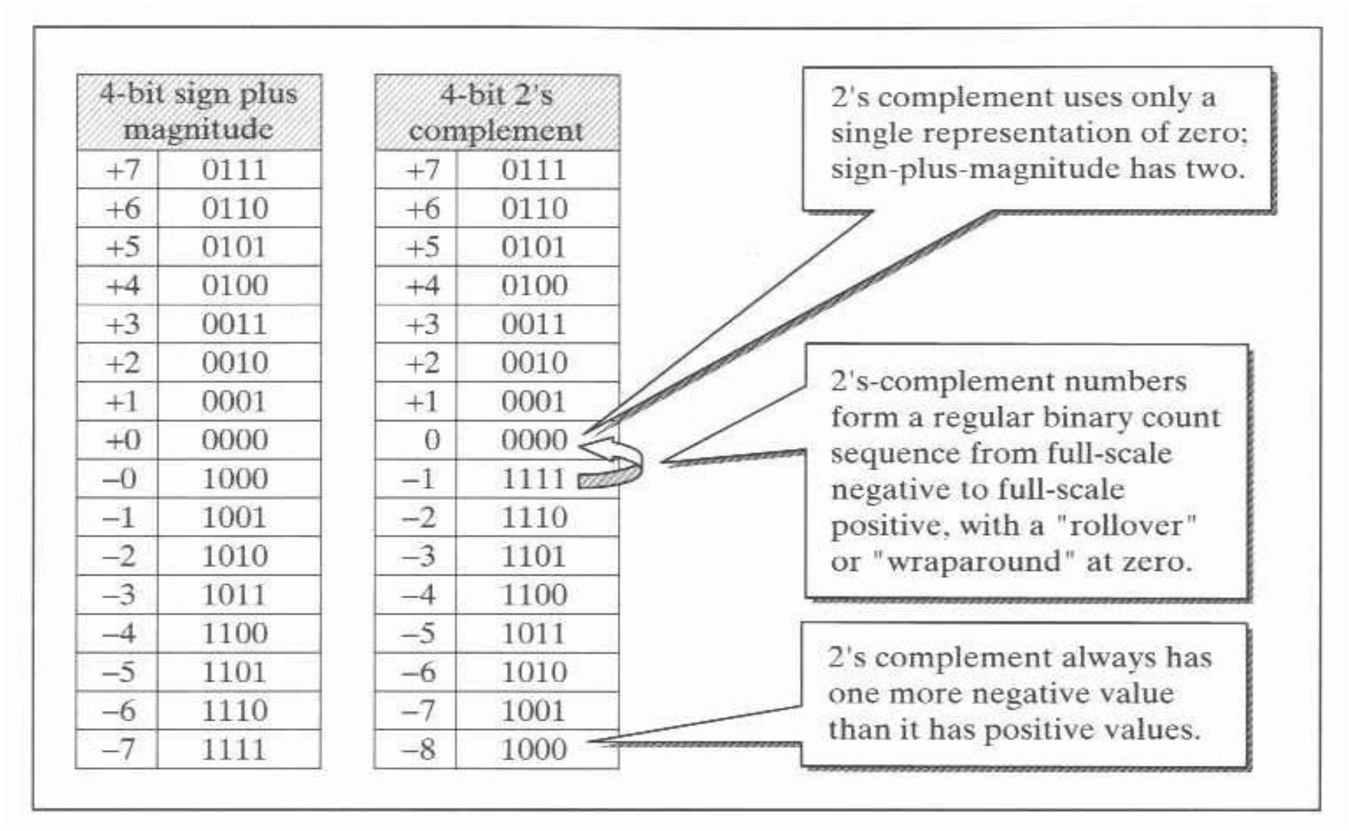
Two’s Complement

The two’s complement algorithm is very simple. First, the original number is converted with the one’s complement. This means that each bit location in the original number is inverted. We now add 1 to the one’s complement number to achieve the two’s complement result.

Example:

```
(+7) 0111      1000   (one’s complement of 7)
                  1  (add 1 to get the two’s complement
               -------
               1001 (two’s complement result = -7)
```

Two's Complement



4-bit sign plus magnitude		4-bit 2's complement	
+7	0111	+7	0111
+6	0110	+6	0110
+5	0101	+5	0101
+4	0100	+4	0100
+3	0011	+3	0011
+2	0010	+2	0010
+1	0001	+1	0001
+0	0000	0	0000
–0	1000	–1	1111
–1	1001	–2	1110
–2	1010	–3	1101
–3	1011	–4	1100
–4	1100	–5	1011
–5	1101	–6	1010
–6	1110	–7	1001
–7	1111	–8	1000

2's complement uses only a single representation of zero; sign-plus-magnitude has two.

2's-complement numbers form a regular binary count sequence from full-scale negative to full-scale positive, with a "rollover" or "wraparound" at zero.

2's complement always has one more negative value than it has positive values.

Source: Fundamentals of Embedded Software, Daniel W. Lewis

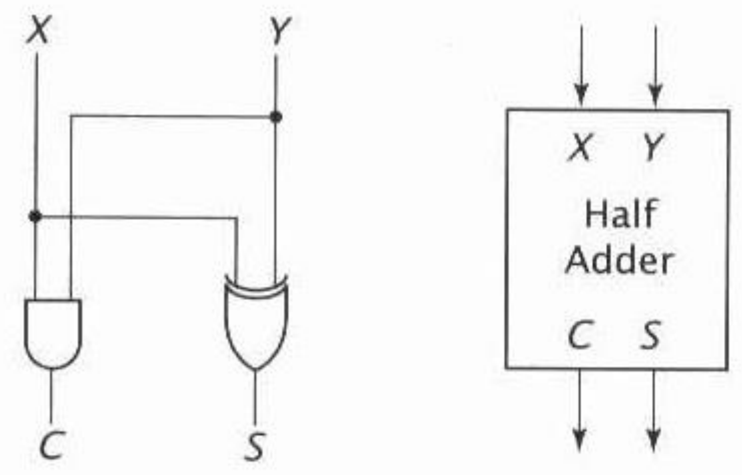
Moving on to Arithmetic Circuits

- Now that we have a basic understanding of arithmetic operation for binary numbers, we can take a look at some digital logic circuits that perform arithmetic operations
- We will explore adders and subtractors

Arithmetic Circuits

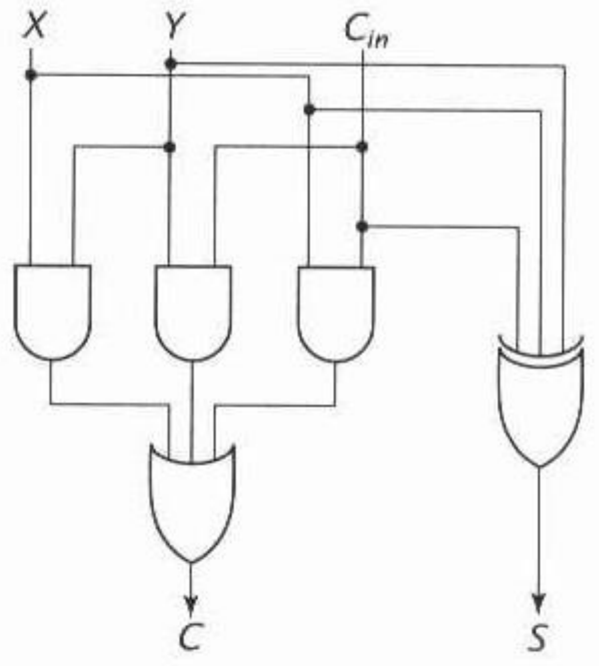
Adders and Subtractors

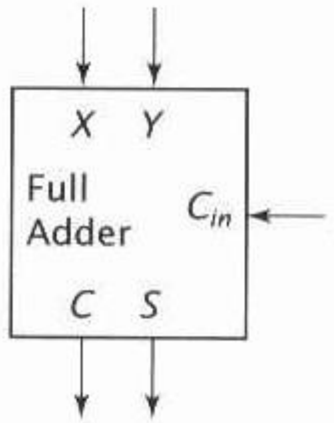
The Half Adder



X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

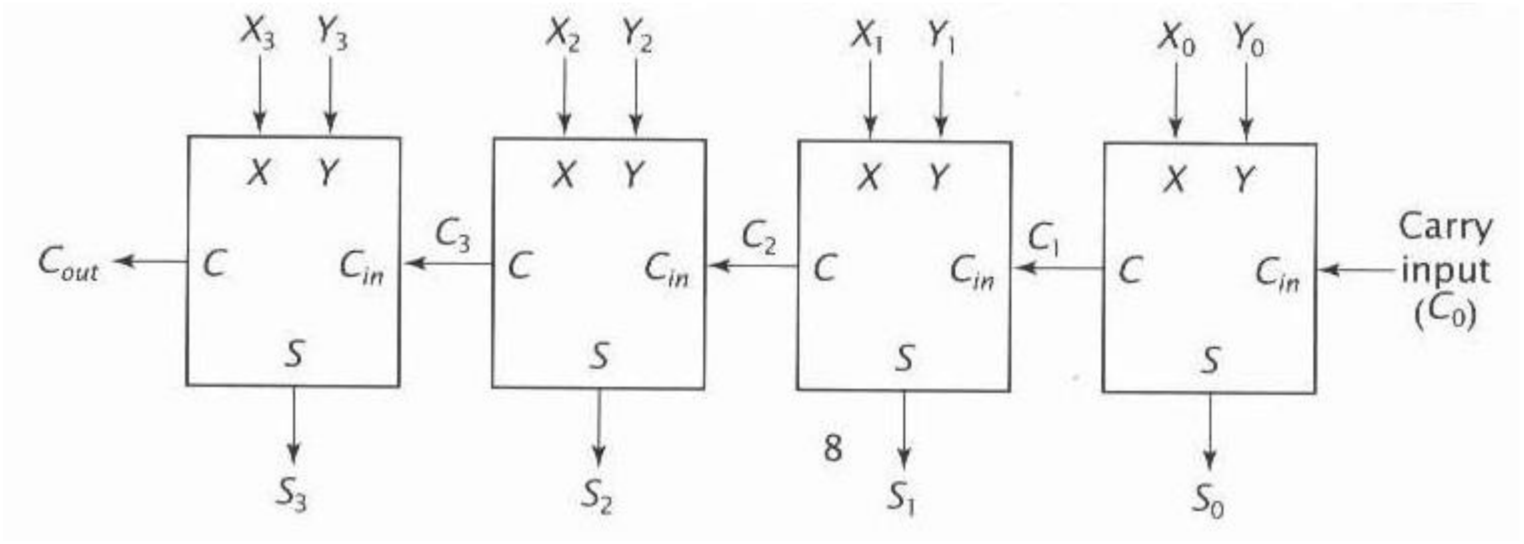
The Full Adder





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

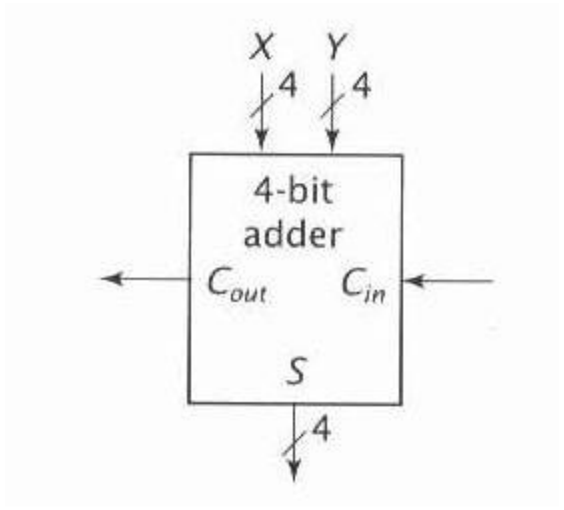
Constructing a 4-bit Carry Lookahead Adder



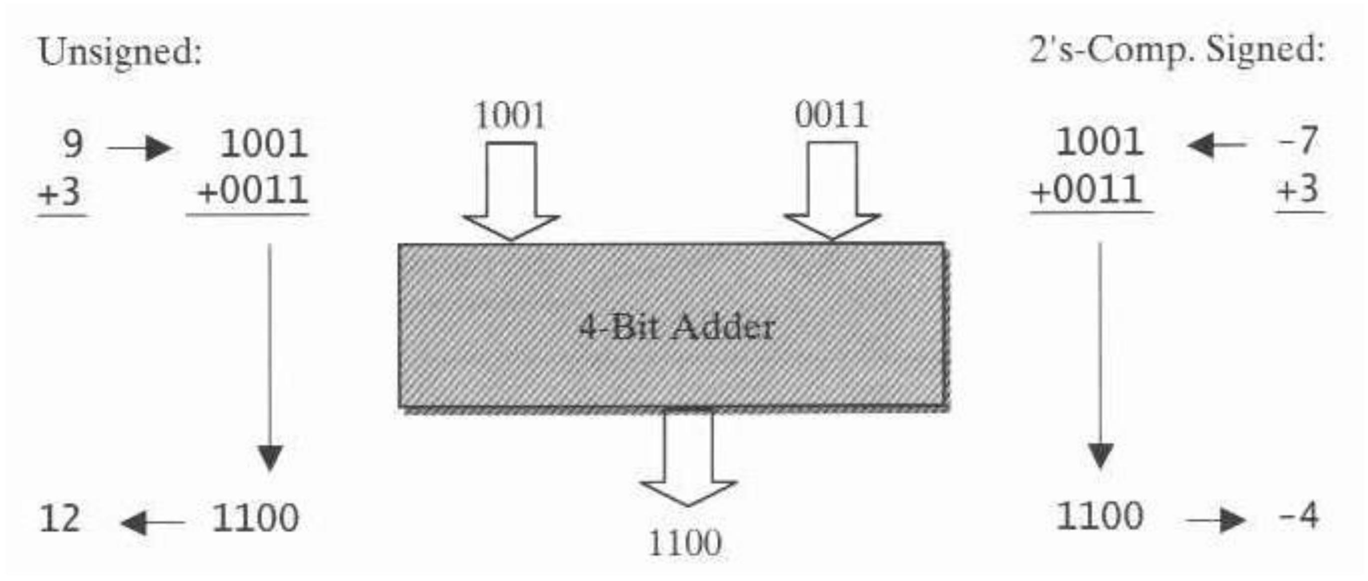
The carry lookahead adder is an advanced design that performs two operations on the input values that determine the output. First, a generate (g) operation is performed by ANDing the X and Y input values. Then a propagate (p) operation is performed by setting p = 1 if either of the input values is 1. This is done by XORing the two inputs. Remember the function of the XOR gate. It is 0 if both values are equal.

The behavior of the carry lookahead adder is expressed by the equation: $C_{i+1} = g_i + p_iC_i$

Representing 4-bit Adder Macrofunction

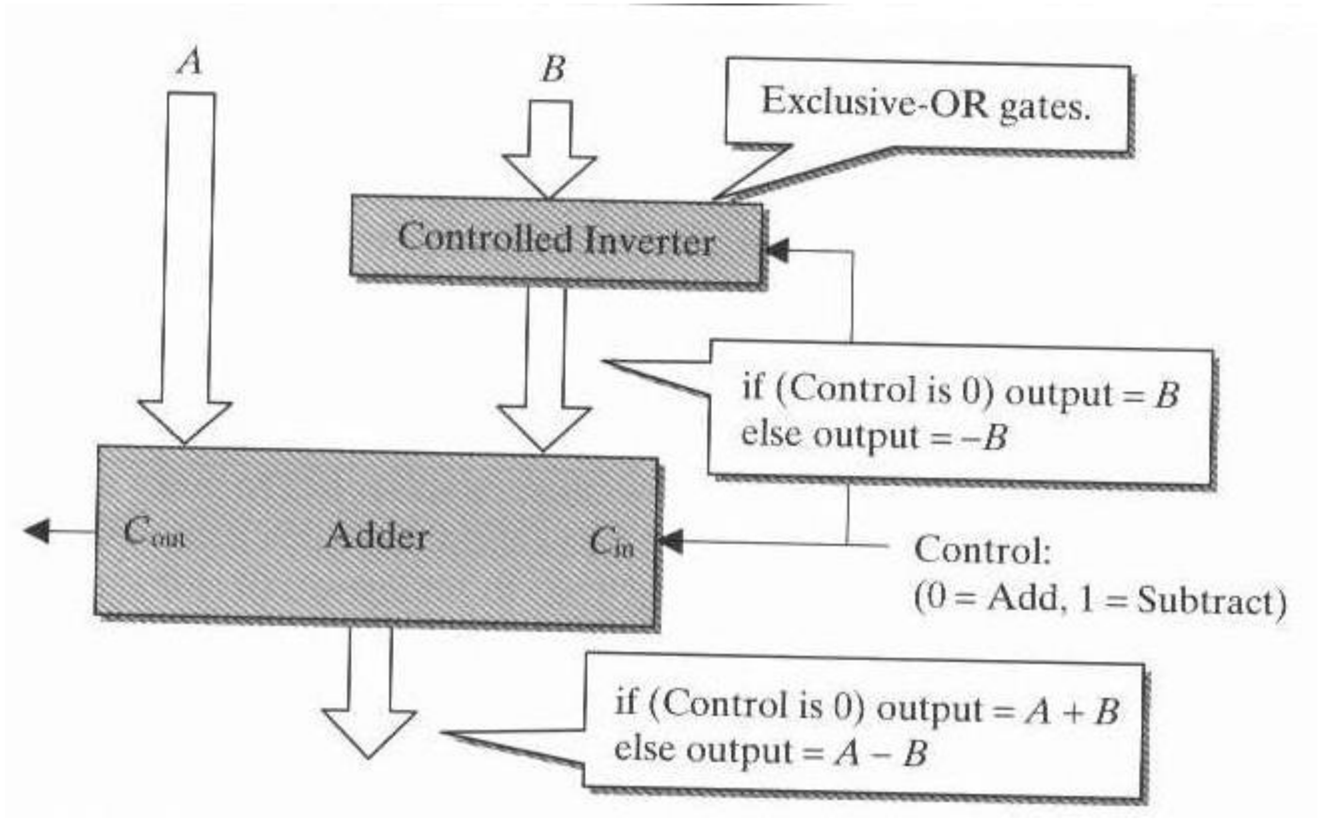


Adding Two 4-bit Numbers



Source: Fundamentals of Embedded Software, Daniel W. Lewis

Performing Subtraction With an Adder?



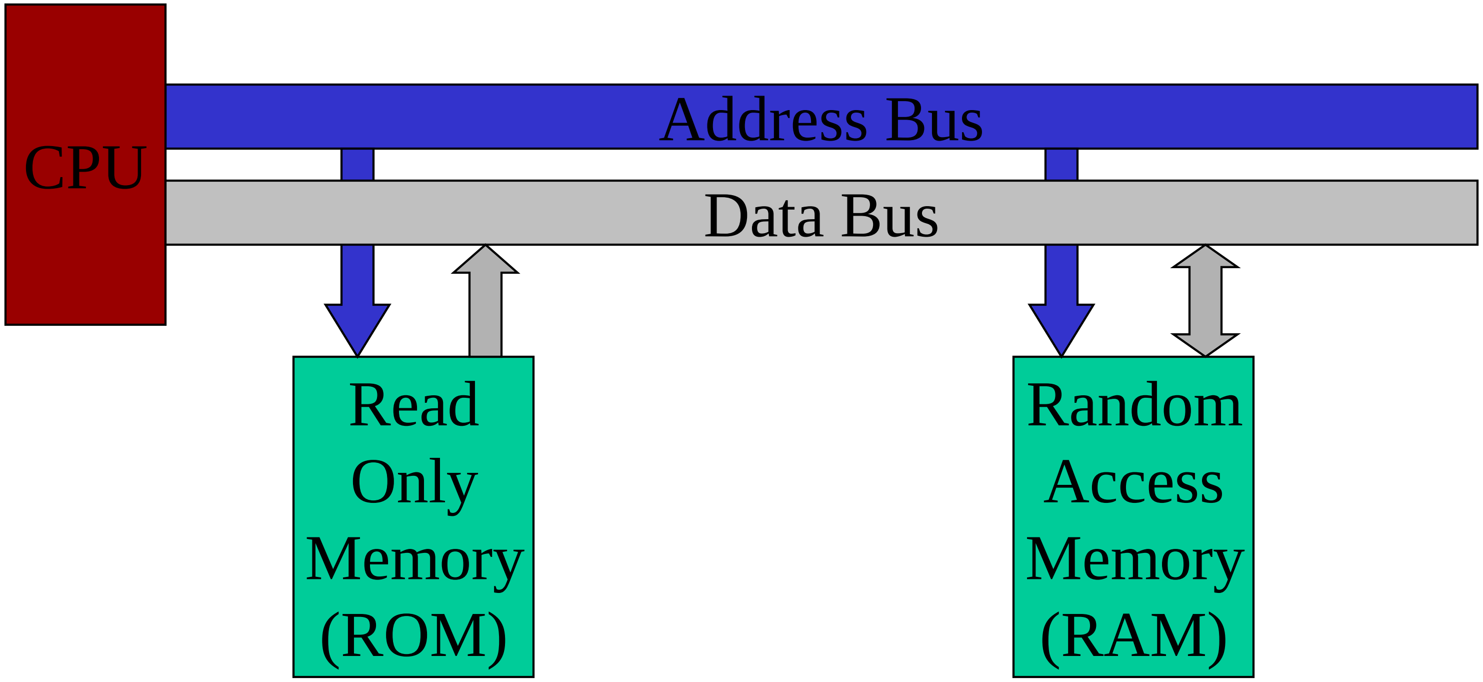
Source: Fundamentals of Embedded Software, Daniel W. Lewis

Memory

Types of Memory

- An ordered sequence of storage cells, each capable of holding a piece of data.
- Volatile Memory
  - RAM – Random Access Memory
- Non-volatile Memory
  - ROM – Read Only Memory

Memories



Factory programmed memory
Used to store BIOS*
Non-volatile memory

User-modifiable memory
Used to store program and data
Volatile memory

*BIOS – Basic Input/Output System