5 Combinatorial Components

Use for data transformation, manipulation, interconnection, and for control:

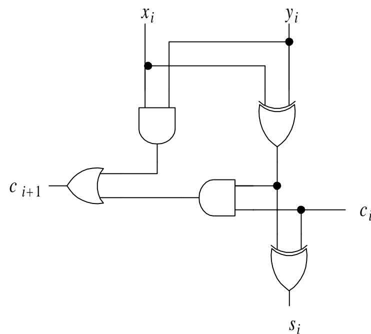
- arithmetic operations - addition, subtraction, multiplication and division.
- logic operations - AND, OR, XOR, and NOT.
- comparison operations - greater than, equal to, and less than.
- bit manipulation operations - shift, rotation, extraction, and insertion.
- interconnect components - selectors, and buses - used to connect different components together.
- conversion components - decoders and encoders - used for conversion between different codes.
- universal components - ROMs and programmable-logic arrays (PLAs) - used primarily in the design of control units.

5.0 Full adder

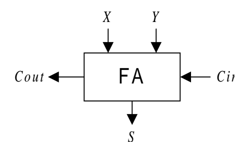
x_i	y_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$
\begin{aligned}
{}_i &= x_i'\, y_i'\, c_i + x_i'\, y_i\, c_i' + x_i\, y_i'\, c_i' + x_i\, y_i\, c_i \\
&= (x_i'\, y_i + x_i\, y_i')\, c_i' + (x_i'\, y_i' + x_i\, y_i)\, c_i \\
&= (x_i \oplus y_i)\, c_i' + (x_i \odot y_i)\, c_i \\
&= (x_i \oplus y_i) \oplus c_i
\end{aligned}
$$

$$
\begin{aligned}
c_{i+1} &= x_i'\, y_i\, c_i + x_i\, y_i'\, c_i + x_i\, y_i\, c_i' + x_i\, y_i\, c_i \\
&= x_i\, y_i\, (c_i' + c_i) + c_i\, (x_i'\, y_i + x_i\, y_i') \\
&= x_i\, y_i \bullet 1 + c_i\, (x_i \oplus y_i)
\end{aligned}
$$

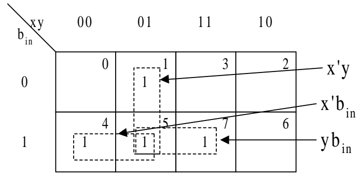


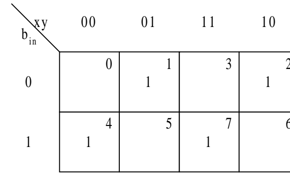
Circuit for Full Adder



Full subtractor

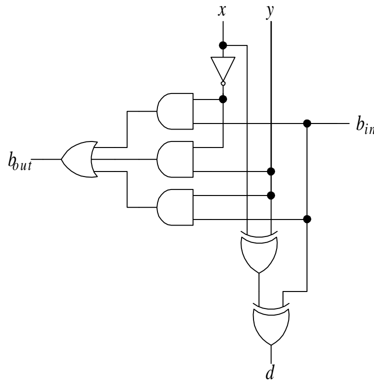
b_{in}	x	y	b_{out}	d	Comment
0	0	0	0	0	x-b_{in}-y = d = 0-0 = 0
0	0	1	1	1	0-0-1 = borrow 2-1 = 1
0	1	0	0	1	1-0-0=1 with no borrow
0	1	1	0	0	1-0-1=0 with no borrow
1	0	0	1	1	x=0-1=-1 because of b_{in}=1, therefore, borrow 2-1=1. Finally, 1-0=1
1	0	1	1	0	x-b_{in}-y=borrow 2-1-1=0
1	1	0	0	0	1-1-0=0 with no borrow
1	1	1	1	1	x-b_{in}=1-1=0. Then 0-1=borrow 2-1=1



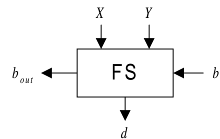


$$b_{out} = x'b_{in} + x'y + yb_{in}$$

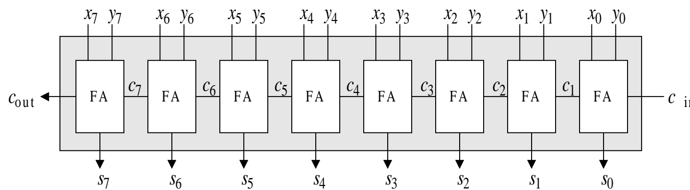
$$\begin{aligned} d &= x'y'b_{in} + xyb_{in} + x'yb'_{in} + xy'b'_{in} \\ &= b_{in}(x'y' + xy) + b'_{in}(x'y + xy') \\ &= b_{in}(x \oplus y)' + b'_{in}(x \oplus y) \\ &= b_{in} \oplus x \oplus y \end{aligned}$$



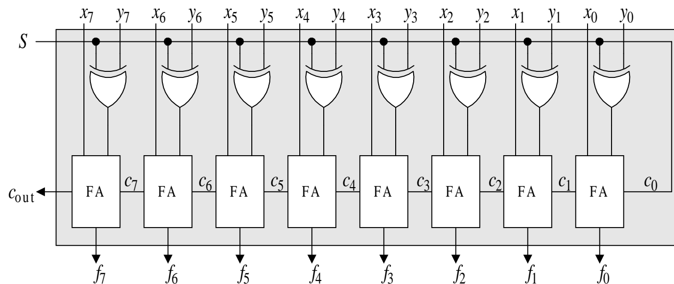
Circuit for Full Subtractor



5.1 Ripple-carry adders



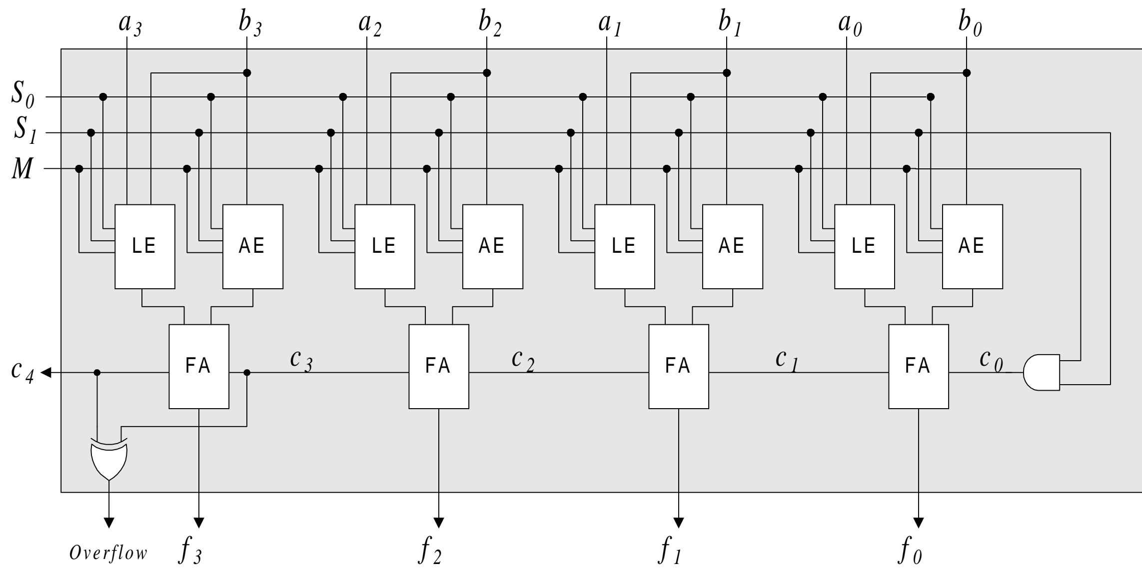
5.3 Adder / Subtractor



S	Function	Comment
0	$X + Y$	Addition
1	$X + Y' + 1$	Subtraction

5.4 Logic Unit

5.5 Arithmetic-Logic Unit (ALU)



Arithmetic Extender (AE)

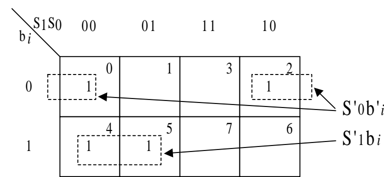
The AE modifies the second operand and passes it to the FA to perform the arithmetic.

M	S_1	S_0	Function Name	Function	X	Y	c_0
1	0	0	Decrement	$A - 1$	A	all 1's	0
1	0	1	Add	$A + B$	A	B	0
1	1	0	Subtract	$A + B' + 1$	A	B'	1
1	1	1	Increment	$A + 1$	A	all 0's	1

Functional table

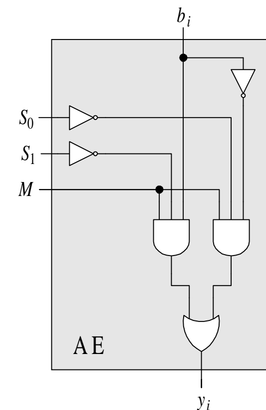
M	S_1	S_0	b_i	y_i
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Truth table



$$y_i = MS_1'b_i + MS_0'b_i'$$

Map representation



Schematic diagram

$$\begin{aligned} y_i &= MS_1'S_0'b_i' + MS_1'S_0'b_i + MS_1'S_0b_i + MS_1S_0'b_i' \\ &= MS_0'b_i'(S_1' + S_1) + MS_1'b_i(S_0' + S_0) \\ &= MS_0'b_i' + MS_1'b_i \end{aligned}$$

Logic Extender (LE)

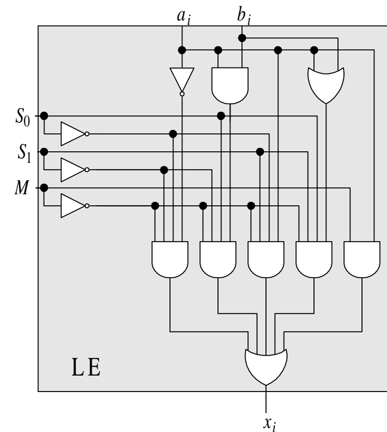
The logic operations are performed in the logic extender. The FAs are used simply as connections for the outputs.

M	S_1	S_0	Function Name	Function	*X*	*Y*	c_0
0	0	0	Complement	*A* ’	*A* ’	0	0
0	0	1	AND	*A* AND *B*	*A* AND *B*	0	0
0	1	0	Identity	*A*	*A*	0	0
0	1	1	OR	*A* OR *B*	*A* OR *B*	0	0

Functional table

M	S_1	S_0	x_i	Indices in map
0	0	0	a_i’	0, 4
0	0	1	$a_i\ b_i$	13
0	1	0	a_i	10, 14
0	1	1	$a_i + b_i$	7, 11, 15
1	X	X	a_i	pass to AE

Truth table

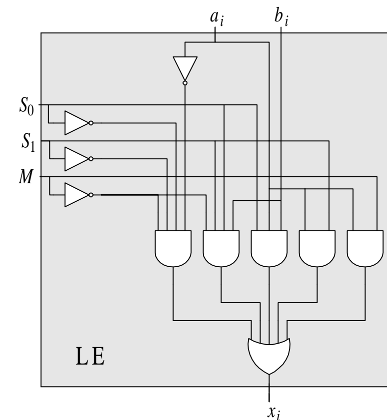


Schematic diagram

$a_i b_i$ \ $S_1 S_0$	M = 0: 00	01	11	10	M = 1: 00	01	11	10
00	0: 1	1	3	2	0	1	3	2
01	4: 1	5	7: 1	6	4	5	7	6
11	12	13: 1	15: 1	14: 1	12: 1	13: 1	15: 1	14: 1
10	8	9	11: 1	10: 1	8: 1	9: 1	11: 1	10: 1

$$x_i = M'S_1'\ S_0'a_i' + M'S_1\ S_0b_i + S_0\ a_i\, b_i + S_1\ a_i + Ma_i$$

Map representation



Simplified schematic diagram

Carry in signal

c_0 – Notice that in the AE functional table, $c_0 = MS_1$

Overflow signal

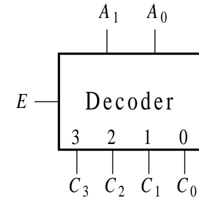
c_4 – The carry-out of the most significant bit represents an overflow in the case of unsigned arithmetic.

Overflow – The XOR of the carry-outs of the two most significant bits represents the overflow in the case of 2’s complement arithmetic.

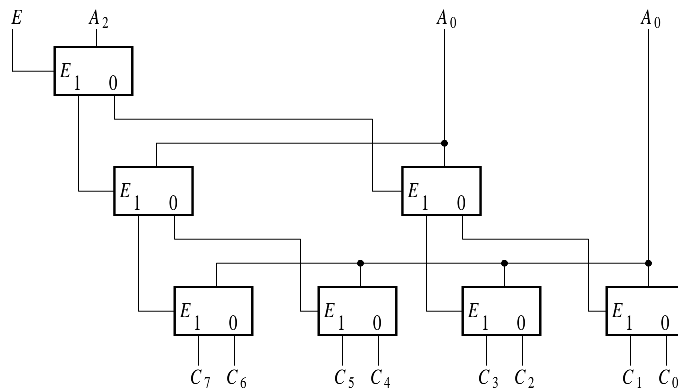
5.6 Decoders / Demultiplexers

To enable only one of *n* components.

E	A_1	A_0	C_3	C_2	C_1	C_0
0	×	×	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0



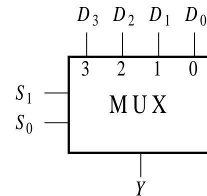
2-to-4 Decoder



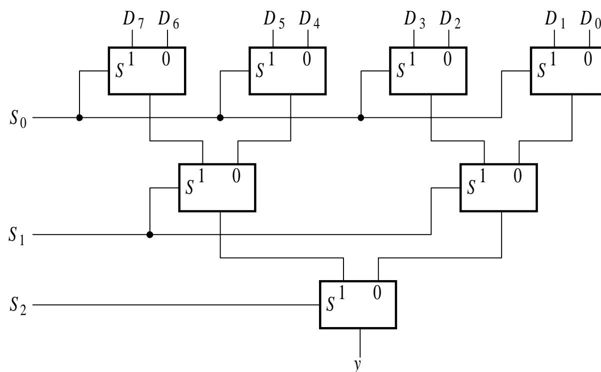
3-to-8 decoder implemented with 1-to-2 decoders

5.7 Selectors / Multiplexers (MUX)

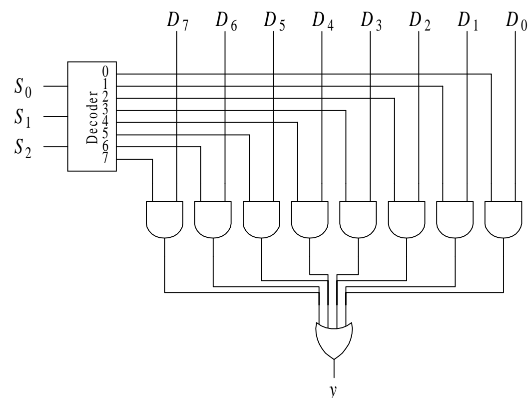
S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



4-to-1 MUX



8-to-1 mux implemented with 2-to-1 muxes



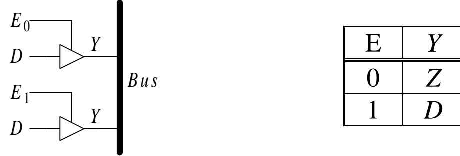
8-to-1 mux implemented with a decoder

5.8 Buses

A **bus** is for connecting several components together, however, only one component can send data to the bus at any one time.

To construct a bus, use a **tristate driver**, whose output provides three different values, 0, 1, and Z.

The value Z represents a high-impedance state which can be thought of as a disconnection from the bus.



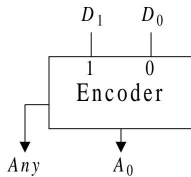
E	Y
0	Z
1	D

5.9 Priority Encoders

A **priority encoder** has:

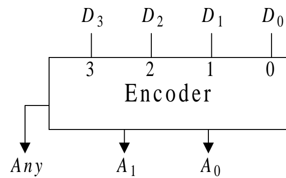
- n inputs, $D_{n-1}, \dots D_0$,
- m outputs, $A_{m-1}, \dots A_0$, where $n = 2^m$, which represents the index (decimal value) of the most significant input bit D_i that has a value equal to 1.
- an additional output call *Any*, which will be 1 whenever any of the inputs has a value that is different from 0.

2-to-1 priority encoder



D_1	D_0	A_0	*Any*
0	0	0	0
0	1	0	1
1	X	1	1

4-to-2 priority encoder



D_3	D_2	D_1	D_0	A_1	A_0	*Any*
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

5.10 Magnitude Comparators

A **comparator** compares two positive integers X and Y and then generates two boolean results G and L as follows:

Test	G	L
$X > Y$	1	
$X \leq Y$	0	
$X < Y$		1
$X \geq Y$		0
$X \neq Y$	1	1
$X = Y$	0	0

For the n-bit integers X and Y, we compare them one bit at a time, starting with the most significant bit,

$G_i = (x_i > y_i)$ or $[(x_i = y_i)$ and $(G_{i-1} > L_{i-1})]$
$L_i = (x_i < y_i)$ or $[(x_i = y_i)$ and $(G_{i-1} < L_{i-1})]$

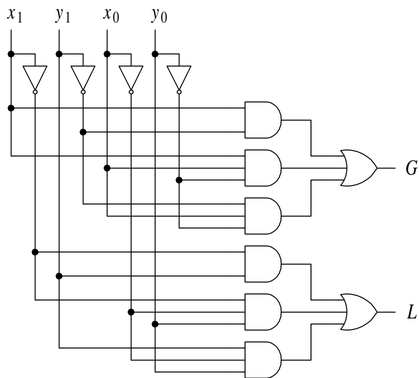
x_1	y_1	x_0	y_0	G	L
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	1	0	1
1	1	1	0	1	0
1	1	1	1	0	0

a_1b_1 \ a_0b_0	00	01	11	10
00	0	1	3	2 1
01	4	5	7	6
11	12	13	15	14 1
10	8 1	9 1	11 1	10 1

$$G = a_1b_1' + a_1a_0b_0' + b_1'a_0b_0'$$

a_1b_1 \ a_0b_0	00	01	11	10
00	0	1 1	3	2
01	4 1	5 1	7 1	6 1
11	12	13 1	15	14
10	8	9	11	10

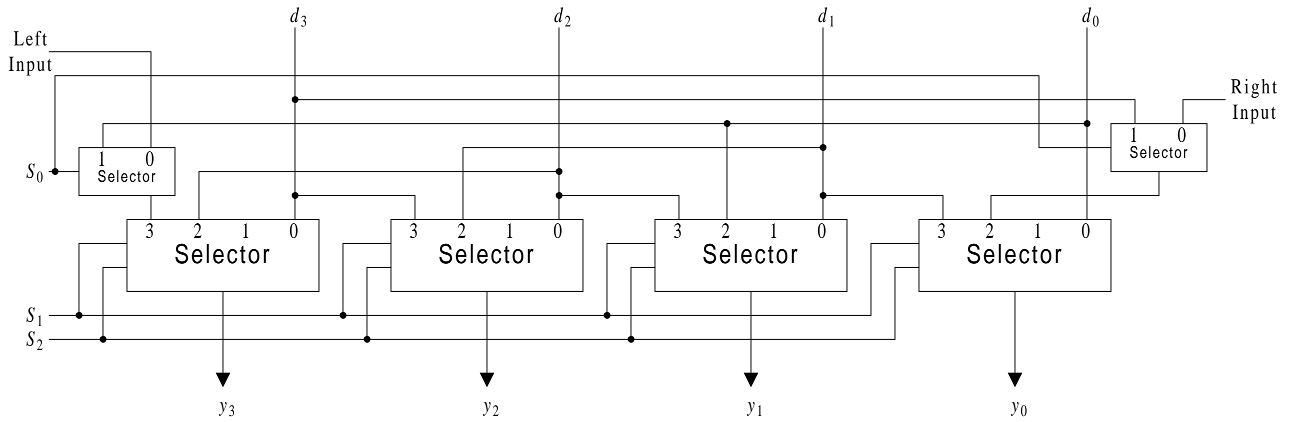
$$L = a_1'b_1 + a_1'a_0'b_0 + b_1a_0'b_0$$



5.11 Shifters and Rotators

Use **selectors** to implement.

S_2	S_1	S_0	Y	Comment
0	0	×	D	No shift
0	1	×		Not used
1	0	0	ShiftLeft(D)	Shift left
1	0	1	RotateLeft(D)	Rotate left
1	1	0	ShiftRight(D)	Shift right
1	1	1	RotateRight(D)	Rotate right

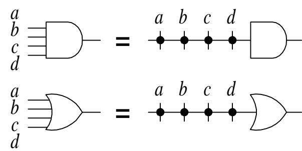


5.12 Read-Only Memories

ROMs can be thought of as a universal logic element that is able to implement concurrently several different Boolean functions that are defined on the same set of variables.

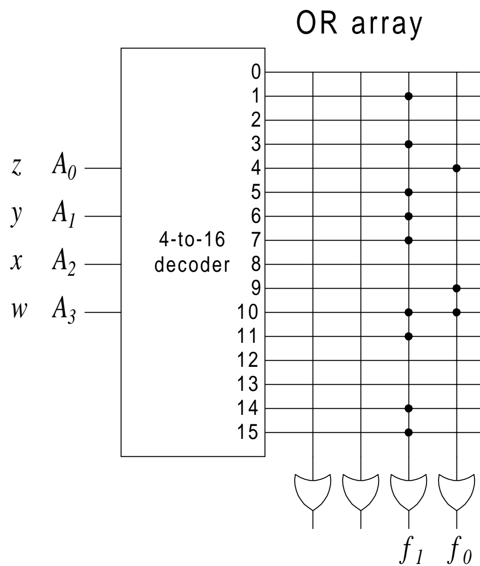
A 16×4 ROM can implement 4 functions.

Programmable ANDs and ORs.



Use a 16×4 ROM to implement the following two functions:

$$f_0 = w\,x'\,y'\,z + w\,x'\,y\,z' + w'\,x\,y'\,z'$$
$$f_1 = x\,y + w'\,z + w\,x'\,y$$



5.13 Programmable Logic Arrays (PLA)

ROMs are not very efficient when we use them to implement sparse functions, i.e. functions with only a small number of 1's because in such cases, many of the words in the ROM will have a value of 0, which is a waste of silicon area.

A PLA differs from a ROM in the address decoder. Instead of a full decoder, PLAs use a programmable decoder call an AND array.

Use a $4 \times 8 \times 4$ PLA to implement the following two functions:

$$f_0 = w\,x'\,y'\,z + w\,x'\,y\,z' + w'\,x\,y'\,z'$$
$$f_1 = w'\,x\,y + w\,y'\,z$$