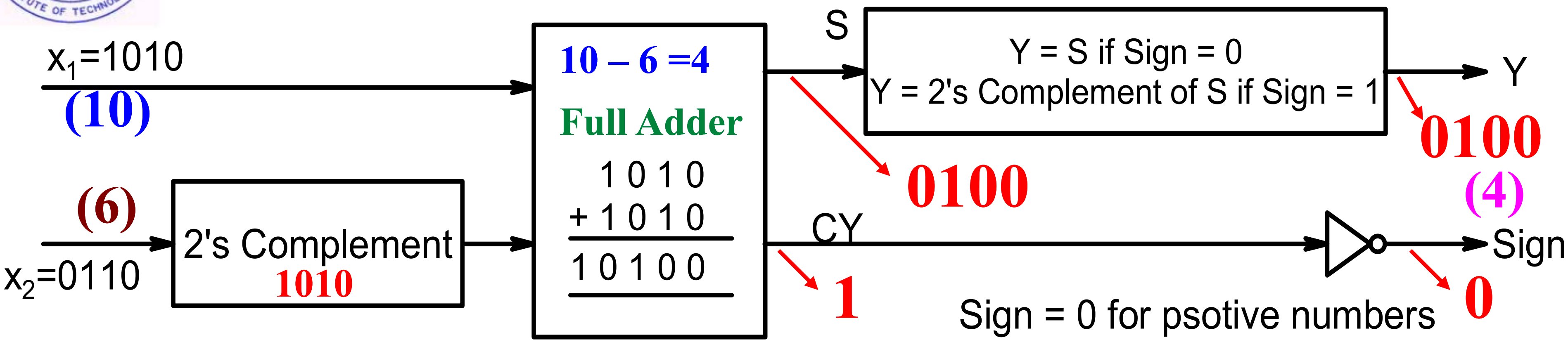
ESc201, Lecture 31: (Digital) Examples of Subtraction

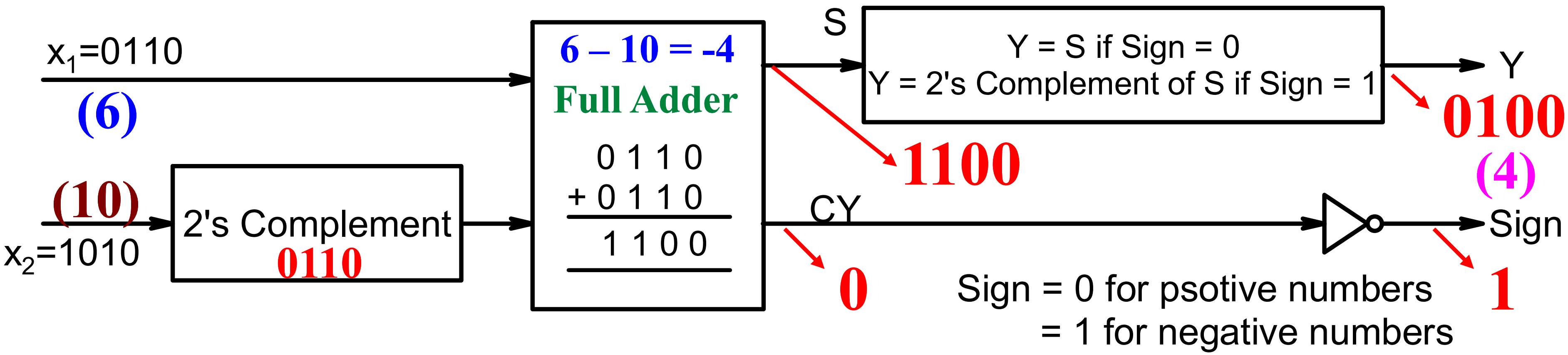
Example of Minuend > Subtrahend



Sign = 0 for psotive numbers
= 1 for negative numbers

An inverter is added to the sign bit to, keep conformity with the positive and negative number sign bit convention.

Example of Subtrahend > Minuend



It makes sense to use adder as a subtractor as well provided additional circuit required for carrying out 2's complement is simple. (find out later that XOR gate is sufficient)

ESc201, Lecture 31: Digital

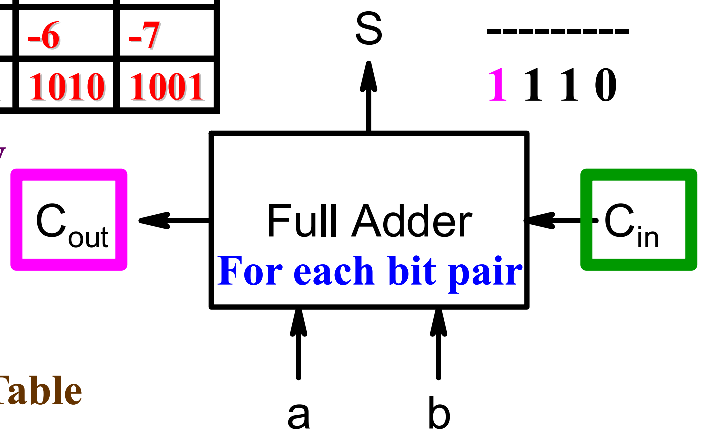
Binary positive and negative number addition & subtraction

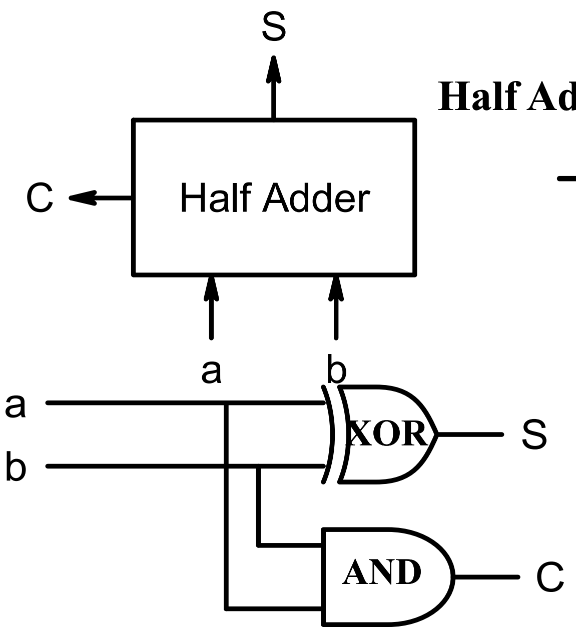
Decimal	0	1	2	3	4	5	6	7
Signed Magnitude	0000	0001	0010	0011	0100	0101	0110	0111
Decimal	-0	-1	-2	-3	-4	-5	-6	-7
Signed Magnitude	1000	1001	1010	1011	1100	1101	1110	1111
Decimal		-1	-2	-3	-4	-5	-6	-7
Signed 1's complement		1110	1101	1100	1011	1010	1001	1000
Decimal		-1	-2	-3	-4	-5	-6	-7
Signed 2's complement		1111	1110	1101	1100	1011	1010	1001

```
   1
 1 1 1
 1 1 0
---------
1 1 1 0
```

```
  1 0 1          1 1 0 1
  1 1 0        + 1 1 1 0
 1 0 1 1       1 1 0 1 1
```

Addition of 3-bit & 4-bit binary numbers need to take the Carry of each Half-Adder sum bits to the next higher bit.





Half Adder Truth Table

a	b	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S=\bar{a}.b+a.\bar{b};$$

$$C=a.b$$

Full-Adder Truth Table

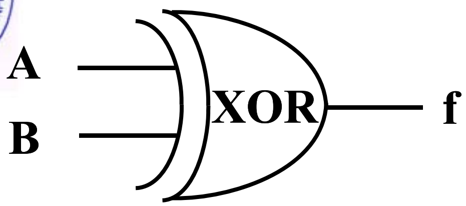
a	b	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S=\bar{a}.\bar{b}.c_{in}+\bar{a}.b.\overline{c_{in}}+a.\bar{b}.\overline{c_{in}}+a.b.c_{in};$$

$$C_{out}=\bar{a}.b.c_{in}+a.\bar{b}.c_{in}+a.b.\overline{c_{in}}+a.b.c_{in}$$

Can also be done with Half - Adder units (Check HA#8)

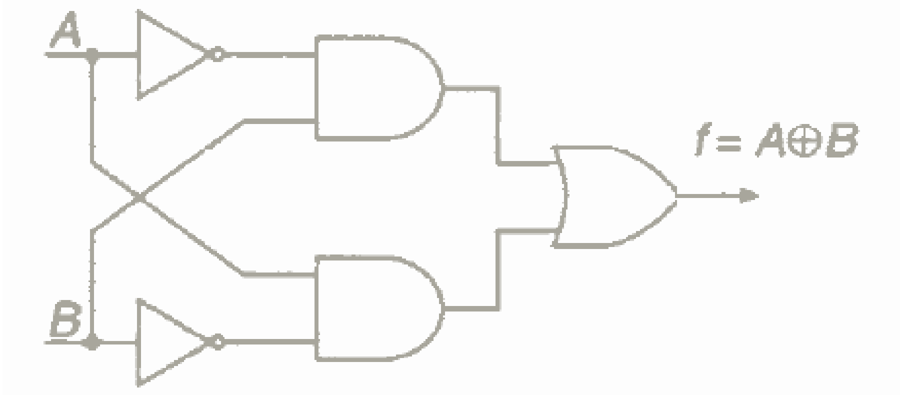
XOR Implementation



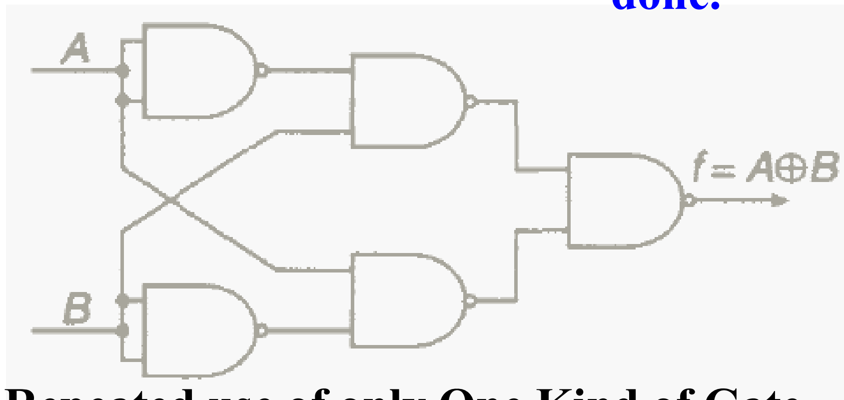
A	0	0	1	1
B	0	1	0	1
f	0	1	1	0

$$f = \bar{A}.B + A.\bar{B}$$

Often there is lot of further optimization that can be done.

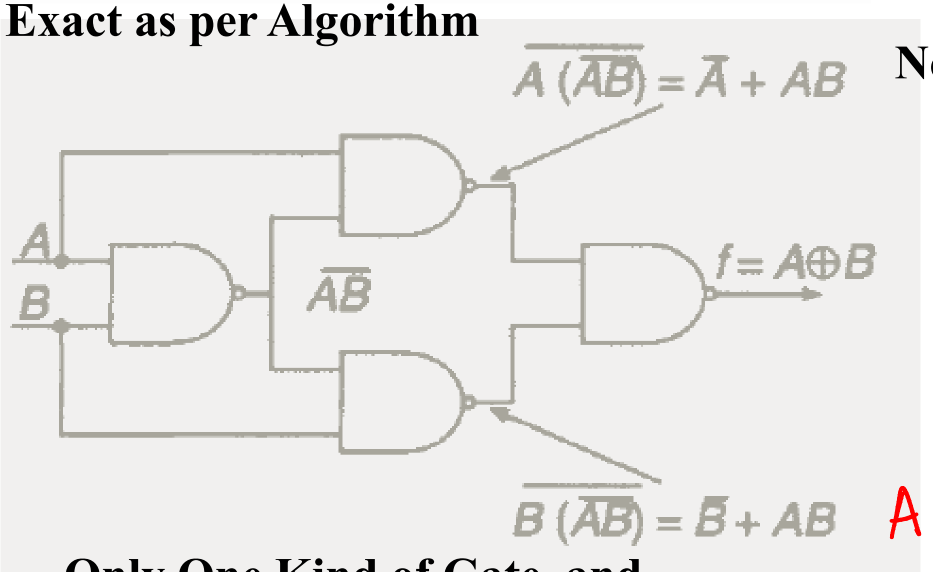


Exact as per Algorithm



Repeated use of only One Kind of Gate

No reduction in the number of gates required



Only One Kind of Gate, and reduction in the number of gates.

$$f = \overline{\left(\overline{\bar{A} + A.B}\right).\left(\overline{\bar{B} + A.B}\right)} = \left(\bar{A} + A.B\right) + \left(\bar{B} + A.B\right)$$

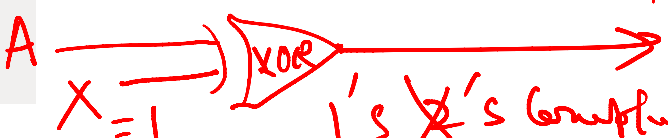
$$= A.\overline{A.B} + B.\overline{A.B} = \cancel{A.\bar{A}} + A.\bar{B} + B.\bar{A} + \cancel{B.\bar{B}}$$

$$= \bar{A}.B + A.\bar{B}$$

2's

0011 → ~~1101~~

1100



1's ~~2~~'s Complement of A

Therefore after XOR add 1 to get 2'compliment.

ESc201, Lecture 31: (Digital) Binary Addition/Subtraction

If both sign bits are same, it's a simple case of addition.

+ 6	00000110	− 6	11111010
+13	00001101	+13	00001101
+19	00010011	+ 7	00000111

No 2's compliment is required.

7 bits + 1 sign bit

2's complement of 6 (00000110) is 11111001

1100101·001
0110011·01

For non-integer add/subtract the binary point hass to be aligned 1[st], otherwise same.

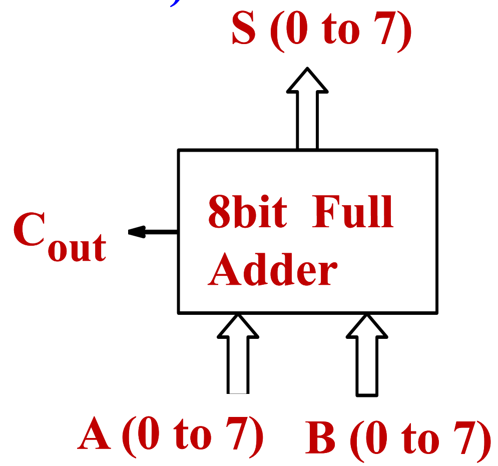
− 6	11111010	+ 6	00000110
−13	11110011	−13	11110011
−19	11101101	− 7	11111001

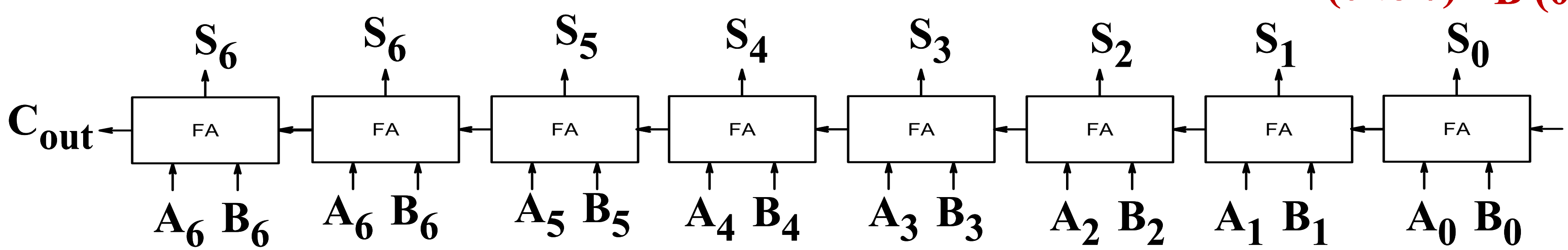
Result is a negative number, hence 2's complement of 7 (00000111) is the result.

The sign bit also turns out to be correct, if one rejects the overflow bit.

Hence 2's Compliment 11111001 has come out as final answer.

Implementation : Example of a 8-bit Adder as required for above.



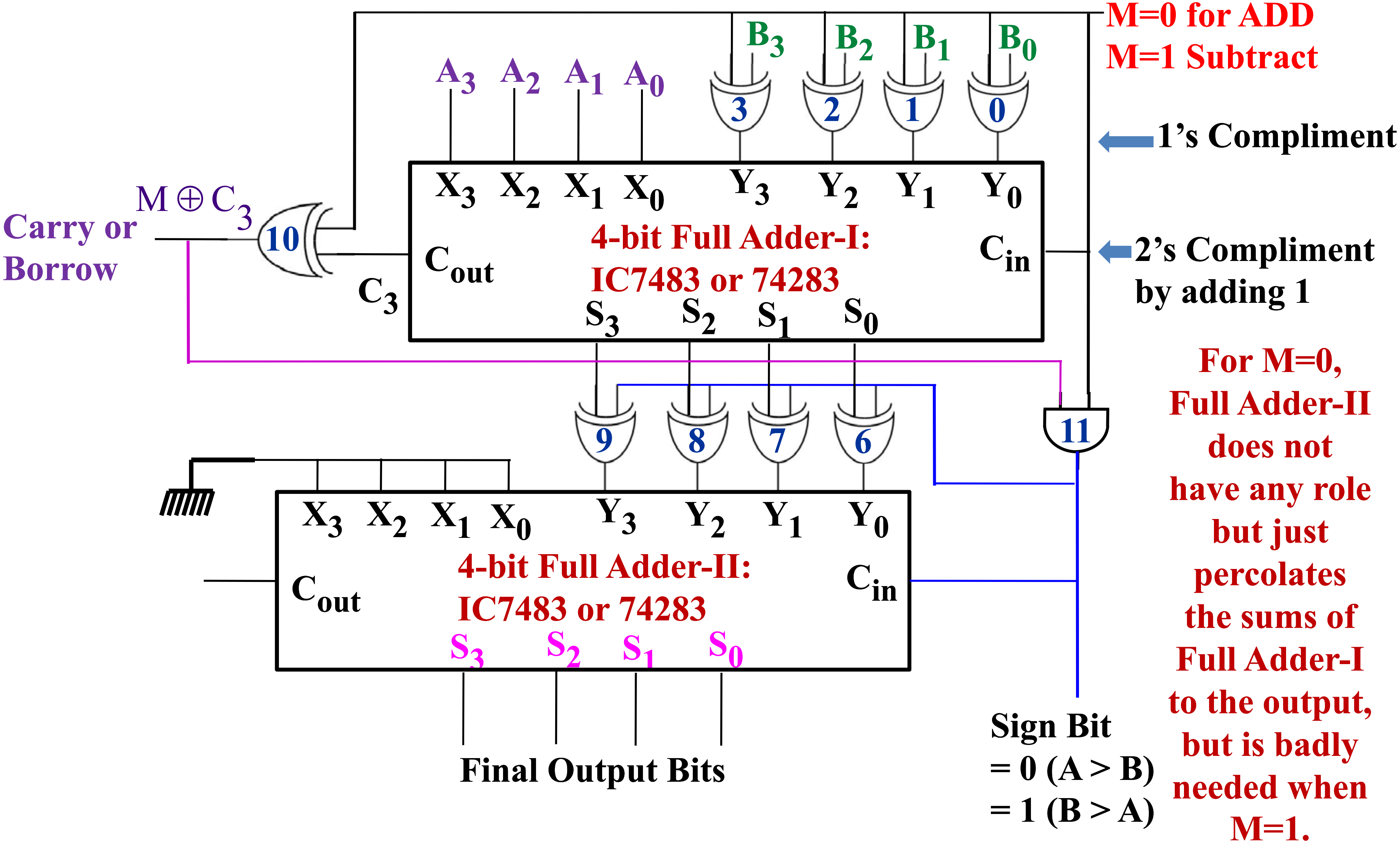


ESc201, Lecture 31: (Digital Adder/Subtractor)

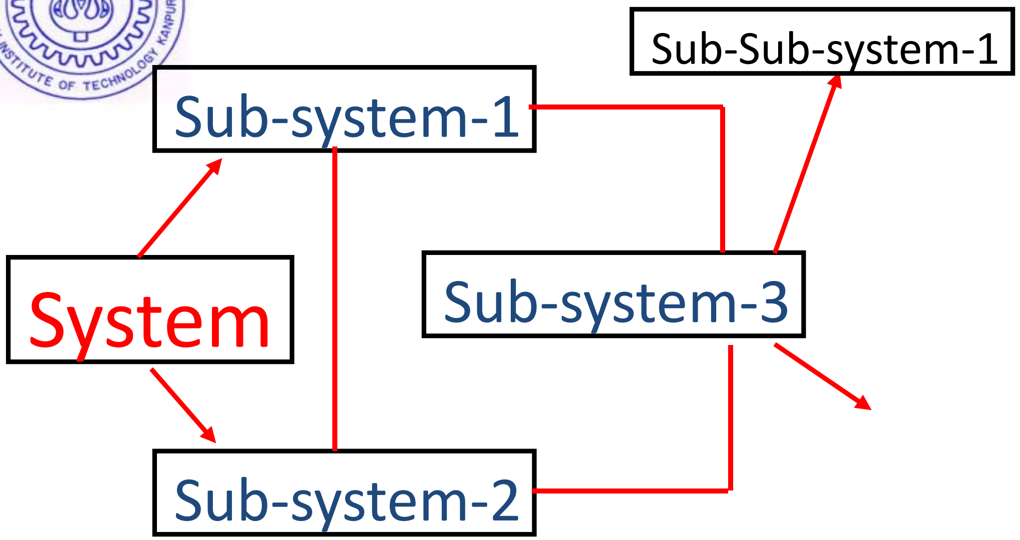
Example with 4-bit words

$$B_0 \oplus 0 = B_0.\bar{0} + \bar{B}_0.0 = B_0$$

$$B_0 \oplus 1 = B_0.\bar{1} + \bar{B}_0.1 = \bar{B}_0$$



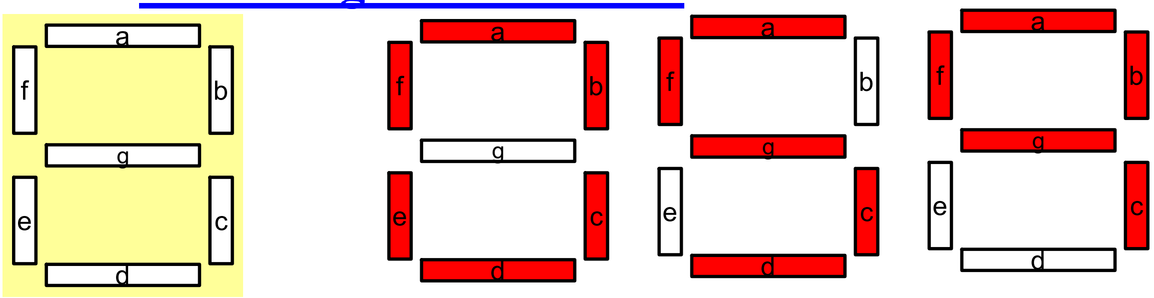
ESc201, Lecture 31: (Digital) General Approach

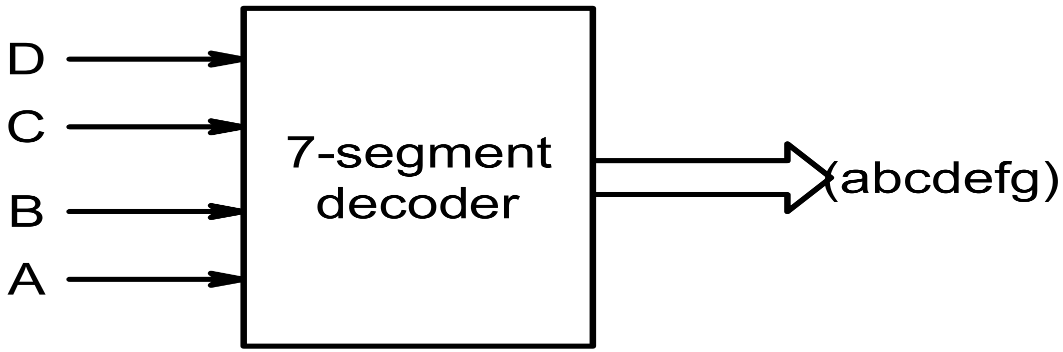


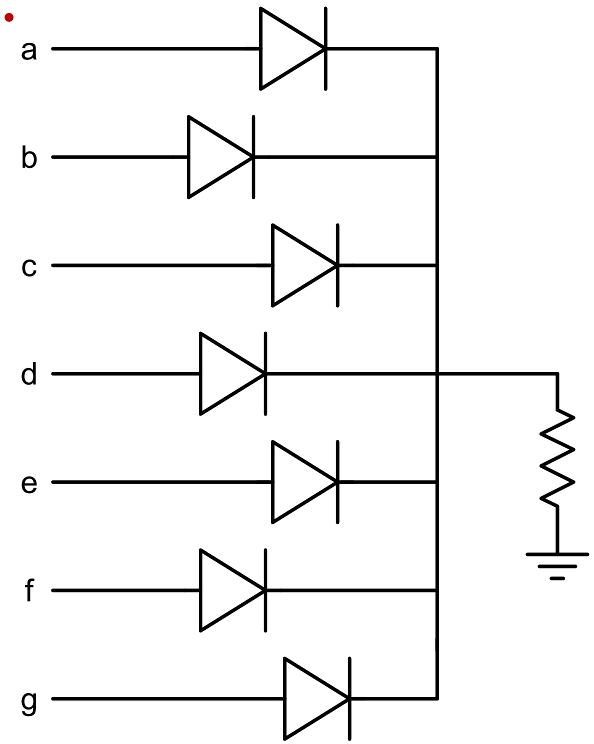
There are certain sub-systems or blocks that are used quite often such as :

1. Decoders, Encoders
2. Multiplexers
3. Adder/Subtractors, Multipliers
4. Comparators
5. Parity Generators
6. ……..

Seven Segment Decoder

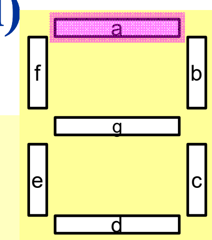






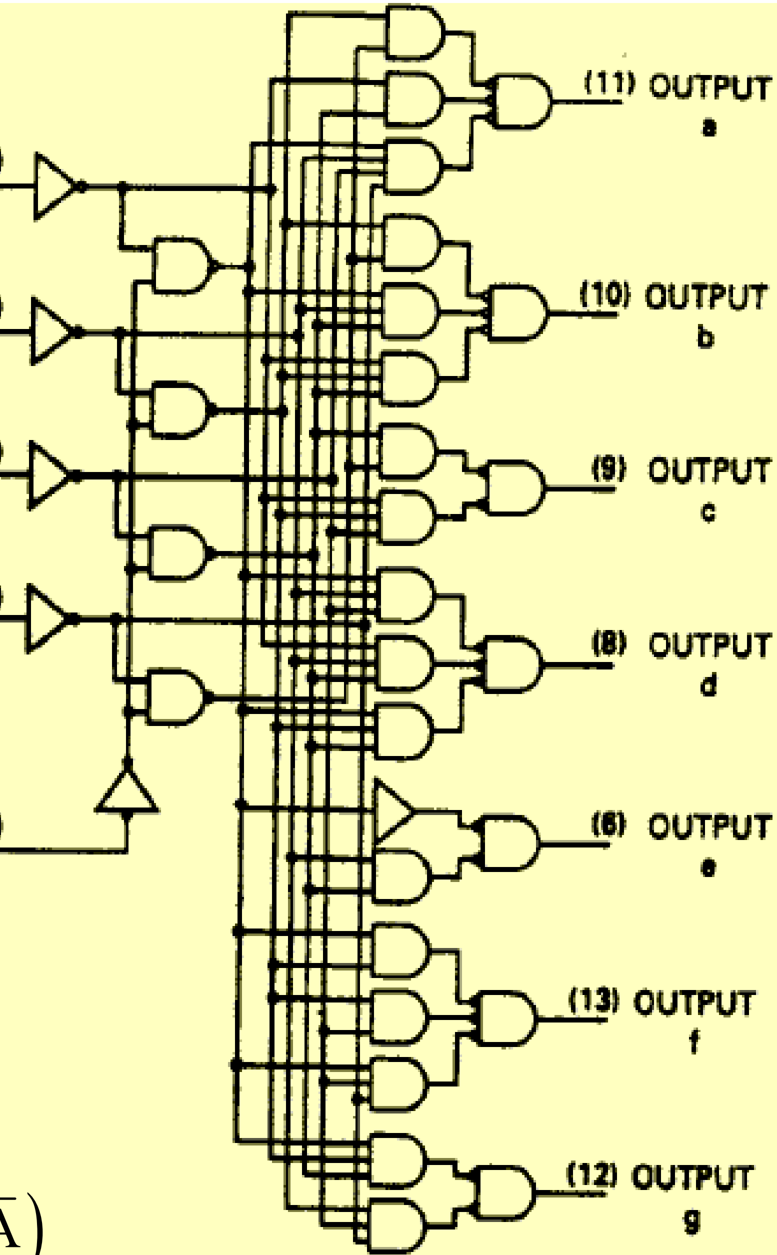
ESc201, Lecture 31: (Digital)

Segement ‘a’ has to be on for: 0, 2, 3, 5, ~~6~~, 7, 8, and 9.



7449 BCD to seven segment decoder

Dec or Function	Input					Output						
	D	C	B	A	BI	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	1	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	1	0	0	1
4	0	1	0	0	1	0	1	1	0	0	1	1
5	0	1	0	1	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	0	1	1
10	1	0	1	0	1	0	0	0	1	1	0	1
11	1	0	1	1	1	0	0	1	1	0	0	1
12	1	1	0	0	1	0	1	0	0	0	1	1
13	1	1	0	1	1	1	0	0	1	0	1	1
14	1	1	1	0	1	0	0	0	1	1	1	1
15	1	1	1	1	1	0	0	0	0	0	0	0
BI	×	×	×	×	0	0	0	0	0	0	0	0



DC \ BA	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	0	0
10	1	1	0	0

output: a

Determine the simplified POS

$$a=\left(\overline{DB}\right).\left(\overline{C\overline{A}}\right).\left(\overline{\overline{D}\,\overline{C}\,\overline{B}A}\right)$$

$$a=\left(\overline{D}+\overline{B}\right).\left(\overline{C}+A\right).\left(D+C+B+\overline{A}\right)$$

ESc201, Lecture 31: (Digital Multiplication)

Advantage in Binary is that the product can only be "1" when both are "1". No question of carry from each bit.

				a_3	a_2	a_1	a_0	Mulitiplicand
				b_3	b_2	b_1	b_0	Multiplier
				p_{30}	p_{20}	p_{10}	p_{00}	Partial product
			p_{31}	p_{21}	p_{11}	p_{01}		
		p_{32}	p_{22}	p_{12}	p_{02}			
	p_{33}	p_{23}	p_{13}	p_{03}				
P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0	Product

Multiplier Using Full Adders

Example of 2-bit multiplication