NEC 304

STLD

Lecture 14

Binary Adders and Subtractors

Rajeev Pandey

Department Of ECE

rajeevvce2007@gmail.com

Overview

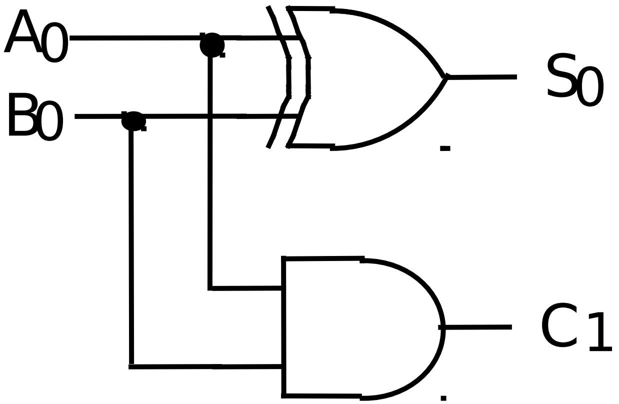
- **Addition and subtraction of binary data is fundamental**
  - **Need to determine hardware implementation**
- **Represent inputs and outputs**
  - **Inputs: single bit values, carry in**
  - **Outputs: Sum, Carry**
- **Hardware features**
  - **Create a single-bit adder and chain together**
- **Same hardware can be used for addition and subtraction with minor changes**
- **Dealing with overflow**
  - **What happens if numbers are too big?**

Half Adder

° **Add two binary numbers**

- **A_0 , B_0 -> single bit inputs**
- **S_0 -> single bit sum**
- **C_1 -> carry out**

A_0	B_0	S_0	C_1
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Dec	Binary
1	1
+1	+1
2	10

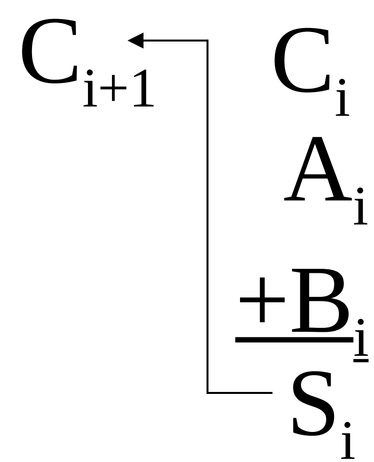
Multiple-bit Addition

- Consider single-bit adder for each bit position.

	A_3	A_2	A_1	A_0
A	0	1	0	1

	B_3	B_2	B_1	B_0
B	0	1	1	1

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



Each bit position creates a sum and carry

Full Adder

- Full adder includes carry in C_i
- Notice interesting pattern in Karnaugh map.

C_i	A_i	B_i	S_i	C_{i+1}
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

C_i \ A_iB_i	00	01	11	10
0		1		1
1	1		1	

S_i

Full Adder

- **Full adder includes carry in C_i**
- **Alternative to XOR implementation**

C_i	A_i	B_i	S_i	C_{i+1}
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_i = !C_i & !A_i & B_i
!C_i & A_i & !B_i
C_i & !A_i & !B_i
C_i & A_i & B_i

Full Adder

- **Reduce and/or representations into XORs**

$$S_i = !C_i \ \& \ !A_i \ \& \ B_i$$
$$\# \ !C_i \ \& \ A_i \ \& \ !B_i$$
$$\# \ C_i \ \& \ !A_i \ \& \ !B_i$$
$$\# \ C_i \ \& \ A_i \ \& \ B_i$$

$$S_i = !C_i \ \& \ (!A_i \ \& \ B_i \ \# \ A_i \ \& \ !B_i)$$
$$\# \ C_i \ \& \ (!A_i \ \& \ !B_i \ \# \ A_i \ \& \ B_i)$$

$$S_i = !C_i \ \& \ (A_i \ \$ \ B_i)$$
$$\# \ C_i \ \& \ !(A_i \ \$ \ B_i)$$

$$S_i = C_i \ \$ \ (A_i \ \$ \ B_i)$$

Full Adder

- **Now consider implementation of carry out**
- **Two outputs per full adder bit (**C_{i+1}, S_i)

C_i	A_i	B_i	S_i	C_{i+1}
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

C_i \ A_iB_i	00	01	11	10
0			1	
1		1	1	1

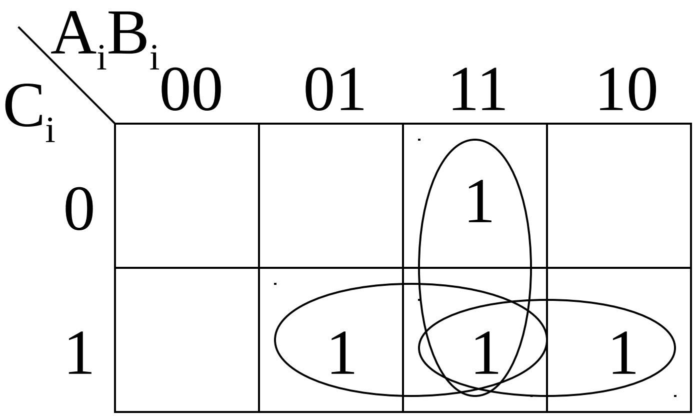
C_{i+1}

Note: 3 inputs

Full Adder

- **Now consider implementation of carry out**
- **Minimize circuit for carry out - C_{i+1}**

C_i	A_i	B_i	S_i	C_{i+1}
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



C_{i+1}

$$C_{i+1} = A_i \ \& \ B_i \ \# \ C_i \ \& \ B_i \ \# \ C_i \ \& \ A_i$$

Full Adder

$$C_{i+1} = A_i \ \& \ B_i$$
$$\# \ C_i \ !A_i \ \& \ B_i$$
$$\# \ C_i \ \& \ A_i \ \& \ !B_i$$

$$C_{i+1} = A_i \ \& \ B_i$$
$$\# \ C_i \ \& \ (!A_i \ \& \ B_i \ \# \ A_i \ \& \ !B_i)$$

$$C_{i+1} = A_i \ \& \ B_i \ \# \ C_i \ \& \ (A_i \ \$ \ B_i)$$

Recall:

$$S_i = C_i \ \$ \ (A_i \ \$ \ B_i)$$

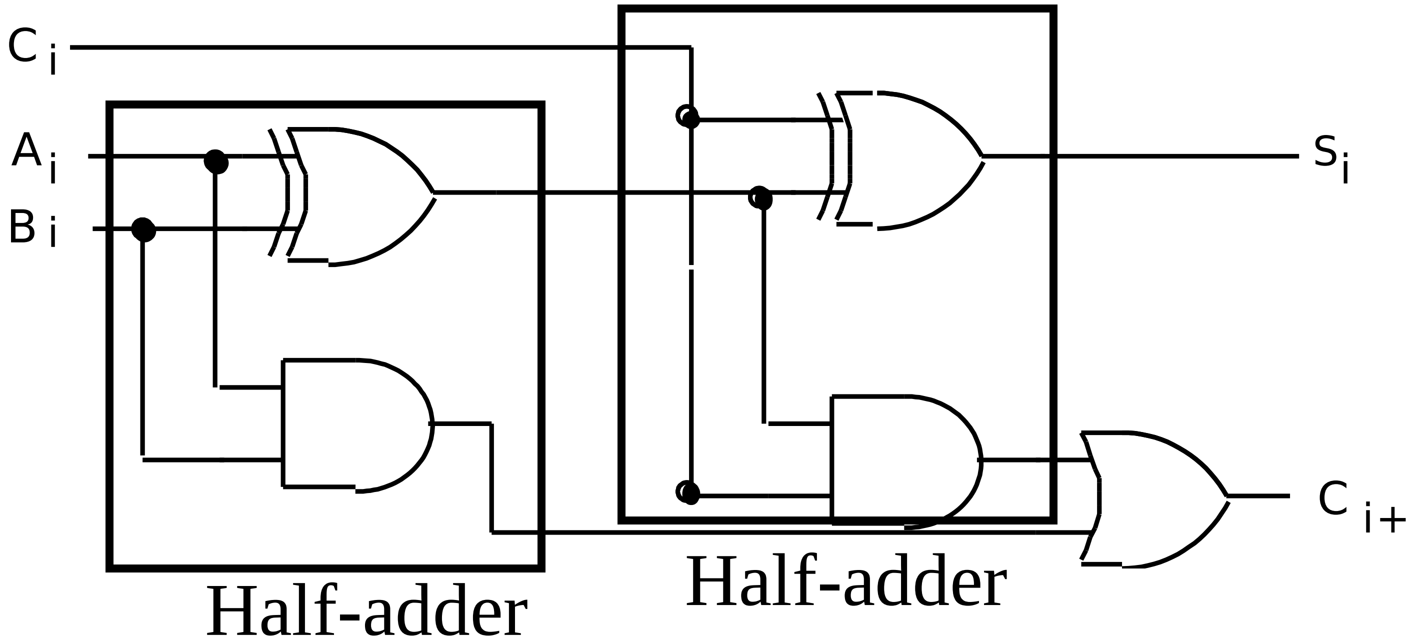
$$C_{i+1} = A_i \ \& \ B_i \ \# \ C_i \ \& \ (A_i \ \$ \ B_i)$$

Full Adder

- **Full adder made of several half adders**

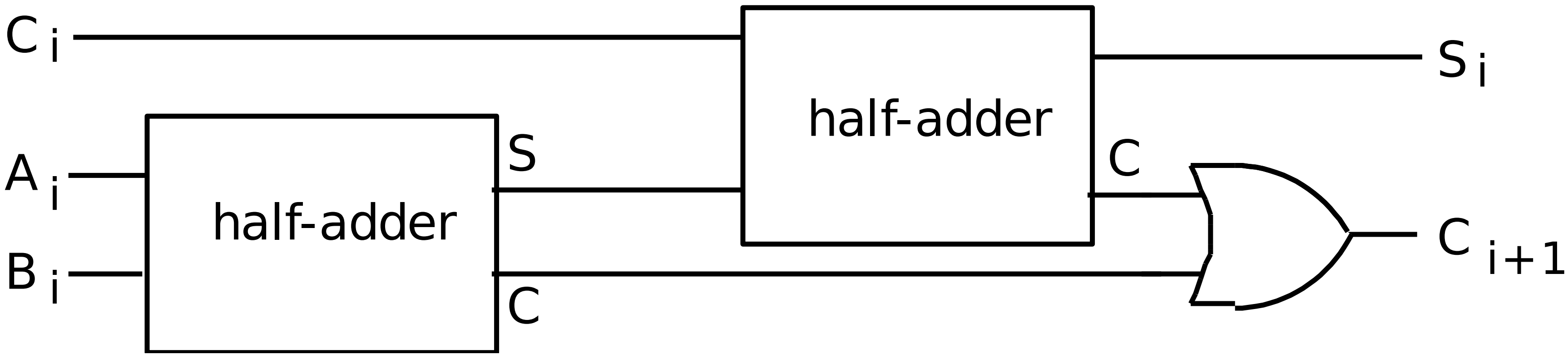
$$S_i = C_i \$ (A_i \$ B_i)$$

$$C_{i+1} = A_i \& B_i \# C_i \& (A_i \$ B_i)$$



Full Adder

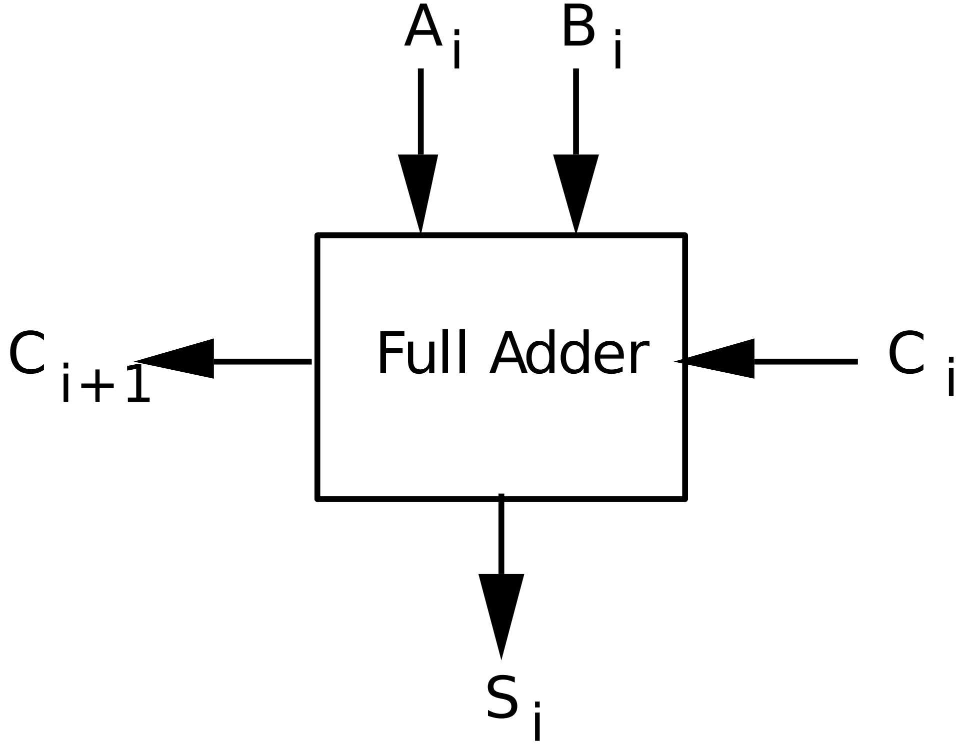
° **Hardware repetition simplifies hardware design**



A full adder can be made from two half adders (plus an OR gate).

Full Adder

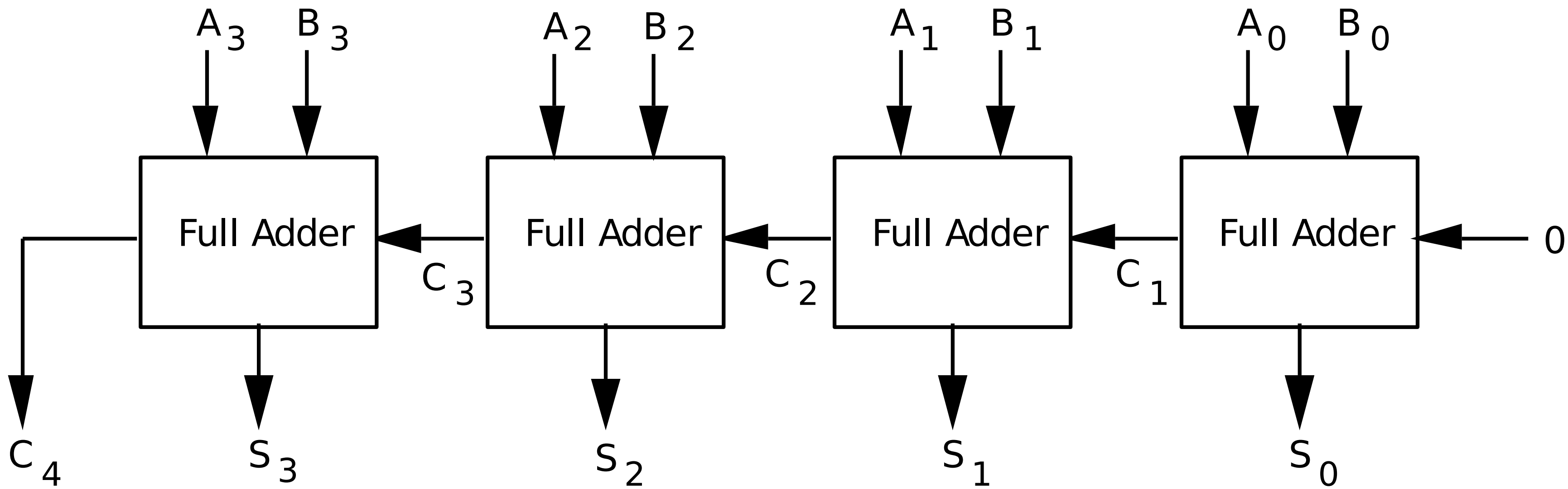
- **Putting it all together**
  - **Single-bit full adder**
  - **Common piece of computer hardware**



Block Diagram

4-Bit Adder

- Chain single-bit adders together.
- What does this do to delay?



C	1 1 1 0
A	0 1 0 1
B	0 1 1 1
S	1 1 0 0

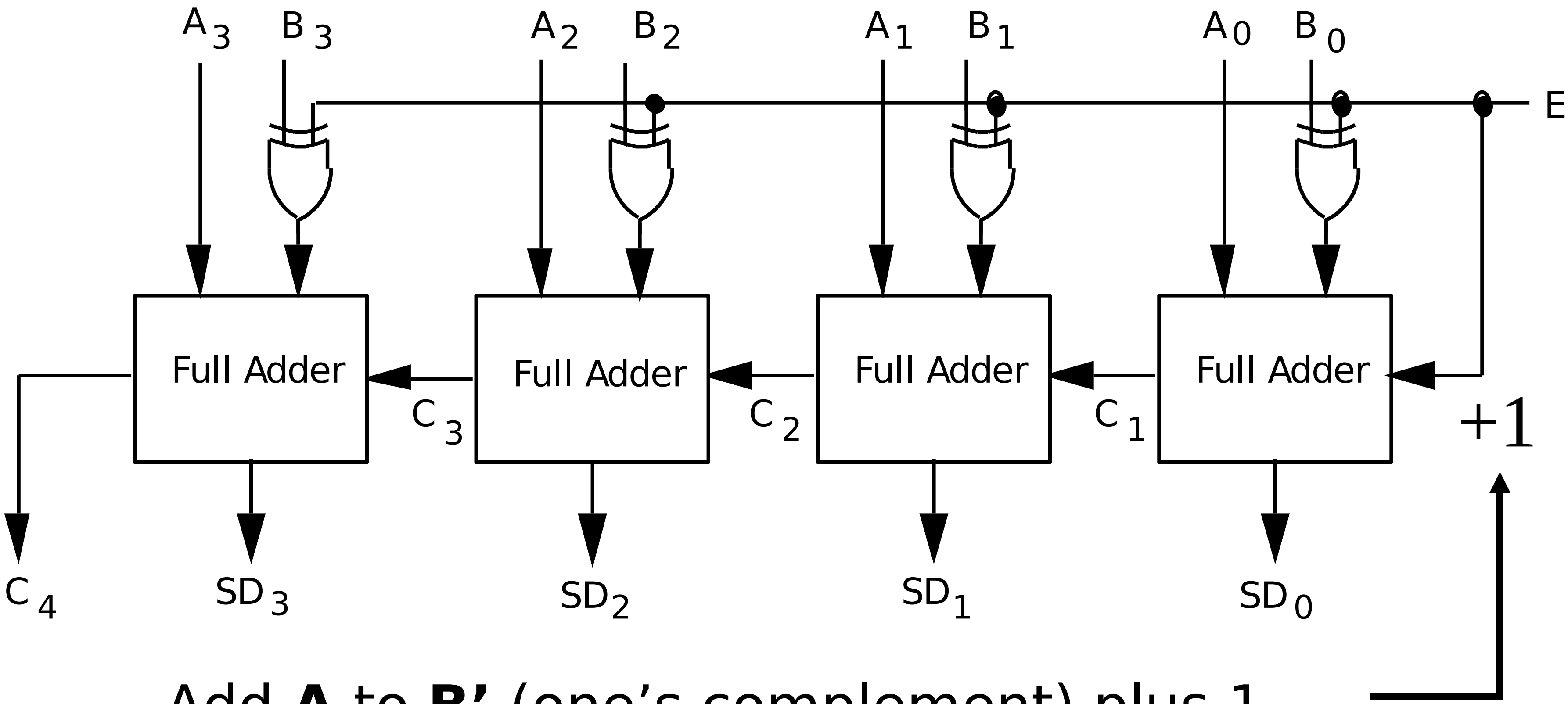
Negative Numbers – 2's Complement.

- **Subtracting a number is the same as:**
  1. **Perform 2's complement**
  2. **Perform addition**
- **If we can augment adder with 2's complement hardware?**

$$1_{10} = 01_{16} = 00000001$$
$$-1_{10} = FF_{16} = 11111111$$

$$128_{10} = 80_{16} = 10000000$$
$$-128_{10} = 80_{16} = 10000000$$

4-bit Subtractor: E = 1

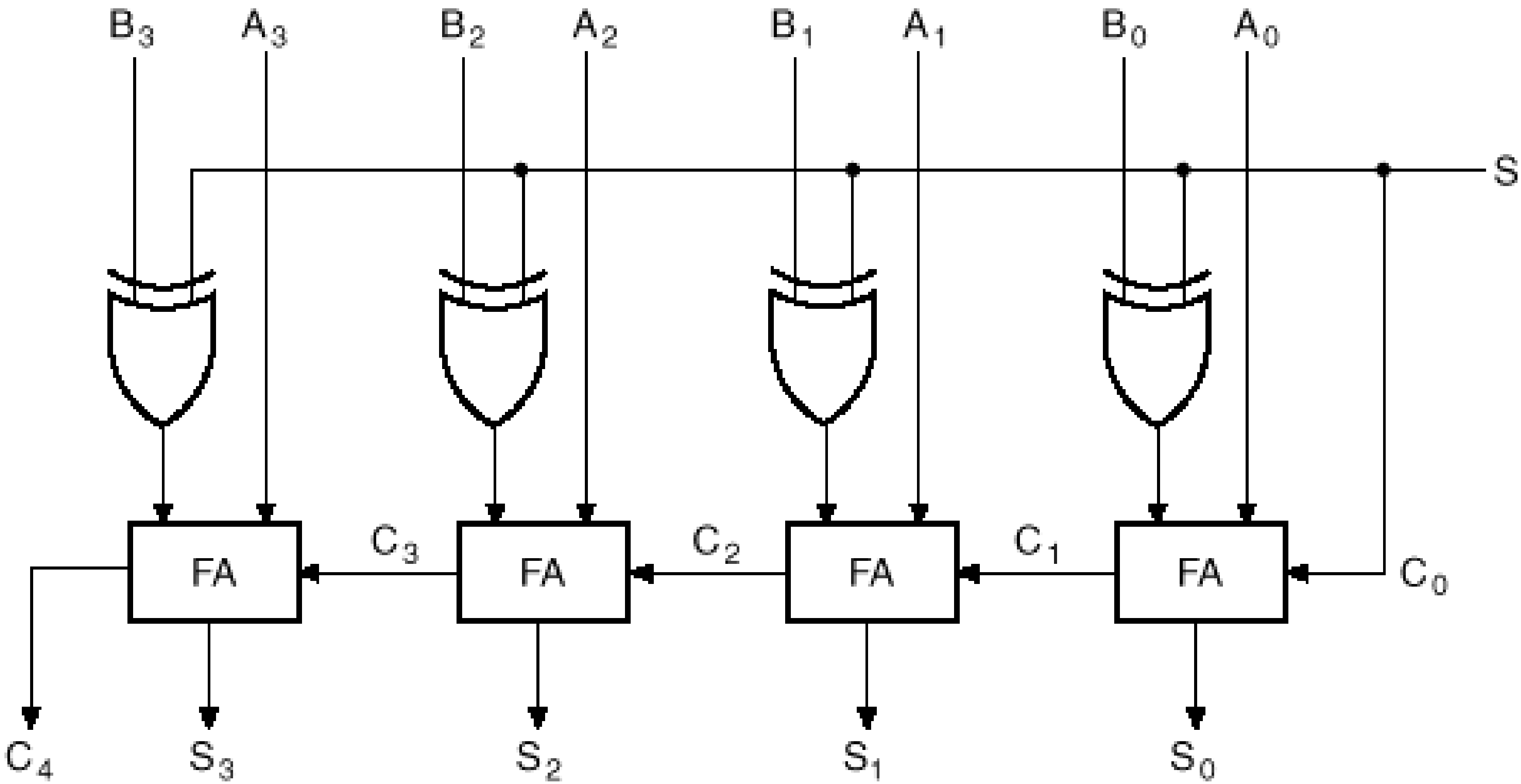


Add **A** to **B'** (one's complement) plus 1

That is, add **A** to two's complement of **B**

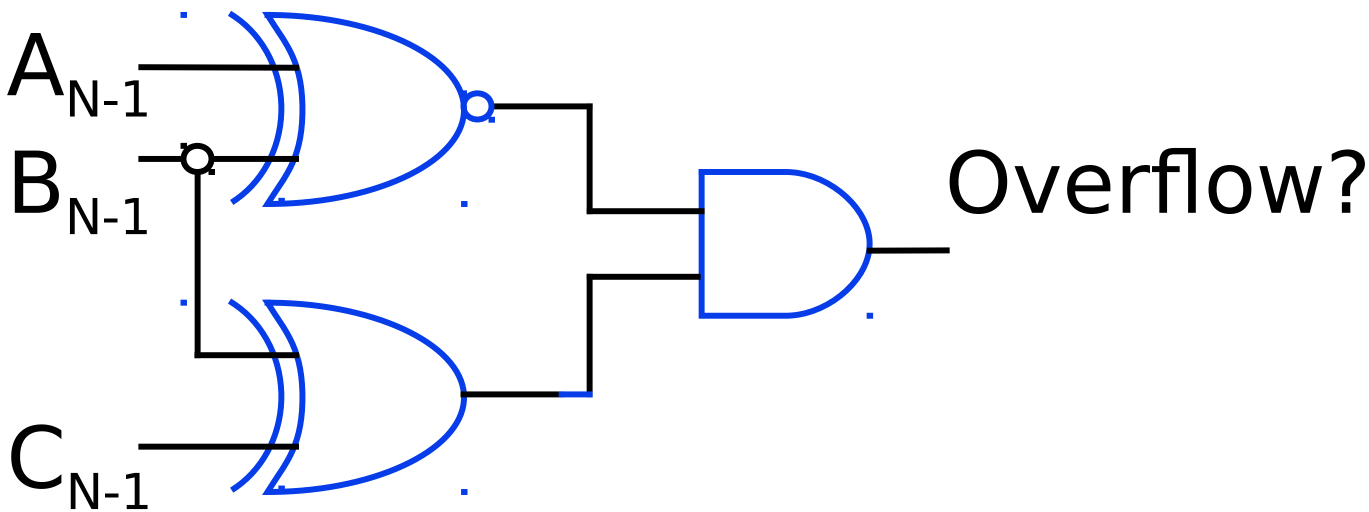
D = **A** - **B**

Adder- Subtractor Circuit



Overflow in two's complement addition

- Definition: When two values of the same signs are added:
  - Result won't fit in the number of bits provided
  - Result has the opposite sign.



Assumes an N-bit adder, with bit N-1 the MSB

Addition cases and overflow

00	01	11	10	00	11
0010	0011	1110	1101	0010	1110
0011	0110	1101	1010	1100	0100
--------	--------	--------	--------	--------	--------
0101	1001	1011	0111	1110	0010
2	3	-2	-3	2	-2
3	6	-3	-6	-4	4
5	-7	-5	7	-2	2
	OFL		OFL		

Summary

- Addition and subtraction are fundamental to computer systems
- Key – create a single bit adder/subtractor
  - Chain the single-bit hardware together to create bigger designs
- The approach is call *ripple-carry* addition
  - Can be slow for large designs
- Overflow is an important issue for computers
  - Processors often have hardware to detect overflow
- Next time: encoders/decoder.