Outline

- ❍ Last time:
  - ➜ Quiz
- ❍ This lecture:
  - ➜ Adders and subtracters: Half adder, full adder
  - ➜ Adder organizations: ripple, carry look-ahead (CLA), carry bypass
  - ➜ Full adder as a full subtracter
  - ➜ Sign-magnitude
  - ➜ One's & Two's Complement
  - ➜ BCD, Floating-point

Bits, Bytes, Nibbles and Words: Some Definitions

- ❍ Data types which are ordered lists of binary digits often have the following names:
  - ➜ Bit: 1 digit
  - ➜ Nibble: 4 digits
  - ➜ Byte: 8 digits
  - ➜ Word: The standard memory bus width in your architecture (e.g. 16-bit, 32-bit, 64-bit words).

Addition

- **Decimal Addition:**

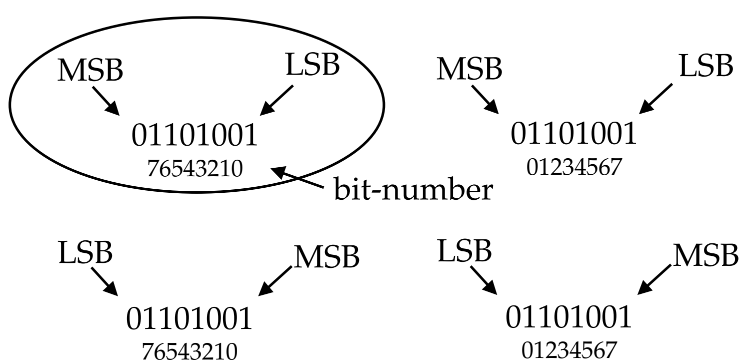
$$\begin{array}{r} 1990 \\ +\ 215 \\ \hline \end{array}$$

- **Binary Addition:**

$$\begin{array}{rcr} 10101 & \longrightarrow & 21 \\ +\ 110 & \longrightarrow & +\ 6 \\ \hline & \longleftarrow & \end{array}$$

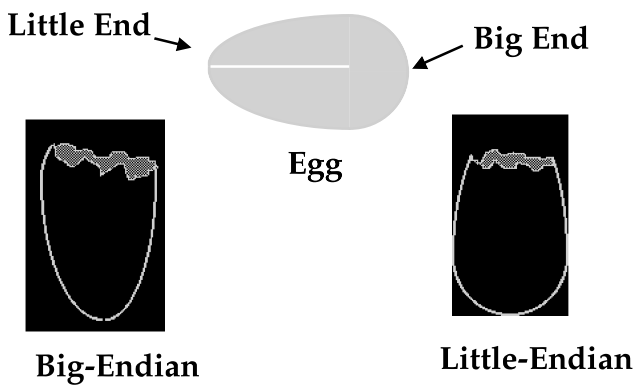
Bit-Ordering, Byte-Ordering and the Endian Wars

- The ordering and naming of digits in a number (binary or decimal) is an arbitrary choice. However, it would be useful if we all agreed on the same choice!

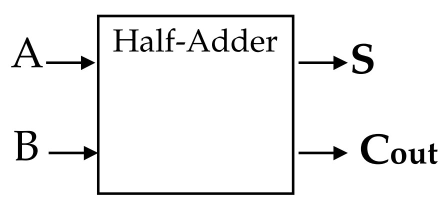


Bit-Ordering, Byte-Ordering and the Endian Wars

❍ In his travels, Gulliver came across the warring races of Big-Endians and Little-Endians. Dr. Danny Cohen (ISI) uses this situation as an analogy:



The Addition of Two Binary Digits: The Half-Adder



A	B	S	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

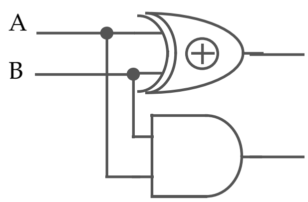
S

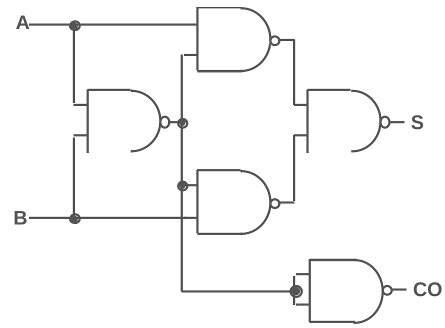
B \ A	0	1
0	0 (0)	1 (2)
1	1 (1)	0 (3)

Cout

B \ A	0	1
0	0 (0)	0 (2)
1	0 (1)	1 (3)

Half Adder Implementation

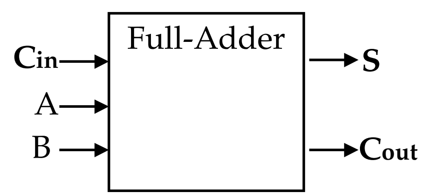


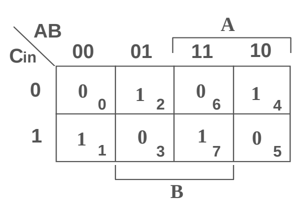


(a) Any gates (b) Two-input NANDS

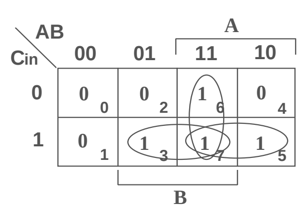
Addition of Two Binary Digits and a Carry-In: The Full Adder

Cin	A	B	S	Cout
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





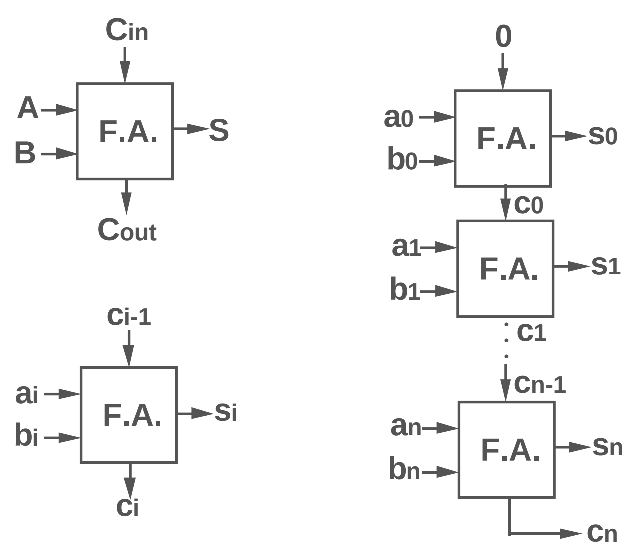
S



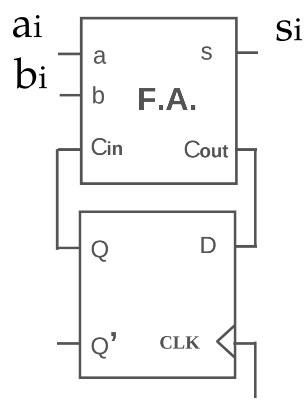
Cout

Full-Adder Implementation

The Ripple-Carry n-Bit Binary Parallel Adder



Serial Adder



Subtraction

❍ **Decimal Subtraction:**

Minuend: 1990

Subtrahend: - 215

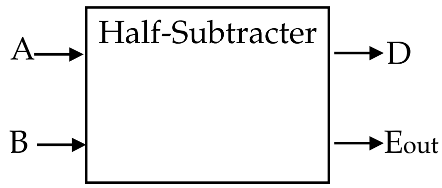
❍ **Binary Subtraction:**

10101 → 21

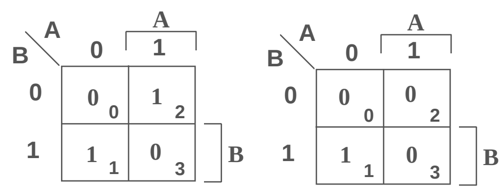
- 110 → - 6

The Subtraction of Two Binary Digits: The Half-Subtracter

Perform:
A-B



AB	D	Eout
0 0	0	0
0 1	1	1
1 0	1	0
1 1	0	0

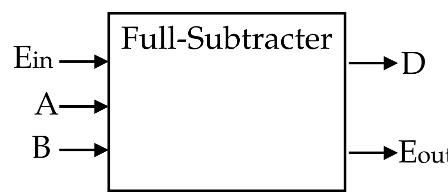


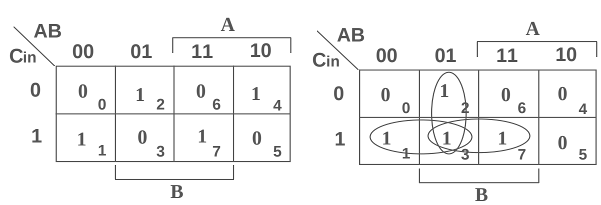
D

Eout

Subtraction of Two Binary Digits and a Borrow-In: The Full Subtracter

Ein	A	B	D	Eout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

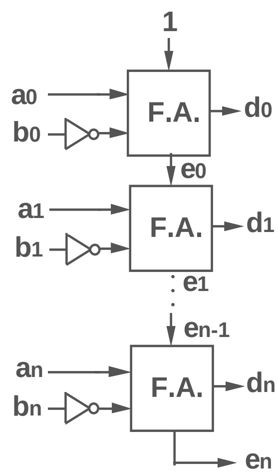




D

Eout

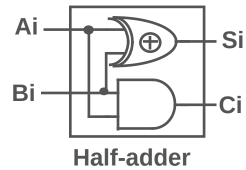
Ripple Subtracter

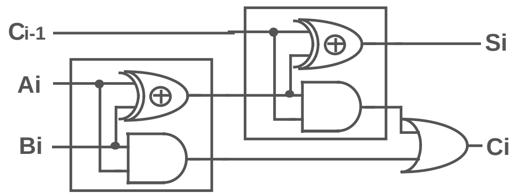


Subtraction & Number Systems

❍ But what about if the result of the subtraction is itself negative? How would we know? How do we represent negative numbers?
- ➜ Signed-magnitude number system
- ➜ Signed-1's-complement number system
- ➜ Signed-2's-complement number system

Full-Adder from Half-Adders



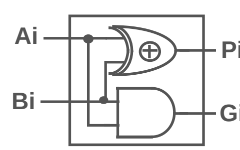


Carry Look-Ahead Addition

$$G_i = A_i B_i$$
$$P_i = A_i \oplus B_i$$

$$\Rightarrow C_i = G_i + P_i C_{i-1}$$

A	B	Cin	S	Cout	
0	0	0	0	0	No carry generated
0	0	1	1	0	No carry generated
0	1	0	1	0	Carry **P**ropagated
0	1	1	0	1	Carry **P**ropagated
1	0	0	1	0	Carry **P**ropagated
1	0	1	0	1	Carry **P**ropagated
1	1	0	0	1	Carry **G**enerated
1	1	1	1	1	Carry **G**enerated



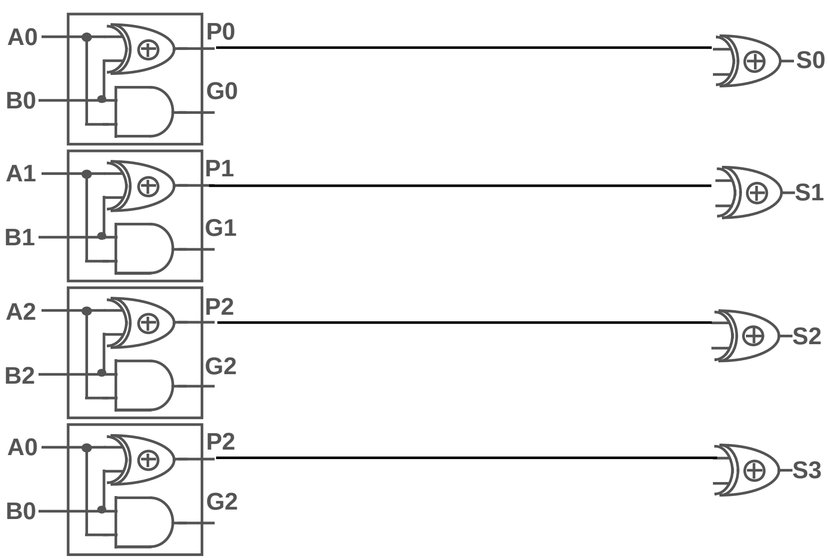
4-Bit Carry Look-Ahead Adder

$G_i = A_i\ B_i$
$P_i = A_i - B_i$

$\Rightarrow C_i = G_i + P_iC_{i-1}$

$C_0 = G_0 + P_0C_{-1} = G_0$
$C_1 = G_1 + P_1C_0 = G_1 + P_1G_0$
$C_2 = G_2 + P_2C_1 = G_2 + P_2\ (\ G_1 + P_1C_0) = G_2 + P_2\ G_1 + P_2\ P_1C_0$
$C_3 = G_3 + P_3C_2 = G_3 + P_3\ (\ G_2 + P_2\ (\ G_1 + P_1C_0\ ))$
$\quad = G_3 + P_3\ (\ \ G_2 + P_2\ G_1 + P_2\ P_1C_0\ )$
$\quad = G_3 + P_3G_2 + P_3P_2\ G_1 + P_3P_2\ P_1C_0\ )$

4-Bit Carry Look-Ahead Adder



Sign-Magnitude Representation: 4-bits

❍ Comments:
- ➜ For n-bit word, left-most bit is sign-bit, 0 for+ve number & 1 for -ve number, and remaining n-1 bits represent integer magnitude.
- ➜ Represents both + and - zero
- ➜ Logic networks to process sign-magnitude awkward to build, so not used often.

0110 =
1001 =
0000 =
1000 =
1111 =
0111 =

0110

sign bit: +0, -1

binary magnitude

Two's Complement Representation (N*): 4 bits

For n-bit positive integer N, define N* = 2^n -N

N		N*
0110	=	
0001	=	
0000	=	
0111	=	

N *		D
1111	=	
1000	=	
0000	=	
0111	=	

❍ Comments:
- ➜ For n-bit word, positive number in 2's complement form is represented by a zero followed by the binary magnitude as in sign-magnitude.
- ➜ Single representation of zero

One's Complement Representation (N'): 4 bits

For n-bit positive integer N, define N' =(2^n - 1) -N

N	N'
0110	=
0001	=
0000	=
0111	=

N '	D
1111	=
1000	=
1101	=
0000	=
0111	=

❍ Comments:
- ➜ For n-bit word, positive number in 1's complement form is represented by a zero followed by the binary magnitude as in sign-magnitude and 2's complement.
- ➜ Representation of +ve and -ve zero
- ➜ 1's complement can also be formed by compelementing all the bits in N

1's and 2's Complement

❍N* = 2^n-N = (2^n - 1 - N) +1 = N' +1

❍N = 2^n-N* and N = (2^n-1) - N'

- ➜Easy way to get N*: start at right-hand-end and complement all bits to the left of the first 1.
- ➜Concept can be extended to the "R's complement" of a number in Base R.
  - ✷Consider R=8 with n=4; the 8's complement of 03458 = 8^4 - 0345 = 74328

Signed Binary Integers (word-length =4): Sign-magnitude, 1's complement, 2's complement

decimal	all systems
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	N/A

decimal	sign & mag	2's comp. N*	1's comp. N'
-0	1000	N/A	1111
-1	1001	1111	1110
-2	1010	1110	1101
-3	1011	1101	1100
-4	1100	1100	1011
-5	1101	1011	1010
-6	1110	1010	1001
-7	1111	1001	1000
-8	N/A	1000	N/A

Addition of 2's Complement Numbers

(a) Positive numbers, sum < 2^(n-1)

+3 0011
+4 0100

+7 0111 (correct answer)

(b) Positive numbers, sum >= 2^(n-1)

+5 0101
+6 0110

1001 (wrong since overflow)

(c) Pos + Neg numbers, Neg > Pos

+5 0101
-6 1010

-1 1111

(d) Pos + Neg numbers, Pos > Neg

-5 1011
+6 0110

+1 0001 (correct if carry ignored)

(e) Negative no's. |sum| <= 2^(n-1)

-3 1101
-4 1100

-7 1001 (correct if carry ignored)

(f) Negative no's. |sum| > 2^(n-1)

-5 1011
-6 1010

0101 (wrong since overflow)

Addition of 1's Complement Numbers

❍ Same as 2's complement except that instead of discarding the last carry, it is added to the n-bit sum in the right-most position ("end-around" carry).

(a) & (b) same as 2's complement

(c) Pos + Neg numbers, Neg > Pos

```
 +5      0101
 -6      1001
 ____________
 -1      1110 (correct answer)
```

(d) Pos + Neg numbers, Pos > Neg

```
 -5      1010
 +6      0110
   (1)   0000
            1 (end-around carry)
 ____________
 +1      0001 (correct; no overflow)
```

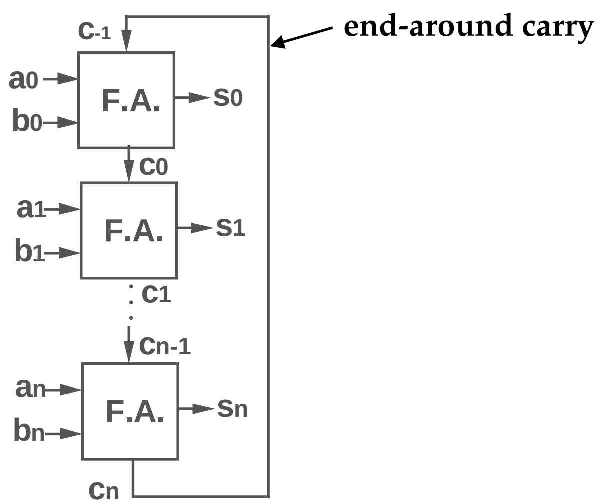
(e) Negative no's. |sum| <= 2^(n-1)

```
 -3     1100
 -4     1011
   (1)  0111
           1 (end-around carry)
 ___________
 -7     1000 (correct; no overflow)
```

(f) Negative no's. |sum| > 2^(n-1)

```
 -5     1010
 -6     1001
   (1)  0011
           1 (end-around carry)
 ___________
        0100 (wrong answer; overflow)
```


One's Complement Adder



Redundant Representations of Numbers

- Normal way of thinking of arithmetic circuit design: "Given a set of binary numbers in 1's complement, 2's complement, etc., find the fastest/smallest circuit that implements the arithmetic functions."
- Another way of asking the questions: "Given that we want the faster/smallest arithmetic circuits, how should the binary numbers be represented?"
  - ➜ Should we use log2(N) bits to represent N numbers?