Binary Arithmetic

COE 202

Digital Logic Design

Dr. Muhamed Mudawar

King Fahd University of Petroleum and Minerals

Presentation Outline

- Binary Addition
- Binary Subtraction
- Binary Multiplication
- Hexadecimal Addition and Subtraction
- BCD Addition
- Signed Numbers and Complement Notation
- Carry and Overflow

Adding Bits

- 1 + 1 = 2, but 2 should be represented as $(10)_2$ in binary
- Adding two bits: the sum is S and the carry is C

X	**0**	**0**	**1**	**1**
+ Y	**+ 0**	**+ 1**	**+ 0**	**+ 1**
C S	**0 0**	**0 1**	**0 1**	**1 0**

- Adding three bits: the sum is S and the carry is C

0	**0**	**0**	**0**	**1**	**1**	**1**	**1**
0	**0**	**1**	**1**	**0**	**0**	**1**	**1**
+ 0	**+ 1**	**+ 0**	**+ 1**	**+ 0**	**+ 1**	**+ 0**	**+ 1**
0 0	**0 1**	**0 1**	**1 0**	**0 1**	**1 0**	**1 0**	**1 1**

Binary Addition

- Start with the least significant bit (rightmost bit)
- Add each pair of bits
- Include the carry in the addition, if present

carry		**1**	**1**	**1**	**1**				
	0	0	1	1	0	1	1	0	(54)
+	0	0	0	1	1	1	0	1	(29)
	0	1	0	1	0	0	1	1	(83)
bit position:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	

Subtracting Bits

- Subtracting 2 bits (X – Y): we get the difference (D) and the **borrow-out** (B) shown as 0 or -1

X	**0**	**0**	**1**	**1**
– Y	**– 0**	**– 1**	**– 0**	**– 1**
B D	**0 0**	**-1 1**	**0 1**	**0 0**

- Subtracting two bits (X – Y) with a **borrow-in = -1**: we get the difference (D) and the **borrow-out** (B)

borrow-in	**-1**	**-1**	**-1**	**-1**	**-1**
	X	**0**	**0**	**1**	**1**
	– Y	**– 0**	**– 1**	**– 0**	**– 1**
	B D	**-1 1**	**-1 0**	**0 0**	**-1 1**

Binary Subtraction

- Start with the least significant bit (rightmost bit)
- Subtract each pair of bits
- Include the borrow in the subtraction, if present

borrow			-1	-1			-1		
	0	0	1	1	0	1	1	0	(54)
–	0	0	0	1	1	1	0	1	(29)
	0	0	0	1	1	0	0	1	(25)
bit position:	7	6	5	4	3	2	1	0	

Binary Multiplication

- Binary Multiplication table is simple:

  0×0=0,   0×1=0,   1×0=0,   1×1=1

$$
\begin{array}{lrl}
\text{Multiplicand} & & 1100_2 = 12 \\
\text{Multiplier} & \times & 1101_2 = 13 \\
\hline
 & & 1100 \\
 & & 0000 \\
 & & 1100 \\
 & & 1100 \\
\hline
\text{Product} & & 10011100_2 = 156
\end{array}
$$

Binary multiplication is easy

0 × multiplicand = 0

1 × multiplicand = multiplicand

- *n*-bit multiplicand × *n*-bit multiplier = 2*n*-bit product
- Accomplished via shifting and addition

Hexadecimal Addition

- Start with the least significant hexadecimal digits
- Let Sum = summation of two hex digits
- If Sum is greater than or equal to 16
  - Sum = Sum – 16 and Carry = 1
- Example:

```
carry          1   1       1
         9   C   3   7   2   8   6   5
  +      1   3   9   5   E   8   4   B
        ------------------------------
         A   F   C   D   1   0   B   0
```

5 + B = 5 + 11 = 16
Since Sum ≥ 16
Sum = 16 – 16 = 0
Carry = 1

Hexadecimal Subtraction

- Start with the least significant hexadecimal digits
- Let Difference = subtraction of two hex digits
- If Difference is negative
  - Difference = 16 + Difference and Borrow = -1
- Example:

borrow		**-1**		**-1**			**-1**	
	9	**C**	**3**	**7**	**2**	**8**	**6**	**5**
-	**1**	**3**	**9**	**5**	**E**	**8**	**4**	**B**
	8	**8**	**A**	**1**	**4**	**0**	**1**	**A**

Since 5 < B, Difference < 0
Difference = 16+5–11 = 10
Borrow = -1

Single Digit BCD Addition

- We use binary arithmetic to add the BCD digits

```
  1000          8
+ 0101        + 5
------        ----------
  1101         13 (>9)
```

- Since the result is more than 9, it must use 2 digits
- To correct the digit, add 6 to the digit sum

```
     1000          8
   + 0101        + 5
   ------        ------------------
     1101         13 (>9)
   + 0110        + 6 (add 6)
   ------        ------------------
   1 0011          3 and a carry
0001 0011 ←───── Final answer in BCD
```

Multiple Digit BCD Addition

Add: 2905 + 1897 in BCD

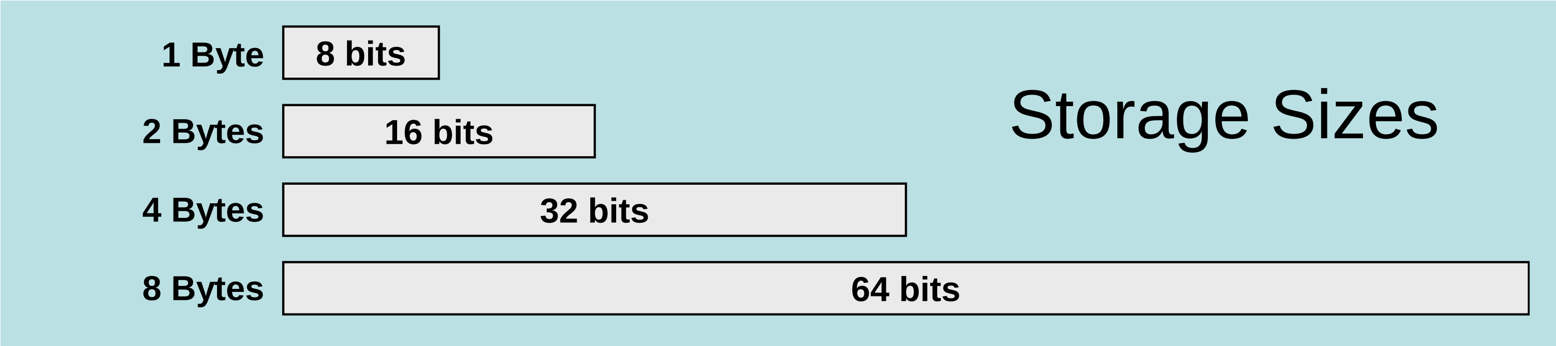
Showing carries and digit corrections

carry	+1	+1	+1	
	0010	1001	0000	0101
+	0001	1000	1001	0111
	0100	10010	1010	1100
digit correction		0110	0110	0110
	0100	1000	0000	0010

Final answer: 2905 + 1897 = 4802

Storage Sizes

- A register stores the bits of a number
- A register consists of a fixed number *n* of storage bits
- The storage size *n* can be 8 bits, 16 bits, 32 bits, or 64 bits
- The Byte size is always equal to 8 bits
- Numbers stored in registers are either unsigned or signed



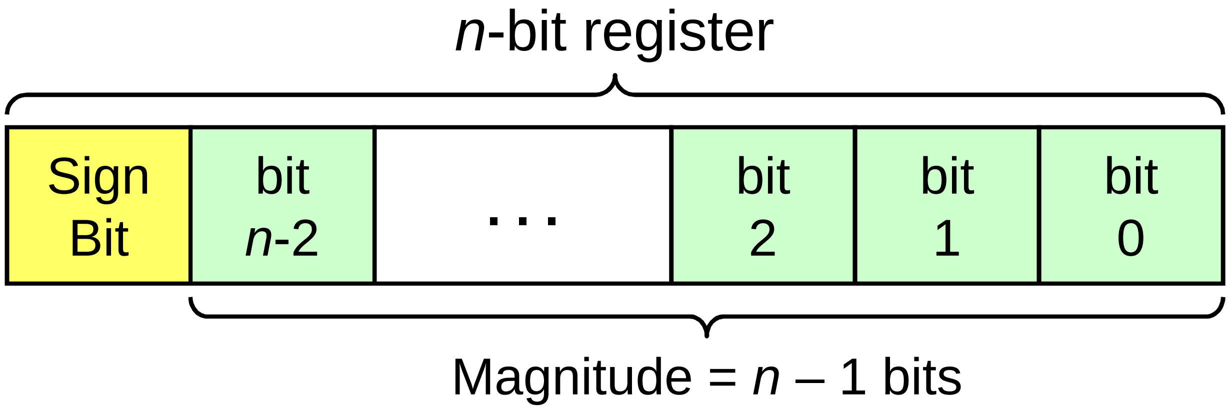
Next . . .

- Binary Addition
- Binary Subtraction
- Binary Multiplication
- Hexadecimal Addition and Subtraction
- BCD Addition
- Signed Numbers and Complement Notation
- Carry and Overflow

Signed Numbers

- Several ways to represent a signed number
  - Sign-Magnitude
  - 1's complement
  - 2's complement
- Divide the range of values into 2 equal parts
  - First part corresponds to the positive numbers (≥ 0)
  - Second part correspond to the negative numbers (< 0)
- The 2's complement representation is widely used
  - Has many advantages over other representations

Sign-Magnitude Representation



- Independent representation of the sign and magnitude
- Leftmost bit is the sign bit: 0 is positive and 1 is negative
- Using *n* bits, largest represented magnitude = $2^{n-1} - 1$

Sign-magnitude representation of +45 using 8-bit register

0	0	1	0	1	1	0	1

Sign-magnitude representation of -45 using 8-bit register

1	0	1	0	1	1	0	1

Properties of Sign-Magnitude

- Two representations for zero: +0 and -0
- Symmetric range of represented values:

  For n-bit register, range is from $-(2^{n-1} - 1)$ to $+(2^{n-1} - 1)$

  For example using 8-bit register, range is -127 to +127
- Hard to implement addition and subtraction
  - Sign and magnitude parts should be processed independently
  - Sign bit should be examined to determine addition or subtraction
  - Addition is converted into subtraction when adding numbers with different signs
  - Increases the cost of the add/subtract circuit

1's Complement Representation

- Given a binary number N

  The 1's complement of N is obtained by reversing each bit in N (0 becomes 1, and 1 becomes 0)

- Example: 1's complement of $(01101001)_2 = (10010110)_2$

- If N consists of n bits then

  1's complement of $N = (2^n - 1) - N$

- $(2^n - 1)$ is a binary number represented by n 1's

- Example: if $n = 8$ then $(2^8 - 1) = 255 = (11111111)_2$

  1's complement of $(01101001)_2 =$

  $(11111111)_2 - (01101001)_2 = (10010110)_2$

2's Complement Representation

- Almost all computers today use 2's complement to represent signed integers

- A simple definition for 2's complement:

  Given a binary number N

  The 2's complement of N = 1's complement of N + 1

- Example: 2's complement of $(01101001)_2$ =

  $(10010110)_2 + 1 = (10010111)_2$

- If N consists of n bits then

  2's complement of $N = 2^n - N$

Computing the 2's Complement

starting value	00100100_2 = +36
step1: reverse the bits (1's complement)	11011011_2
step 2: add 1 to the value from step 1	+ 1_2
sum = 2's complement representation	11011100_2 = -36

2's complement of 11011100_2 (-36) = 00100011_2 + 1 = 00100100_2 = +36

The 2's complement of the 2's complement of *N* is equal to *N*

Another way to obtain the 2's complement:

Start at the least significant 1

Leave all the 0s to its right unchanged

Complement all the bits to its left

Binary Value

= 00100$\boxed{1}$00 least significant 1

2's Complement

= 11011$\boxed{1}$00

Unsigned and Signed Value

- Positive numbers
  - Signed value = Unsigned value
- Negative numbers
  - Signed value = Unsigned value – 2^n
  - n = number of bits
- Negative weight for MSB
  - Another way to obtain the signed value is to assign a negative weight to most-significant bit

1	0	1	1	0	1	0	0
-128	64	32	16	8	4	2	1

-128 + 32 + 16 + 4 = -76

8-bit Binary value	Unsigned value	Signed value
00000000	0	0
00000001	1	+1
00000010	2	+2
. . .	. . .	. . .
01111110	126	+126
01111111	127	+127
10000000	128	-128
10000001	129	-127
. . .	. . .	. . .
11111110	254	-2
11111111	255	-1

Properties of the 2's Complement

- The 2's complement of N is the negative of N
- The sum of N and 2's complement of N must be zero

  The final carry is ignored

- Consider the 8-bit number $N = 00101100_2 = +44$

  -44 = 2's complement of $N = 11010100_2$

  $00101100_2 + 11010100_2 =$ **1** 00000000_2 (8-bit sum is 0)

  Ignore final carry

- In general: Sum of N + 2's complement of $N = 2^n$

  where 2^n is the final carry (1 followed by n 0's)

- There is only one zero: 2's complement of 0 = 0

Ranges of Unsigned/Signed Integers

For n-bit unsigned integers: Range is 0 to $(2^n - 1)$

For n-bit signed integers: Range is -2^{n-1} to $(2^{n-1} - 1)$

Positive range: 0 to $(2^{n-1} - 1)$

Negative range: -2^{n-1} to -1

Storage Size	Unsigned Range	Signed Range
8 bits (byte)	0 to $(2^8 - 1)$ = 0 to 255	-2^7 to $(2^7 - 1)$ = -128 to +127
16 bits	0 to $(2^{16} - 1)$ = 0 to 65,535	-2^{15} to $(2^{15} - 1)$ = -32,768 to +32,767
32 bits	0 to $(2^{32} - 1)$ = 0 to 4,294,967,295	-2^{31} to $(2^{31} - 1)$ = -2,147,483,648 to +2,147,483,647
64 bits	0 to $(2^{64} - 1)$ = 0 to 18,446,744,073,709,551,615	-2^{63} to $(2^{63} - 1)$ = -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807

Sign Extension

Step 1: Move the number into the lower-significant bits

Step 2: Fill all the remaining higher bits with the sign bit

- This will ensure the correctness of the signed value
- Examples
  - Sign-Extend 01100010_2 to 16 bits

    $\mathbf{01100010_2 = +98} \Rightarrow \boxed{\mathbf{00000000}}\ \mathbf{01100010 = +98}$

  - Sign-Extend 10110011_2 to 16 bits

    $\mathbf{10110011_2 = -77} \Rightarrow \boxed{\mathbf{11111111}}\ \mathbf{10110011 = -77}$

- Infinite 0's can be added to the left of a positive number
- Infinite 1's can be added to the left of a negative number

Subtraction with 2's Complement

- When subtracting A – B, convert B to its 2's complement
- Add A to (2's complement of B)

```
borrow:   -1 -1       -1           carry: 1  1        1  1
          0 1 0 0 1 1 0 1                 0 1 0 0 1 1 0 1
    –                          ⇨     +
          0 0 1 1 1 0 1 0                 1 1 0 0 0 1 1 0   (2's complement)
        _________________               _________________
          0 0 0 1 0 0 1 1                 0 0 0 1 0 0 1 1   (same result)
```

- Final carry is ignored, because
  - Negative number is sign-extended with 1's
  - You can imagine infinite 1's to the left of a negative number
  - Adding the carry to the extended 1's produces extended zeros

Carry and Overflow

- Carry is important when …
  - Adding or subtracting unsigned integers
  - Indicates that the unsigned sum is out of range
  - Either < 0 or >maximum unsigned *n*-bit value
- Overflow is important when …
  - Adding or subtracting signed integers
  - Indicates that the signed sum is out of range
- Overflow occurs when
  - Adding two positive numbers and the sum is negative
  - Adding two negative numbers and the sum is positive
  - Can happen because of the fixed number of sum bits

Carry and Overflow Examples

❖ We can have carry without overflow and vice-versa

❖ Four cases are possible (Examples are 8-bit numbers)

				1					
	0	0	0	0	1	1	1	1	15
+	0	0	0	0	1	0	0	0	8
	0	0	0	1	0	1	1	1	23

Carry = 0 Overflow = 0

1	1	1	1	1					
	0	0	0	0	1	1	1	1	15
+	1	1	1	1	1	0	0	0	248 (-8)
	0	0	0	0	0	1	1	1	7

Carry = 1 Overflow = 0

	1								
	0	1	0	0	1	1	1	1	79
+	0	1	0	0	0	0	0	0	64
	1	0	0	0	1	1	1	1	143 (-113)

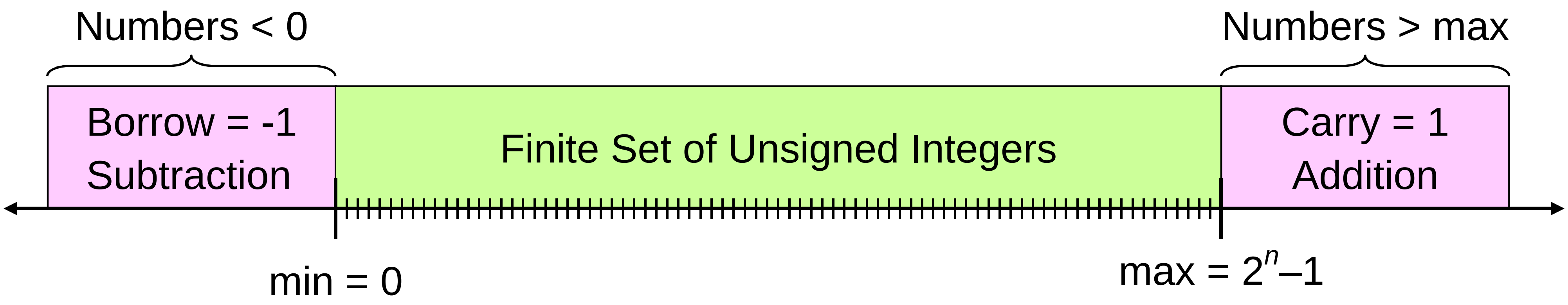
Carry = 0 Overflow = 1

1			1	1					
	1	1	0	1	1	0	1	0	218 (-38)
+	1	0	0	1	1	1	0	1	157 (-99)
	0	1	1	1	0	1	1	1	119

Carry = 1 Overflow = 1

Range, Carry, Borrow, and Overflow

- Unsigned Integers: *n*-bit representation



- Signed Integers: 2's complement representation