NAAC ACCREDITED "B++" (CGPA 2.89)

Yogoda Satsanga Mahavidyalaya

Jagannathpur, Dhurwa, Ranchi-834004

www.ysmranchi.net

ysmranchi4@gmail.com

Course :Computer System Architecture

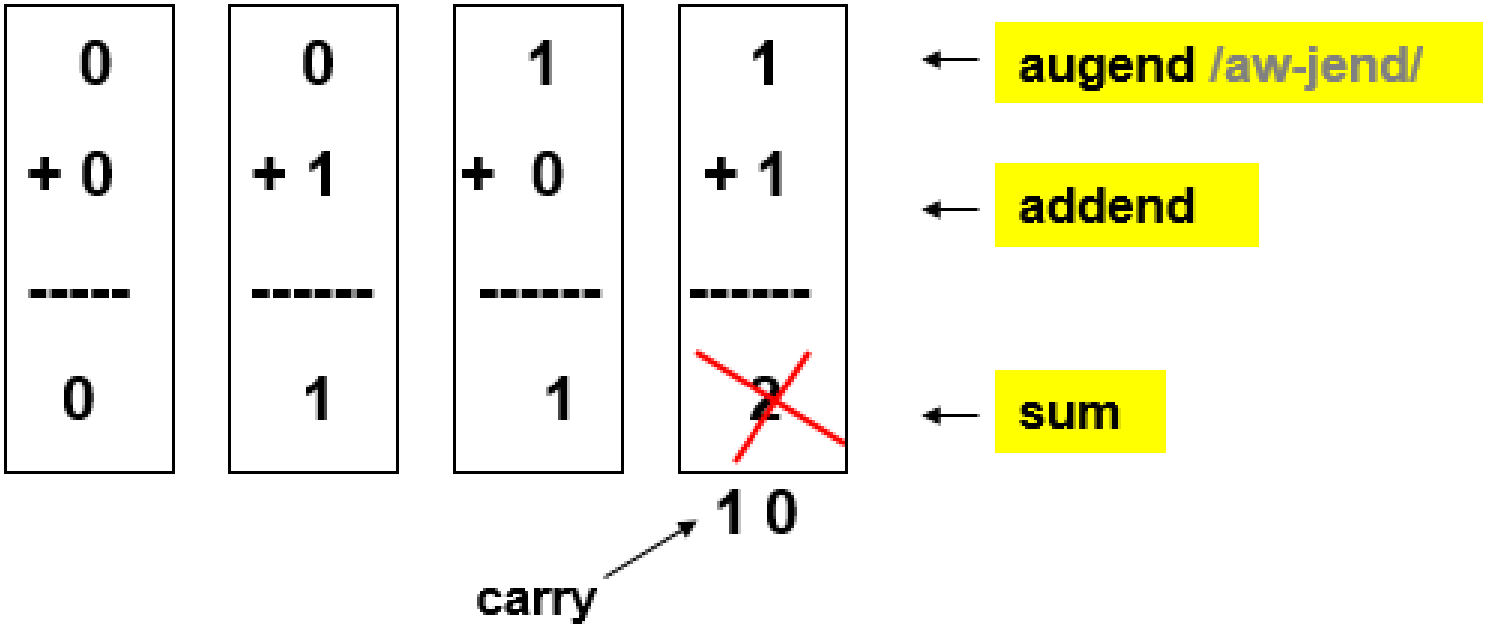
Class : Sem-1

Lesson :Number System Contd..

By :Goutam Sanyal

Binary Addition

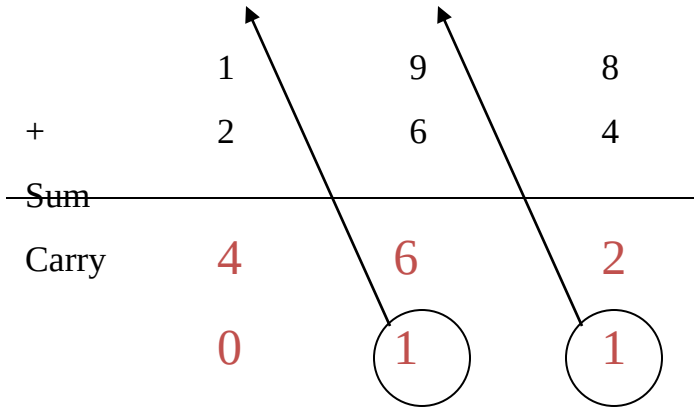
One bit addition:

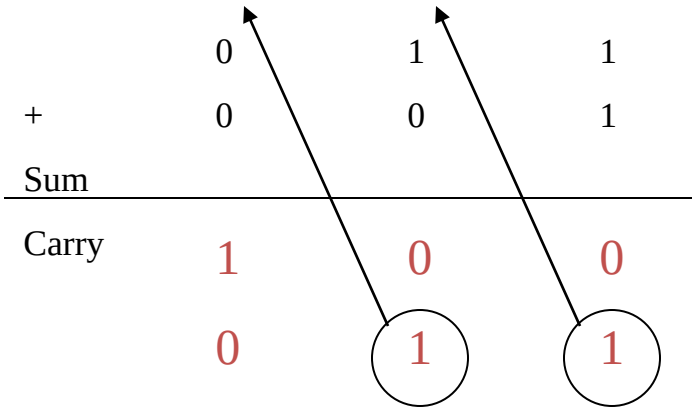


2 doesn't exist in binary!

Addition

- From right to left, we add each pair of digits
- We write the sum, and add the carry to the next column on the left





Binary Sums and Carries

a	b	Sum
0	0	0
0	1	1
1	0	1
1	1	0

a	b	Carry
0	0	0
0	1	0
1	0	0
1	1	1

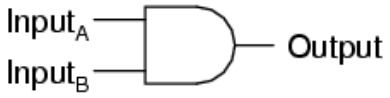
XOR

Exclusive-OR gate

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

AND

2-input AND gate



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

Binary Addition (cont.)

Example:

```
   1         1 1 1        <- carries
   1 1 0 0 0 0 1 1 1 1
+  0 1 1 1 1 0 1 0 1 0
  ---------------------
  1 0 0 1 1 1 1 1 0 0 1   <- sum
```

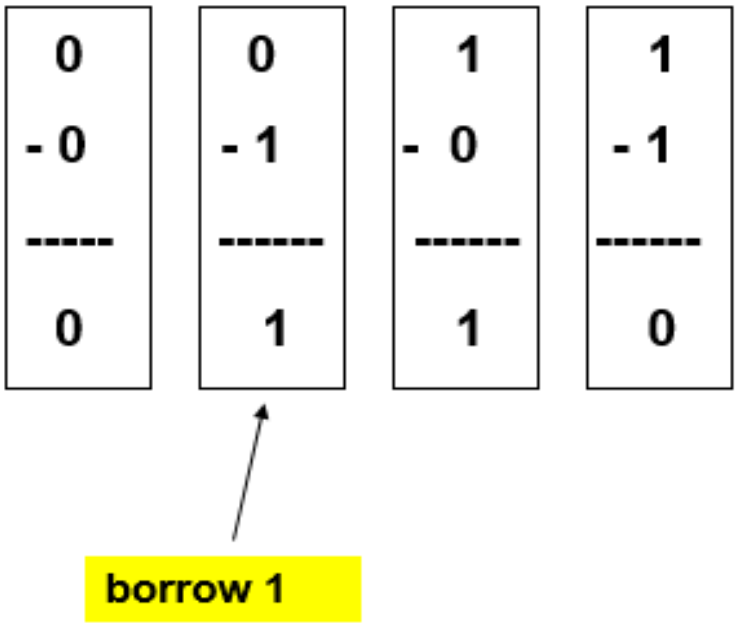
Q: How to verify?

A: Convert to decimal

```
   783
+  490
  -----
  1273
```

Binary Subtraction

One bit subtraction:



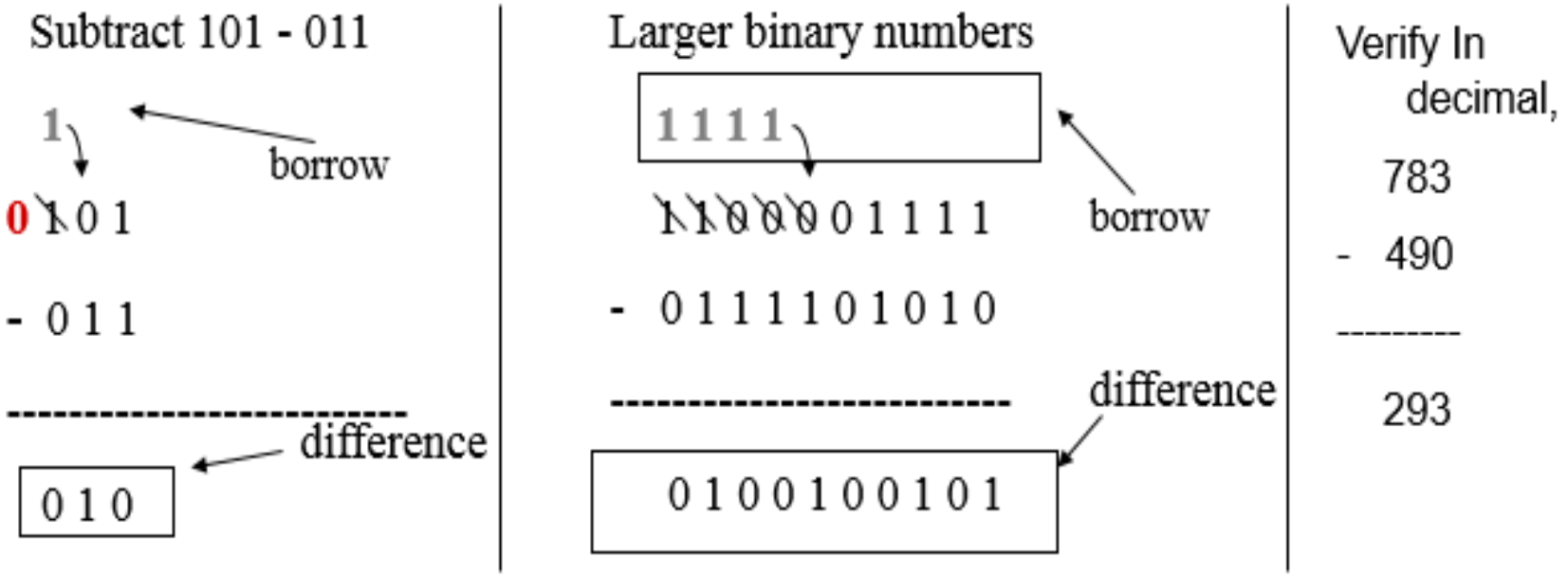
← **minuend** /men-u-end/

← **subtrahend** /sub-tra-hend/

← **difference**

- In binary addition, there is a **sum** and a **carry**.
- In binary subtraction, there is a **difference** and a **borrow**
- **Note: 0 – 1 = 1 borrow 1**

Binary Subtraction (cont.)



- In Decimal subtraction, the borrow is equal to 10.
- In Binary, the borrow is equal to 2. Therefore, a '1' borrowed in binary will generate a $(10)_2$, which equals to $(2)_{10}$ in decimal

One’s Complement

- The additive inverse of a one’s complement representation is found by inverting each bit.

- Inverting each bit is also called taking the one’s complement

Example

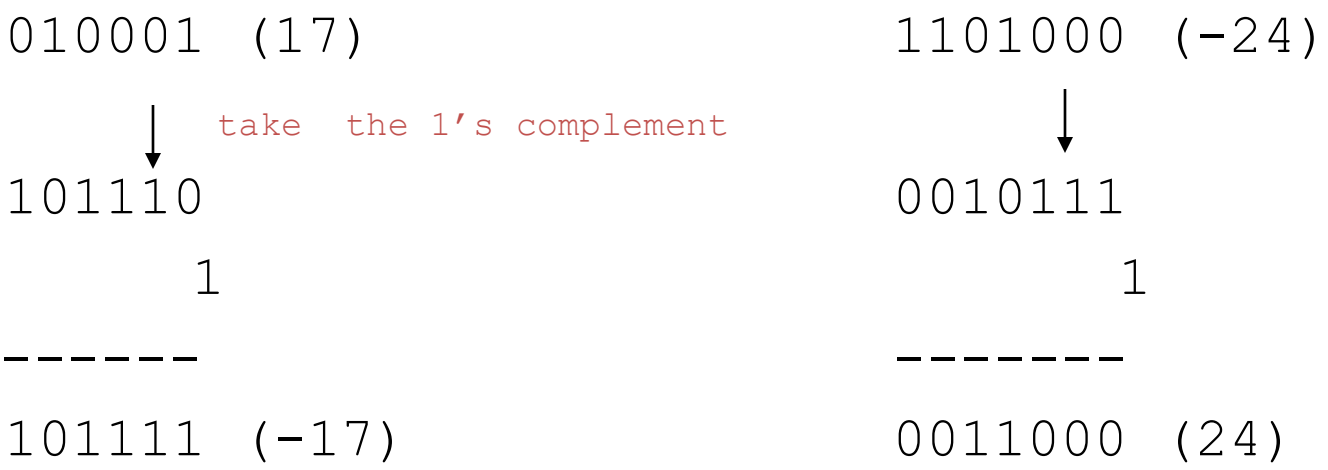
```
0000 0011 (3)
1111 1100 (-3)

1110 1000 (-23)
0001 0111 (23)

0000 0000 (0)
1111 1111 (0)
```

Note: There are two representations of zero

Two’s complement

- The additive inverse of a two’s complement integer can be obtained by adding 1 to its one’s complement

Example


010001 (17)
take the 1's complement
101110
1

101111 (-17)
1101000 (-24)
0010111
1

0011000 (24)

Subtracting the large number from the small number

Let's subtract 61 from 45:

```
  45
- 61
----
 -24
```

I just followed the usual rules for subtraction: 5-1 is 4; 4-6 is -2.
So is the answer -24? No, because if we add 61 to -24 we get

```
  61
- 24
----
  37
```

not 45. What went wrong?

Subtracting the large number from the small number

The problem is that when we got that negative sign, it meant only that the 2 in the tens place was negative; the 4 is still positive! So what we realy found was that

$$(40 + 5) - (60 + 1) = (40 - 60) + (5 - 1) = -20 + 4$$

That is not -24, but -16, which is the right answer to the problem.

So we can't subtract a larger number from a smaller one columnwise, because the sign gets mixed up.

The usual way to do this is to apply the fact that

$$a - b = -(b - a)$$

to say that

$$45 - 61 = -(61 - 45) = -16$$

Subtracting the large number from the small number

You would do the same thing in binary (and these are in fact the same numbers):

```
  101101     111101
- 111101   - 101101
--------   --------
        = -(010000)
```

That is, you reverse the order of the numbers, subtract, and take the negative.

Group Activity

$$\begin{array}{rccccc} & 1 & 0 & 0 & 0 & 1 \\ + & 1 & 1 & 1 & 0 & 1 \\ \hline \end{array}$$

$$\begin{array}{rccccc} & 1 & 0 & 1 & 1 & 1 \\ + & 1 & 1 & 0 & 1 & 0 & 1 \\ \hline \end{array}$$

- $10001 + 11101 = 101110$:

$$\begin{array}{rccccc} & 1 & & & & 1 \\ & 1 & 0 & 0 & 0 & 1 \\ + & 1 & 1 & 1 & 0 & 1 \\ \hline 1 & 0 & 1 & 1 & 1 & 0 \end{array}$$

- $10111 + 110101 = 1001100$:

$$\begin{array}{rccccccc} & 1 & 1 & & 1 & 1 & 1 & \\ & & 1 & 0 & 1 & 1 & 1 \\ + & 1 & 1 & 0 & 1 & 0 & 1 \\ \hline 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{array}$$

Group Activity

$$\begin{array}{cccccccc} & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ - & & & 1 & 0 & 0 & 1 & 0 \\ \hline \end{array}$$

- $1011011 - 10010 = 1001001$:

$$\begin{array}{cccccccc} & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ - & & & 1 & 0 & 0 & 1 & 0 \\ \hline & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{array}$$

How To Create A Negative Number

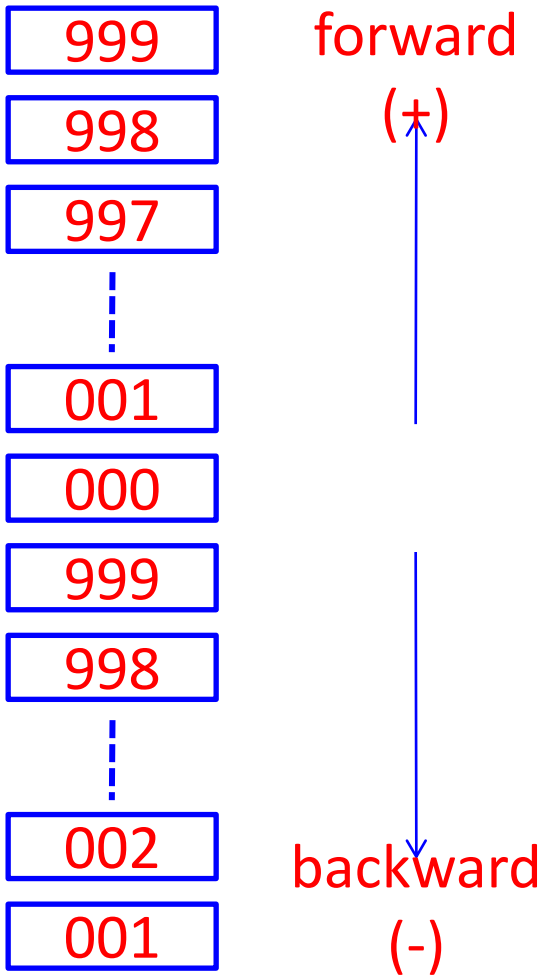
- In digital electronics you cannot simply put a minus sign in front of a number to make it negative.
- You must represent a negative number in a *fixed-length* binary number system. All signed arithmetic must be performed in a *fixed-length* number system.
- A physical *fixed-length* device (usually memory) contains a fixed number of bits (usually 4-bits, 8-bits, 16-bits) to hold the number.

3-Digit Decimal Number System

A bicycle odometer with only three digits is an example of a fixed-length decimal number system.

The problem is that without a negative sign, you cannot tell a +998 from a -2 (also a 998). Did you ride forward for 998 miles or backward for 2 miles?

Note: Car odometers do not work this way.



Negative Decimal

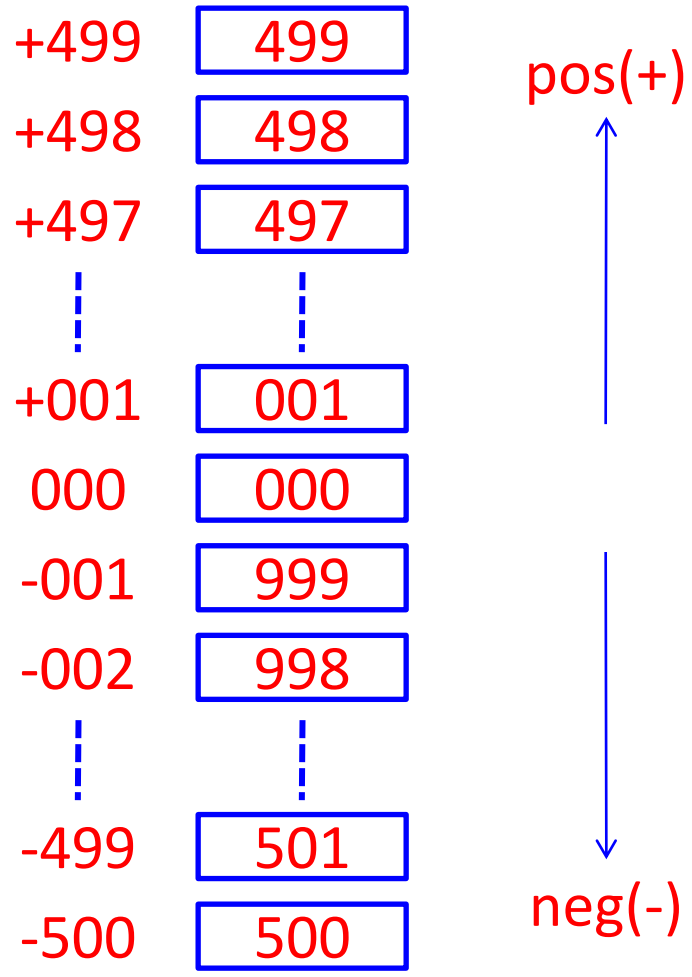
How do we represent negative numbers in this 3-digit decimal number system without using a sign?

→Cut the number system in half.

→Use 001 – 499 to indicate positive numbers.

→Use 500 – 999 to indicate negative numbers.

→Notice that 000 is not positive or negative.



“Odometer” Math Examples

3 + 2 = 5	003 + 002 = 005
6 + (-3) = 3	006 + 997 = 1]003

Disregard Overflow

(-5) + 2 = (-3)	995 + 002 = 997
(-2) + (-3) = (-5)	998 + 997 = 1]995

Disregard Overflow

It Works!

Complex Problems

- The previous examples demonstrate that this process works, but how do we *easily* convert a number into its negative equivalent?
- In the examples, converting the negative numbers into the 3-digit decimal number system was fairly easy. To convert the (-3), you simply counted backward from 1000 (i.e., 999, 998, 997).
- This process is not as easy for large numbers (e.g., -214 is 786). How did we determine this?
- To convert a large negative number, you can use the 10's Complement Process.

10’s Complement Process

The **10’s Complement** process uses base-10 (decimal) numbers. Later, when we’re working with base-2 (binary) numbers, you will see that the **2’s Complement** process works in the same way.

First, complement all of the digits in a number.

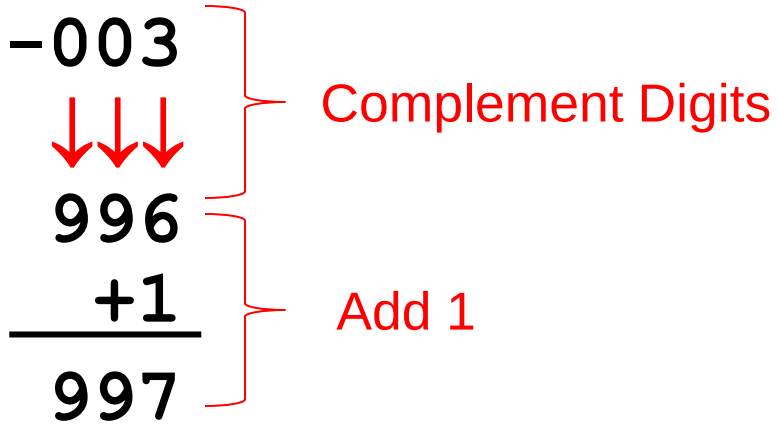
- A digit’s complement is the number you add to the digit to make it equal to the largest digit in the base (i.e., 9 for decimal). The complement of 0 is 9, 1 is 8, 2 is 7, etc.

Second, add 1.

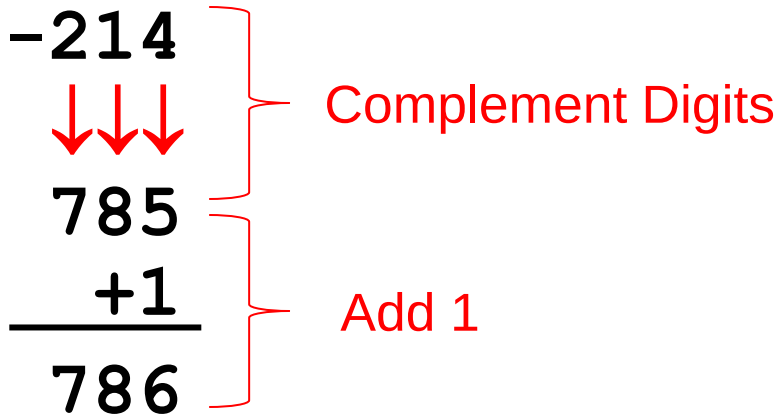
- Without this step, our number system would have two zeroes (+0 & -0), which no number system has.

10’s Complement Examples

Example #1



Example #2



Signed Integers

- So far we have treated all of our numbers as unsigned (or positive only)
  - To implement signed integers (or signed fractions), we need a mechanism to denote the sign itself (positive or negative)
    - Unfortunately, this introduces new problems, so we will see 3 different approaches, all of which add a special bit known as the *sign bit*
  - If the sign bit is 0, the number is positive
  - If the sign bit is 1, the number is negative
    - If we have an 8 bit number, does this mean that we now need 9 bits to store it with one bit used exclusively for the sign?

Signed Magnitude

- The first signed integer format is signed magnitude where we add a bit to the front of our numbers that represents the sign
  - In 4 bits, 3 = 0011 and –3 = 1011
    - Notice in 4 bits, we can store 16 numbers in unsigned magnitude (0000 to 1111, or decimal 0 to 15) but in signed magnitude we can only store 15 numbers (between –7, or 1111, and +7, 0111), so we lose a number
  - Two problems:
    - 0 is now represented in two ways: 0000, 1000, so we lose the ability to store an extra number since we have two 0s
    - We cannot do ordinary arithmetic operations using signed magnitude
      - we have to "strip" off the sign bit, perform the operation, and insert the sign bit on the new answer – this requires extra hardware

One's Complement

- An alternative approach to signed magnitude is one's complement where the first bit is again a sign bit
- But negative numbers are stored differently from positive numbers
  - Positive number stored as usual
  - Negative number – all bits are inverted
    - 0s become 1s, 1s become 0s
  - Example: +19 in 6 bits = 010011, -19 = 101100
    - The first bit is not only the sign bit, but also part of the number
  - Notice that we still have two ways to represent 0, 000000 and 111111
    - So, we won't use one's complement

Two's Complement

- Positive numbers remain the same
- Negative numbers: derived by flipping each bit and then adding 1 to the result
  - +19 in 6 bits = 010011,
  - -19 in 6 bits = 101101
    - 010011 ➔ 101100 +1 ➔ 101101
  - To convert back, flip all bits and add 1
    - 101101 ➔ 010010 + 1 ➔ 010011
  - While this is harder, it has two advantages
    - Only 1 way to represent 0 (000000) so we can store 1 extra value that we lost when we tried signed magnitude and one's complement
    - Arithmetic operations do not require "peeling" off the sign bit

4-bit Two's Complement	
Binary	Decimal
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

Some Examples

Represent 83 and –83 using 8 bits in all 3 signed representations:

+83 = 64 + 16 + 2 + 1 = 01010011 (in all 3 representations)

-83:
Signed magnitude = 11010011 (sign bit is 1 for negative)
One's complement = 10101100 (flip all bits from +83)
Two's complement = 10101101 (flip all bits from +83 and add 1)

Convert 11110010 into a decimal integer in all 4 representations
Unsigned magnitude = 128 + 64 + 32 + 16 + 2 = 242
Signed magnitude = -114 (negative, 1110010 = 114)
One's complement = -13 (leading bit = 1, the number is negative,
flip all bits ➔00001101 = 13)
Two's complement = -14 (negative, so flip all bits and add 1 ➔
00001101 + 1 = 00001110 = 14)

Addition

- This operation is much like decimal addition except that you are only adding 1s and 0s
  - Add each column as you would in decimal, write down the sum and if the sum > 1, carry a 1 to the next column
  - Four possibilities:
    - Sum of the two digits (and any carry in) = 0, write 0, carry 0
    - Sum = 1, write 1, carry 0
    - Sum = 2, write 0, carry 1 (this represents 10 = 2)
    - Sum = 3, write 1, carry 1 (this represents 11 = 3)

Examples:

1 + 1 = 2, write 0, carry 1

```
  01000101
+ 00001111
  01010100
```

```
  11111111
+ 10101010
 110101001
```

The carry out of this last bit causes *overflow*

Subtraction

- There are two ways we could perform subtraction
  - As normal, we subtract from right to left with borrows now being 2 instead of 10 as we move from one column to the next
  - Or, we can negate the second number and add them together (36 – 19 = 36 + -19)
    - We will use the latter approach when implementing a subtraction circuit as it uses the same circuit as addition

Examples:

borrow 2 from the previous column

```
 11010100
- 00110011
 10100001
```

```
 11010100 ➔  11010100
- 00110011   + 11001101
            110100001
```

Notice the overflow in this case too, but it differs from the last example because we are using two's complement

Overflow Rules

- In unsigned magnitude addition
  - a carry out of the left-most bit is also an overflow
- In unsigned magnitude subtraction
  - overflow will occur in subtraction if we must borrow prior to the left-most bit
- In two's complement addition/subtraction
  - if the two numbers have the same sign bit and the sum/difference has a different sign bit, then overflow

Below we see examples of four *signed* additions

Expression	Result	Carry?	Overflow?	Correct Result?
0100(+4)+0010(+2)	0110(+6)	No	No	Yes
0100(+4)+0110(+6)	1010(-6)	No	Yes	No
1100(-4)+1110(-2)	1010(-6)	Yes	No	Yes
1100(-4)+1010(-6)	0110(+6)	Yes	Yes	No

2’s Complement Arithmetic

This presentation will demonstrate

- That subtracting one number from another is the same as making one number negative and adding.
- How to create negative numbers in the binary number system.
- The 2’s Complement Process.
- How the 2’s complement process can be use to add (and subtract) binary numbers.

Negative Numbers?

- Digital electronics requires frequent addition and subtraction of numbers. You know how to design an adder, but what about a subtract-er?
- A subtract-er is not needed with the 2's complement process. The 2's complement process allows you to easily convert a positive number into its negative equivalent.
- Since subtracting one number from another is the same as making one number negative and adding, the need for a subtract-er circuit has been eliminated.

8-Bit Binary Number System

Apply what you have learned to the binary number systems. How do you represent negative numbers in this 8-bit binary system?

→Cut the number system in half.

→Use 00000001 – 01111111 to indicate positive numbers.

→Use 10000000 – 11111111 to indicate negative numbers.

→Notice that 00000000 is not positive or negative.

127	01111111
126	01111110
125	01111101
0	00000000
-1	11111111
-2	11111110
-3	11111101
-4	11111100
-5	11111011
-127	10000001
-128	10000000

Sign Bit

- What did do you notice about the most significant bit of the binary numbers?
- The MSB is (0) for all positive numbers.
- The MSB is (1) for all negative numbers.
- The MSB is called the sign bit.
- In a signed number system, this allows you to instantly determine whether a number is positive or negative.

127	01111111
126	01111110
125	01111101
0	00000000
-1	11111111
-2	11111110
-3	11111101
-4	11111100
-5	11111011
-127	10000001
-128	10000000

2’S Complement Process

The steps in the **2’s Complement** process are similar to the 10’s Complement process. However, you will now use the base two.

First, complement all of the digits in a number.

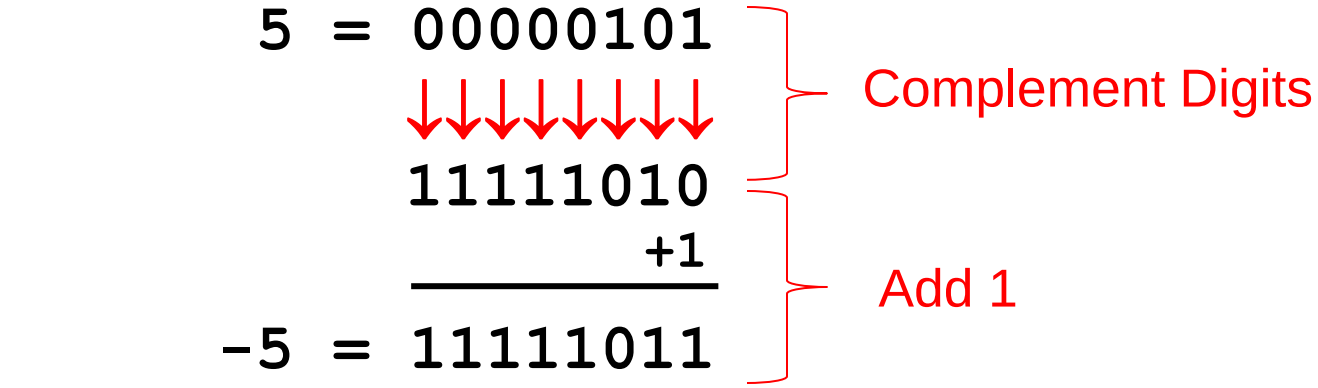
- A digit’s complement is the number you add to the digit to make it equal to the largest digit in the base (i.e., 1 for binary). In binary language, the complement of 0 is 1, and the complement of 1 is 0.

Second, add 1.

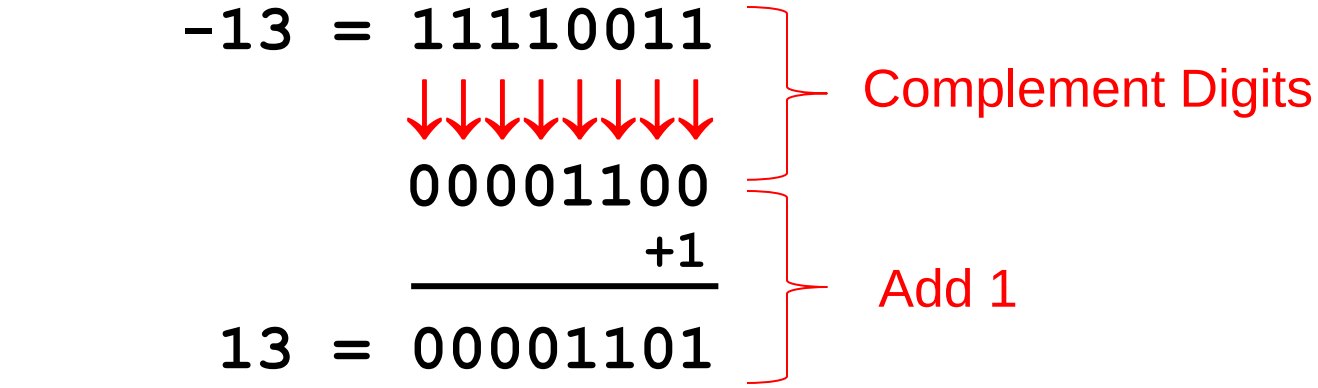
- Without this step, our number system would have two zeroes (+0 & -0), which no number system has.

2's Complement Examples

Example #1



Example #2



Using The 2's Compliment Process

Use the 2's complement process to add together the following numbers.

POS		9		NEG		(-9)
+ POS	⇒	+ 5		+ POS	⇒	+ 5
POS		14		NEG		- 4

POS		9		NEG		(-9)
+ NEG	⇒	+ (-5)		+ NEG	⇒	+ (-5)
POS		4		NEG		- 14

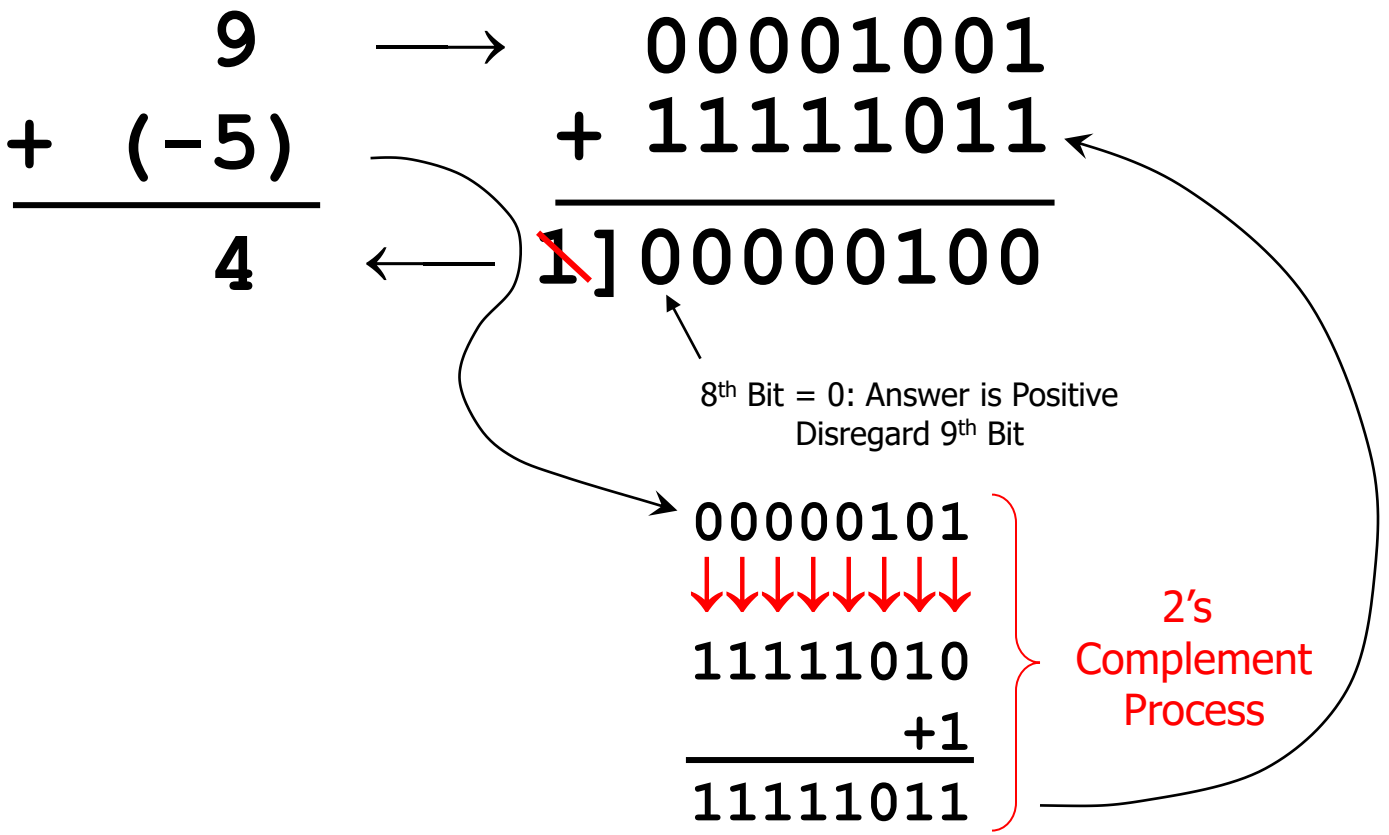
POS + POS → POS Answer

If no 2's complement is needed, use regular binary addition.

```
    9  ⟶    00001001
+   5  ⟶  + 00000101
-------    --------
   14  ⟵    00001110
```

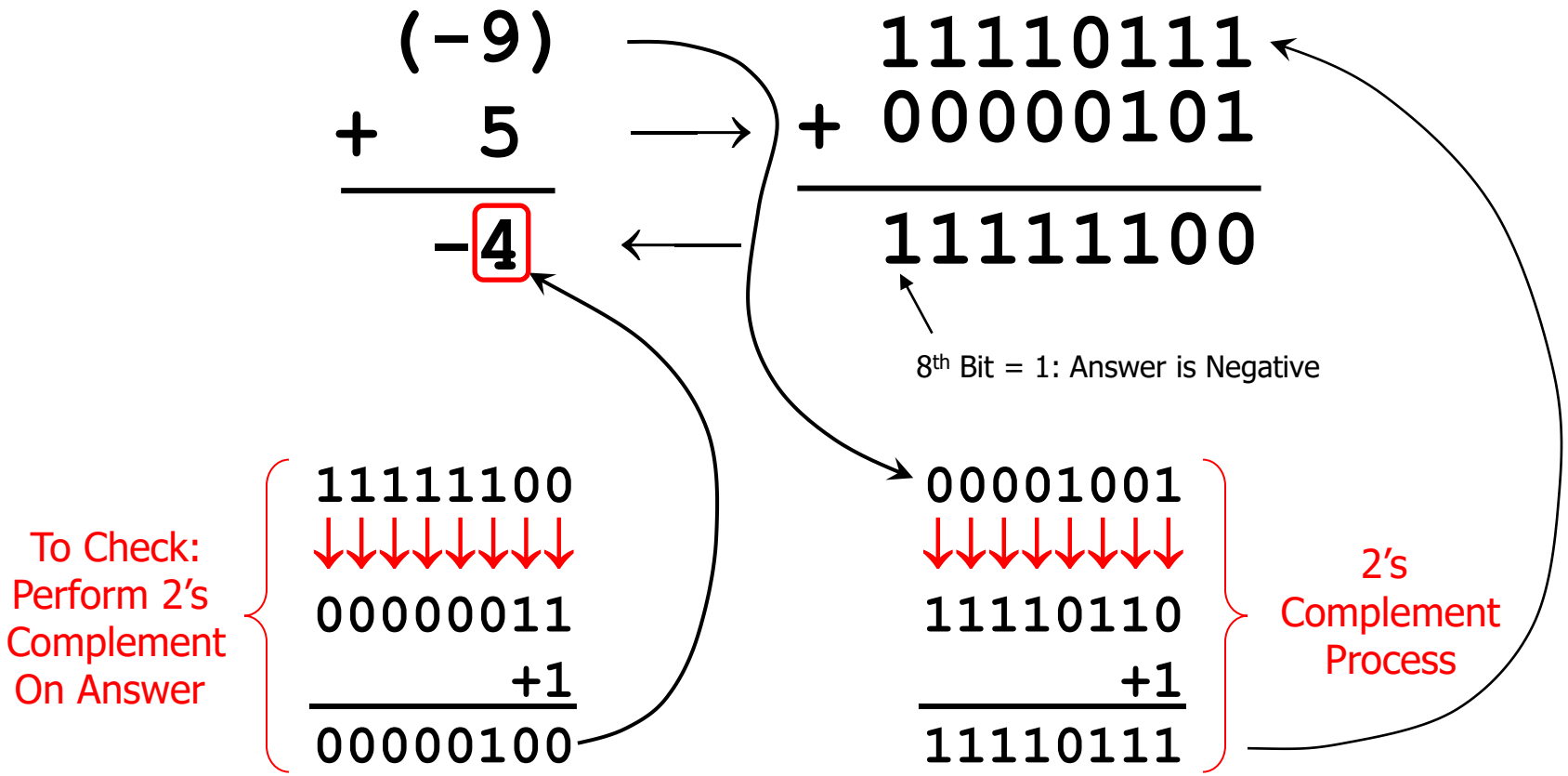
POS + NEG → POS Answer

Take the 2's complement of the negative number and use regular binary addition.



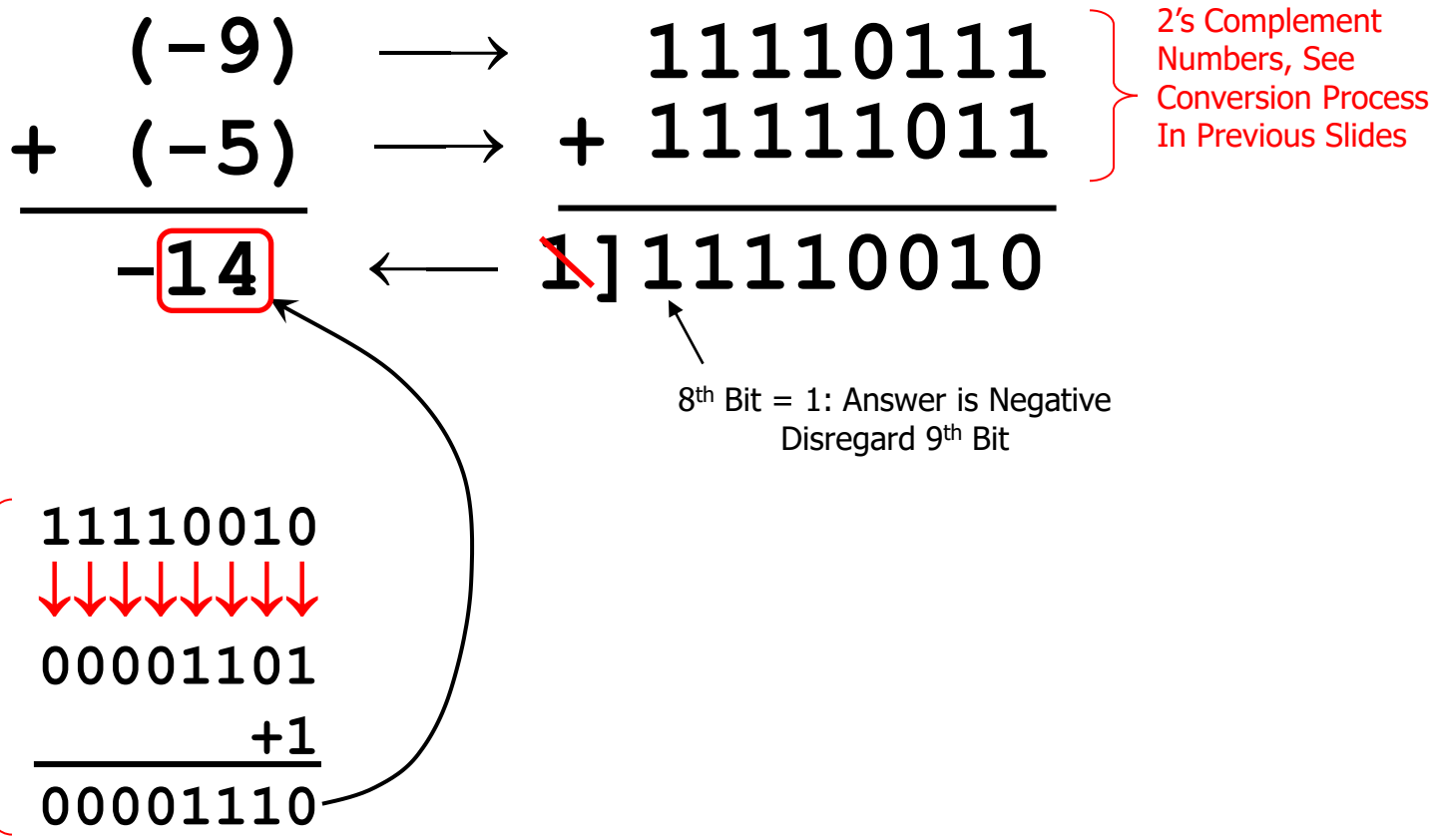
POS + NEG → NEG Answer

Take the 2's complement of the negative number and use regular binary addition.



NEG + NEG → NEG Answer

Take the 2's complement of both negative numbers and use regular binary addition.



Summary