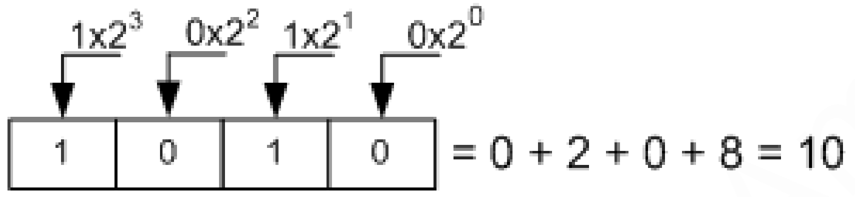
Number Systems

- **Binary-To-Decimal Conversion:**



- **Decimal-To-Binary Conversion:**

Suppose you want to convert a decimal number 25 into a corresponding binary number, then express 25 as a sum of powers of 2 and then write 1s and 0s in the appropriate digit positions:

$$25 = 16 + 8 + 0 + 0 + 1 \longleftarrow 11001$$

Binary					Decimal
0	0	0	0		0
0	0	0	1		1
0	0	1	0		2
0	0	1	1		3
0	1	0	0		4
0	1	0	1		5
0	1	1	0		6
0	1	1	1		7
1	0	0	0		8
1	0	0	1		9
1	0	1	0		10
1	0	1	1		11
1	1	0	0		12
1	1	0	1		13
1	1	1	0		14
1	1	1	1		15

Binary to Decimal conversion Table

Decimal to Binary equivalences

- Whenever a binary number has all 1s , then we can find its decimal equivalent by using the following formulae:

$$\text{Decimal} = 2^B - 1$$

where B is the number of bits.

												1	1	2 bits	2^2-1 = 3
										1	1	1	1	4 bits	2^4-1 = 15
					1	1	1	1		1	1	1	1	8 bits	2^8-1 = 255
1	1	1	1		1	1	1	1		1	1	1	1	12 bits	2^{12}-1 = 4095

Hexadecimal Numbers

- ❑ **Hexadecimal numbers** are much shorter than binary numbers and easy to write and remember. The hexadecimal number system consists of : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F and as shown in the table:

- ❑ **Hexadecimal to Binary Conversion**

To convert a hexadecimal number to a binary number, convert each hexadecimal number to its 4-bit equivalent using the table. For example, convert a hexadecimal number 8CF to binary;

$$\begin{tikzcd}
8 \arrow[d] & C \arrow[d] & F \arrow[d] \\
1000 & 1100 & 1111
\end{tikzcd}$$

- ❑ **Binary to Hexadecimal Conversion**

Again, use the table to do the conversion. For example convert a binary 1010 1110 to hexadecimal;

$$\begin{tikzcd}
1010 \arrow[d] & 1110 \arrow[d] \\
A & E
\end{tikzcd}$$

Binary				Hexadecimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	A (10)
1	0	1	1	B (11)
1	1	0	0	C (12)
1	1	0	1	D (13)
1	1	1	0	E (14)
1	1	1	1	F (15)

The hexadecimal number system has a base of 16.

Example

- A computer memory contains binary instructions and data as shown below. The memory has 65,536 addresses, each storing a byte. Convert these bytes to their hexadecimal equivalents.

16-bit Address	D7	D6	D5	D4	D3	D2	D1	D0
0000 0000 0000 0000	0	0	0	0	0	0	0	0
0000 0000 0000 0001	1	1	0	0	1	0	0	0
----------	------------------							
0000 0001 1001 1101	0	0	0	0	0	1	1	0
-----------	-------------------							
1100 1111 1101 0001	0	0	0	0	1	0	0	1
------------	--------------------							
1110 1110 1011 0000	0	1	0	1	1	0	1	1
---------------	---------------------							
1111 1111 1111 1110	1	0	1	1	0	1	1	0
1111 1111 1111 1111	1	1	1	0	1	0	1	0

Memory Content

- Answer:

Address in Hexadecimal	Memory content in Hexadecimal
0000	00
0001	C8
019D	06
CFD1	09
EED0	5B
FFFE	B6
FFFF	EA

Binary Arithmetic

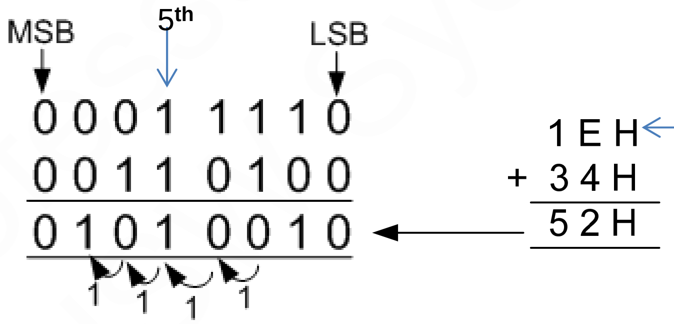
- **Binary Addition:** Four basic cases of binary addition are given below:

$$\begin{array}{r} 0 \\ +\,0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 0 \\ +\,1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\,0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\,1 \\ \hline 1\,0 \end{array}$$

Carry =1

- **8-Bit Arithmetic:** Using the above four basic rules, we can add column by column to find the sum of two binary number, regardless of how long they may be.

- Add the following 8- bit numbers and show the same numbers in hexadecimal notation.



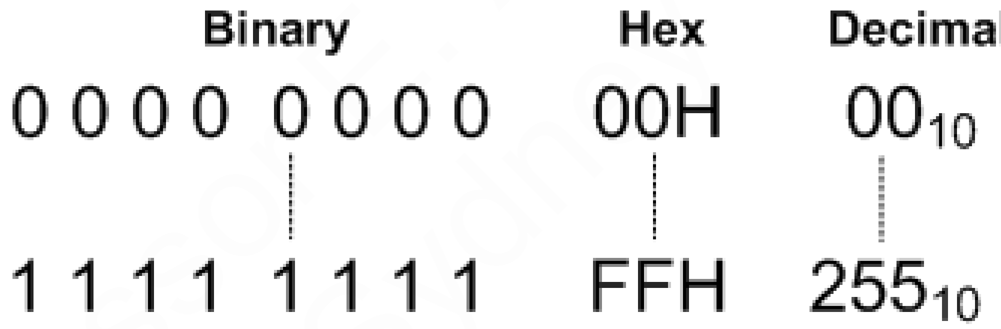
The letter H is used to signify Hexadecimal numbers

In the fifth column, 1 + 1 = 10 + 1 (carry) = 11

Unsigned Binary Numbers

- **Addition of unsigned numbers:** In applications where all data are either positive or negative, we can ignore the signs and concentrate on the absolute value of the numbers.

- For example, the smallest 8 - bit number is 0000 0000 (00H) and the largest is 1111 1111 (FFH) and this is equivalent to a decimal 0 to 255 respectively.



Binary	Hex	Decimal
0 0 0 0 0 0 0 0	00H	00_{10}
⋮	⋮	⋮
1 1 1 1 1 1 1 1	FFH	255_{10}

Unsigned Binary Numbers

- **Overflow:** In 8-bit arithmetic, if two unsigned numbers are added and if the sum is greater than 255, an overflow will occur, ie. A carry into the 9th column will happen and this will cause the 8-bit sum to be invalid. See examples below:

Example 1

```
MSB            LSB
 ↓               ↓
 1 0 0 1  1 1 1 0     158₁₀
 0 1 0 1  0 1 0 0      84₁₀
 ----------------
 1 1 1 1  0 0 1 0     242₁₀
 ----------------
```

The sum is between 0 and 255, hence no overflow

Example 2

```
                MSB            LSB
                 ↓               ↓
                 1 0 0 1  1 1 1 0     158₁₀
                 0 1 1 0  1 0 0 0     100₁₀
                 ----------------
Overflow ——→  1  0 0 0 0  0 1 1 0       6₁₀
                 ----------------
```

- ❖ **With 8-bit arithmetic, only lower 8-bits are used and it gives the sum as 6 as opposed to 258. the 8-bit answer is wrong.**
- ❖ **If the overflow is taken into account, the answer is correct, but then we are no longer using 8-bit arithmetic.**

1’s and 2’s Complement Representation

- **2’s Complement representation** is used to represent the positive and negative numbers in binary arithmetic.

- **1’s Complement** of a binary number is obtained when we complement (i.e invert) each bit of the number. Examples of 1’s complements are:

$$\begin{array}{rl} & 1\,0\,1\,0 \\ \text{1'complement} \rightarrow & 0\,1\,0\,1 \end{array} \quad \text{Invert} \qquad \begin{array}{rl} & 0000\;0000 \\ \text{1'complement} \rightarrow & 1111\;1111 \end{array}$$

- **2’Complement** of a binary number is obtained when we add 1 to the 1’s complement of the number.

2’complement = 1’s complement +1

- Find the 2’s complement of $1000\ 1001_2$

$$\begin{array}{rr} & 1000\;1001 \\ \text{1'complement} \rightarrow & 0111\;0110 \\ \text{Add 1 to the LSB} & 1 \\ \hline \text{2'complement} \rightarrow & 0111\;0111 \end{array}$$

Positive and Negative Numbers

- The table shows how positive and negative numbers are represented in 2's complement form.
- The MSB is the sign bit (see table): 0 represents a positive number (blue) and 1 represents a negative (red) number.
- The negative numbers in the table are the 2's complements of the positive numbers (except for the last entry)
- 2's complement of a positive binary number corresponds to negative binary number

```
                        0 1 0 1  =  +5 (see table)
1'complement ──►        1 0 1 0
Add 1 to the LSB              1
                       ─────────
2'complement ──►        1 0 1 1  =  -5 (see table)
```

After taking the 2's complement of 0101 (= +5) , we get 1011, which represents -5.

In 2's complement representation, positive numbers always have the MSB zero and the negative numbers always have the MSB of 1

Decimal numbers represented as 2's complements

MSB			LSB	Decimal
0	1	1	1	+7
0	1	1	0	+6
0	1	0	1	+5
0	1	0	0	+4
0	0	1	1	+3
0	0	1	0	+2
0	0	0	1	+1
0	0	0	0	0
1	1	1	1	-1
1	1	1	0	-2
1	1	0	1	-3
1	1	0	0	-4
1	0	1	1	-5
1	0	1	0	-6
1	0	0	1	-7
1	0	0	0	-8

Note: when using 4-bit binary number +7 is the largest positive number and -8 largest negative number

2’Complement Arithmetic - Addition

- **Addition:** As long as positive and negative numbers are expressed in 2’s complement representation, the result of an addition is always correct (assuming that the digital sum is within the negative to positive range)

CASE1: Add two positive numbers +82 and +15

```
  0 1 0 1  0 0 1 0        +  82₁₀
  0 0 0 0  1 1 1 1        +  15₁₀
  ----------------          ------
  0 1 1 0  0 0 0 1  <-----    97₁₀
```

CASE2: Add a positive and a negative numbers +124 and -67

```
  0 1 1 1  1 1 0 0        +  124₁₀
  1 0 1 1  1 1 0 1        -   67₁₀
  ----------------          -------
1 0 0 1 1  1 0 0 1  <-----    57₁₀
```

Disregard the final carry, because it is meaningless in 8-bit 2's complement arithmetic

CASE3: Add two negative numbers -44 and -77

```
  1 1 0 1  0 1 1 0        -  44₁₀
  1 0 1 1  0 0 0 1        -  77₁₀
  ----------------          ------
1 1 0 0 0  0 1 1 1  <---  - 121₁₀
```

We ignore the final carry into the ninth column

NOTE: In all three cases the results (sum) are correct. As long as the decimal sum is within the +127 to -128 range (for an 8-bit 2’s complement representation) the addition works. If the added sum is outside this range, we will get an overflow.

2’Complement Arithmetic - Subtraction

- **Subtraction:** The format for subtraction is that the Subtrahend (bottom number) is taken away from the Minuend (top number).
- In 2’s complement representation, we can use an adder to do subtraction.
- We know that adding a negative number is equivalent to subtracting a positive number
- If we take the 2’s complement of the subtrahend and add that to the minuend, we get the correct answer.
- Perform 8-bit subtraction of the numbers +82 and +15;

Procedure: First take the 2’s complement of +15 to obtain -15 and then add +82 and -15 to get the result

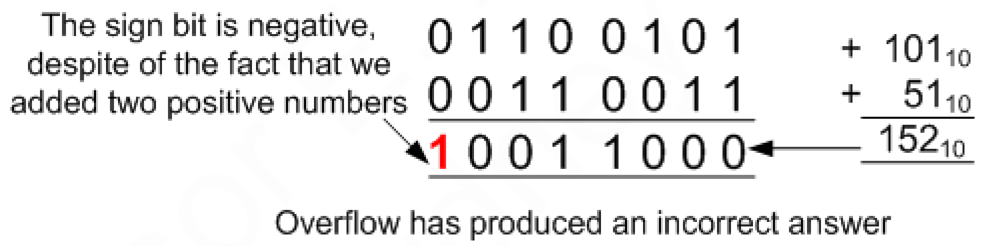
$+82_{10} = 0\,1\,0\,1\;\;0\,0\,1\,0_2$ and $+15_{10} = 0\,0\,0\,0\;\;1\,1\,1\,1_2$

$-15_{10} = 1\,1\,1\,1\;\;0\,0\,0\,1_2$ (take 2’s complement of +15)

$$\begin{array}{rl|l} & 0\,1\,0\,1\;\;0\,0\,1\,0 & +\;82_{10} \\ & 1\,1\,1\,1\;\;0\,0\,0\,1 & +\,(-15)_{10} \\ \hline \underset{\text{Ignore}}{1} & 0\,1\,0\,0\;\;0\,0\,1\,1 \leftarrow & 67_{10} \end{array}$$

2’Complement Arithmetic - Overflow

- **Overflow:** In 8-bit arithmetic, if the sum is outside the range of -128 to +127, there is an overflow into the sign bit causing a sign change.
- Overflow can arise only when two positive numbers or two negative numbers are added.
- **Case1:** Suppose the numbers being added are +101 and +51 and the sum is +152, so an overflow occurs into the MSB position causing the sign bit of the answer to change.

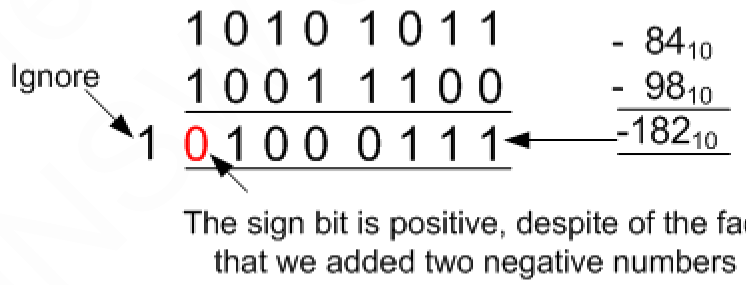


The sign bit is negative, despite of the fact that we added two positive numbers

```
  0 1 1 0  0 1 0 1      + 101₁₀
  0 0 1 1  0 0 1 1      +  51₁₀
  1 0 0 1  1 0 0 0  <--   152₁₀
```

Overflow has produced an incorrect answer

- **Case 2:** When the numbers -84 and -98 are added an overflow will occur.



```
         1 0 1 0  1 0 1 1      - 84₁₀
Ignore   1 0 0 1  1 1 0 0      - 98₁₀
       1 0 1 0 0  0 1 1 1  <-- -182₁₀
```

The sign bit is positive, despite of the fact that we added two negative numbers

We must test for an overflow after each addition and subtraction by comparing the sign bits of the two numbers being added or subtracted.

Chapter 1b: Exercise1

- What are the decimal equivalents for each of the following binary numbers?

  (a) 010101_2 (b) 11101.011_2 (c)$1100\ 1000\ 0111\ 0011_2$

- Convert the following decimal numbers to binary numbers:

  (a) 28_{10} (b) 105_{10} (c) 108.354_{10}

- What are the hexadecimal numbers that follow each of the following?

  (a) FECD (b)105B (c) 7FFF

- Convert the following hexadecimal numbers to binary numbers:

  (a) F6 (b) BAD (c) 8AF5

- Convert the binary numbers into hexadecimal numbers:

  (a) $1100\ 1000_2$ (b) $0110\ \ 0111_2$ (c) $1111\ 1110\ \ 1101\ \ 1001_2$

- Convert the hexadecimal number EF59 to its decimal equivalent.

- What is the hexadecimal equivalent of decimal 65302_{10}?

Chapter 1b: Exercise 2

- What is the binary sum of $0101\ 1111_2 + 1000\ 1011_2$?
- Show the binary addition of 756_{10} and 637_{10} using 16-bit numbers?
- Which of the following additions produces an overflow with 8-bit unsigned arithmetic?

  (a) $58_{10} + 78_{10}$ (b) $CF_{16} + 68_{16}$
- Convert each of the following to a 2's complement representation?

  (a) $+78_{10}$ (b) -25_{10} (c) -122_{10}
- Perform an 8-bit addition of the following decimal numbers in 2's complement representation?

  (a) +39 , +62 (b) +67, -99, (c) -25, -40
- Perform an 8-bit subtraction of the following decimal numbers in 2's complement representation?

  (a) -39 , -62 (b) +67, -43, (c) +25, +40