Overflow

for Addition Operation

House:

Computers:

Registers

↳ Store Data ✓

n bits + n bits

Data Data

⇓

Sometimes ✓ Correct Result

needs n+1 bits. ∋ overflow

Unsigned numbers :⇒ 0 to 15

4 bits Register :

$R_1 = 4$
$R_2 = 6$

4+6

No OF

$R_1 = 10$
$R_2 = 13$

$R_1 + R_2 = 23$

Overflow (OF)

Cannot Put in 4 bits

Overflow: Depends on

- Representation of Data
  - Unsigned
  - Signed
    - SM
    - 1's Comp
    - 2's Comp
- "Operation"
  - Addition;
  - Subtraction;
  - Multiplication;
  - ---

Unsigned Numbers : (4 bits)

Range : 0 to 15

$R_1 = 10$
$R_2 = 2$

} overflow ? ⇒ Depend on the operation !

Unsigned Numbers ; (4 bits)

Range : 0 to 15

$R_1 = 10$
$R_2 = 2$
} overflow ?

Operation	of
Add	X
multiplication	
Exponentiation	

10+6 overflow

Gate Syllabus: for Overflow:

Operation: "Addition"

Representation: { Unsigned
Signed [1. SM
2. 1's Comp
3. 2's Comp]

Overflow Definition :

a , b ⇒ n-bits ; Some Operation #

a # b = final Correct Answer Cannot be Represented in n-bits. ⇒ Overflow

4-bits: $R_1 = 4$, $R_2 = 5$; Addition

Unsigned	Signed (2's Comp)
Range: 0 to 15	Range: −8 to +7
$R_1 + R_2$ ⇒ No	$R_1 + R_2$ =
↓	↓
9 ⇒ OF	9 OF

$R_1 = -4$; $R_2 = -4$ ⇒ Signed Numbers (4 bits)

→ Addition

1's Comp	2's Comp
Range : −7 to +7	Range : −8 to +7
$R_1 + R_2$	$R_1 + R_2$
↓	↓
−8 ⇒ OF	−8 ⇒ No OF

Representation of Negative Numbers

Up to this point we have been working with unsigned positive numbers. The most common methods for representing both positive and negative numbers are sign and magnitude, 2's complement, and 1's complement. In each of these methods, the leftmost bit of a number is 0 for positive numbers and 1 for negative numbers. As discussed below, if n bits are used to represent numbers, then the sign and magnitude and 1's complement methods represent numbers in the range $-(2^{(n-1)} - 1)$ to $+(2^{(n-1)} - 1)$ and both have two representations for 0, a positive 0 and a negative 0. In 2's complement, numbers in the range $-2^{(n-1)}$ to $+(2^{(n-1)} - 1)$ are represented and there is only a positive 0. If an operation, such as addition or subtraction, is performed on two numbers and the result is outside the range of representation, then we say that an *overflow* has occurred.

In order to obtain a correct answer, we must ensure that the result has a sufficient number of bits to accommodate the sum. If we start with two n-bit numbers and the sum occupies $n + 1$ bits, we say that an overflow occurs. When one performs the addition with paper and pencil, an overflow is not a problem, because we are not limited by the width of the page. We just add another 0 to a positive number or another 1 to a negative number in the most significant position to extend the number to $n + 1$ bits and then perform the addition. Overflow is a problem in computers because the number of bits that hold a number is finite, and a result that exceeds the finite value by 1 cannot be accommodated.

The complement form of representing negative numbers is unfamiliar to those used to the signed-magnitude system. To determine the value of a negative number in signed-2's complement, it is necessary to convert the number to a positive number to place it in a more familiar form. For example, the signed binary number 11111001 is negative because the leftmost bit is 1. Its 2's complement is 00000111, which is the binary equivalent of +7. We therefore recognize the original negative number to be equal to −7.

overflow

big problem

What went wrong?

The fault was quickly identified as a software bug in the rocket's Inertial Reference System. The rocket used this system to determine whether it was pointing up or down, which is formally known as the horizontal bias, or informally as a BH value. This value was represented by a 64-bit floating variable, which was perfectly adequate.

However, problems began to occur when the software attempted to stuff this 64-bit variable, which can represent billions of potential values, into a 16-bit integer, which can only represent 65,535 potential values. For the first few seconds of flight, the rocket's acceleration was low, so the conversion between these two values was successful. However, as the rocket's velocity increased, the 64-bit variable exceeded 65k, and became too large to fit in a 16-bit variable. It was at this point that the processor encountered an operand error, and populated the BH variable with a diagnostic value.

We will study Overflow only for "Addition" operation.

Next Topic : Overflow

In case of Unsigned Addition

Overflow Definition : Always Same ✓

Overflow Detection Condition ⇒ Depend on Operation ⇒ Addition ✓ & Representation

1 bit

Range: 0 to 1

0 + 0 = 0

0 + 1 = 1

0 + 1 = 1

1 + 1 = 10

Need two bits

No of

No of

No of

of L

Unsigned Addition :

Representation Operation

Of

Detection

Conditi

final Carry ⟺ overflow

||

Carry out of MSB

3. Adding Unsigned Numbers

1. Let's first solve the problem for addition of **one-bit** quntities:

```
0 + 0 =  0
0 + 1 =  1
1 + 0 =  1
1 + 1 = 10
```

   - The last line indicates that we have a *carry output*.
   - That is, **one-bit** quantity cannot accommodate **(1 + 1)**.
   - Therefore, larger data type is required for **(1 + 1)** to succeed.

2. When **multi-bit** unsigned quantities are added, overflow occurrs if there is a **carry out** from the leftmost (*most significant*) bit.

Ex: 2 bits: → Range: 0 to 3

OF Definition

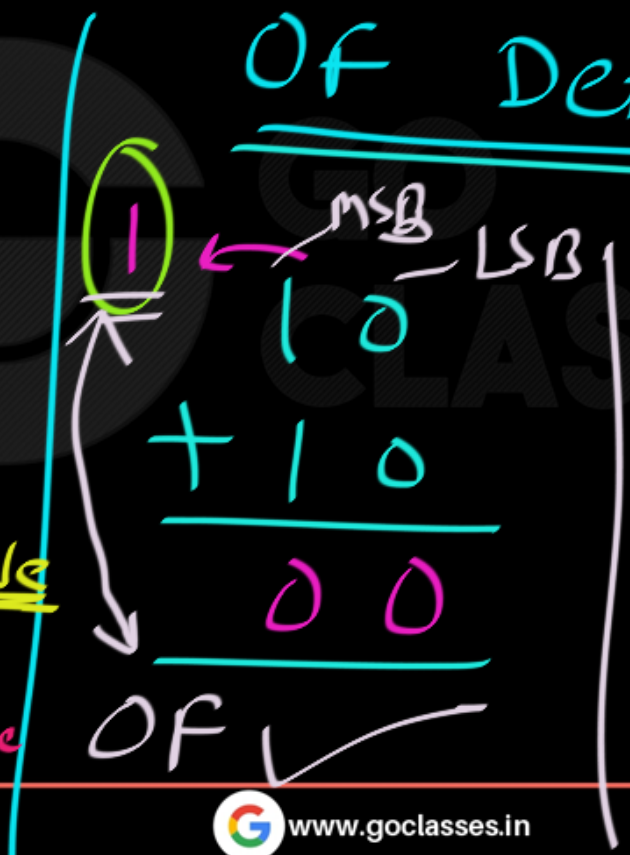
2+2 = OF ✓

2+1 = OF ✗

(-3)+(-4) Nonsense

(+3)+(-1) Nonsense

OF Detection Condition



Result: unsigned Numbers :

$A = a_3\ a_2\ a_1\ a_0$

$+\quad B = b_3\ b_2\ b_1\ b_0$

$r_4\ r_3\ r_2\ r_1\ r_0$

Carry out of MSB

OF occurs iff $r_4 = 1$

Theorem: Addition of two N-bit unsigned numbers $A = a_{N-1}a_{N-2}...a_1a_0$ and $B = b_{N-1}b_{N-2}...b_1b_0$ to produce sum $C = A + B = c_Nc_{N-1}c_{N-2}...c_1c_0$, overflows if and only if the carry out c_N of the addition is a 1 bit.

Next Topic : Overflow

Signed Numbers Addition

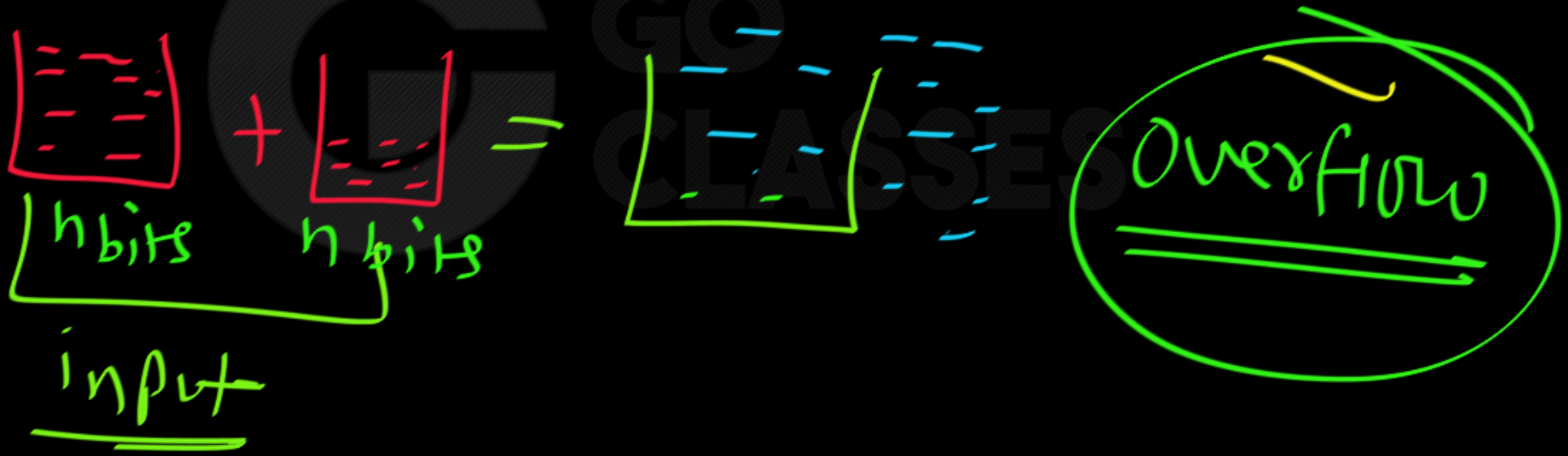
Range: $[-, a, -, 0, +, +, b, +, \ldots -]$

Signes Addition: OF can NEVER occur if numbers have opposite sign.

Overflow

- If two numbers have different signs, their sum will never overflow.
- If they have the same sign, they might overflow.


Nonsense


n bits
n bits
input
Overflow

Next Topic : Overflow

Signed Numbers Addition

1. Sign Magnitude Addition

4 bits : Range : -7 to $+7$

Addition

(1) $(+5)$, $(+3)$ of ✓

(2) ~~$(+5)$, $(+10)$~~ Nonsense

(3) (-7), $(+5)$ of can NEVER occur

4 bits: Range: −7 to +7

Addition

① +3, +4 No of

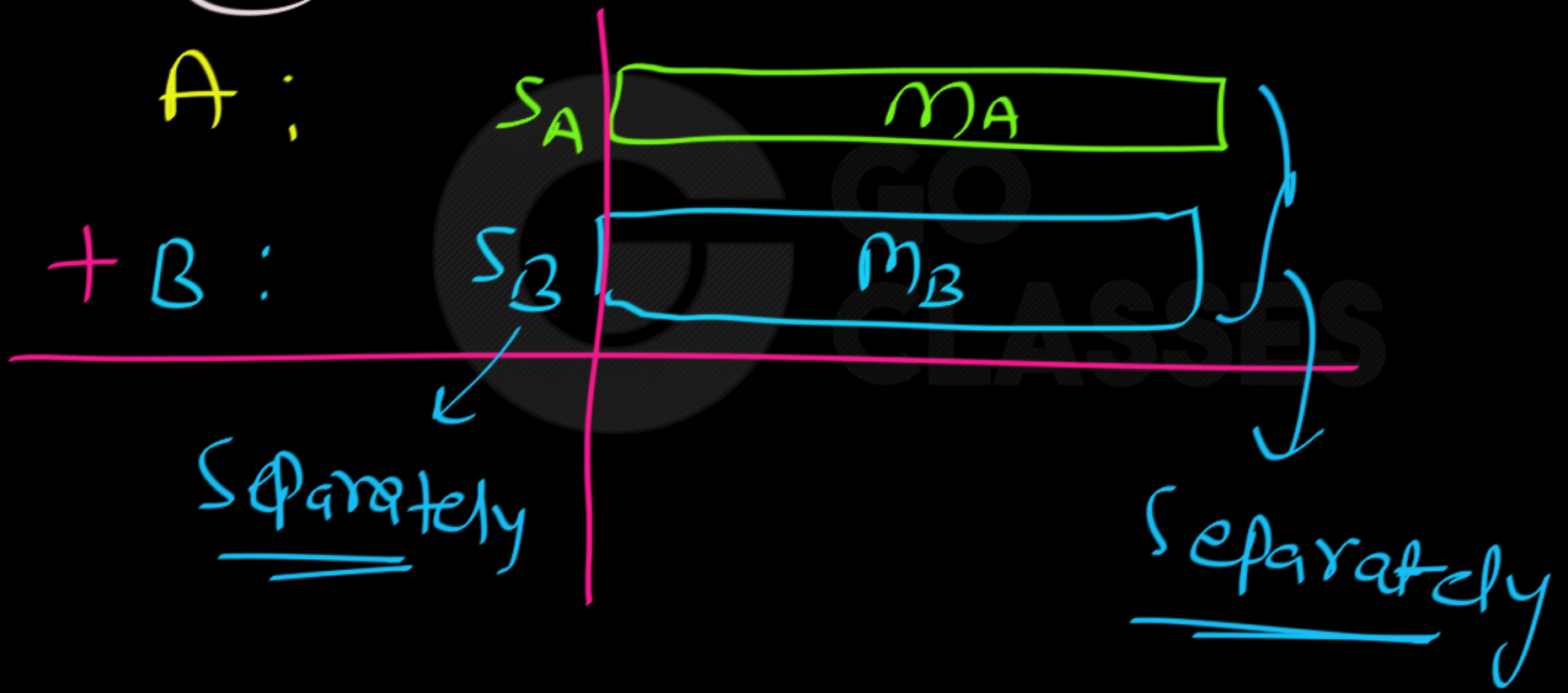
② +3, +5 of

④ (−6), (−2) of

③ (+4), (−3) of can NEVER occur

OF Detection Condition for Signed Number Addition in SM system?

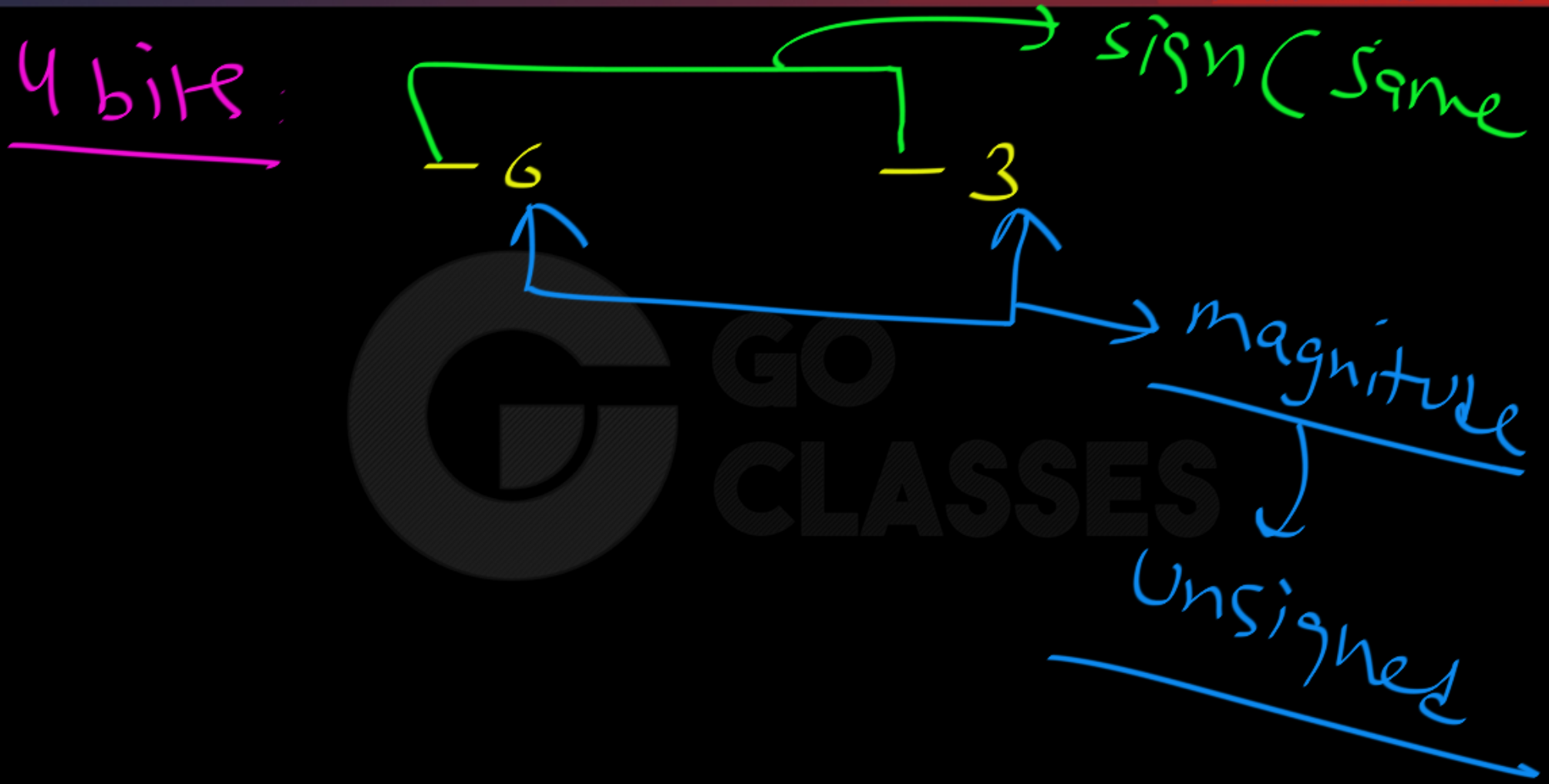
In SM System : A + B



① If $S_A \neq S_B$ then No overflow.

② If $S_A = S_B$ then Magnitude addition Decides Overflow.

(Unsigned number)


4 bits
sign (same
6
3
magnitude
Unsigned

(2) If $S_A = S_B$ then

Overflow occurs iff there is

Carry out of MSB of magnitude.

4 bits :→ Range : −7 to +7

(−6) + (−3) ⟹ ofL

(by Definition)

1 1 1 0 1 1

sign same

1 1 1 0
+ 0 1 1 1
0 0 1

⟹ of
=
L

4 bits :→ Range : −7 to +7

(+6) + (−5) ⇒ No of (By Definition of)

0 110
1 101

↳ Sign Different ⇒ No of ← by of Detection Condition

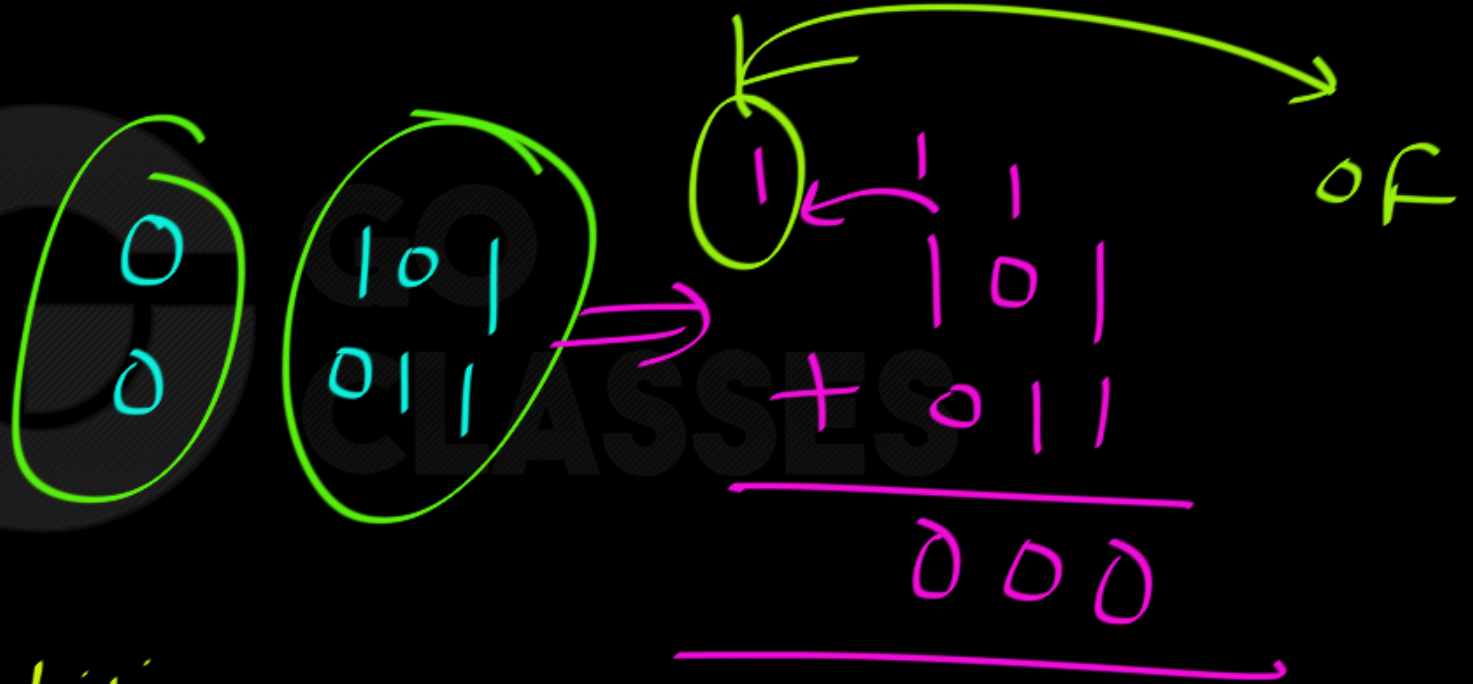
4 bits :→ Range : −7 to +7

+5, +3

sign same

↓

Magnitud Addition



4 bits :→ Range : −7 to +7

+5, +2

0 0

101 010

No carry out of MSB

$$\begin{array}{r} 101 \\ +010 \\ \hline 111 \end{array}$$

No of

Same

Rocket :→ 4 bits Registers

→ SM system

+5

+3

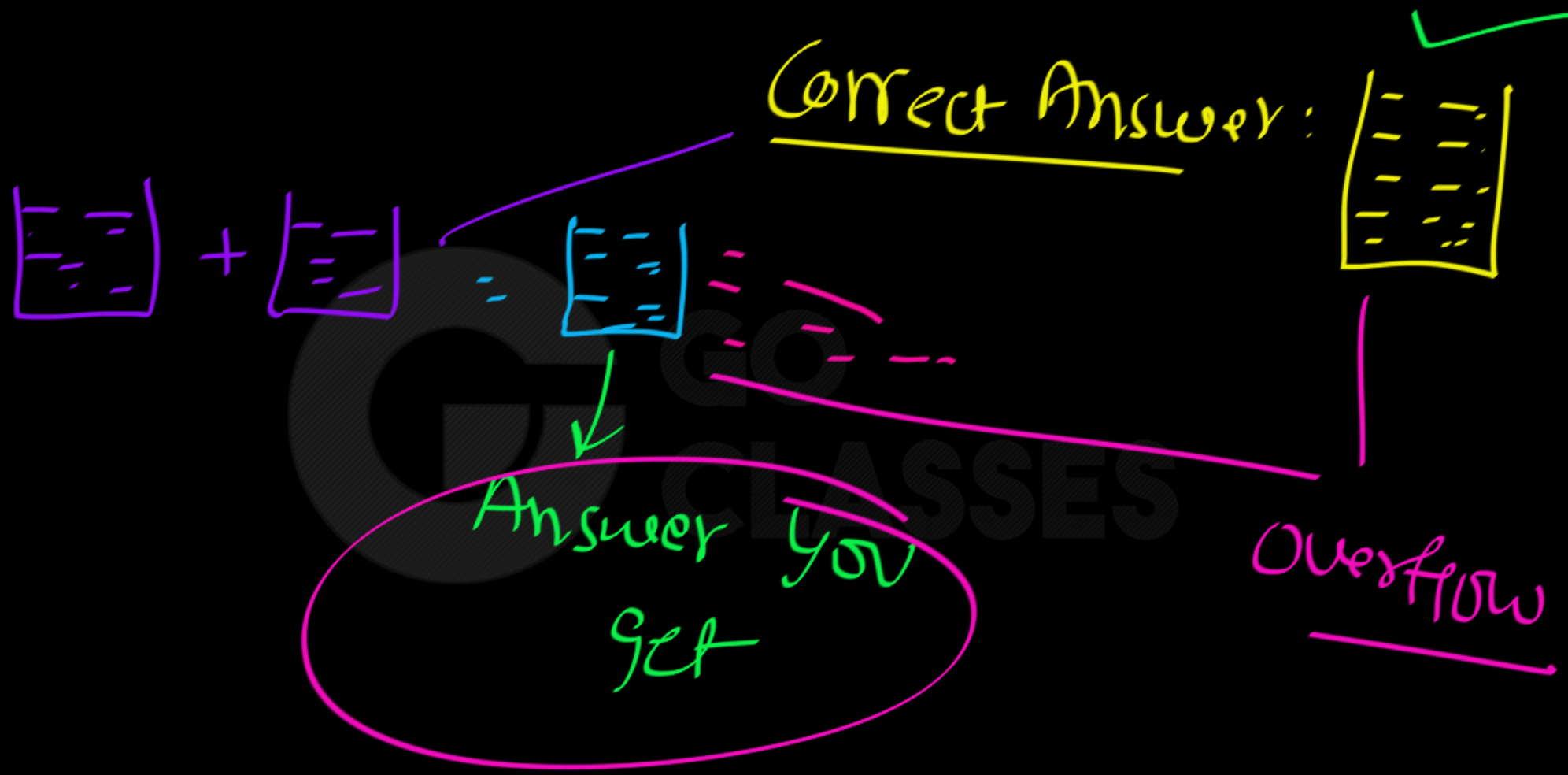
+

Crash

wrong

→ Data

because of Overflow


Correct Answer :
+
=
Answer you get
Overflow

If sign are same and

sign magnitude -- when there is a carry out of the msb of the magnitude

```
    1 1000 (-8)
  + 1 1001 (-9)
    -----
    1 0001 (-1)   (carry out of msb of magnitude)
```

Of occurs.

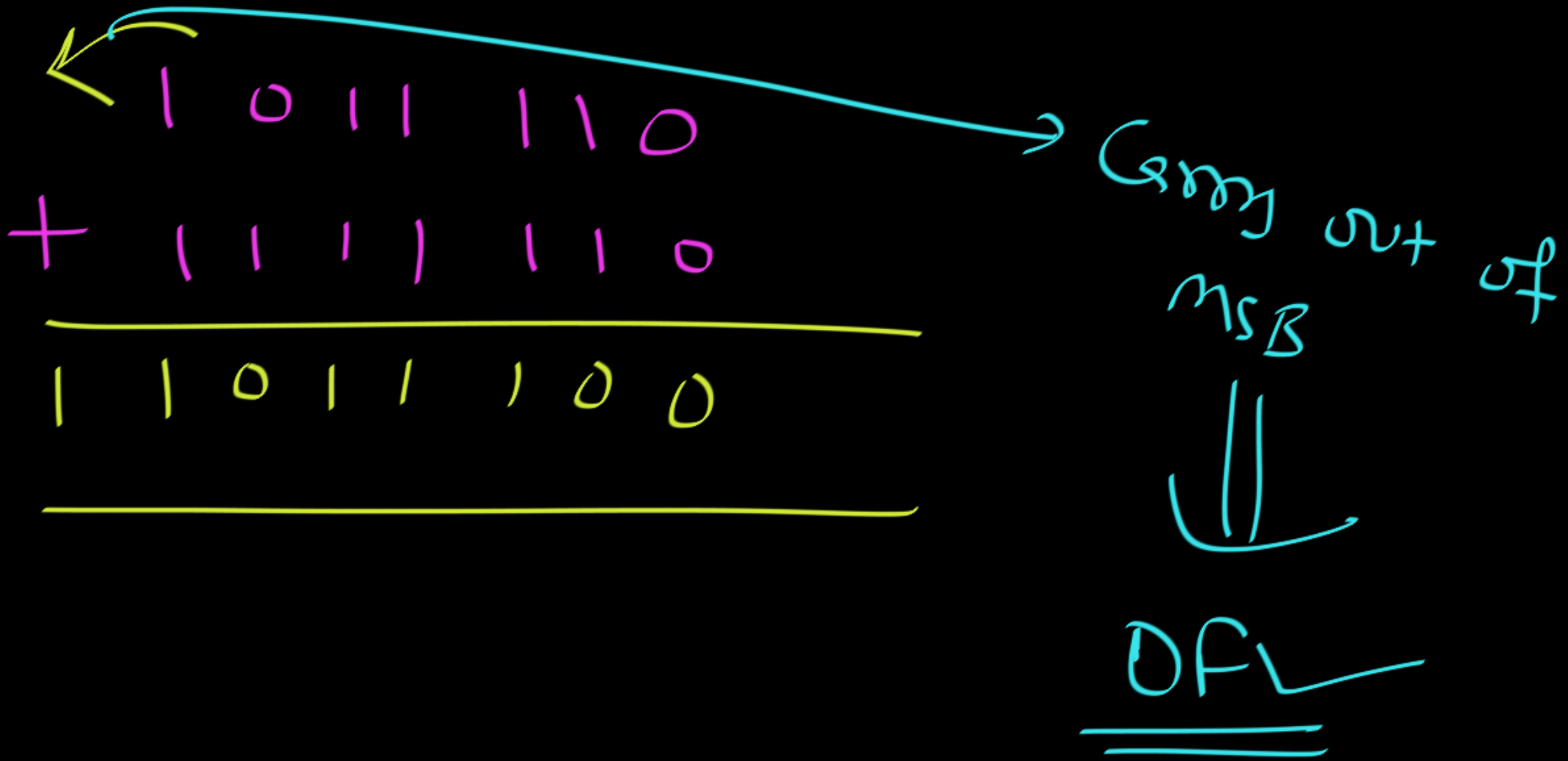
Ex: In SM system :⟶ 8 bits

$$\overline{=}\,01011101 + \overline{=}\,10110110$$

No of

$$\overline{=}\,01011110 + \overline{=}\,01111110$$

+


1 0 1 1 1 1 0
+ 1 1 1 1 1 1 0
1 1 0 1 1 1 0 0
Carry out of MSB
1
OF

A : S_A | a_m - — — —

B : S_B | b_m - - — - -

MSB of magnitude of B

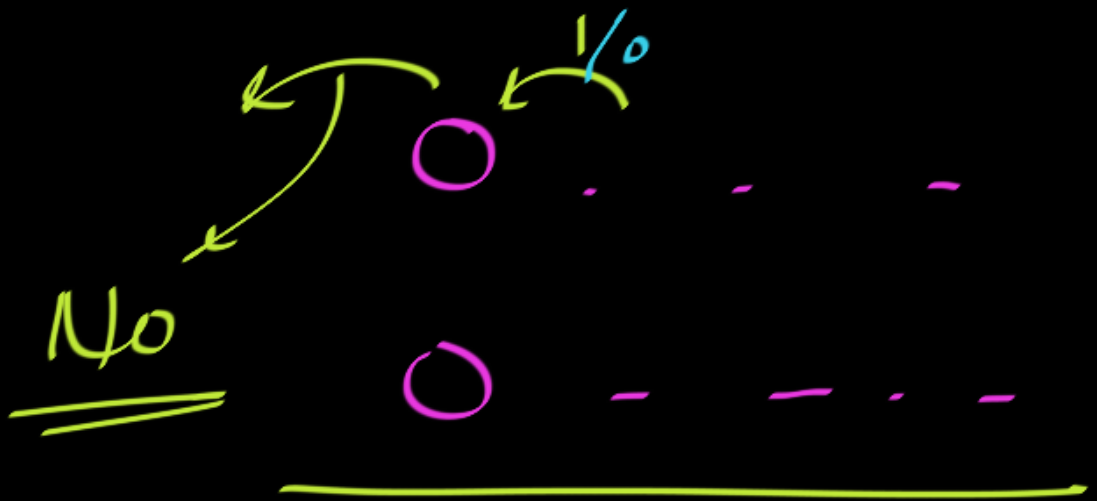
$a_m = b_m = 0 \Rightarrow$ No of

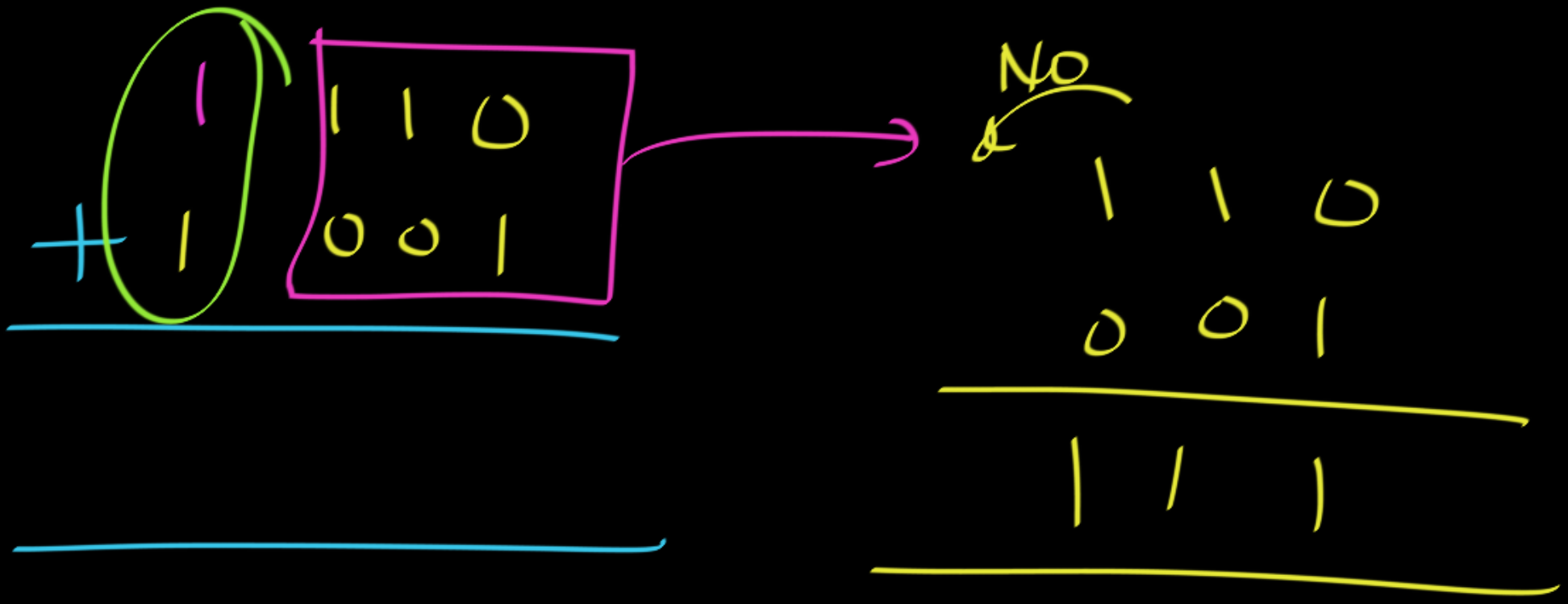
$a_m = b_m = 1 \Rightarrow$ of

$a_m \neq b_m \Rightarrow$ may or may not Overflow.

If $S_A \neq S_B \Rightarrow$ No of

$S_A = S_B \Rightarrow$ may or may not of.


1/0
No



No of

"Overflow!"

$C_{out} \oplus C_{in} = 1$

"Overflow!"

$C_{out} \oplus C_{in} = 1$ ⇒ for 2's Complement Addition

in future

Next Topic : Overflow

Signed Numbers Addition

"2's Complement Addition"

4 bits: Range: -8 to +7

+4, +5 — Never of

+4, +5 — of

-4, -4 ⟶ No of

-8, +1 ⟶ Never of

-4, -5

of

4 bits :

	SM	2's Comp
-4, -4	Of ✓	No Of

BINARY		DECIMAL
sign bit	data bit	
0	0	0
0	1	1
1	1	-1
1	0	-2

Recall that to represent 2's complement negative number, we must

1. Flip all bits
2. Add **1**.

2 bits:

Range: −2 to +1 ✓

~~+2, +1~~ Nonsense

+1, +1 OF ✓

+1, −2 No OF

−2, −1 OF ✓

−1, −1 No OFF

n bits:

Range: -2^{n-1} to $+(2^{n-1}-1)$

9. Signed Numbers Addition

Two-bit signed data type:

BINARY		DECIMAL
sign bit	data bit	
0	0	0
0	1	1
1	1	-1
1	0	-2

Notice that when operands have **opposite signs**, their sum will never overflow:

```
1 + -2 = -1
1 + -1 = 0
```

Therefore, *overflow can only occur when the operands have the same sign*:

```
  1 + 1 = 2
-2 + -2 = -4
-2 + -1 = -3
```

None of the results `{ 2, -4, -3 }` can fit into the **two-bit** signed data type.

There is **no overflow**, if:

both operands have Different Signs.

→ works for all Systems for Signed Number Addition

There is **no overflow**, if:

both operands are positive and the sum is positive.

both operands are negative and the sum is negative.

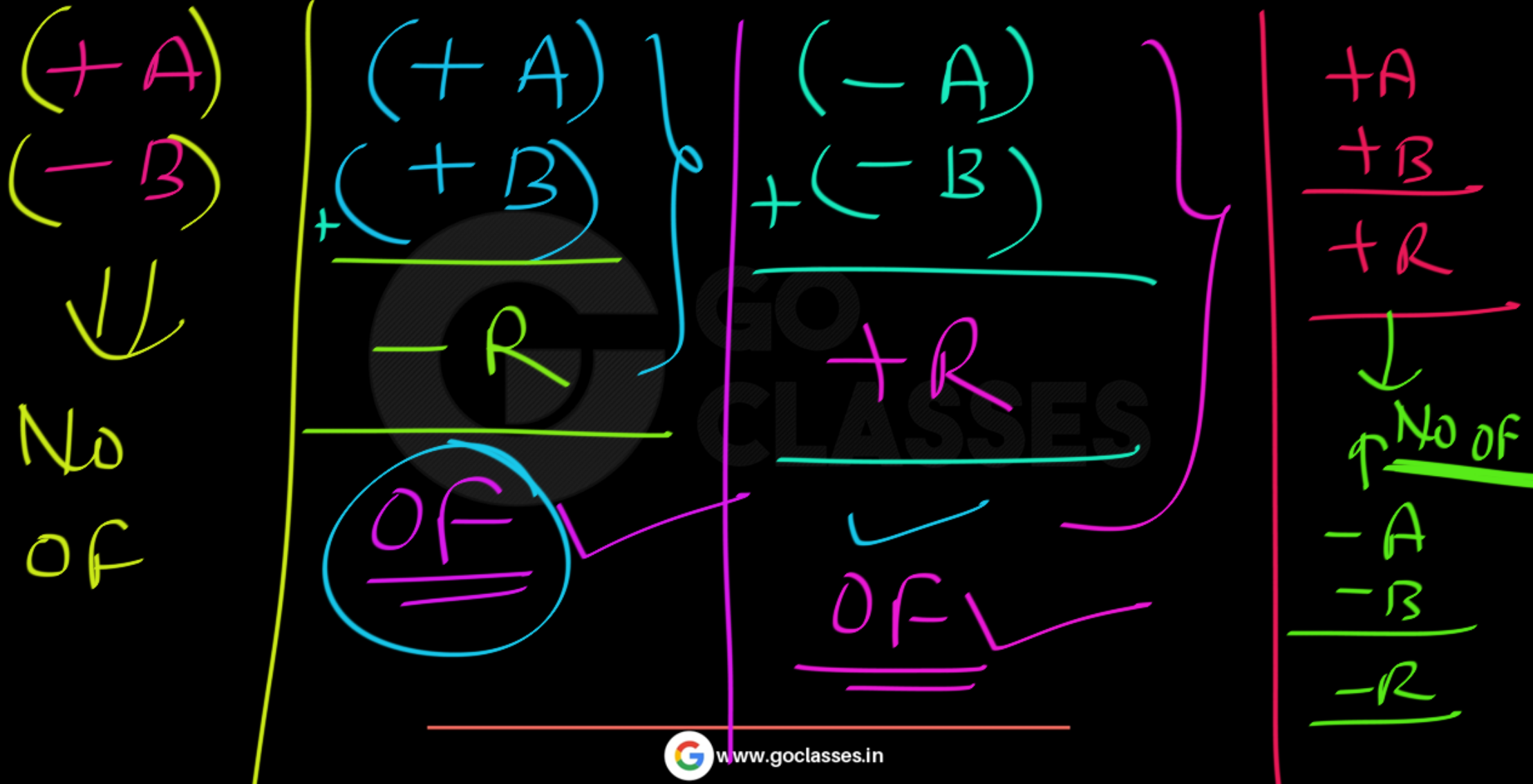
Works 2's Comp Addition
1's Comp Addition

There is [redacted] **overflow**, if:

both operands are positive and the sum is negative

both operands are negative and the sum is positive

Works 2's Comp Addition
1's Comp Addition



4 bits : (2's Comp Addition)

→ Range : −8 to +7

(+3), (+3)

No of

+4 +4

Overflow

+4 , +4 = +8 Correct Answer

0 1 0 0 — +ve
+ 0 1 0 0 — +ve
1 0 0 0 — -ve

Overflow

8 ⇒ Final Result → Wrong Answer.

When two signed 2's complement numbers are added, overflow is detected if:

1. both operands are positive and the sum is negative, or
2. both operands are negative and the sum is positive.

2's Complement Addition

$A = a_3\, a_2\, a_1\, a_0$

$+ B = b_3\, b_2\, b_1\, b_0$

$r_3\, r_2\, r_1\, r_0$ → final sum we get

OF occurs iff

($a_3 = 0$, $b_3 = 0$, $r_3 = 1$) OR ($a_3 = 1$, $b_3 = 1$, $r_3 = 0$)

2's Complement Addition

$A = a_3 a_2 a_1 a_0$

$+ B = b_3 b_2 b_1 b_0$

$r_3 \; r_2 \; r_1 \; r_0$

final sum we get

OF occurs iff

$$\overline{a_3}\,\overline{b_3}\,r_3 + a_3 b_3 \overline{r_3} = 1$$

$A = a_n \; a_{n-1} \; \cdots \; a_1$

$B = b_n \; b_{n-1} \; \cdots \; b_1$

$a_n = MS_B$ of A

$b_n = MS_B$ of B

OF occurs iff $\bar{a_n} \bar{b_n} \gamma_n + a_n b_n \bar{\gamma_n} = 1$

Addition of 2's Complement Numbers

The addition of n-bit signed binary numbers is straightforward using the 2's complement system. The addition is carried out just as if all the numbers were positive, and any carry from the sign position is ignored. This will always yield the correct result except when an overflow occurs. When the word length is n bits, we say that an *overflow* has occurred if the correct representation of the sum (including sign) requires more than n bits. The different cases which can occur are illustrated below for $n = 4$.

1. Addition of two positive numbers, sum $< 2^{n-1}$

+3	0011
+4	0100
+7	0111 (correct answer)

4 bits:

Range: −8 to +7

2. Addition of two positive numbers, sum $\geq 2^{n-1}$

+5	0101	
+6	0110	
	1011	← wrong answer because of overflow (+11 requires 5 bits including sign)

$\bar{q}_n \bar{b}_n r_n + q_n b_n \bar{r}_n = 1$

3. Addition of positive and negative numbers (negative number has greater magnitude)

+5	0101	
−6	1010	
−1	1111	(correct answer)

No of

4. Same as case 3 except positive number has greater magnitude

+5	1011	
+6	0110	
+1	(1)0001	← correct answer when the carry from the sign bit is ignored (this is *not* an overflow)

No of

5. Addition of two negative numbers, $|\text{sum}| \leq 2^{n-1}$

−3	1101	
−4	1100	
−7	(1)1001	← correct answer when the last carry is ignored (this is *not* an overflow)

Discarded

6. Addition of two negative numbers, $|\text{sum}| > 2^{n-1}$

−5	1011	
−6	1010	
	(1)0101	← wrong answer because of overflow (−11 requires 5 bits including sign)

Note that an overflow condition (cases 2 and 6) is easy to detect because in case 2 the addition of two positive numbers yields a negative result, and in case 6 the addition of two negative numbers yields a positive answer (for four bits).

(-5)

$+(-6)$

-11

OF ✓

A $1\ 0\ 1\ 1$

B $+\ 1\ 0\ 1\ 0$

$1\ \boxed{0\ 1\ 0\ 1}$

$a_n = 1$

$b_n = 1$

$s_n = 0$

OF ✓

Discarded

final Result

Next Topic : Overflow

Signed Numbers Addition

3. 1's Complement Addition

There is **no overflow**, if:

both operands are positive and the sum is positive.

both operands are negative and the sum is negative.

When two signed 1's complement numbers are added, overflow is detected if:

1. both operands are positive and the sum is negative, or
2. both operands are negative and the sum is positive.

A general rule for detecting overflow when adding two n-bit signed binary numbers (1's or 2's complement) to get an n-bit sum is:

An overflow occurs if adding two positive numbers gives a negative answer or if adding two negative numbers gives a positive answer.

Addition of 1's Complement Numbers

The addition of 1's complement numbers is similar to 2's complement except that instead of discarding the last carry, it is added to the n-bit sum in the position furthest to the right. This is referred to as an *end-around* carry. The addition of positive numbers is the same as illustrated for cases 1 and 2 under 2's complement. The remaining cases are illustrated below ($n = 4$).

3. Addition of positive and negative numbers (negative number with greater magnitude)

+5	0101	
−6	1001	
−1	1110	(correct answer)

No of 1's

Range:

−7 to +7

4. Same as case 3 except positive number has greater magnitude

−5	1010	
+6	0110	
	(1) 0000	
	→ 1	(end-around carry)
	0001	(correct answer, *no* overflow)

-3
-4
-7

No of

NOT r_n

A 1 1 0 0

+B 1 0 1 1

1 0 1 1 1

+1

1 0 0 0

final Result r_n

$a_n = 1$
$b_n = 1$
$r_n = 1$

No of

NOT the final Result

-5

-6

-11

or ✓

A 1 0 1 0

B + 1 0 0 1

1 0 0 1 1

$+1$

0 1 0 0

$a_n = 1$

$b_n = 1$

$r_n = 0$

OF

r_n

Wrong Answer

final Result

5. Addition of two negative numbers, $|\text{sum}| < 2^{n-1}$

−3	1100	
−4	1011	
	(1) 0111	
	↳ 1	(end-around carry)
	1000	(correct answer, *no* overflow)

6. Addition of two negative numbers, $|\text{sum}| \geq 2^{n-1}$

−5	1010	
−6	1001	
	(1) 0111	
	↳ 1	(end-around carry)
	0100	(wrong answer because of overflow)

Again, note that the overflow in case 6 is easy to detect because the addition of two negative numbers yields a positive result.

The following examples illustrate the addition of 1's and 2's complement numbers for a word length of $n = 8$:

1. Add −11 and −20 in 1's complement.

$+11 = 00001011 \qquad +20 = 00010100$

taking the bit-by-bit complement,

−11 is represented by 11110100 and −20 by 11101011

$$\begin{array}{rl} 11110100 & (-11) \\ \underline{11101011} & \underline{+(-20)} \\ (1)\,11011111 & \\ \quad \longrightarrow \underline{\;\;1\;\;} & \text{(end-around carry)} \\ 11100000 = -31 & \end{array}$$

2. Add −8 and +19 in 2's complement

+8 = 00001000

complementing all bits to the left of the first 1, −8, is represented by 11111000

$$\begin{array}{rl} 11111000 & (-8) \\ \underline{00010011} & \underline{+19} \\ (\cancel{1})00001011 = & +11 \end{array}$$

↑ (discard last carry)

Note that in both cases, the addition produced a carry out of the furthest left bit position, but there is no overflow because the answer can be correctly represented by eight bits (including sign). A general rule for detecting overflow when adding two n-bit signed binary numbers (1's or 2's complement) to get an n-bit sum is:

> An overflow occurs if adding two positive numbers gives a negative answer or if adding two negative numbers gives a positive answer.

An alternative method for detecting overflow in 2's complement addition is as follows:

> An overflow occurs if and only if the carry out of the sign position is not equal to the carry into the sign position.

2's Complement Addition :

OF Detection Condition :

$$\overline{a_n}\,\overline{b_n}\,\overline{r_n} + a_n\,b_n\,\overline{r_n} = 1$$

When $a_n b_n \overline{r_n} = 1$?

$A = a_n a_{n-1} - - a_1$

$B = b_n b_{n-1} - - b_1$

$C_{in} = 1$

1 - - - - - -

1 - - - - - - -

0

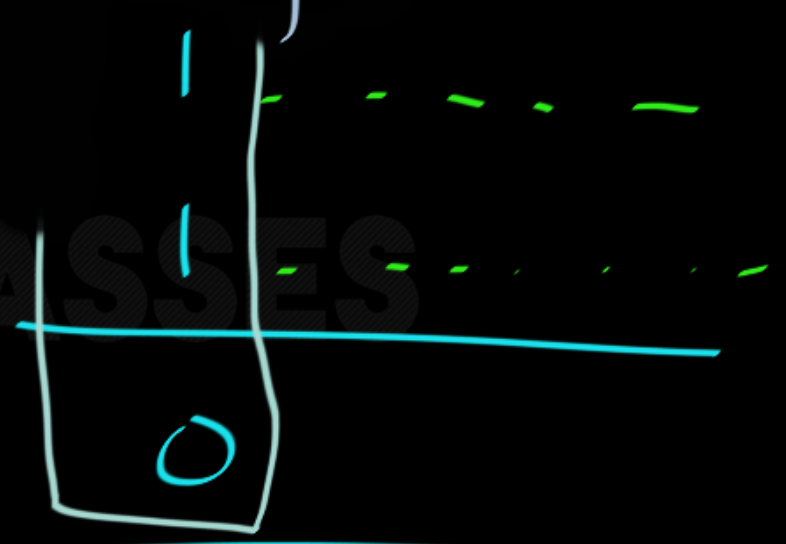
When $a_n b_n \overline{r_n} = 1$?

$C_{out} = 1$

$0 = C_{in}$

$A = a_n a_{n-1} - - a_1$

$B = b_n b_{n-1} - - b_1$



When $a_n b_n \overline{r_n} = 1$ iff $C_{in} = 0$ and $C_{out} = 1$

C_{out} C_{in}

$A = a_n a_{n-1} - - a_1$

$B = b_n b_{n-1} - - b_1$

C_{in} = in-Carry into MSB

C_{out} = out Carry out of MSB

When

$\overline{a_n}\ \overline{b_n}\ r_n = 1$ iff $c_{in} = 1$

$C_{out} = 0$

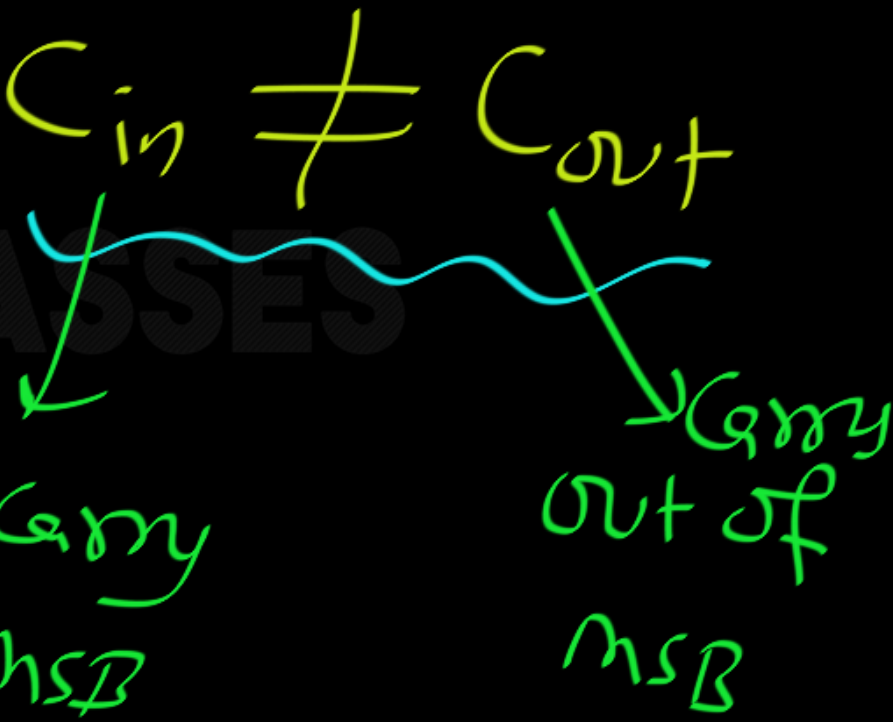
$C_{out} = 0$ ← 0 → $c_{in} = 1$

0 _ _ _

1 _ _ _ _ _ _

2's Complement Addition:

OF occurs iff $C_{in} \neq C_{out}$



2's Complement Addition:

OF occurs iff $C_{in} \oplus C_{out} = 1$

1's Complement Addition :

OF occurs iff $C_{in} \oplus C_{out} = 1$

→ WRONG

Ex: $C_{out}=1$ $C_{in}=0$

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

In 2's Comp

$a_n=1$ $r_n=0$

$b_n=1$

$\Rightarrow$ OF

$C_{out} \oplus C_{in} = 1 \oplus 0 = 1 \iff$ OF

1111 = −1

1000 = −8

In 2's Comp

−9

OF

Ex: $C_{out}=1$ $C_{in}=0$

$$\begin{array}{r} 1111 \\ 1000 \\ \hline 0111 \\ +1 \\ \hline 1000 \end{array}$$

$C_{out}=1, C_{in}=0$

r_n

In 1's Comp

$a_n=1$
$b_n=1$
$r_n=1$

No of

1111 = −0

1000 = −7

In 1's Comp

−7

No of 1's

1. Overflow Condition

- Arithmetic operations have a potential to run into a condition known as *overflow*.
- Overflow occurs with respect to the **size of the data type** that must accommodate the result.
- Overflow indicates that the result was *too large* or *too small* to fit in the original data type.
- When two signed 2's complement numbers are added, overflow is detected if:
    1. both operands are positive and the result is negative, or
    2. both operands are negative and the result is positive.
- When two unsigned numbers are added, overflow occurrs if
    - there is a **carry out** of the leftmost bit.

2. Binary Arithmetic

- Computers don't know the difference between signed and unsigned binary numbers.
- This is a good thing, because it makes logic circuits fast.
- This is also a bad thing, because distinguishing between signed and unsigned is our responsibility.
- The distinction is very important when detecting an **overflow** after addition or subtraction.
- Correct approach to **detect the overflow** is to consider two separate cases:
    1. Overflow when adding unsigned numbers.
    1. Overflow when adding signed numbers.

Misconceptions about Overflow

- Specific overflow detection requires knowing the *operation* and the *representation*.
- Overflow occurs when you do some operation to two valid representations...

  ...and the result can not be represented in the representation because the value is *too large* or *too smal*.
- Overflow detection is detecting overflow for a *specific representation*...

  ...Too often people mistake overflow condition for unsigned overflow, when the **carry out** is **1**.
- Overflow detection for 2's complement addition is different:

  One way to detect it is to XOR the **carry in** and the **carry out**.

final Conclusion : (Summary)

(1) Unsigned Addition :

OF iff Carry out of MSB.

(2) Signed Numbers Addition :

(All System)

If Sign Different $\Longrightarrow$ No of

If Signs Same $\Longrightarrow$ May or may not of.

(3) Sign-Magnitude Addition :

(1) Check sign :

Diff ⇒ No of

Same ⇒ magnitude Addition

of occurs iff Carry out of MSB of magnitude

(4) 1's Comp ; 2's Comp :-

a_n - - - -

b_n - - -

r_n - - - - -

→ in final Result

OF happens iff

$\bar{a}_n \bar{b}_n r_n + a_n b_n \bar{r}_n = 1$

5

Only for 2's Comp Addition:

Cout Cin

a_n - - - -

b_n - - -

r_n - - - - -

in final Result

OF happens iff

$C_{in} \neq C_{out}$

6

for 1's Comp Addition:

Cout Cin

a_n - - - -

b_n - - -

s_n - - - - -

in final Result

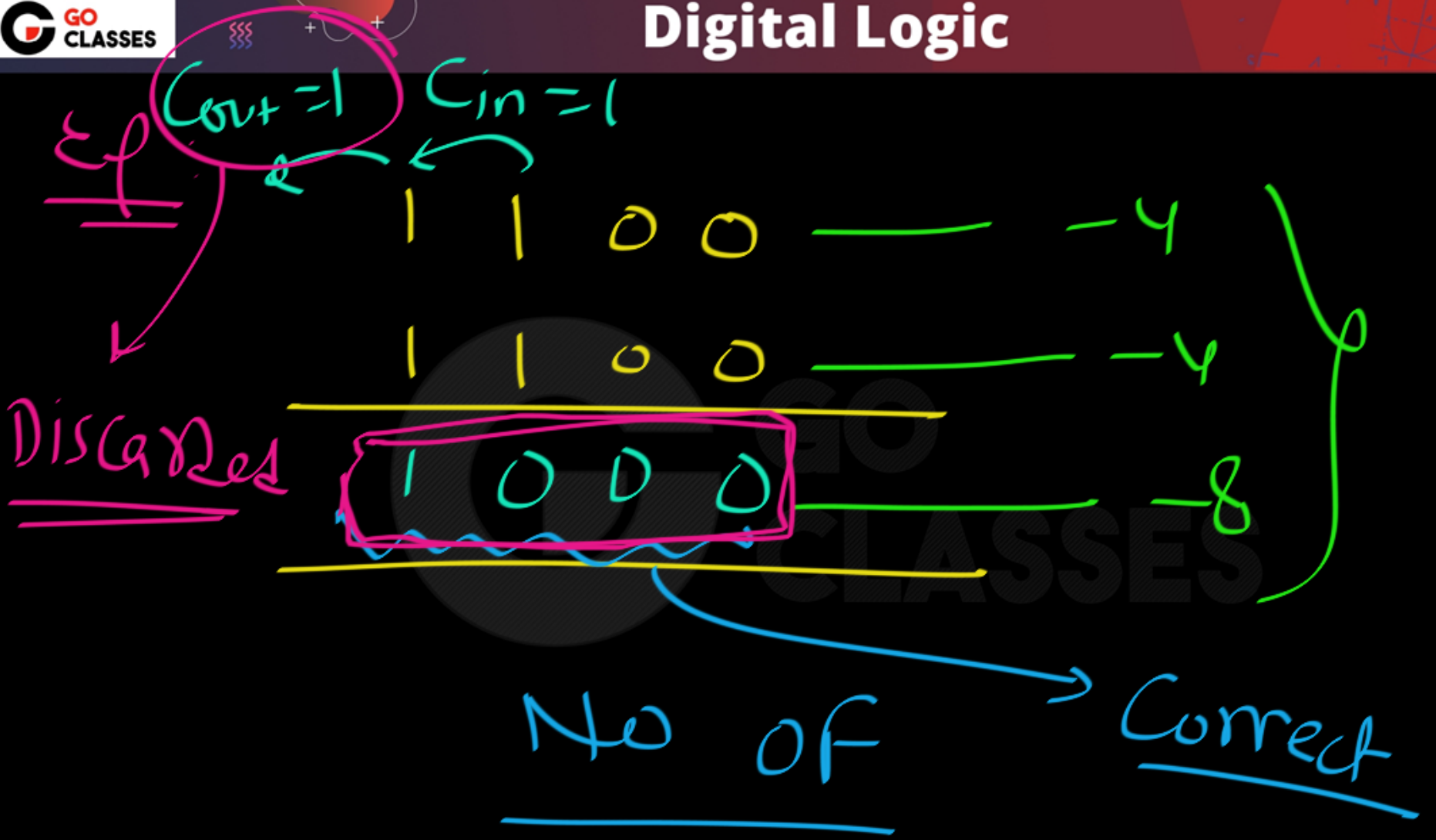
OF happens iff

$C_{in} \neq C_{out}$

Note: In 2's Comp Addition :

If $C_{in} = 1 = C_{out} \Rightarrow$ No of

$1 \oplus 1 = 0$


Cout = 1
Cin = 1
1 1 0 0 — -4
1 1 0 0 — -4
1 0 0 0 — -8
Discarded
No of
Correct