What is the max number given n position?

$$N = 3 \times 4^{n-1} + 3 \times 4^{n-2} + \cdots + 3 \times 4^2 + 3 \times 4^1 + 3 \times 4^0$$

$$N = 3 \times (4^{n-1} + 4^{n-2} + \cdots + 4^2 + 4^1 + 4^0) \rightarrow \text{geometric series}$$

$$N = 3 \times \left(\frac{4^n - 1}{4 - 1}\right)$$

$$N = 4^n - 1$$

Example: $n = 10$

$$4^{10} - 1 = 1{,}048{,}575$$

The other way around, What is the required position for a number N.

$$4^n - 1 = N$$

$$4^n = N + 1$$

$$\log_4 4^n = \log(N+1)$$

$$n = \log_4(N+1)$$

BASE-64 | RADIX-64

$(A,B,C, \ldots,Z, a,b,c,\ldots,z,0,1,2,\ldots,9,+,/)_{64}$

1992: RFC 1341

MIME (MULTIPURPOSE INTERNET MAIL EXTENSIONS)

Mechanisms For Specifying And Describing The Format Of Internet Message Bodies

Digit	Value		Digit	Value		Digit	Value		Digit	Value
A	0		Q	16		g	32		w	48
B	1		R	17		h	33		x	49
C	2		S	18		i	34		y	50
D	3		T	19		j	35		z	51
E	4		U	20		k	36		0	52
F	5		V	21		l	37		1	53
G	6		W	22		m	38		2	54
H	7	→	X	23	→	n	39	→	3	55
I	8		Y	24		o	40		4	56
J	9		Z	25		p	41		5	57
K	10		a	26		q	42		6	58
L	11		b	27		r	43		7	59
M	12		c	28		s	44		8	60
N	13		d	29		t	45		9	61
O	14		e	30		u	46		+	62
P	15		f	31		v	47		/	63

10's Complement

10^0	10^0
0	-9
1	-8
2	-7
3	-6
4	-5
5	-4
6	-3
7	-2
8	-1

Base-10 equal share

Signed complement base 10

Note: (*) We ignore the last carry, while adding or subtract.

(*) $x - y = x + (r\text{'s comp } Y)$

Adding in Base 16 (ex: base 16)

$$
\begin{array}{rcccccc}
 & & 1 & 1 & & 1 & \\
 & 0 & 2 & A_{(10)} & . & {}_{(14)}E & 5 & 4 \\
+ & 4 & B_{(11)} & F_{(16)} & . & 2 & B_{(11)} & 0 \\
 & & \frac{27}{16} = 1 \; r \; 11 & & & \frac{17}{16} = 1 \; r \; 1 & \frac{14}{16} = 1 \; r \; 0 & \\
\hline
 & 4 & E_{(14)} & B_{(11)} & & 1 & 0 & 4
\end{array}
$$

$\Rightarrow$ 4 E B 1 0 4

Substracting (ex: base 10)

(*)

$$
\begin{array}{rcccccc}
 & & +10 & & & & \\
 & -1 & 0 & 2 & 11 & . & 1 & 5 & 4 \\
 & - & 4 & 3 & 2 & . & 0 & 6 & 0 \\
\hline
 & 5 & 8 & 9 & . & 0 & 9 & 4
\end{array}
$$

(not complete)

Not possible

$\Rightarrow (021.154)_{10} - (432.060)_{10} = (021.154)_{10} + (1000.000)_{10} - (1000.000)_{10} - (432.060)_{10}$

$= -(1000.000)_{10} + (589.094)_{10}$

$= -\left[(1000.000)_{10} - (589.094)_{10}\right]$

$$
\begin{array}{r}
0\;9\;9\;9\;\;9\;9 \\
\cancel{1}\,\cancel{0}\,\cancel{0}\,\cancel{0}.\,\cancel{0}\,\cancel{0}\,\cancel{0} \\
-\;589.094 \\
\hline
410.906
\end{array}
$$

(complete)

Signed numbers (negative numbers)

(*) SIGNED: $(-x)+(-y) = -(x+y)$

ex: Base 11

```
   1  2  A(10).  E(15)  5  4
+  1  B  F(15)   2     B(11) 0
      (11) 26/16 = 1 r 10   18/16 = 1 r 2   16/16 = 1 r 0
   ----------------------------------------
   2  E  A      2     0  4
```

(*) SIGNED: $(+x)+(-y) = -(-x+y)$

ex: Base 16

```
   +16  +16  +16  13  +16
-1  -1   1   A10.  E   5 4
-   1    B(11) F15.  2 B(11) 0
    16/16 = 1/0
   ----------------------------
    1   6  B   B  A 4
```

(not complete)

$\Rightarrow (2A.E54)_{16} - (BF.2B)_{16}$

$= (2A.E54)_{16} + (100.000)_{16} - (100.000)_{16} - (1BF.2B)_{16}$

$\Rightarrow -(100.000)_{16} + (6B.BA4)_{16}$

$\Rightarrow -\left[(100.000)_{16} - (6B.BA4)_{16}\right]$

```
   0 15 15 15 15 16
   X 0 0. 0 0 0
-    6 B. B A 4
     11 11
   ----------------
     9 4. 4 5 C
```

(complete)

ex: Base 2

$$(27)_{10} - [-14]_{10} = (27)_{10} + (14)_{10}$$

$$\begin{array}{l|l}
27 & 2 \\ \hline
26 & 13 \\ \hline
1 & 12 \\ \hline
 & 1 \\
 & 6 \\ \hline
 & 0 \\
 & 6 \\ \hline
 & 3 \\
 & 2 \\ \hline
 & 1 \\
 & 1 \\ \hline
 & 0 \\
 & 0 \\ \hline
 & 1
\end{array}
\qquad
\begin{array}{l|l}
13 & 2 \\ \hline
6 & 6 \\ \hline
3 & 3 \\ \hline
1 & 1 \\ \hline
0 & 0
\end{array}
\qquad
\begin{array}{l|l}
6 & 2 \\ \hline
3 & 3 \\ \hline
1 & 1 \\ \hline
0 & 0
\end{array}
\qquad
\begin{array}{l|l}
3 & 2 \\ \hline
1 & 1 \\ \hline
0 & 0
\end{array}
\qquad
\begin{array}{l|l}
2 & 2 \\ \hline
1 & 1 \\ \hline
0 & 0
\end{array}
\qquad
\begin{array}{l|l}
1 & 2 \\ \hline
0 & 0
\end{array}$$

$$(27)_{10} = (11011)_2$$

$$\begin{array}{l|l}
14 & 2 \\ \hline
14 & 7 \\ \hline
0 & 6 \\ \hline
 & 1 \\
 & 3 \\ \hline
 & 2 \\
 & 1 \\ \hline
 & 1 \\
 & 1 \\ \hline
 & 0 \\
 & 0 \\ \hline
 & 1
\end{array}
\qquad
\begin{array}{l|l}
7 & 2 \\ \hline
6 & 3 \\ \hline
1 & 1 \\ \hline
0 & 0
\end{array}
\qquad
\begin{array}{l|l}
3 & 2 \\ \hline
2 & 1 \\ \hline
1 & 0 \\ \hline
 & 1
\end{array}
\qquad
\begin{array}{l|l}
1 & 2 \\ \hline
0 & 0
\end{array}$$

$$(14)_{10} = (1110)_2$$

$$(11011)_2 + (1110)_2$$

$$\begin{array}{cccccc}
 & & 1 & 1 & 1 & \\
1 & & 1 & 1 & 0 & 1 \; 1 \\
+ & & 1 & 1 & 1 & 0 \\
 & \cancel{2}=1 & \cancel{2}=1 & \cancel{2}=1 & \cancel{2}=1 & \\
\hline
1 & 0 & 1 & 0 & 0 &
\end{array}$$

1. Minterm

x' vs. x

x' → Complement form

x → Normal form

$m_0 = x'$

$m_1 = x$

2 binary variables appear either in one of these forms: $2^2 = 4$

yx (m_3) ; yx' (m_2) ; $y'x$ (m_1) , $y'x'$ (m_0)

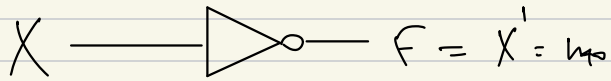
3 binary variables appear as : $2^3 = 8$

zyx, zyx', $zy'x'$, $z'y'x'$; $z'y'x$; $z'yx$, $zy'x$, $z'yx'$

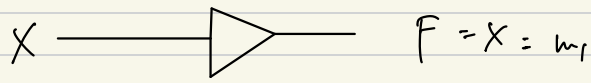
Therefore: 2^n

Truth tables

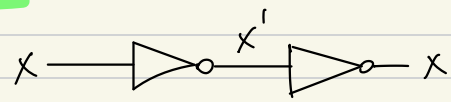
x	$F(x) = x' = m_0$
0	1
1	0



x	$F(x) = x = m_1$
0	0
1	1

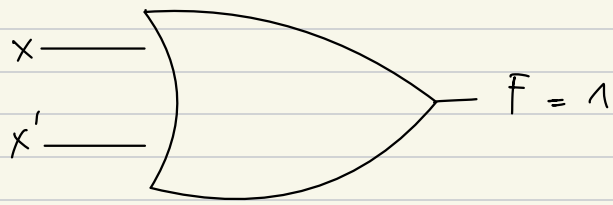


Note:

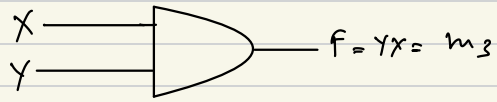


OR (+)

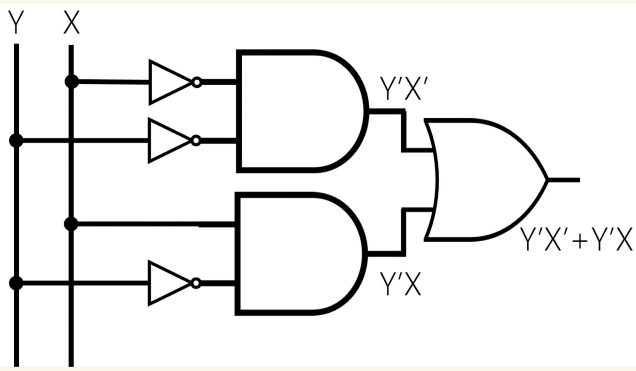
x'	x	$x'+x$
1	0	1
0	1	1



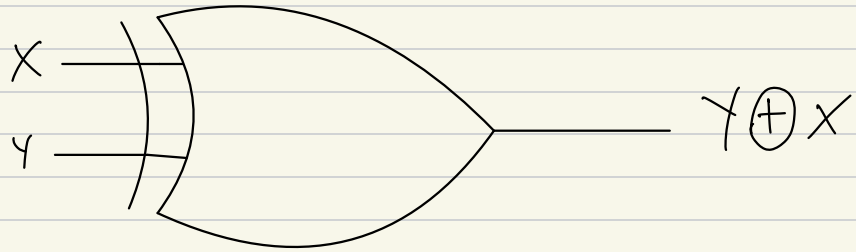
Y	X	$F = F(Y,X) = YX = m_3$
0	0	0
0	1	0
1	0	0
1	1	1



Y	X	$F = F(Y,X) = \sum m(0,1) = m_0 + m_1 = Y'X' + Y'X$
0	0	1
0	1	1
1	0	0
1	1	0



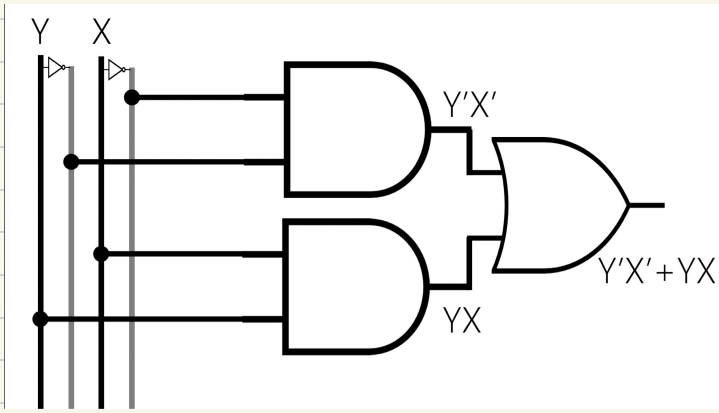
Exclusive - OR (X OR)



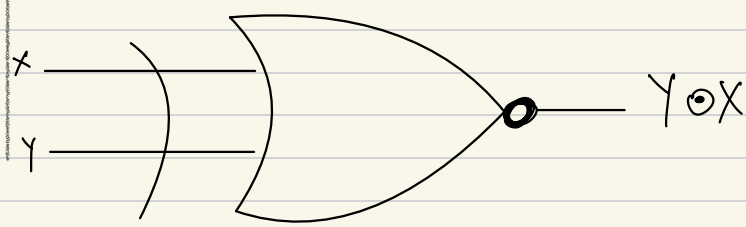
Y	X	$F = F(Y,X) = Y'X + YX' = m_1 + m_2$
0	0	0
0	1	1
1	0	1
1	1	0

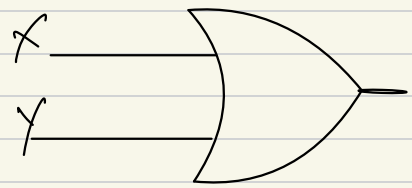
Ex

Y	X	$F = F(Y,X) = m_0 + m_3 = Y'X' + YX = \sum m(0,3)$
0	0	1
0	1	0
1	0	0
1	1	1



Y	X	$F = F(Y, X) = Y'X' + YX = m_0 + m_3$	$F = F(Y, X) = Y'X + YX' = m_1 + m_2$
0	0	1	0
0	1	0	1
1	0	0	1
1	1	1	0



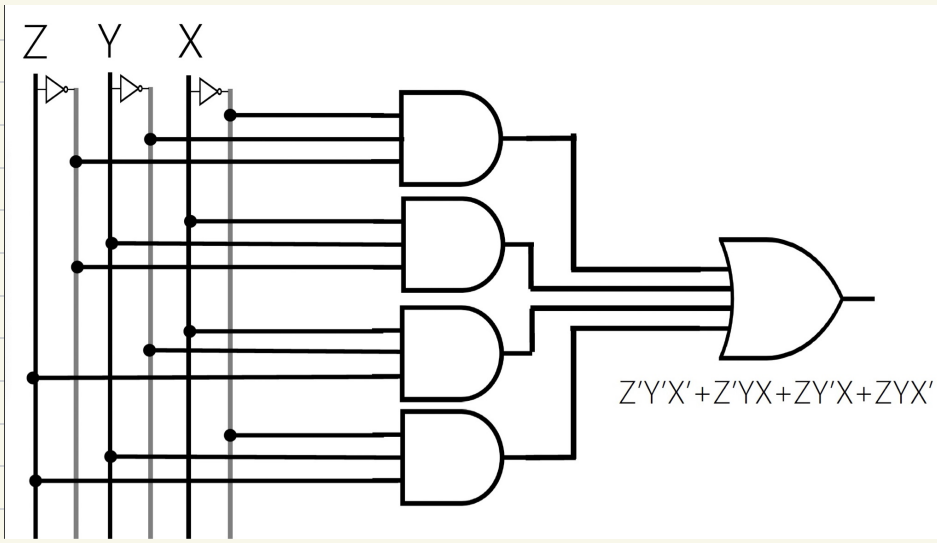


NOT EXclusive-OR

(NXOR ⟶ XNOR)

Example: Given 3 inputs, design a circuit to determine if there is even number of 1.

Z	Y	X	$F(Z,Y,X) = Z'Y'X' + Z'YX + ZY'X + ZYX' = m_0 + m_3 + m_5 + m_6 = \Sigma m(0,3,5,6)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



Show the remainder (MOD) NUMBER % 3 = ?

⊛ Range of input = $[0, 15]_{10} = [0, 1111]_2 = [0000, 1111]_2$

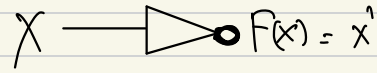
⊛ Range of output (The remainder of any number divided by 3 is 0,1,2)

$\Rightarrow [0, 2]_{10} = [0, 10]_2 = [00, 10]_2$

W	Z	Y	X	$F_1=\sum m(2,5,8,11,14)$	$F_2=\sum m(1,4,7,10,13)$
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	0	0
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	1	0	1
1	1	1	0	1	0
1	1	1	1	0	0

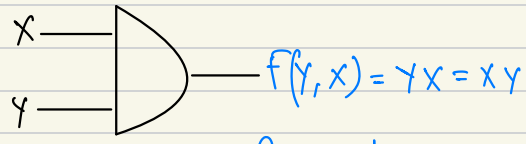
1's Complement

X	NOT X	Inverter X	X'	$\bar{X}$
0	1			
1	0			



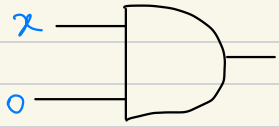
AND

Y	X	X AND Y	Y.X	Y*X
0	0	0		
0	1	0		
1	0	0		
1	1	1		

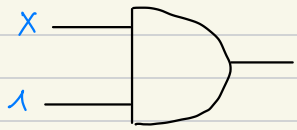


Commutative

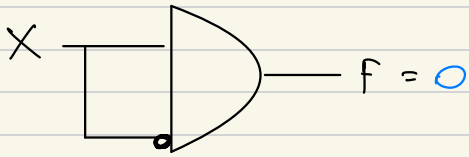
⊛ Boolean functions:

⊛ 

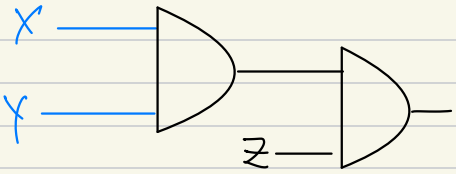
$F(0,x) = 0$; $F(x,0) = x \cdot 0 = 0$

⊛ 

$F = x$; $F = x1 = 1111 \times 1111 = x$

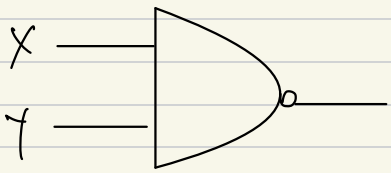
⊛ 

$F = Xx' = 0$

⊛ 

$F = z(Yx) = z(xY) = (zx)Y = (xz)Y$
$= xzY$

NAND (Not - AND)



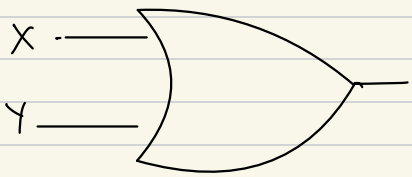
$F = (Yx)'$

⊛ $F = (YX)' = (XY)' = Y \uparrow X = X \uparrow Y$

Commutative

⊛ $F = (XX)' = X \uparrow X = X'$

OR



$F = X + Y = Y + X$

Commutative

X	Y	Y OR X	Y + X
0	0	0	
0	1	1	
1	0	1	
1	1	1	

Note: ⊛ $F = X + 1 = 1$

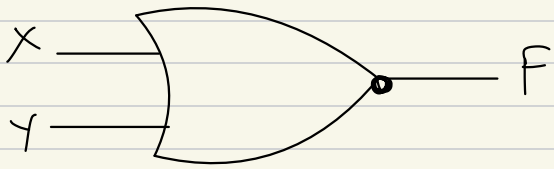
⊛ $F = X + 0 = X$

⊛ $F = X + X = X$

⊛ $F = Z + (Y + X) = Z + (X + Y) = (Z + X) + Y = Z + Y + X$

Associative.

NOR (Not-OR)



⊛ Commutative

Exercises

⊛ $F = A(B + C + D) + BCD \Rightarrow$ NAND-only

① The following binary numbers have a sign in the leftmost position and if negative, are in 2s complement form. Perform the indicated arithmetic operations and verify the answers. Indicate whether overflow occurs for each computation

(a) 100111 + 111001

(b) 110001 − 010010

Note How to verify if a number is negative (signed)

eg 110011 $n=6$ →

- check $n=6$
- the max with $n=6$ is $(111111)_2$
- divide by 2 : $\frac{111111}{2} = 011111$
- check if the 110011 is less than 011111, if so

Exclusive-OR (XOR) $\oplus$

XOR ; $x \oplus y = x\bar{y} + \bar{x}y$

Truth table

x	y	$\bar{x}$	$\bar{y}$	$x\bar{y}$	$\bar{x}y$	$x \oplus y$
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	1	0	1
1	1	0	0	0	0	0

exclusive-NOR (XNOR), also known as equivalence, performs the following Boolean operation:

$$x \odot y = xy + x'y'$$

Note: * Discrete entities can be counted while a continuous entity can be half

e.g → time is continuous (i.e time and half)

→ weight can be both

→

One's complement

One's compliment is a way to represent negative numbers in binary. To find the one's complement of a binary number, you simply flip all the bits : change 0s to 1s and 1s to 0s.

Example: For the binary number 1010, the one's complement would be 0101.

Two compliment

Two's compliment is another method for representing negative, and it's the most commonly used in computer systems. To find the two's compliment:

1. First, find the one's compliment (flip the bits).
2. Then add 1 to the significant bit (the rightmost bit).

Example

$$(1010)_2 = \underset{\text{one's compliment}}{(0101)_2} \Rightarrow 0101 + 0001 = \begin{array}{r} 0101 \\ +0001 \\ \hline 0110 \end{array} \Rightarrow \underset{\text{two's compliment.}}{(0110)_2}$$

OR it can be calculated by using: (-2^{n-1}) for n positions

SMF (Signed Magnitude Form)

eg: $(110110)_2 = -(\cancel{0}10110)_2 = -(\underset{4\,3\,2\,1\,0}{10110})_2 = -(16+4+2)_{10} = -(22)_{10}$

⊛ In c/c++, a variable can be designed as unsigned to indicate that the variable does not support negative numbers. This is like when we have number number system with no negative numbers:

```c
#include <stdio.h>
int main(void) {

    setbuf(stdout, NULL);
    unsigned int a;

    printf("Enter an unsigned integer number:\n");
    scanf("%u", &a);

    printf("The number is: \n");
    //printf("Binary: %b \n", a); There is no option for binary!
    printf("Octal: %o \n", a);
    printf("Decimal: %u \n", a);
    printf("Hexadecimal: %x \n", a); //Alphabet in small letters
    printf("HEXAdecimal: %X \n", a); //Alphabet in capital letters
    return 0;
}
```

However, there is implicit casting, when you enter -45, the decimal part would be a random number. We fix it like so:

```c
#include <stdio.h>
int main()
{

    signed int a;
    printf("Enter an unsigned integer number:\n");
    scanf("%d", &a);
    printf("The number is: \n");
    printf("Octal is: %o \n", a);
    printf("Decimal is: %d \n", a);
    printf("Hexadecimal is: %x \n", a); //Alphabet in small letters
    printf("HEXAdecimal is: %X \n", a); // Alphabet in capital letters

    return 0;
}
```

Before

```
→ lab05_ngabira gcc draft.c
→ lab05_ngabira ./a.out
Enter an unsigned integer number:
-45
The number is:
Octal is: 4294967251
Decimal is: 4294967251
Hexadecimal is: ffffffd3
HEXAdecimal is: FFFFFFD3
```

After

```
→ lab05_ngabira gcc draft.c
→ lab05_ngabira ./a.out
Enter an unsigned integer number:
-45
The number is:
Octal is: 37777777723
Decimal is: -45
Hexadecimal is: ffffffd3
HEXAdecimal is: FFFFFFD3
```

From the library <limits.h> we can find the minimum for a given type in c/c++. Here, we used UINT_MAX for the maximum unsigned integer, which is equal to 4294967295 in c/c++;

lab05_ngabira > C draft.c > ...

```c
#include <stdio.h>
#include<limits.h>

int main()
{

    unsigned int a;
    printf("Enter an unsigned integer number between %u and %u:\n", 0, UINT_MAX);
    scanf("%u", &a);

    printf("The number is: \n");
    printf("Octal is: %o \n", a);
    printf("Decimal is: %u \n", a);
    printf("Hexadecimal is: %x \n", a); //Alphabet in small letters
    printf("HEXAdecimal is: %X \n", a); // Alphabet in capital letters

    return 0;
}
```

output

Notice how (-45) is still wrong.

```
➜  lab05_ngabira ./a.out
Enter an unsigned integer number between 0 and 4294967295:
-45
The number is:
Octal is: 37777777723
Decimal is: 4294967251
Hexadecimal is: ffffffd3
HEXAdecimal is: FFFFFFD3
```

```
➜  lab05_ngabira ./a.out
Enter an unsigned integer number between 0 and 4294967295:
4294967295
The number is:
Octal is: 37777777777
Decimal is: 4294967295
Hexadecimal is: ffffffff
HEXAdecimal is: FFFFFFFF
```

Let's create an overflow by entering the maximum number +1

```
➜  lab05_ngabira ./a.out
Enter an unsigned integer number between 0 and 4294967295:
4294967296
The number is:
Octal is: 0
Decimal is: 0
Hexadecimal is: 0
HEXAdecimal is: 0
```

It is because this number would be 33 bits (instead of 32 bits) with the highest bit equal to 1 and all other remaining bits equal to 0

lab05_ngabira > C draft.c > main()

```c
#include <stdio.h>
#include<limits.h>

int main()
{

    int a;
    printf("Enter an unsigned integer number between %d and %d:\n", INT_MIN, INT_MAX);
    scanf("%d", &a);

    printf("The number is: \n");
    printf("Octal is: %o \n", a);
    printf("Decimal is: %d \n", a);
    printf("Hexadecimal is: %x \n", a); //Alphabet in small letters
    printf("HEXAdecimal is: %X \n", a); // Alphabet in capital letters

    return 0;
}
```

Overflow

```
Enter a signed integer number between -2147483648 and 2147483647:
2147483648
The number is:
Octal: 20000000000
Decimal: -2147483648
Hexadecimal: 80000000
HEXAdecimal: 80000000

Enter a signed integer number between -2147483648 and 2147483647:
-2147483649
The number is:
Octal: 17777777777
```

```
Decimal: 2147483647
Hexadecimal: 7fffffff
HEXAdecimal: 7FFFFFFF
```

Alternative Code

Line 32: decimal = (decimal << 1) | (binary[i] - '0');

⇒ first it shifts the decimal by one bit to the left e.g 000011 → 000110

⇒ Bitwise (OR) operator combine the shifted values

→ binary[i] - '0': This converts the character at binary[i] (either '0' or '1') into its integer value in C, characters are represented by their ASCII values.

POSTULATE (assumption, Axiom)

A statement that is taken to be true

Serve as a premise or starting point for further reasoning.

Proving Postulates

$X + X = X$

$X + X + \cdots + X = X$

$X + X = (X+X) \wedge$

$(X+X)(X+X')$

$XX + XX' +$

Duality

OR $\rightleftarrows$ AND ; $1 \rightleftarrows 0$

DUAL (F) $\neq$ NOT (F)

Ex

$x+1 \leftrightarrows x0$

$xx' \rightleftarrows x+x'$

Absorption law

$X + XY = X$

$X + XY + XZW + \dots + XWAD = X.$

$X + XY =$

$= (X1 + XY)$ using the identity element $= 1$

$= X(1+Y)$ using distributive property of x over $+$.

$= X1$ using $X+1 = 1$

$= X$ using identity $1x = 1$

K-Map

2 variable input

Y \ X	0	1
0	m0	m1
1	m2	m3

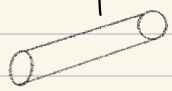
3-variable input

Z \ XY	00	01	11	10
0	m0	m1	m12	m3
1	m4	m5	m6	m7

4- variable input

WZ \ YX	00	01	11	10
00	m0	m1	m3	m2
01	m4	m5	m7	m6
11	m12	m13	m15	m14
10	m8	m9	m11	m10

!! K-map could be viewed as a roll



Connecting files

As seen, header (.h) files contain only the signatures of the functions and not the bodies.

e.g.

```
# include "arithmetic.h"
# include "complement.h"
```

Advanced!

We know the following facts:

i) 2's-complement of a binary is 1's-complement plus 1.
ii) 1's-complement of a binary number is equal to NOT of each bit.
iii) XOR of each binary variable b with 1 make it NOT b: $b \oplus 1 = b'$
iv) XOR of each binary variable b with 0 makes no change: $b \oplus 0 = b$
v) In C/C++, the XOR operator is '^'. So, b^1=b' and b^0=b

So, the 2's-complement of b is (b^1) + 1. A nice trick would be to have one single function for both addition and subtraction:

$$(b\hat{}m)+m = \begin{cases} 2's - comp(b) & \text{if } m = 1 \\ b & \text{if } m = 0 \end{cases}$$

```
void func_signed_2s_arithmetic(int a[], int b[], int m, int result[]){...}
```

Then,

```
func_signed_2s_arithmetic(a, b, 0, result)#addition
func_signed_2s_arithmetic(a, b, 1, result)#subtraction
```

⊛ **Minterm**: also called standard product (standard ANDing) expressed as sum of products (SOP)

⊛ **Maxterm**: also called standard sum (standard ORing) expressed as a product of sum (POS).

S	$F = S'I_0 + SI_1$
0	I_0
1	I_1

S \ $I_1 I_0$	00	01	11	10
0	0	1	1	0
1	0	0	1	1

Multiplexer

MUX $2^n \times 1$

$\overset{100}{21^0} = 2^2 = 4$ my

Finals Reading

Minterm = ANDing

eg: $y'x + y'x'$

Maxterm = ORing

$\rightarrow F_1(x,y) = \bar{y}x + \bar{y}\bar{x}$

$= m_3 + m_0$

$F_3(z,y,x) = \bar{y}x + zy$

Canonical form = SoP or PoS
Min Max

$E = F_1 + F_2$

$E = 1$ if $F_1 = 0$ + $F_2 = 1$ | 1 + 0 | 1 + 1

↓ no need

$E = 1$ if $m_3 + m_0 = 1$
$yx + \bar{y}\bar{x}$

$E = 1$ if $\bar{y}x + zy = 1$

X	Y	Z	$\bar{X}$	$\bar{Y}$	F_1	F_2	E
0	0	0	1	1	1	0	1
0	0	1	1	1	1	0	1
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	1	0	1	1
1	0	1	0	1	0	1	1
1	1	0	0	0	1	0	1
1	1	1	0	0	1	1	1

$E = F_1 + F_2 = yx + \bar{y}\bar{x} + \bar{y}x + zy$

$= y(x+z) + \bar{y}(\bar{x}+x)$ factorizing

$\Rightarrow y(x+z) + \bar{y}(1) \rightarrow$ identity law

$= y(x+z) + \bar{y}$

or

$= yx + yz + \bar{y}$

(b) $G = F_1 F_2$

$G = 1$ if $F_1 = 1$
$F_2 = 1$

X	Y	Z	$\bar{X}$	$\bar{Y}$	F_1	F_2	G
0	0	0	1	1	1	0	0
0	0	1	1	1	1	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	1	0	1	0
1	0	1	0	1	0	1	0
1	1	0	0	0	1	0	0
1	1	1	0	0	1	1	1

So for $G = 1$ it means that $F_1 = 1$ and $F_2 = 1$, therefore G equals to the minterms that exist in both F_1 and F_2.

② (a) $\Rightarrow \Sigma$.

De Morgan's Law

$(Y+X)' = Y'X'$

X	Y	$\bar{X}$	$\bar{Y}$	$(Y+X)'$	$\bar{Y}\bar{X}$
0	0	1	1	1	1
0	1	1	0	0	0
1	0	0	1	0	0
1	1	0	0	0	0

XOR