International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Invisible Unicode Programming

Dilip Kumar Rai

Dilip Rai,House-77,Phase-2,Kailash Nagar,Bhopal,(M.P.),Pin-462010,India
DOI: ***https://doi.org/10.55248/gengpi.2022.3.4.1***

A B S T R A C T

Invisible Unicode programming (IUP) can be defined as the art and science of hiding information in data that can be read by the computer. **Invisible Unicode Programming (IUP)** has the explicit goal of converting traditional character encodings to invisible encoding. **This paper** presents a new method to hide text. This systematic method uses the binary format of invisible character to hide and extract secret information. Creating a secret message involves four main steps, first using the binary format of the original letters in the message.nextcreating the appropriate **Binary Masking Value (BMV)**to cover the text, and subtracting the **Binary Value (BV)** from the **Binary Masking Value (BMV)**text to get **Binary Invisible Unicode (BIU)**, and finally masking secret text using Binary Invisible Character (BIU). **This Invisible Unicode Programming (IUP)** Concept uses the 54 Invisible Unicode characters to make the text invisible.
The results of the experiments show that this **IUP method** creates highly secured invisible information using the multi-level complexity algorithm to avoid the hackers.

Keywords: Invisible Unicode, Invisible Coding, Hiding information, Invisible Characters

1. UNICODE STANDARD

Unicode is a universal character encoding standard that is used to support non-ASCII characters. Initially, all text editors were created based on ASCII encoding, which contains characters of the English alphabet and consists of only 128 characters.
Unicode provides support for all the world's languages and their unique character sets. Unicode can support more than 1 million characters. The reason is that Unicode can use more position bits to represent a character, which are units of information in computers. ASCII characters only require 7 bits, while Unicode can use 16 bits. This is necessary because some languages, such as Chinese and Arabic, require more position bits.
At the same time, the Unicode table for characters in a language such as Arabic includes languages such as Persian, Urdu, Pashto, Sindhi, and Kurdish. The standard provides detailed explanations of implementation methods, including the letter-join method, right-to-left text insertion, and much more.
For our research, we will rely on the work, where we are interested in Unicode codes for spaces.
The Unicode Consortium
The Unicode Consortium develops the Unicode Standard. Their goal is to replace the existing character sets with its standard Unicode Transformation Format (UTF). The Unicode Standard has become a success and is implemented in HTML, XML, Java, JavaScript, E-mail, ASP, PHP, etc. The Unicode standard is also supported in many operating systems and all modern browsers. The Unicode Consortium cooperates with the leading standards development organizations, like ISO, W3C, and ECMA.
The Unicode Consortium developed the UTF-8 and UTF-16 standards, because the ISO-8859 character-sets are limited, and not compatible a multilingual environment.
The Unicode Standard covers (almost) all the characters, punctuations, and symbols in the world.

* *Corresponding author.* Tel.: +91 7869849003.
E-mail address: info@diliprai.com

2. Introduction

With the ceaseless usage of web and other online services, it has turned out that copying, sharing, and transmitting digital media over the Internet are amazingly simple. Since the text is one of the main available data sources and most widely used digital media on the Internet, the significant part of websites, books, articles, daily papers, and so on is just the plain text Currently IUP technique is applied for saving privacy and originality of HTML Source Code . Thereby, Invisible Unicode Programming considers as a challenging mission that tenuous adjustment in HTML Source coding can be specified. Invisible character technique is used to hide the html source code without anyone can be seen the hidden processing.

There are different techniques like steganography, cryptography; coding, etc have been utilized. Now we are using IUP to hide html source code. There are 54 invisible Unicode characters are considering the flag of communicating in a hidden style.

In view of this digital invisible Unicode programming conceals even the evidence of encrypted messaging, many methods have been used to hide information by using the recorder with tales of steganography and cryptography through times of war or peace, Moreover, IUP is the art and

science which hides information in any computer readable data in a way that an invisible Unicode character should be not distinguishable from origin cover neither by a human nor by computer looking for statistical pattern.

3. Illustrations

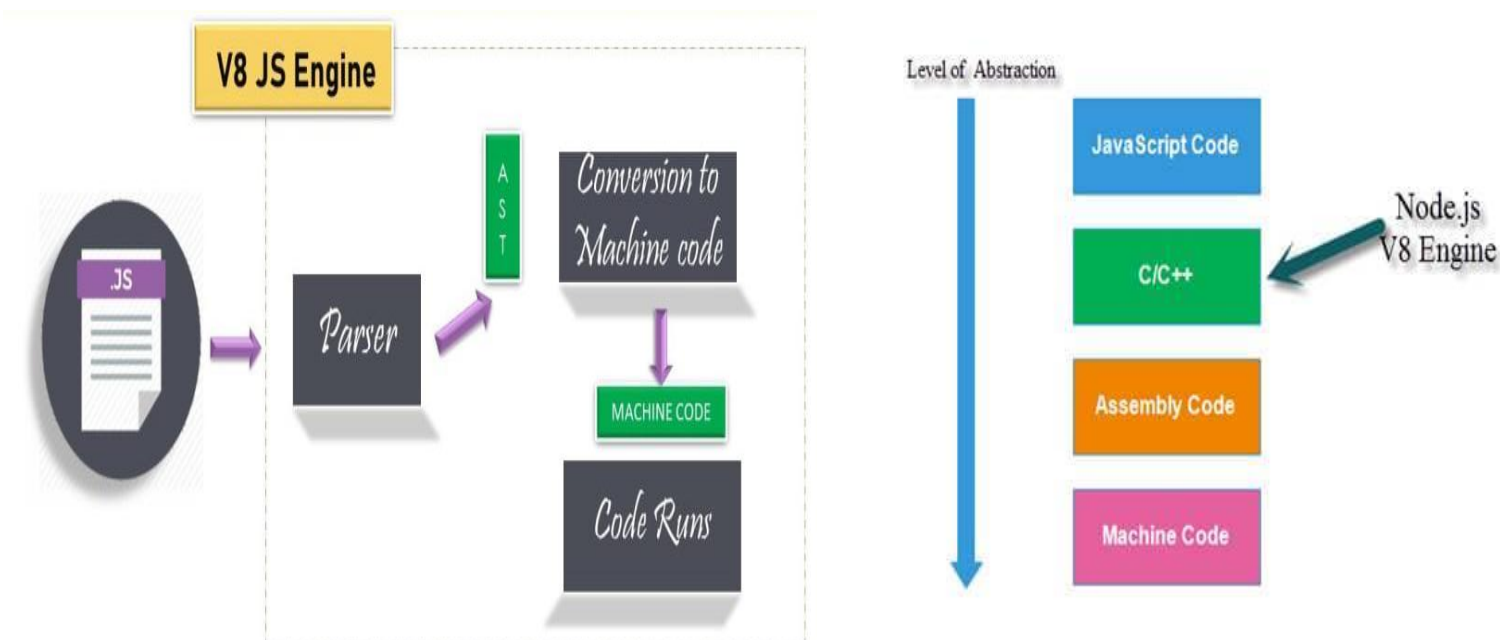


Fig. 1 - (a) first picture; (b) second picture.

4. Related Work

4.1 **Whitespace Programming Language**, designed in 2003 by Edwin Brady and Chris Morris, is an imperative, stack-based, esoteric programming language that uses only whitespace characters—space, tab, and linefeed—as syntax. All other characters are ignored. Whitespace got a brief moment of fame when it was posted on Slashdot on April 1st, 2003. Most people took it as an April fool's joke, while it wasn't. UP is superset of Whitespace programming Language because Whitespace Programming language only uses 4 characters, but IUP use 54 Invisible Unicode characters. Much more secure and powerful.

4.2 **Steganography** is the practice of concealing a message within another message or a physical object. In computing/electronic contexts, a computer file, message, image, or video is concealed within another file, message, image, or video. The word steganography comes from Greek steganographia, which combines the words steganós (στεγανός), meaning "covered or concealed", and -graphia (γραφή) meaning "writing".

4.3 **Cryptography** is the study of secure communications techniques that allow only the sender and intended recipient of a message to view its contents. The term is derived from the Greek word kryptos, which means hidden.

4.4 **Watermarking** is the technique and art of hiding additional data (such as watermarked bits, logo and text message) in the host signal which includes image, video, audio, speech, text, without any perceptibility of the existence of additional information

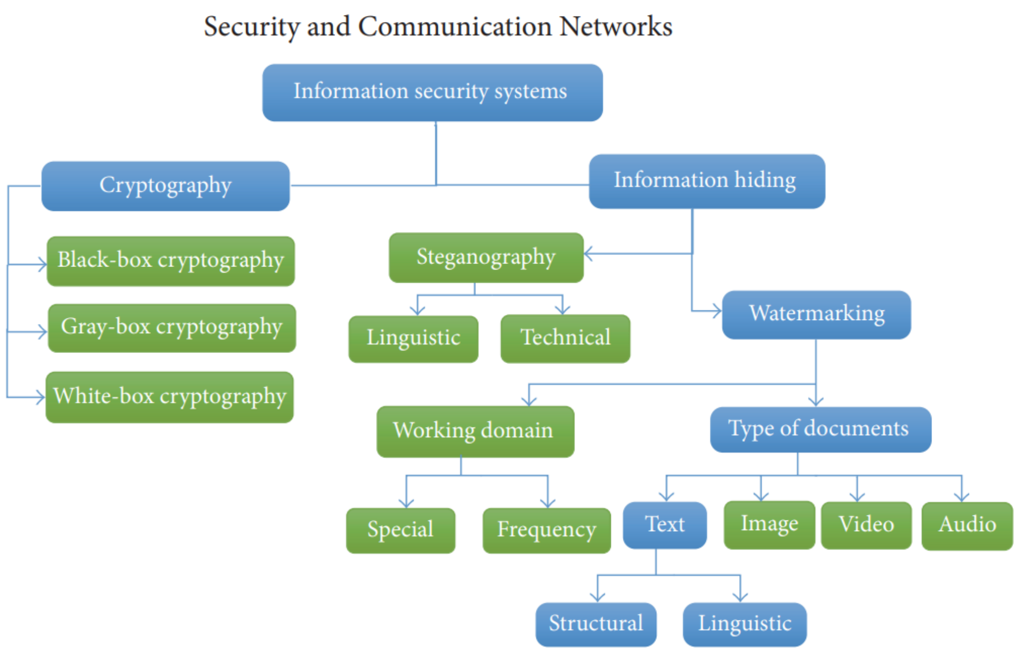


FIGURE 1: Different categories of information security systems.

5. Hiding processing

1. ASCII BINARY + INVISIBLE UNICODE BINARY = Masking Binary Value (MBV)
2. ASCII BINARY – Masking Binary Value = INVISIBLE UNICODE BINARY
3. MBV – IUB = ASCII Character.

TABLE - I
Invisible Unicode in the Algorithm

ENCODE INVISIBLE UNICODE CHARSET			UTF-8,UTF-16 & UTF-32 ENCODE					
SNO	**Space**	**Unicode**	**ENCODE TYPE**	**hex**	**dec(bytes)**	**dec**	**binary**	**HTML**
1	CHARACTER TABULATION	U + 0009	**1 byte**	09	9	9	00001001	
2	SPACE	U+0020	**1 byte**	20	32	32	00100000	
3	NO-BREAK SPACE	U+00A0	**2 byte**	C2 A0	194 160	49824	11000010 10100000	
4	SOFT HYPHEN	U+00AD	**2 byte**	C2 AD	194 173	49837	11000010 10101101	
5	COMBINING GRAPHEME JOINER	U+034F	**2 byte**	CD 8F	205 143	52623	11001101 10001111	͏
6	ARABIC LETTER MARK	U+061C	**2 byte**	D8 9C	216 156	55452	11011000 10011100	؜
7	HANGUL CHOSEONG FILLER	U+115F	**3 byte**	E1 85 9F	225 133 159	14779807	11100001 10000101 10011111	ᅟ
8	HANGUL JUNGSEONG FILLER	U+1160	**3 byte**	E1 85 A0	225 133 160	14779808	11100001	ᅠ

							10000101 10100000	
9	KHMER VOWEL INHERENT AQ	U+17B4	**3 byte**	E1 9E B4	225 158 180	14786228	11100001 10011110 10110100	឴
10	KHMER VOWEL INHERENT AA	U+17B5	**3 byte**	E1 9E B5	225 158 181	14786229	11100001 10011110 10110101	឵
11	MONGOLIAN VOWEL SEPARATOR	U+180E	**3 byte**	E1 A0 8E	225 160 142	14786702	11100001 10100000 10001110	&#6158
12	EN QUAD	U+2000	**3 byte**	E2 80 80	226 128 128	14844032	11100010 10000000 10000000	
13	EM QUAD	U+2001	**3 byte**	E2 80 81	226 128 129	14844033	11100010 10000000 10000001	
14	EN SPACE	U+2002	**3 byte**	E2 80 82	226 128 130	14844034	11100010 10000000 10000010	
15	EM SPACE	U+2003	**3 byte**	E2 80 83	226 128 131	14844035	11100010 10000000 10000011	
16	THREE-PER-EM SPACE	U+2004	**3 byte**	E2 80 84	226 128 132	14844036	11100010 10000000 10000100	
17	FOUR-PER-EM SPACE	U+2005	**3 byte**	E2 80 85	226 128 133	14844037	11100010 10000000 10000101	
18	SIX-PER-EM SPACE	U+2006	**3 byte**	E2 80 86	226 128 134	14844038	11100010 10000000 10000110	
19	FIGURE SPACE	U+2007	**3 byte**	E2 80 87	226 128 135	14844039	11100010 10000000 10000111	
20	PUNCTUATION SPACE	U+2008	**3 byte**	E2 80 88	226 128 136	14844040	11100010 10000000 10001000	
21	THIN SPACE	U+2009	**3 byte**	E2 80 89	226 128 137	14844041	11100010 10000000 10001001	
22	HAIR SPACE	U+200A	**3 byte**	E2 80 8A	226 128 138	14844042	11100010 10000000 10001010	
23	ZERO WIDTH SPACE	U+200B	**3 byte**	E2 80 8B	226 128 139	14844043	11100010 10000000 10001011	
24	ZERO WIDTH NON-JOINER	U+200C	**3 byte**	E2 80 8C	226 128 140	14844044	11100010 10000000 10001100	‌
25	ZERO WIDTH JOINER	U+200D	**3 byte**	E2 80 8D	226 128 141	14844045	11100010 10000000 10001101	‍
26	LEFT-TO-RIGHT MARK	U+200E	**3 byte**	E2 80 8E	226 128 142	14844046	11100010 10000000 10001110	‎
27	RIGHT-TO-LEFT MARK	U+200F	**3 byte**	E2 80 8F	226 128 143	14844047	11100010 10000000 10001111	‏
28	NARROW NO-BREAK SPACE	U+202F	**3 byte**	E2 80 AF	226 128 175	14844079	11100010 10000000 10101111	
29	MEDIUM MATHEMATICAL SPACE	U+205F	**3 byte**	E2 81 9F	226 129 159	14844319	11100010 10000001 10011111	
30	WORD JOINER	U+2060	**3 byte**	E2 81 A0	226 129 160	14844320	11100010 10000001 10100000	
31	FUNCTION APPLICATION	U+2061	**3 byte**	E2 81 A1	226 129 161	14844321	11100010 10000001 10100001	⁡
32	INVISIBLE TIMES	U+2062	**3 byte**	E2 81 A2	226 129 162	14844322	11100010 10000001 10100010	⁢
33	INVISIBLE SEPARATOR	U+2063	**3 byte**	E2 81 A3	226 129 163	14844323	11100010 10000001 10100011	⁣
34	INVISIBLE PLUS	U+2064	**3 byte**	E2 81 A4	226 129 164	14844324	11100010 10000001 10100100	⁤
35	INHIBIT SYMMETRIC SWAPPING	U+206A	**3 byte**	E2 81 AA	226 129 170	14844330	11100010 10000001 10101010	⁪
36	ACTIVATE SYMMETRIC SWAPPING	U+206B	**3 byte**	E2 81 AB	226 129 171	14844331	11100010 10000001 10101011	⁫

37	INHIBIT ARABIC FORM SHAPING	U+206C	**3 byte**	E2 81 AC	226 129 172	14844332	11100010 10000001 10101100	⁬
38	ACTIVATE ARABIC FORM SHAPING	U+206D	**3 byte**	E2 81 AD	226 129 173	14844333	11100010 10000001 10101101	⁭
39	NATIONAL DIGIT SHAPES	U+206E	**3 byte**	E2 81 AE	226 129 174	14844334	11100010 10000001 10101110	⁮
40	NOMINAL DIGIT SHAPES	U+206F	**3 byte**	E2 81 AF	226 129 175	14844335	11100010 10000001 10101111	⁯
41	IDEOGRAPHIC SPACE	U+3000	**3 byte**	E3 80 80	227 128 128	14909568	11100011 10000000 10000000	
42	BRAILLE PATTERN BLANK	U+2800	**3 byte**	E2 A0 80	226 160 128	14852224	11100010 10100000 10000000	⠀
43	HANGUL FILLER	U+3164	**3 byte**	E3 85 A4	227 133 164	14910884	11100011 10000101 10100100	ㅤ
44	ZERO WIDTH NO-BREAK SPACE	U+FEFF	**3 byte**	EF BB BF	239 187 191	15711167	11101111 10111011 10111111	
45	HALFWIDTH HANGUL FILLER	U+FFA0	**3 byte**	EF BE A0	239 190 160	15711904	11101111 10111110 10100000	ﾠ
46	MUSICAL SYMBOL NULL NOTEHEAD	U+1D159	**4 byte**	F0 9D 85 99	240 157 133 153	4036855193	11110000 10011101 10000101 10011001	𝅙
47	MUSICAL SYMBOL BEGIN BEAM	U+1D173	**4 byte**	F0 9D 85 B3	240 157 133 179	4036855219	11110000 10011101 10000101 10110011	𝅳
48	MUSICAL SYMBOL END BEAM	U+1D174	**4 byte**	F0 9D 85 B4	240 157 133 180	4036855220	11110000 10011101 10000101 10110100	𝅴
49	MUSICAL SYMBOL BEGIN TIE	U+1D175	**4 byte**	F0 9D 85 B5	240 157 133 181	4036855221	11110000 10011101 10000101 10110101	𝅵
50	MUSICAL SYMBOL END TIE	U+1D176	**4 byte**	F0 9D 85 B6	240 157 133 182	4036855222	11110000 10011101 10000101 10110110	𝅶
51	MUSICAL SYMBOL BEGIN SLUR	U+1D177	**4 byte**	F0 9D 85 B7	240 157 133 183	4036855223	11110000 10011101 10000101 10110111	𝅷
52	MUSICAL SYMBOL END SLUR	U+1D178	**4 byte**	F0 9D 85 B8	240 157 133 184	4036855224	11110000 10011101 10000101 10111000	𝅸
53	MUSICAL SYMBOL BEGIN PHRASE	U+1D179	**4 byte**	F0 9D 85 B9	240 157 133 185	4036855225	11110000 10011101 10000101 10111001	𝅹
54	MUSICAL SYMBOL END PHRASE	U+1D17A	**4 byte**	F0 9D 85 BA	240 157 133 186	4036855226	11110000 10011101 10000101 10111010	𝅺

Example: Suppose after APPLYING Server side Analysis Algorithm we got UNIQUE ID 10024 result using JavaScript AI (invisibleCGIvisible.ai) AI File use in Image D above on **www.god.com** Domain.

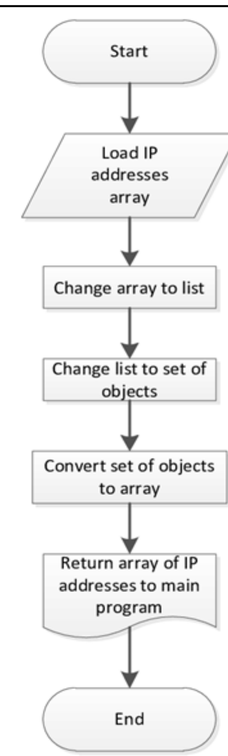
Suppose after APPLYING Server side Analysis Algorithm we got **UNIQUE ID 10024 result.**

1. UID – 10024
2. Now Shuffling **UNICODE CHARSET** Table – 1 based on UID in Loop and rearrange and generate **new customized Array Set for www.god.com and Generate TABLE-2**

TABLE-2

A	CHARACTER TABULATION	U + 0009
B	SPACE	U+0020
C	NO-BREAK SPACE	U+00A0
D	**SOFT HYPHEN**	**U+00AD**
E	COMBINING GRAPHEME JOINER	U+034F
F	ARABIC LETTER MARK	U+061C
G	**HANGUL CHOSEONG FILLER**	**U+115F**
H	HANGUL JUNGSEONG FILLER	U+1160
I	KHMER VOWEL INHERENT AQ	U+17B4
J	KHMER VOWEL INHERENT AA	U+17B5
K	MONGOLIAN VOWEL SEPARATOR	U+180E

UNIQUE ID FLOWCHART



L	EN QUAD	U+2000
M	EM QUAD	U+2001
N	EN SPACE	U+2002
O	**_EM SPACE**	**U+2003**
P	THREE-PER-EM SPACE	U+2004
Q	FOUR-PER-EM SPACE	U+2005
R	SIX-PER-EM SPACE	U+2006
S	FIGURE SPACE	U+2007
T	PUNCTUATION SPACE	U+2008
U	THIN SPACE	U+2009
V	HAIR SPACE	U+200A
W	ZERO WIDTH SPACE	U+200B
X	ZERO WIDTH NON-JOINER	U+200C
Y	ZERO WIDTH JOINER	U+200D
Z	LEFT-TO-RIGHT MARK	U+200E

1	2	3
G	**O**	**D**
01000111	01001111	01000100

5.2 Binary Adder and Subtractor for Above Calculation

We are going to look at the Binary Adder and Subtractor Circuits. We will learn about the Half Adder, Full Adder, Parallel Adder (using multiple Full Adders), and Half Subtractor, Full Subtractor and a Parallel Adder / Subtractor combination circuit.

Binary Addition Circuits

Addition and Subtraction are two basic Arithmetic Operations that must be performed by any Digital Computer. If both these operations can be properly implemented, then Multiplication and Division tasks become easy (as multiplication is repeated addition and division is repeated subtraction).
Consider the operation of adding two binary numbers, which is one of the fundamental tasks performed by a digital computer. The four basic addition operations two single bit binary numbers are:

- 0 + 0 = 0
- 1 + 0 = 1
- 0 + 1 = 1
- 1 + 1 = (Carry)1 0

	1	1	0	0
	+1	+0	+1	+0
(carry)1	0	1	1	0

In the first three operations, each binary addition gives sum as one bit , i.e., either 0 or 1. But for the fourth addition operation (where the inputs are 1 and 1), the result consists of two binary digits. Here, the lower significant bit is called as the 'Sum Bit', while the higher significant bit is called as the 'Carry Bit'.

For single bit additions, there may not be an issue. The problem may arise when we try to add binary numbers with more than one bit.

The logic circuits which are designed to perform the addition of two binary numbers are called as Binary Adder Circuits. Depending on how they handle the output of the '1+1' addition, they are divided into:

- Half Adder
- Full Adder

Let us take a look at the binary addition performed by various adder circuits.

Half Adder

A logic circuit used for adding two 1-bit numbers or simply two bits is called as a Half Adder circuit. This circuit has two inputs and two outputs. The inputs are the two 1-bit binary numbers (known as Augend and Addend) and the outputs are Sum and Carry.

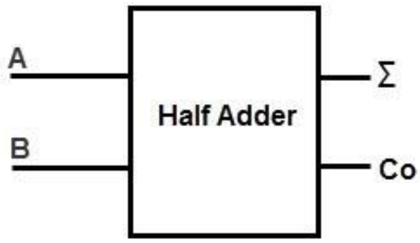
The following image shows the block diagram of Half Adder.

The truth table of the Half Adder is shown in the following table.

INPUT		**OUTPUT**	
A	**B**	**Sum**	**Carry**
0	0	0	0
0	1	**1**	0
1	0	**1**	0
1	1	0	**1**

If we observe the 'Sum' values in the above truth table, it resembles an Ex-OR Gate. Similarly, the values for 'Carry' in the above truth table resembles an AND Gate.

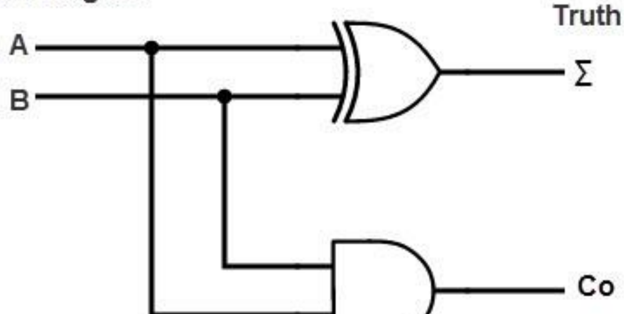
So, to properly implement a Half Adder, you need two Logic Gates: an XOR gate for 'Sum' Output and an AND gate for 'Carry' output. The following image shows the Logic Diagram of a Half Adder.



Block Diagram

A	B	Sum (∑)	Carry Out (Co)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Truth Table



Logic Diagram

In the above half adder circuit, inputs are labeled as A and B. The 'Sum' output is labeled as summation symbol (∑) and the Carry output is labeled with C_O.

Half adder is mainly used for addition of augend and addend of first order binary numbers i.e., 1-bit binary numbers. We cannot add binary numbers with more than one bit as the Half Adder cannot include the 'Carry' information from the previous sum.

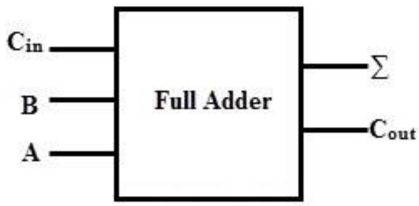
Due to this limitation, Half Adder is practically not used in many applications, especially in multi-digit addition. In such applications, carry of the previous digit addition must be added along with two bits; hence it is a three bit addition.

Full Adder

A Full Adder is a combinational logic circuit which performs addition on three bits and produces two outputs: a Sum and a Carry. As we have seen that the Half Adder cannot respond to three inputs and hence the full adder is used to add three digits at a time.

It consists of three inputs, of which two are input variables representing the two significant bits to be added, whereas the third input terminal is the carry from the previous addition. The two outputs are a Sum and Carry outputs.

The following image shows a block diagram of a Full Adder where the inputs are labelled as A, B and C_{IN}, while the outputs are labelled as ∑ and C_{OUT}.



Block Diagram of Full Adder

C_{in}	B	A	∑	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

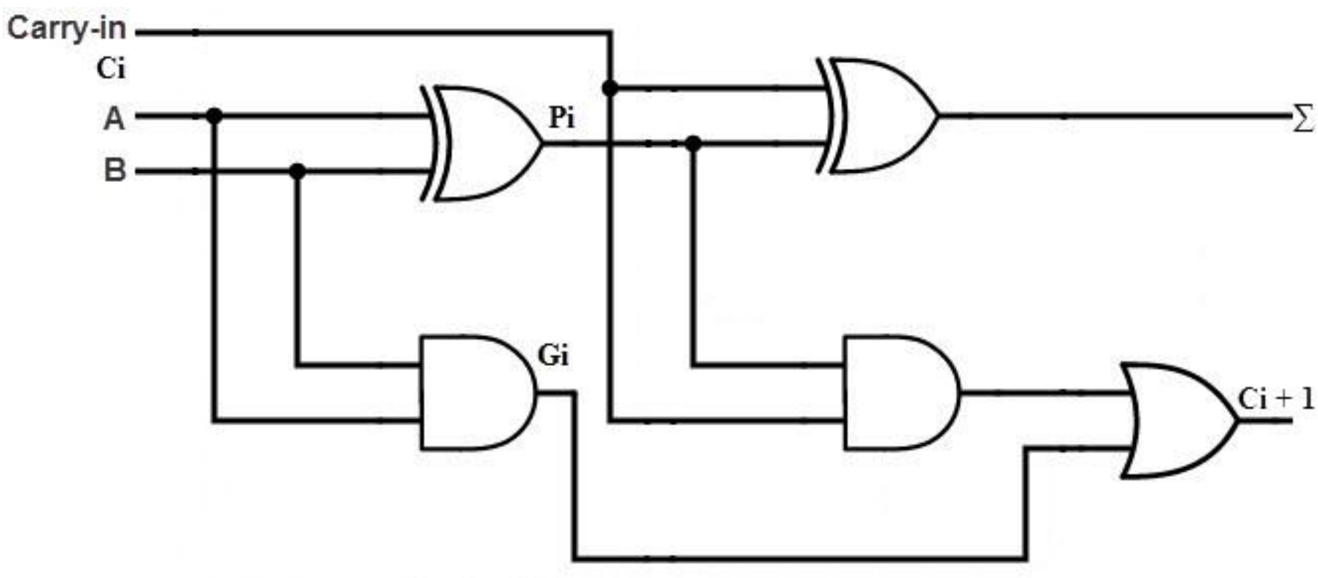
Truth Table

INPUT			OUTPUT	
A	B	C_{IN}	Sum	C_{OUT}
0	0	0	0	0
0	0	1	**1**	0
0	1	0	**1**	0
0	1	1	0	**1**
1	0	0	**1**	0
1	0	1	0	**1**
1	1	0	0	**1**
1	1	1	**1**	**1**

Coming to the truth table, the following table shows the truth table of a Full Adder.

From the above truth table, we can obtain the Boolean Expressions for both the Sum and Carry Outputs. Using those expressions, we can build the logic circuits for Full Adder. But by simplifying the equations further, we can derive at a point that a Full Adder can be easily implemented using two Half Adders and an OR Gate.

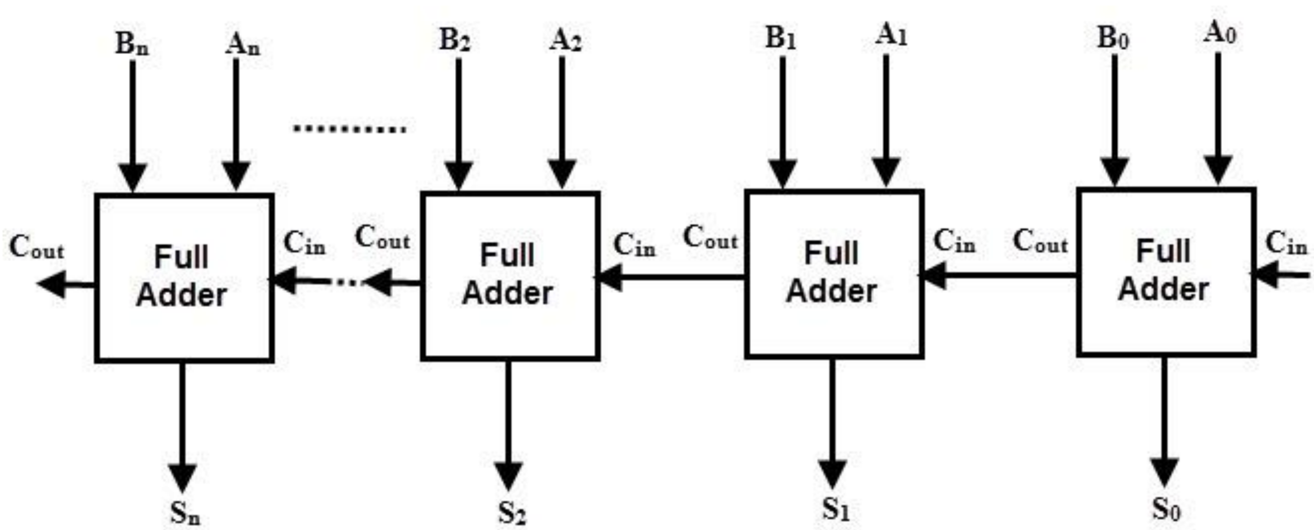
The following image shows a Full Adder Circuit implemented using two Half Adders and an OR Gate. Here, A and B are the main input bits, C_{IN} is the carry input, ∑ and C_{OUT} are the Sum and Carry Outputs respectively.



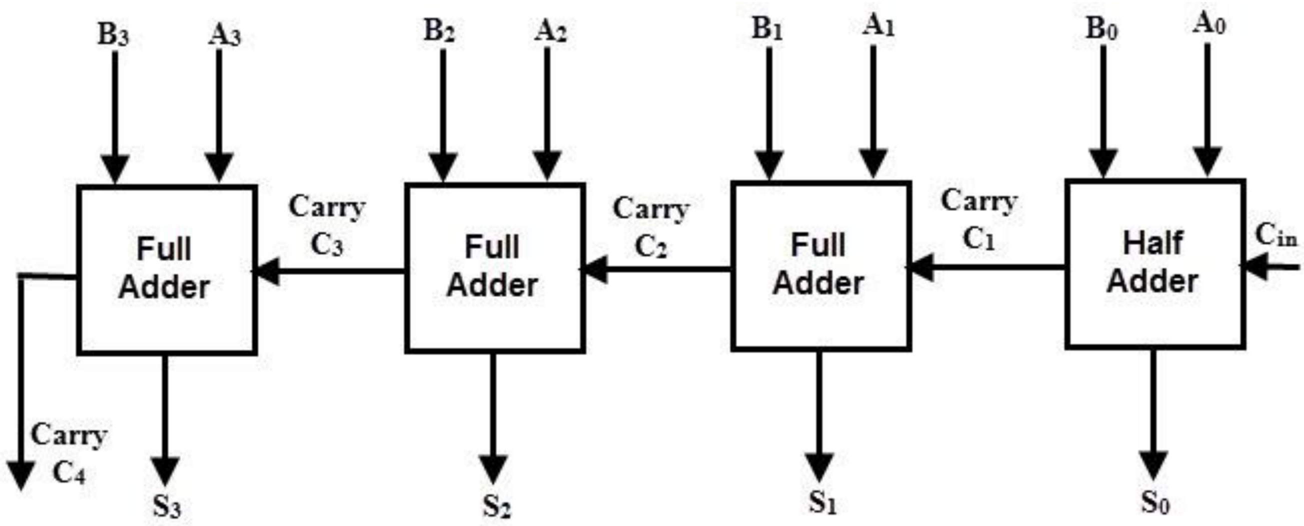
Parallel Binary Adders

As we discussed, a single Full Adder performs the addition of two one bit numbers and also the carry input. For performing the addition of binary numbers with more than one bit, more than one full adder is required and the number of Full Adders depends on the number bits. Thus, a Parallel Adder, is a combination of Multiple Full Adders and is used for adding all bits of the two numbers simultaneously.

By connecting 'n' number of full adders in parallel, an n-bit Parallel Adder can be constructed. From the below figure, it is to be noted that there is no carry at the least significant position, hence we can use either a half adder or make the carry input of full adder as zero at this position.



The following figure shows a Parallel 4-bit Binary Adder, which has three full adders and one half adder. The two binary numbers to be added are 'A_3 A_2 A_1 A_0' and 'B_3 B_2 B_1 B_0', which are applied to the corresponding inputs of the Full Adders. This parallel adder produces their result as 'C_4 S_3 S_2 S_1 S_0', where C_4 is the final carry.



In the 4 bit adder, first block is a half-adder that has two inputs as A_0 B_0 and produces a sum S_0 and a carry bit C_1. The first block can also be a full adder and if so, then the input Carry C_0 must be 0.

Next three blocks should be full adders, as there are three inputs applied to them (two main binary bits and a Carry bit from the previous stage).

Hence, the second block full adder produces a sum S_1 and a carry C_2. This will be followed by other two full adders and thus the final result is C_4 S_3 S_2 S_1 S_0.

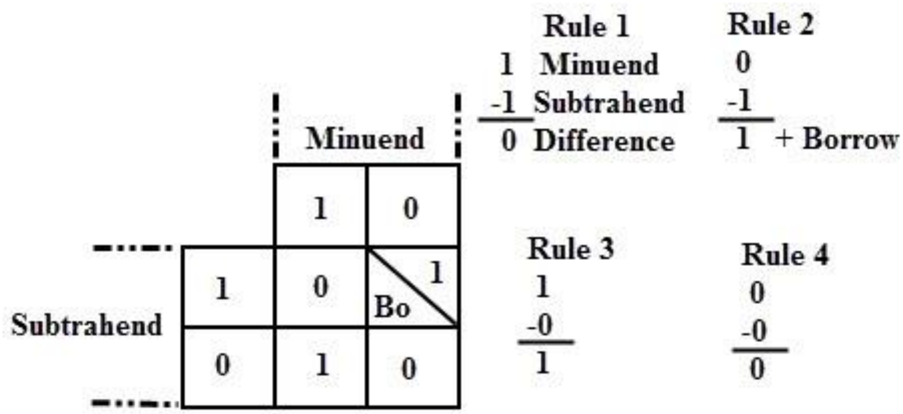
Commonly, the Full Adders are designed in dual in-line package integrated circuits. 74LS283 is a popular 4-bit full adder IC. Arithmetic and Logic Unit or ALU of an unit computer consist of these parallel adders to perform the addition of binary numbers.

Binary Subtraction Circuits

Another basic arithmetic operation to be performed by Digital Computers is the Subtraction. Subtraction is a mathematical operation in which one integer number is deducted from another to obtain the equivalent quantity. The number from which other number is to be deducted is called as 'Minuend' and the number subtracted from the minuend is called 'Subtrahend'.

Similar to the binary addition, binary subtraction is also has four possible basic operations. They are:

- 0 – 0 = 0
- 0 – 1 = (Borrow)1 1
- 1 – 0 = 1
- 1 – 1 = 0



The above figure shows the four possible rules or elementary operations of the binary subtractions. In all the operations, each subtrahend bit is deducted from the minuend bit.

But in the second rule, minuend bit is smaller than the subtrahend bit, hence 1 is borrowed to perform the subtraction. Similar to the adder circuits, basic subtraction circuits are also of two types:

- Half Subtractor
- Full Subtractor

Half Subtractors

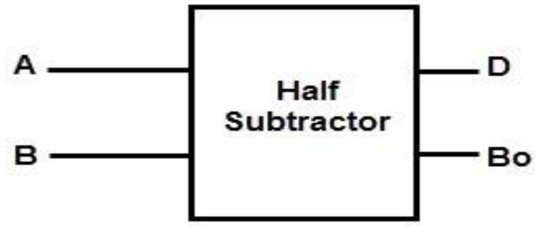
A Half Subtractor is a multiple output Combinational Logic Circuit that does the subtraction of two 1-bit binary numbers. It has two inputs and two outputs. The two inputs correspond to the two 1-bit binary numbers and the two outputs corresponds to the Difference bit and Borrow bit (in contrast to Sum and Carry in Half Adder).

Following table shows the truth table of a Half Subtractor.

INPUT		**OUTPUT**	
A	**B**	**Difference**	**Borrow**
0	0	0	0
0	1	**1**	**1**
1	0	**1**	0
1	1	0	0

From the above truth table, we can say that the ‘Difference’ output of the Half Subtractor is similar to an XOR output (which is also same as the Sum output of the Half Adder). Thus, the Half Subtraction is also performed by the Ex-OR gate with an AND gate with one inverted input and one normal input, requiring to perform the Borrow operation.

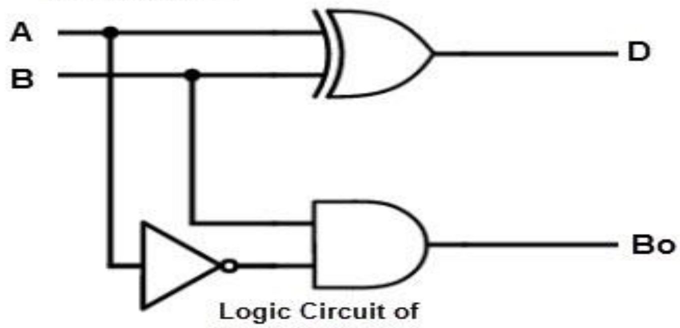
The following image shows the logic circuit of a Half Adder.



Block Diagram

A	B	D	B_o
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Truth Table



Logic Circuit of Half Subtractor

This circuit is similar to that of the Half Adder with only difference being the minuend input i.e., A is complemented before applied at the AND gate to implement the borrow output.

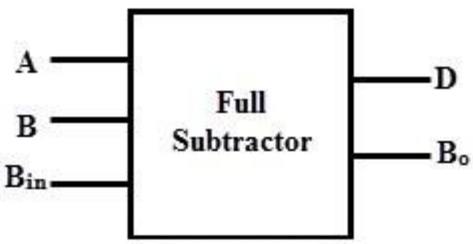
In case of multi-digit subtraction, subtraction between the two digits must be performed along with borrow of the previous digit subtraction, and hence a subtractor needs to have three inputs, which is not possible with a Half Subtractor. Therefore, a half subtractor has limited set of applications and strictly speaking, it is not used in practice.

Full Subtractor

A Full Subtractor is a combinational logic circuit which performs a subtraction between the two 1-bit binary numbers and it also considers the borrow of the previous bit i.e., whether 1 has been borrowed by the previous minuend bit.

So, a Full Subtractor has three inputs, in which two inputs corresponding to the two bits to be subtracted (minuend A and subtrahend B), and a borrow bit, usually represented as B_{IN}, corresponding to the borrow operation. There are two outputs, one corresponds to the difference D output and the other Borrow output B_O.

The following image shows the block diagram of a full subtractor.



Block Diagram of Full Subtractor

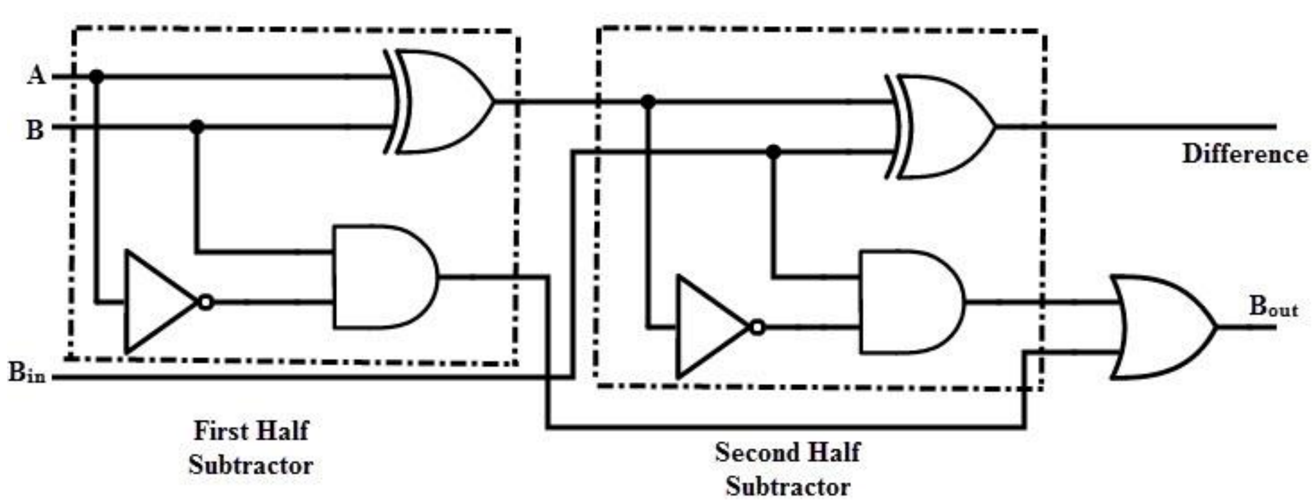
A	B	B_{in}	D	B_o
0	0	0	**0**	**0**
0	0	1	**1**	**1**
0	1	0	**1**	**1**
0	1	1	**0**	**1**
1	0	0	**1**	**0**
1	0	1	**0**	**0**
1	1	0	**0**	**0**
1	1	1	**1**	**1**

Truth Table

The following table shows the truth table of a Full Subtractor.

INPUT			
A	**B**	**B_{IN}**	**D**
0	0	0	0
0	0	1	**1**
0	1	0	**1**
0	1	1	0
1	0	0	**1**
1	0	1	0
1	1	0	0
1	1	1	**1**

By deriving the Boolean expression for the full subtractor from above truth table, we get the expression that tells that a full subtractor can be implemented with half subtractors with OR gate as shown in figure below.



By comparing the adder and subtractor circuits and truth tables, we can observe that the output D in the full subtractor is exactly same as the output S of the full adder. And the only difference is that input variable A is complemented in the full subtractor.

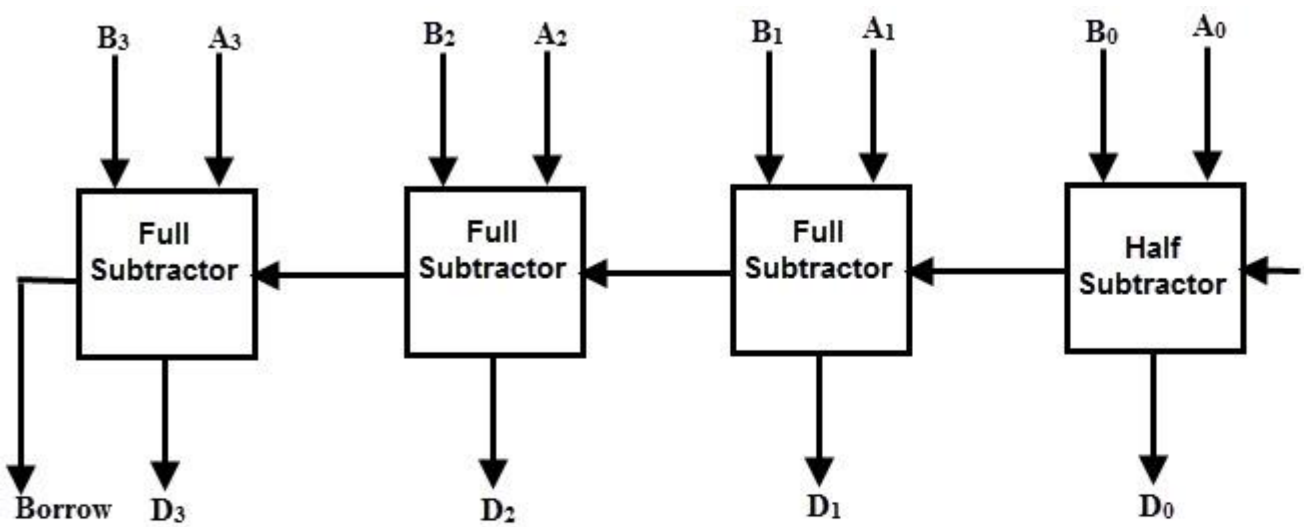
Therefore, it is possible to convert the full adder circuit into full subtractor by simply complementing the input A before it is applied to the gates to produce the final borrow bit output Bo.

Parallel Binary Subtractors

To perform the subtraction of binary numbers with more than one bit, we have to use the Parallel Subtractors. This parallel subtractor can be designed in several ways, including combination of half and full subtractors, all full subtractors, all full adders with subtrahend complement input, etc.

The below figure shows a 4 bit Parallel Binary Subtractor formed by connecting one half subtractor and three full subtractors.

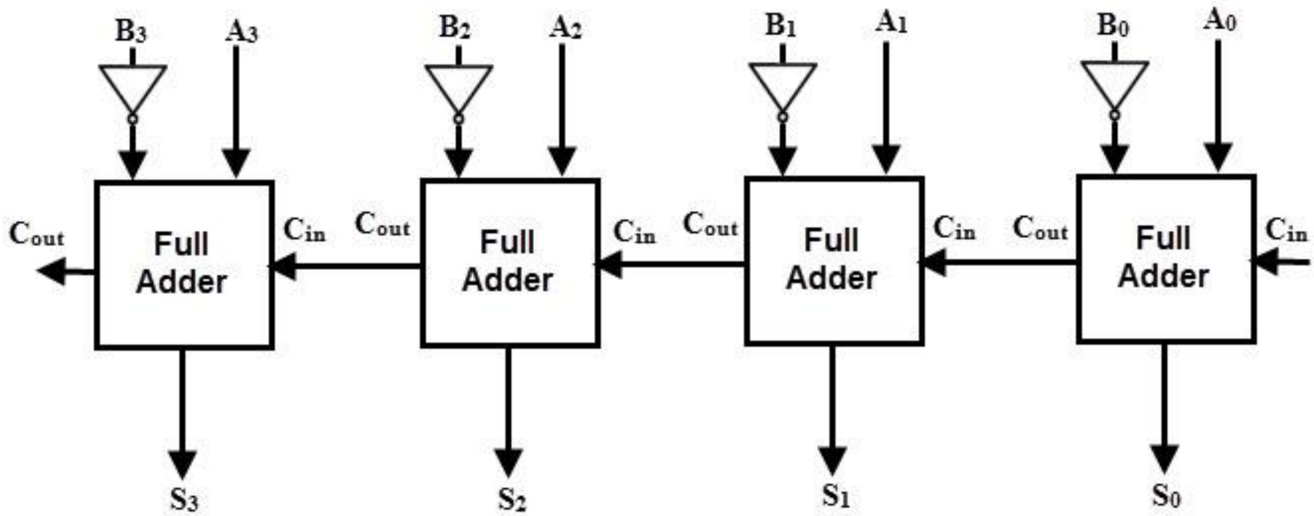
In this subtractor, 4 bit minuend 'A_3 A_2 A_1 A_0' is subtracted by 4 bit subtrahend 'B_3 B_2 B_1 B_0' and the result is the difference output 'D_3 D_2 D_1 D_0'. The borrow output of each subtractor is connected as the borrow input to the next subtractor.



It is also possible to design a 4 bit parallel subtractor using 4 full adders as shown in the below figure. This circuit performs the subtraction operation by considering the principle that the addition of minuend and the complement of the subtrahend is equivalent to the subtraction process.

We know that the subtraction of A by B is obtained by taking 2's complement of B and adding it to A. The 2's complement of B is obtained by taking 1's complement and adding 1 to the least significant pair of bits.

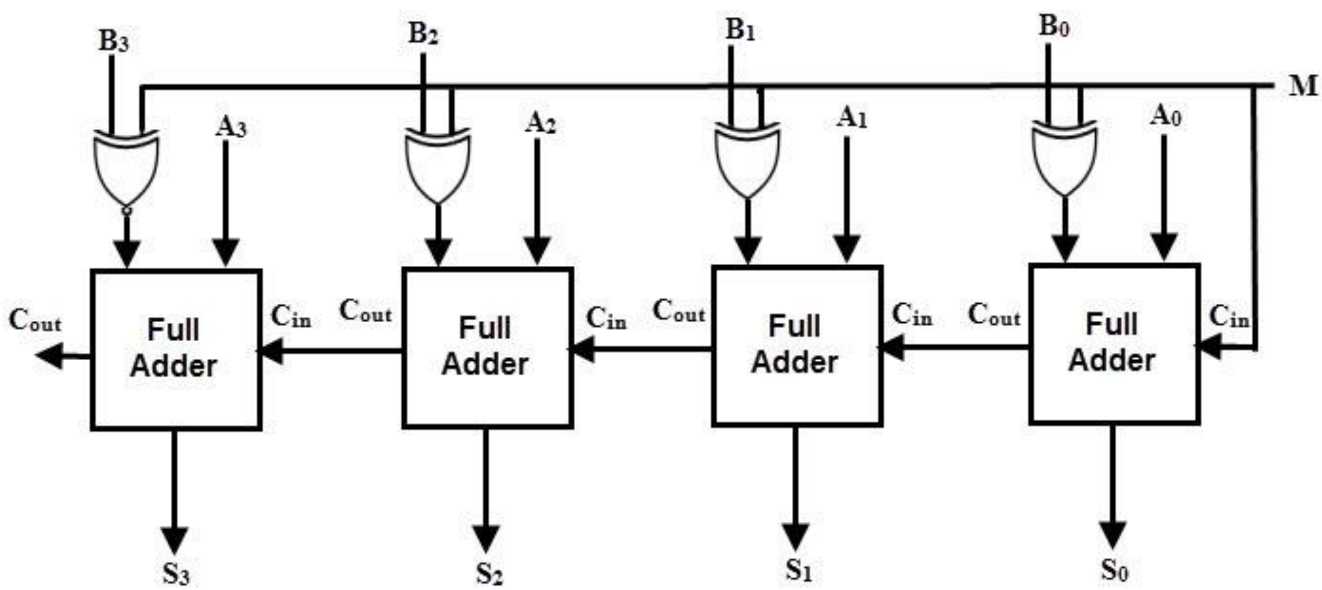
Hence, in this circuit 1's complement of B is obtained with the inverters (NOT gate) and a 1 can be added to the sum through the input carry.



Parallel Adder / Subtractor

The operations of both addition and subtraction can be performed by a one common binary adder. Such binary circuit can be designed by adding an Ex-OR gate with each full adder as shown in below figure. The figure below shows the 4 bit parallel binary adder/subtractor which has two 4 bit inputs as 'A_3 A_2 A_1 A_0' and 'B_3 B_2 B_1 B_0'.

The mode input control line M is connected with carry input of the least significant bit of the full adder. This control line decides the type of operation, whether addition or subtraction.



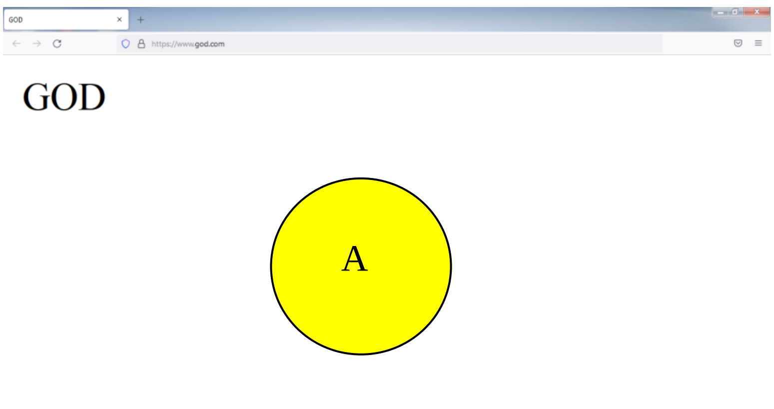
When M= 1, the circuit is a subtractor and when M=0, the circuit becomes adder. The Ex-OR gate consists of two inputs to which one is connected to the B and other to input M. When M = 0, B Ex-OR of 0 produce B. Then, full adders add the B with A with carry input zero and hence an addition operation is performed.

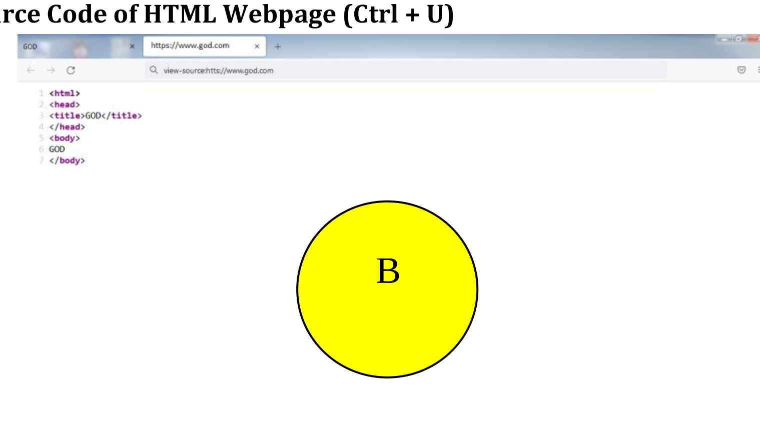
When M = 1, B Ex-OR of 0 produce B complement and also carry input is 1. Hence the complemented B inputs are added to A and 1 is added through the input carry, nothing but a 2’s complement operation. Therefore, the subtraction operation is performed.

6. Experimental Result

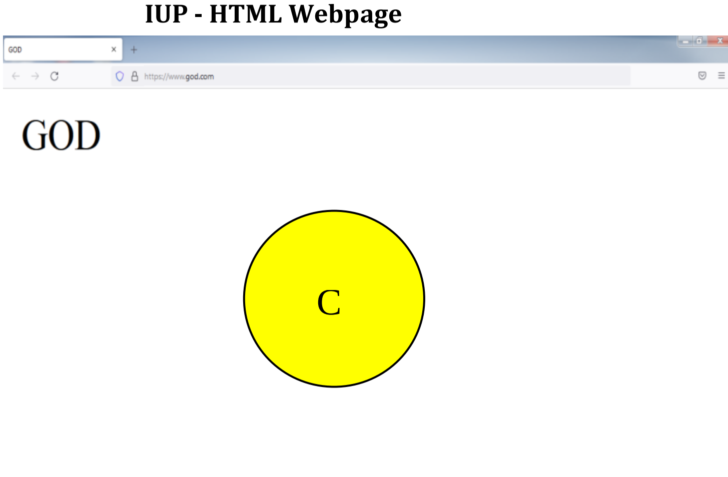
Practical Example of Invisible Unicode Programming
Normal HTML Webpage Example

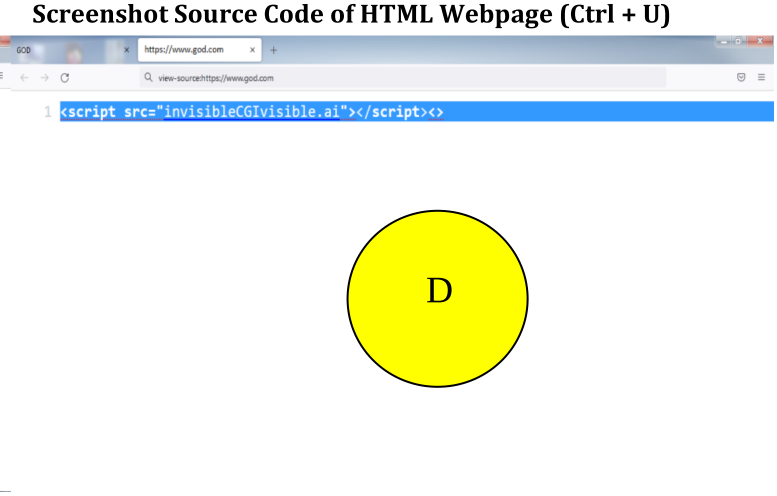
Screenshot OUTPUT ON BROWSER **Screenshot Source Code of HTML Webpage (Ctrl + U)**





IUP - HTML Webpage **Screenshot Source Code of HTML Webpage (Ctrl + U)**





Implementation Algorithm of IUP

First: Suppose after APPLYING Server side Analysis Algorithm we got UNIQUE ID 10024 result using JavaScript AI (invisibleCGIvisible.ai) AI File use in Image D above **on www.god.com Domain.**

3. UID – 10024
4. Now Shuffling **UNICODE CHARSET** Table – 1 based on UID in Loop and rearrange and generate **new customized Array set for www.god.com and Generate TABLE-2**

Logic to get MBV (Masking Binary Value)
And Logic to make Visible and Invisible character using (MBV)

1. ASCII BINARY + INVISIBLE UNICODE BINARY = Masking Binary Value (MBV)
2. ASCII BINARY – Masking Binary Value = INVISIBLE UNICODE BINARY
3. MBV – IUB = ASCII Character

➔ ‘G’

ASCII BINARY + INVISIBLE UNICODE BINARY = Masking Binary Value (MBV)

01000111 **(‘G’ From Table - 3)** + 11100001 10000101 10011111 **(‘G’ From Table - 2) =**
0111000011000010111100110

Characters	Binary
‘G’	01000111
Masking Binary Value	**0111000011000010111100110**
U+ 115F (Unicode Invisible Character)	11100001 10000101 10011111

➔ ENCODE ‘G’ to Invisible ‘G’

0111000011000010111100110 – 01000111 = 11100001 10000101 10011111

MBV –‘G’	‘G’	INVISIBLE UNICODE CHARACTER of ‘G’

➔ DECODE Invisible ‘G’ to Visible ‘G’

0111000011000010111100110 – **111000011000010110011111** = 01000111

MBV –‘G’	INVISIBLE UNICODE CHARACTER of ‘G’	‘G’

➔ ‘O’

Characters	Binary
‘O’	01001111
Masking Binary Value	**01110001010000000011010010**
U+ 2003 (Unicode Invisible Character)	11100010 10000000 10000011

➔ ‘D’

Characters	Binary
‘D’	01000100
Masking Binary Value	**01100001011110001**
U+ 00AD (Unicode Invisible Character)	11000010 10101101

7. Conclusion

IUP is revolutionary technology and science to add additional invisible obfuscation layer, it is a built-in security method, sometimes referred to as application self-protection, it is also an additional layer of security in digital world, so that we can Create Computer Document like word file, Excel file, PowerPoint File, Video File, Audio File, Images etc all type of File is convert into invisible file.

Reference:

[1] D. Parnas, “On the Criteria to Be Used in Decomposing Systems Into Modules”, Communication of the ACM , vol. 15, no. 12, December 1972, pp. 1053-1058.

[2] W. Bender, D. Gruhl, N. Morimoto, and A. Lu, “Techniques for data hiding”, IBM syst. J., vol. 35, nos. 3 – 4, 1996, , pp. 313 – 336.

[3] J. Brassil, S. Low, N. F. Maxemchuk, and L. O'Garman., “Marking Text Features of Document Images to Deter Illicit Dissemination”, IEEE, 1994, pp. 315-319.

[4] J. Brassil, S. Low, N. F. Maxemchuk, and L. O'Garman, “Electronic Marking and Identification Techniques to Discourage Document Copying ”, IEEE, Oct. 1994, pp. 1278-1287.

[5] J. Brassil, S. Low, N. F. Maxemchuk, and L. O'Garman. “Electronic Marking and Identification Techniques to Discourage Document Copying”. IEEE Journal on Selected Areas in Communications , Vol. 13, Oct. 1995, pp. 1495-1504.

[6] J. Brassil, S. Low, N. F. Maxemchuk, and L. O'Garman, “Copyright Protection for the Electronic Distribution of Text Documents”, Proceedings of IEEE, Vol. 87, July 1999.

[7] J. Brassil, S. Low, N. F. Maxemchuk, and L. O'Garman. "Document Marking and Identification using Both Line and Word Shifting," Proc. Infocom95, IEEE CS Press, Los Alamitos, Calif., 1995.

[8] Chen Chao, Wang Shuozhong, Zhang Xinpeng. "Information Hiding In Text Using Typesetting Tools with Stego-Encoding", ICICIC, 2006.

[9] P. Wayner, "Mimic functions", Cryptologia archive, vol. 16, Issue 3, July 1992, pp.193 – 214.

[10] P. Wayner. " Strong Theoretical Steganography". Cryptologia, XIX (3), July 1995, pp. 285-299.

[11] M. Chapman, G. Davida. "Hiding the Hidden: A Software System for Concealing Ciphertext as Innocuous Text ". Master Thesis, Milwaukee: University of Wisconsin-Milwaukee, 1998.

[12] "Spammimic", 2000. [Online] Available: http://www.spammimic.com [Accessed: March 8, 2008].

[13] N. F. Johnson, S. Jajodia, "Exploring Steganography: Seeing the Unseen," IEEE Computer, February 1998, pp.26–34.

[14] P. Wayner, Disappearing Cryptography: Being and Nothingness on the Net, Academic Press, Inc., 1996.

[15] Richard Bergmair, "Towards Linguistic Steganography: A Systematic Investigation of Approaches", Systems, and Issues, Technical Report, Nov 2004.

[16] "Spy Gadgets in World War II: Microdots", 2007. [Online]. Available: http://www.mi5.gov.uk/output/Page303.html [Accessed: Feb. 15, 2008].

[17] N. Provos, P. Honeyman, "Hide and Seek: An Introduction to Steganography", The IEEE Computer Security, 2003.

[18] Fabien A. P. Petitcolas, Ross J. Anderson and Markus G. Kuhn. "Information Hiding – A Survey", Proceedings of the IEEE, special issue on protection of multimedia content, July 1999, pp. 1062 – 1078.

[19] Bret Dunbar, " A Detailed Look At Steganographic Techniques and Their Use in Open-Systems Environment", SANS Institute, 2002.

[20] B. Pfiztzmann, "Information Hiding Terminology." pp. 347-350, ISBN 3-540-61996-8, results of an informal plenary meeting and additional proposals, 1996.

[21] K. Bennett, "Linguistic Steganography: Survey, Analysis, and Robustness Concerns for Hiding Information in Text", Purdue University, CERIAS Tech. Report, 2004.

[22] Dr. Mohammed Al-Mualla and Prof. Hussain Al-Ahmad, "Information Hiding: Steganography and Watermarking". [Online]. Available: http://www.emirates.org/ieee/information_hiding.pdf [Accessed: March 12, 2008].

[23] Matthew Kwan, "The SNOW Home Page", 1998. [Online].Available: http://www.darkside.com.au/snow/ [Accessed: March 12, 2008].

[24] Sabu M. Thampi. "Information Hiding Techniques: A Tutorial Review", ISTE-STTP on Betwork Security & Cryptography, LBSCE 2004.

[25] M. Chapman, G. Davida, and M. Rennhard, "A Practical and Effective Approach to Large-Scale Automated Linguistic Steganography", Proceedings of the Information Security Conference, October 2001, pp. 156-165.

[26] Nursery Rhymes - lyrics and origins. [Online].Available:http://www.famousquotes.me.uk/nursery_rhymes/nursery_rhymes_index.htm [Accessed: March 31, 2008]

[27] M. Hassan Shirali-Shahreza, Mohammad Shirali-Shahreza. "Text Steganography in Chat", IEEE, 2007.

[28] M. Hassan Shirali-Shahreza, Mohammad Shirali-Shahreza. "Text Steganography in SMS", International Conference on Convergence Information Technology, 2007.

[29] K. Beare, "SMS-Texting", English as 2nd Language. [Online]. Available: http://esl.about.com/ [Accessed: March 10 2008].

[30] "Bits, Bytes and Bandwidth Reference Guide". [Online]. Available: http://www.speedguide.net/read_articles.php?id=115 [Accessed: 22 April 2008].

[31] Osamu Takizawa, Akihiro Yamamura, Hiroshi Nakagawa, Tsutomu Matsumoto, Ichiro Murase, Kyoko Makino, Shingo Inoue and Hiroyuki Ohno, "A Proposal of Steganography on Plain Text and XML", The 1st NLP and XML Workshop, Nov 2001.

[32] Chen Chao, Wang Shuozhong, Zhang Xinpeng. "Information Hiding In Text Using Typesetting Tools with Stego-Encoding", ICICIC, 2006.