Practical – 01

AIM: Design a Finite State Machine (FSM) that accepts all strings over input symbols {0, 1} having three consecutive 1's as a substring.

As per the AIM, set of valid strings are represented by set A:

A = {111, 0111, 1110, 0101011110101,...}

means any string should be declared valid if it contains 111 as a substring. Let M be the machine for above AIM, hence it can be define as M(Q, Σ, δ, q0, F) where

Q: set of states: {A, B, C, D}

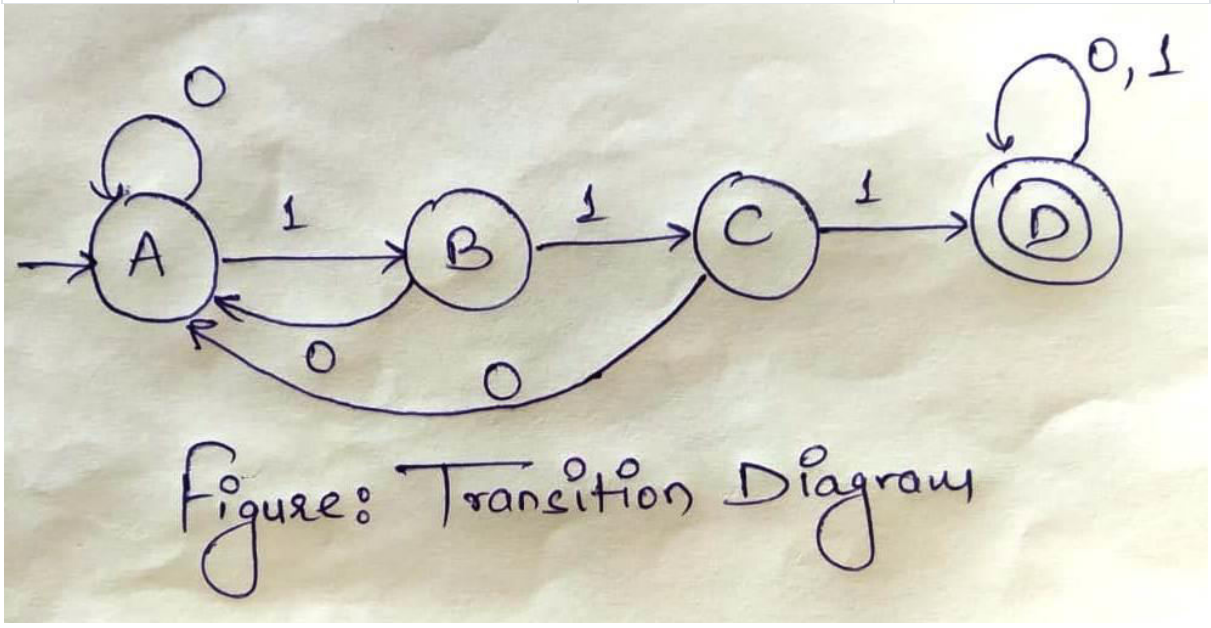
Σ: set of input symbols: {0, 1}

q0: initial state (A)

F: set of Final states: {D}

δ: Transition Function: (Transition state diagram is shown in Figure 1.)

State	Input	
-	0	1
A	A	B
B	A	C
C	A	D
D	D	D



Transition Diagram

Figure 1: FSM - accepting three consecutive 1's as a substring.

Code in C++

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
void main()
{
        char Input[100];
        clrscr();
        cout<<"Enter a string to validate (input string should be of 0 and
1)\n";
        gets(Input);
        int i=-1;
        A:
                i++;
                if(Input[i]=='0')
                {
                        goto A;
                }
                else if(Input[i]=='1')
                {
                        goto B;
                }
                else if(Input[i]=='\0')
                {
                        goto Invalid;
                }
                else
                {
                        goto Wrong;
                }
        B:
                i++;
                if(Input[i]=='0')
                {
                        goto A;
                }
                else if(Input[i]=='1')
                {
                        goto C;
                }
                else if(Input[i]=='\0')
                {
                        goto Invalid;
                }
                else
                {
                        goto Wrong;
                }
        C:
                i++;
                if(Input[i]=='0')
                {
                        goto A;
                }
```

```
                else if(Input[i]=='1')
                {
                        goto D;
                }
                else if(Input[i]=='\0')
                {
                        goto Invalid;
                }
                else
                {
                        goto Wrong;
                }
        D:
                i++;
                if(Input[i]=='0')
                {
                        goto D;
                }
                else if(Input[i]=='1')
                {
                        goto D;
                }
                else if(Input[i]=='\0')
                {
                        goto Valid;
                }
                else
                {
                        goto Wrong;
                }
        Valid:
                cout<<"\n Output: Valid String";
                goto exit;
        Invalid:
                cout<<"\n Output: Invalid String";
                goto exit;
        Wrong:
                cout<<"\n Please enter binary string {format of 0, 1}";
        exit:
                getch();
}
```

Practical - 02

AIM: Design a Finite State Machine (FSM) that accepts all strings over input symbols {0, 1} which are divisible by 3.

Discussion:

As per the AIM, set of valid strings are represented by set A:

A = {0, 00, 000, 11, 011, 110, ...}

means any binary string that when divide by three gives remainder zero. Let M be the machine for above AIM, hence it can be define as M(Q, Σ, δ, q0, F) where

Q: set of states: {q, q0, q1, q2}

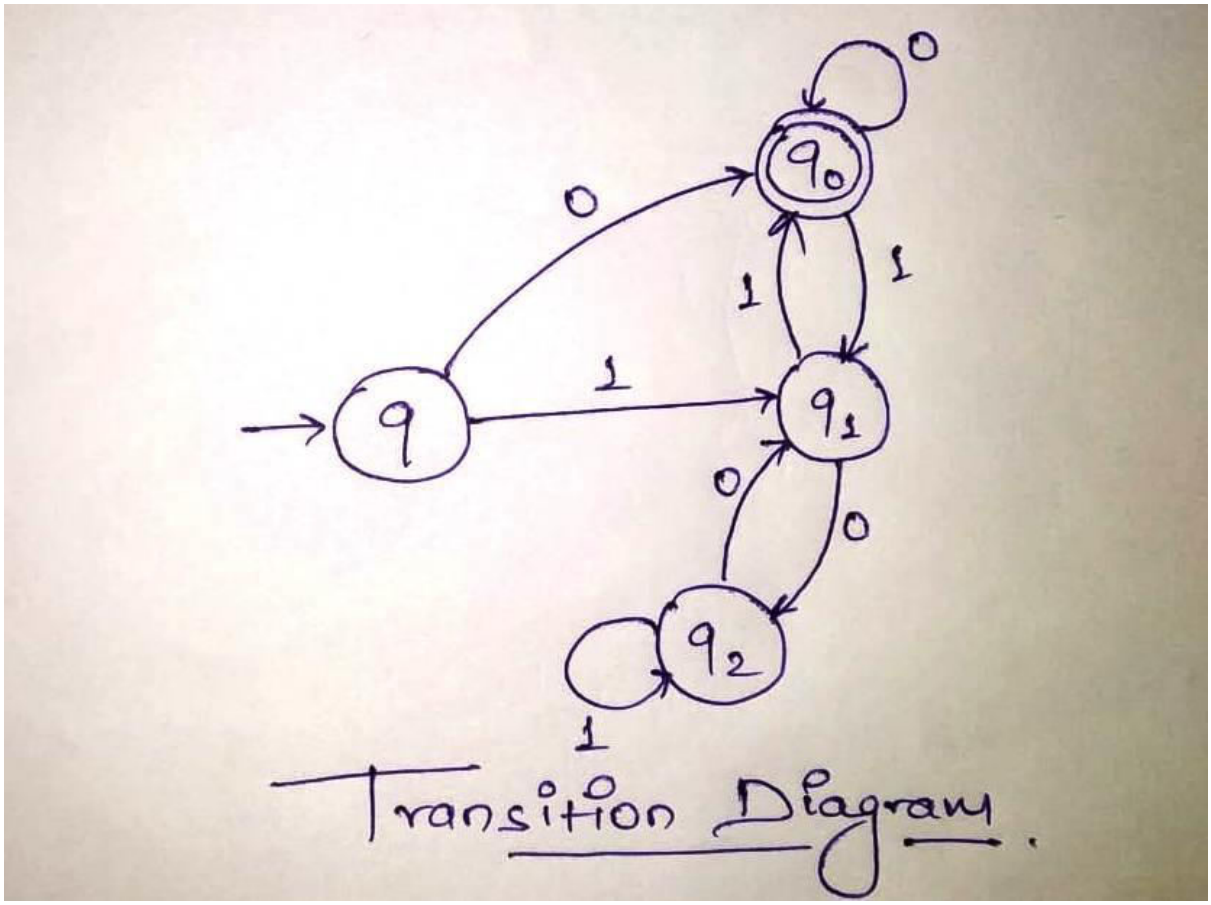
Σ: set of input symbols: {0, 1}

q0: initial state (q)

F: set of Final states: {q0}

δ: Transition Function: (Transition state diagram is shown in Figure 1.)

State	Input	
-	0	1
q	q0	q1
q0	q0	q1
q1	q2	q0
q2	q1	q2



Transition Diagram

Figure 1: FSM - accepting binary string if divisible by 3.

Code in C++

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
void main()
{
        char Input[100];
        clrscr();
        cout<<"Enter a string to validate (input string should be of 0 and
1)\n";
        gets(Input);
        int i=-1;
        q:
                i++;
                if(Input[i]=='0')
                {
                        goto q0;
                }
                else if(Input[i]=='1')
```

```
                {
                        goto q1;
                }
                else if(Input[i]=='\0')
                {
                        goto Invalid;
                }
                else
                {
                        goto Wrong;
                }
        q0:
                i++;
                if(Input[i]=='0')
                {
                        goto q0;
                }
                else if(Input[i]=='1')
                {
                        goto q1;
                }
                else if(Input[i]=='\0')
                {
                        goto Valid;
                }
                else
                {
                        goto Wrong;
                }
        q1:
                i++;
                if(Input[i]=='0')
                {
                        goto q2;
                }
                else if(Input[i]=='1')
                {
                        goto q0;
                }
                else if(Input[i]=='\0')
                {
                        goto Invalid;
                }
                else
                {
                        goto Wrong;
                }
        q2:
                i++;
                if(Input[i]=='0')
                {
                        goto q1;
                }
                else if(Input[i]=='1')
                {
                        goto q2;
```

```
                }
                else if(Input[i]=='\0')
                {
                        goto Invalid;
                }
                else
                {
                        goto Wrong;
                }
        Valid:
                cout<<"\n Output: Valid String";
                goto exit;
        Invalid:
                cout<<"\n Output: Invalid String";
                goto exit;
        Wrong:
                cout<<"\n Please enter binary string {format of 0, 1}";
        exit:
                getch();
}
```

Practical - 03

AIM: Design a Finite State Machine (FSM) that accepts all decimal string which are divisible by 3.

Discussion:

As per the AIM, set of valid strings are represented by set A:

A = {0, 3, 6, 9, 03, 06, 09, 12, 012, ..}

means any decimal number string that when divided by three gives remainder zero. Let M be the machine for above AIM, hence it can be define as M(Q, Σ, δ, q0, F) where

Q: set of states: {q, q0, q1, q2}

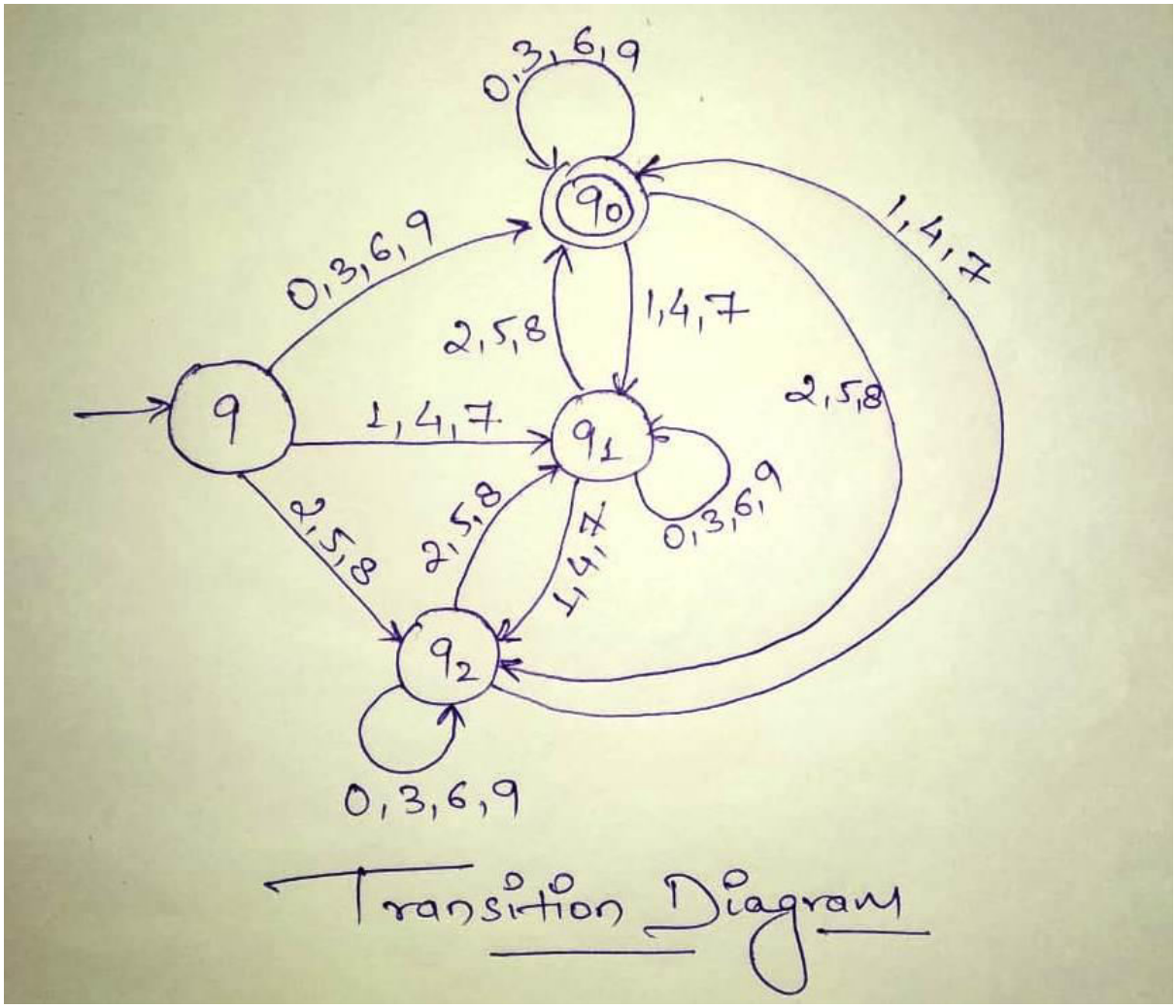
Σ: set of input symbols: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

q0: initial state (q)

F: set of Final states: {q0}

δ: Transition Function: (Transition state diagram is shown in Figure 1.)

State	Input		
-	0, 3, 6, 9	1, 4, 7	2, 5, 8
q	q0	q1	q2
q0	q0	q1	q2
q1	q1	q2	q0
q2	q2	q0	q1



Transition Diagram

Figure 1: FSM - accepting decimal string if divisible by 3.

Code in C++

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
void main()
{
        char Input[100];
        clrscr();
        cout<<"Enter a string to validate (input string should be decimal
number (i.e constructed from 0,1,2,3,4,5,6,7,8,9 digits)\n";
        gets(Input);
        int i=-1;
        q:
                i++;
```

```
                if(Input[i]=='0'|| Input[i]=='3'|| Input[i]=='6'||
Input[i]=='9')
                {
                        goto q0;
                }
                else if(Input[i]=='1'|| Input[i]=='4'|| Input[i]=='7')
                {
                        goto q1;
                }
                else if(Input[i]=='2'|| Input[i]=='5'|| Input[i]=='8')
                {
                        goto q2;
                }
                else if(Input[i]=='\0')
                {
                        goto Invalid;
                }
                else
                {
                        goto Wrong;
                }
        q0:
                i++;
                if(Input[i]=='0'|| Input[i]=='3'|| Input[i]=='6'||
Input[i]=='9')
                {
                        goto q0;
                }
                else if(Input[i]=='1'|| Input[i]=='4'|| Input[i]=='7')
                {
                        goto q1;
                }
                else if(Input[i]=='2'|| Input[i]=='5'|| Input[i]=='8')
                {
                        goto q2;
                }
                else if(Input[i]=='\0')
                {
                        goto Valid;
                }
                else
                {
                        goto Wrong;
                }
        q1:
                i++;
                if(Input[i]=='0'|| Input[i]=='3'|| Input[i]=='6'||
Input[i]=='9')
                {
                        goto q1;
                }
                else if(Input[i]=='1'|| Input[i]=='4'|| Input[i]=='7')
                {
                        goto q2;
                }
                else if(Input[i]=='2'|| Input[i]=='5'|| Input[i]=='8')
```

```
                {
                        goto q0;
                }
                else if(Input[i]=='\0')
                {
                        goto Invalid;
                }
                else
                {
                        goto Wrong;
                }
        q2:
                i++;
                if(Input[i]=='0'|| Input[i]=='3'|| Input[i]=='6'||
Input[i]=='9')
                {
                        goto q2;
                }
                else if(Input[i]=='1'|| Input[i]=='4'|| Input[i]=='7')
                {
                        goto q0;
                }
                else if(Input[i]=='2'|| Input[i]=='5'|| Input[i]=='8')
                {
                        goto q1;
                }
                else if(Input[i]=='\0')
                {
                        goto Invalid;
                }
                else
                {
                        goto Wrong;
                }
        Valid:
                cout<<"\n Output: Valid String";
                goto exit;
        Invalid:
                cout<<"\n Output: Invalid String";
                goto exit;
        Wrong:
                cout<<"\n Please enter valid decimal number string";
        exit:
                getch();
}
```

Practical - 04

AIM: Design a Push Down Automata (PDA) that accepts all string having equal number of 0's and 1's over input symbol {0, 1} for a language 0^n1^n where n >= 1.

Discussion:

As per the AIM, set of valid strings that can be generated by given language is represented in set A:

A = {01, 0011, 000111, ...}

means all string having n number of 0's followed by n numbers of 1's where n can be any number greater than equal tp and count of 0's must be equal to count of 1's. Block diagram of push down automata is shown in Figure 1.

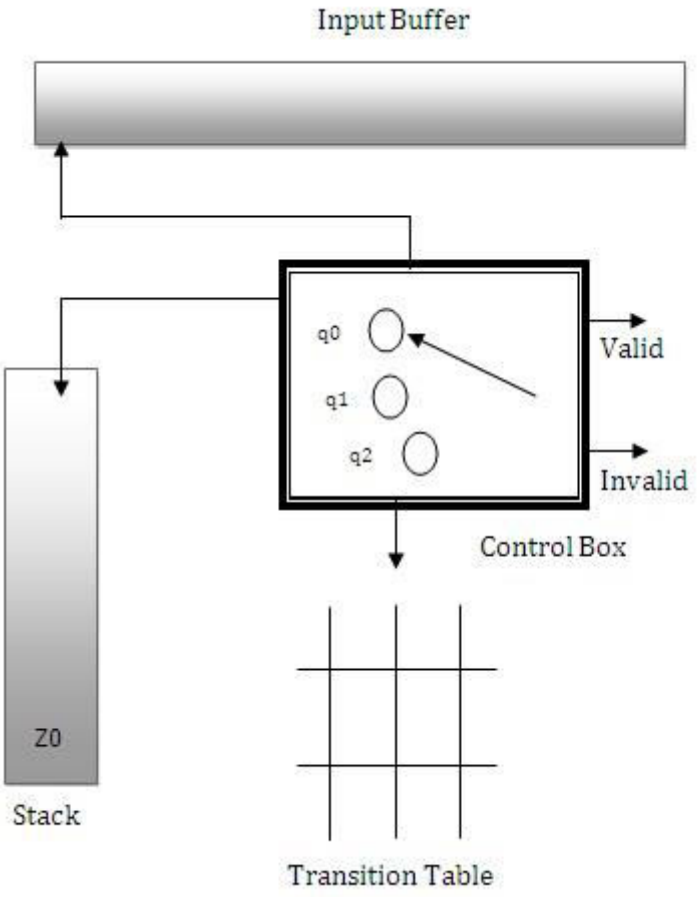


Figure 1: Block Diagram of Push Down Automata.

Input string can be valid or invalid, valid if it follows the language 0^n1^n where n >= 1 else invalid. PDA has to determine whether the input string is according to the language or not.

Let M be the PDA machine for above AIM, hence it can be define as M(Q, Σ, Γ, δ, q0, Z0, F) where

Q: set of states: {q0, q1, q2}

Σ: set of input symbols: {0, 1}

Γ: Set of stack symbols: {A, Z0}

q0: initial state (q0)

Z0: initial stack symbol (Z0)

F: set of Final states: { } [Note: Here, set of final states is null as decision of validity of string is based on stack whether it is empty or not.]

δ: Transition Function: (Transition state diagram is shown in Figure 2.)

$$\delta(q0, 0, Z0) \rightarrow (q0, AZ0)$$

$$\delta(q0, 0, 0) \rightarrow (q0, 00)$$

$$\delta(q0, 1, 0) \rightarrow (q1, \varepsilon)$$

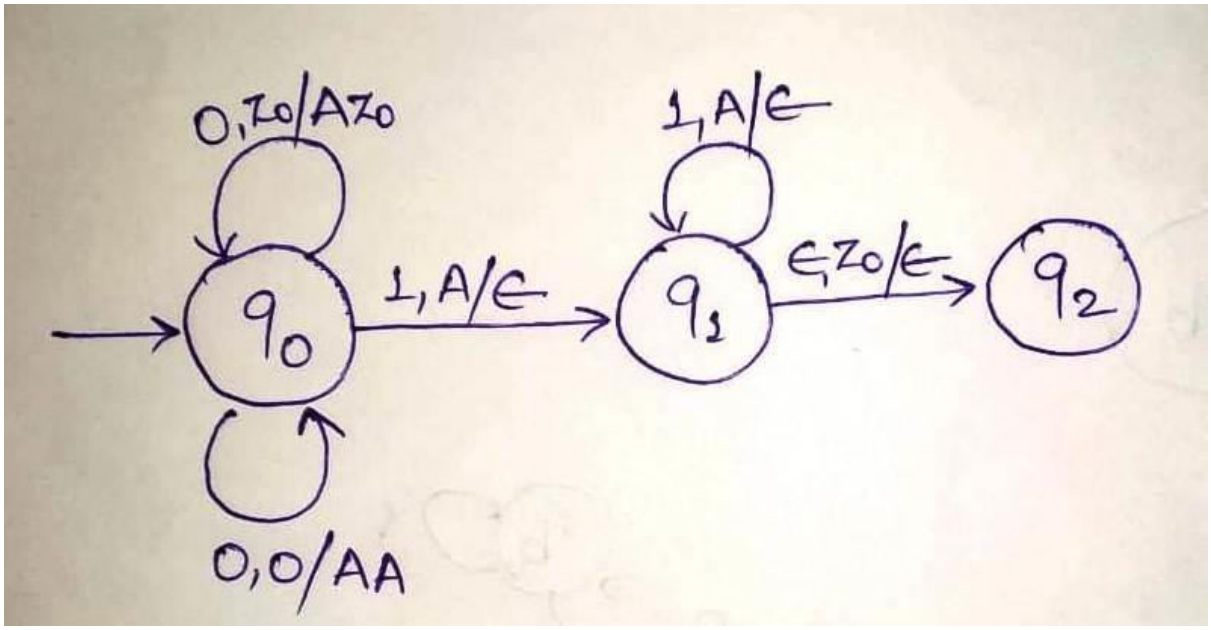
$$\delta(q1, 1, 0) \rightarrow (q1, \varepsilon)$$

$$\delta(q1, \varepsilon, Z0) \rightarrow (q2, \varepsilon)$$

Rules for implementing PDA for a given language

Initial Setup: Load the input string in input buffer, push Z0 as an initial symbol in stack and consider the machine at initial state q0.
Rules:

1. **It is must that the first symbol should be 0. If the first symbol of input string is zero then push symbol 'A' into stack and read the next character in input string.**
2. **If next character is again 0, then push A again in stack and repeat the same process for all consecutive 0's in input string.**
3. **If next character is 1, then pop A and change its state from q0 to q1.**
4. **If again the next character is 1 and top symbol of stack is 'A', then pop A and repeat the same process for all consecutive 1's in input string.**
5. **If all the characters of input string are parsed and stack top is Z0, it means string is valid, pop Z0 from stack and change the state from q1 to q2.**



Transition Diagram

Figure 2: PDA - accepting string for a language 0^n1^n where n >= 1.

Code in C++

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
void main()
{
        char Input[100];
        char stack[100]; //Implementing stack through array.
        int Top = -1;
        clrscr();
        cout<<"Enter binary string to validate (input string should be of 0
and 1)\n";
        gets(Input);
        stack[++Top] = 'Z';//Taking 'Z'as an initial stack symbol.
        int i=-1;
        q0:
                i++;
                if(Input[i]=='0' && stack[Top]== 'Z')
                {
                        stack[++Top]= 'A';
                        goto q0;
                }
                else if(Input[i]=='0' && stack[Top]== 'A')
                {
                        stack[++Top]= 'A';
                        goto q0;
                }
                else if(Input[i]=='1' && stack[Top]== 'A')
                {
                        Top--;
```

```
                        goto q1;
                }
                else
                {
                        goto Invalid;
                }
        q1:
                i++;
                if(Input[i]=='1' && stack[Top]== 'A')
                {
                        Top--;
                        goto q1;
                }
                else if(Input[i]=='\0' && stack[Top]== 'Z')
                {
                        Top--;
                        goto q2;
                }
                else
                {
                        goto Invalid;
                }
        q2:
                cout<<"\n Output: Valid String";
                goto exit;
        Invalid:
                cout<<"\n Output: Invalid String";
                goto exit;
        exit:
                getch();
}
```

Practical - 05

AIM: Design a Program to create PDA machine that accept the well-formed parenthesis.

Discussion:

As per the AIM, set of valid strings that can be generated by given language is represented in set A:

A = {(),(()), ()(), (()), ((()(()))) ...}

means all the parenthesis that are open must closed or combination of all legal parenthesis formation. Here, opening par is '(' and closing parenthesis is ')'. Block diagram of push down automata is shown in Figure 1.

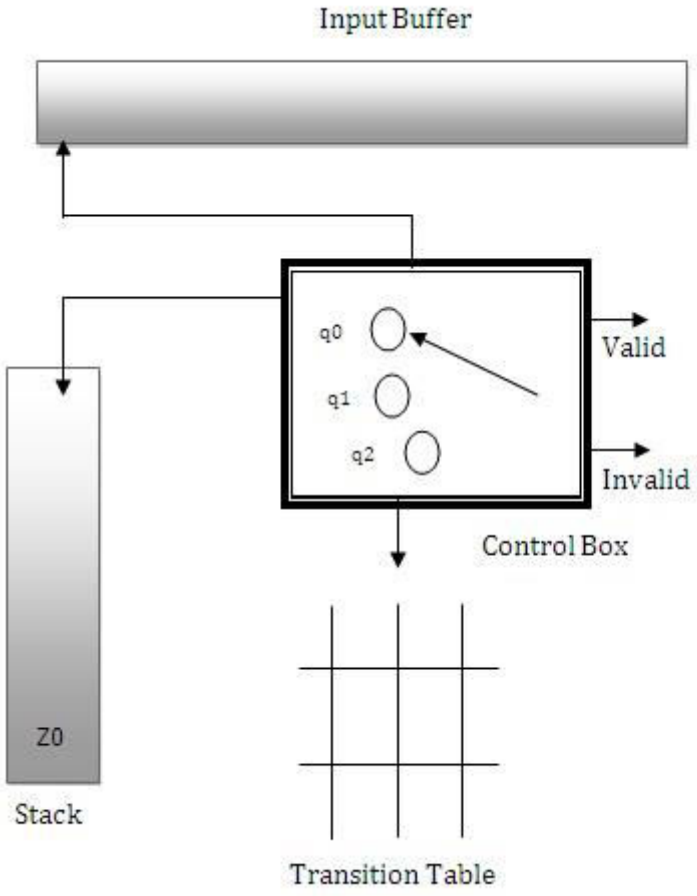


Figure 1: Block Diagram of Push Down Automata.

Input string can be valid or invalid, valid if the input string follow set A (define above). PDA has to determine whether the input string is according to the language or not.

Let M be the PDA machine for above AIM, hence it can be define as M(Q, Σ, Γ, δ, q0, Z0, F) where

Q: set of states: {q0, q1}

Σ: set of input symbols: {(,)}

Γ: Set of stack symbols: {(, Z}

q0: initial state (q0)

Z0: initial stack symbol (Z)

F: set of Final states: { } [Note: Here, set of final states is null as decision of validity of string is based on stack whether it is empty or not. If empty means valid else invalid.]

δ: Transition Function: (Transition state diagram is shown in Figure 2.)

$$\delta(q0, (, Z) \rightarrow (q0, (Z)$$

$$\delta(q0, (, () \rightarrow (q0, (()$$

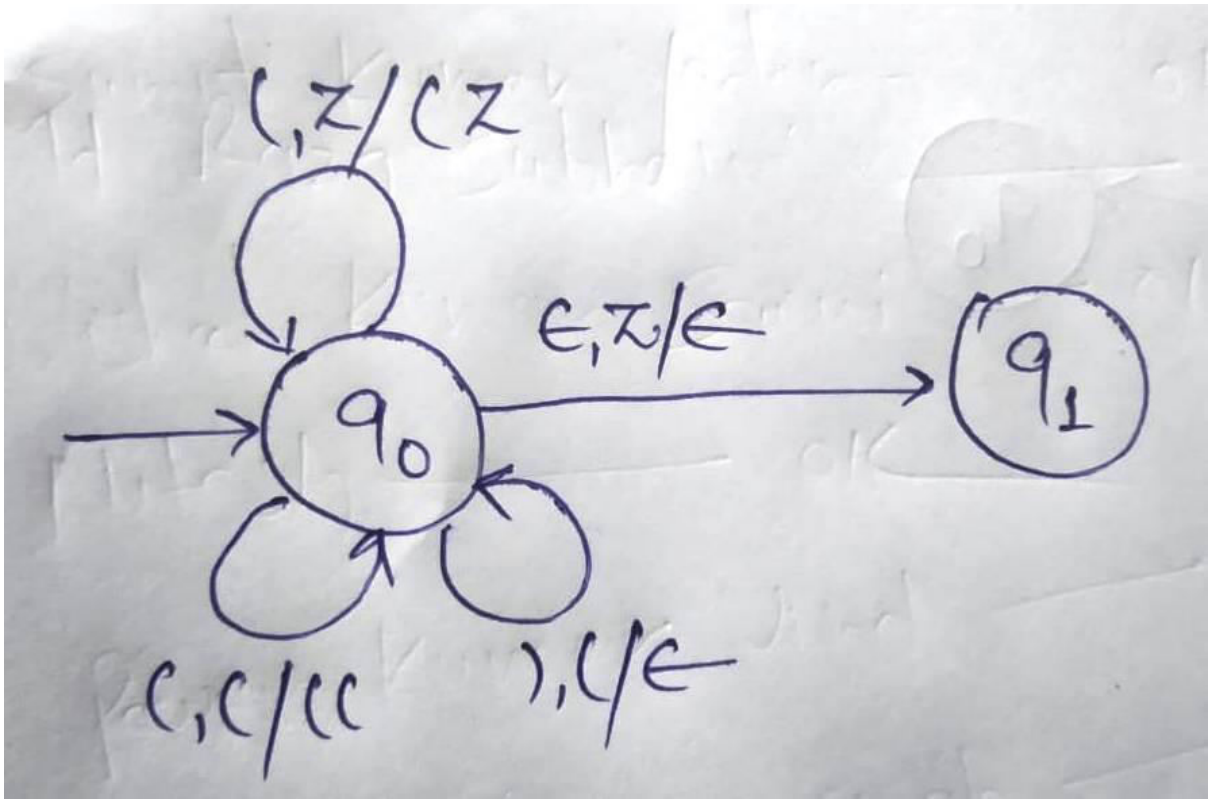
$$\delta(q0,), () \rightarrow (q0, \varepsilon)$$

$$\delta(q0, \varepsilon, Z) \rightarrow (q1, \varepsilon)$$

Rules for implementing PDA for a given language

Initial Setup: Load the input string in input buffer, push Z as an initial stack symbol and consider the machine in at initial state q0.
Rules:

1. **It is must that the first symbol should be '('.**
2. **If the input symbol of string is '(' and stack top is Z then push symbol '(' into stack and read the next character in input string.**
3. **If next character is again '(', then push '(' again in stack and repeat the same process for all '(' in input string.**
4. **If character is ')' and stack top is '(', then pop '(' from stack.**
5. **If all the characters of input string are parsed and stack top is Z, it means string is valid, pop Z from stack and change the state from q0 to q1.**



Transition Diagram

Figure 2: PDA that accept the well-formed parenthesis.

Code in C++

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
void main()
{
        char Input[100];
        char stack[100];
        int Top = -1;
        clrscr();
        cout<<"Enter parenthesis string (string character should be '(' and
')')\n";
        gets(Input);
        stack[++Top] = 'Z';//Taking 'Z'as an initial stack symbol.
        int i=-1;
        q0:
                i++;
                if(Input[i]=='(' && stack[Top]== 'Z')
                {
                        stack[++Top]= '(';
                        goto q0;
                }
                else if(Input[i]=='(' && stack[Top]== '(')
                {
                        stack[++Top]= '(';
```

```
                        goto q0;
                }
                else if(Input[i]==')' && stack[Top]== '(')
                {
                        Top--;
                        goto q0;
                }
                else if(Input[i]=='\0' && stack[Top]== 'Z')
                {
                        Top--;
                        goto q1;
                }
                else
                {
                        goto Invalid;
                }
        q1:
                cout<<"\n Output: Valid String";
                goto exit;
        Invalid:
                cout<<"\n Output: Invalid String";
                goto exit;
        exit:
                getch();
}
```

Practical - 06

AIM: Design a PDA to accept WCW^R where w is any binary string and W^R is reverse of that string and C is a special symbol.

Discussion:

As per the AIM, set of valid strings that can be generated by given language is represented in set A:

A = {0C0, 1C1, 011000110C011000110, 101011C110101, ...}

means string must have some binary string followed by special character 'C' followed reverse of binary string that appears before 'C'. Block diagram of push down automata is shown in Figure 1.

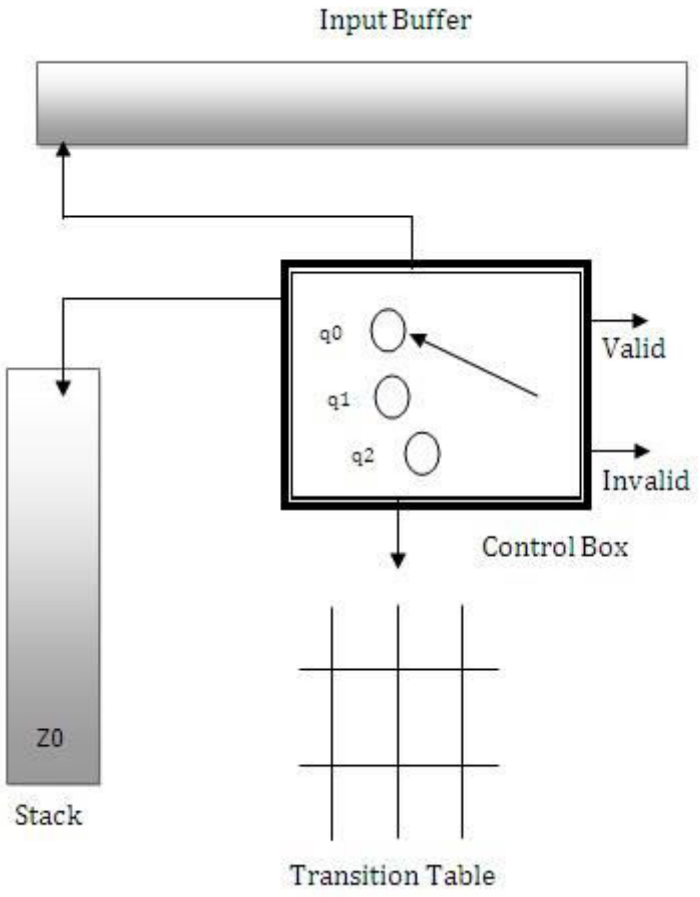


Figure 1: Block Diagram of Push Down Automata.

Input string can be valid or invalid, valid if the input string follow set A (define above). PDA has to determine whether the input string is according to the language or not.

Let M be the PDA machine for above AIM, hence it can be define as M(Q, Σ, Γ, δ, q0, Z0, F) where

Q: set of states: {q0, q1, q2}

Σ: set of input symbols: {0, 1, C}

Γ: Set of stack symbols: {A, B, Z}

q0: initial state (q0)

Z0: initial stack symbol (Z)

F: set of Final states: { } [Note: Here, set of final states is null as decision of validity of string is based on stack whether it is empty or not. If empty means valid else invalid.]

δ: Transition Function: (Transition state diagram is shown in Figure 2.)

$$\delta(q0, 0, Z) \rightarrow (q0, AZ)$$

$$\delta(q0, 1, Z) \rightarrow (q0, BZ)$$

$$\delta(q0, 0, A) \rightarrow (q0, AA)$$

$$\delta(q0, 0, B) \rightarrow (q0, AB)$$

$$\delta(q0, 1, A) \rightarrow (q0, BA)$$

$$\delta(q0, 1, B) \rightarrow (q0, BB)$$

$$\delta(q0, C, A) \rightarrow (q1, A)$$

$$\delta(q0, C, B) \rightarrow (q1, B)$$

$$\delta(q1, 0, A) \rightarrow (q1, \varepsilon)$$

$$\delta(q1, 1, B) \rightarrow (q1, \varepsilon)$$

$$\delta(q1, \varepsilon, Z) \rightarrow (q2, \varepsilon)$$

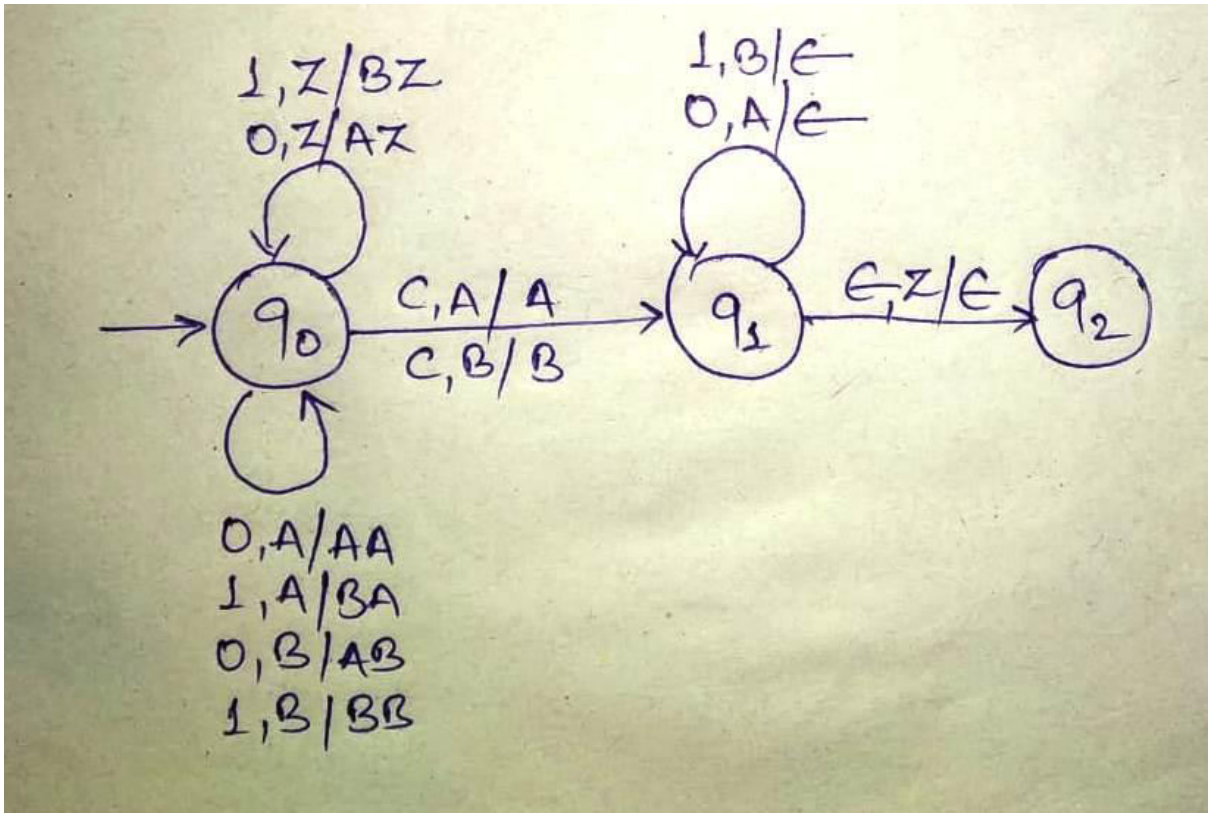
Rules for implementing PDA for a given language

Initial Setup: Load the input string in input buffer, push Z as an initial stack symbol and consider the machine in at initial state q0.

Rules:

1. **If the input symbol of string is '0' and stack top is Z then push symbol 'A' into stack and read the next character in input string.**
2. **If the input symbol of string is '1' and stack top is Z then push symbol 'B' into stack and read the next character in input string.**
3. **If the input symbol of string is '0' and stack top is A or B then push symbol 'A' into stack and read the next character in input string.**
4. **If the input symbol of string is '1' and stack top is A or B then push symbol 'B' into stack and read the next character in input string.**

5. If the input symbol of string is 'C' and stack top is A or B then change state from q0 to q1 and read the next character in input string.
6. If the input symbol of string is '0', machine state is q1 and stack top is A then pop stack top and read the next character in input string.
7. If the input symbol of string is '1', machine state is q1 and stack top is B then pop stack top and read the next character in input string.
8. If all the characters of input string are parsed, stack top is Z and machine state is q1, it means string is valid, pop Z from stack and change the state from q1 to q2.



Transition Diagram

Figure 2: PDA that accept language WCW^R where w is any binary string and W^R is reverse of that string and C is a special symbol.

Code in C++

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
void main()
{
        char Input[100];
        char stack[100];
        int Top = -1;
        clrscr();
        cout<<"Enter string to be validate\n";
        gets(Input);
        stack[++Top] = 'Z';//Taking 'Z'as an initial stack symbol.
```

```
        int i=-1;
        q0:
                i++;
                if(Input[i]=='0' && stack[Top]== 'Z')
                {
                        stack[++Top]= 'A';
                        goto q0;
                }
                else if(Input[i]=='1' && stack[Top]== 'Z')
                {
                        stack[++Top]= 'B';
                        goto q0;
                }
                else if(Input[i]=='0' && (stack[Top]== 'A'|| stack[Top]==
'B'))
                {
                        stack[++Top]= 'A';
                        goto q0;
                }
                else if(Input[i]=='1' && (stack[Top]== 'A'|| stack[Top]==
'B'))
                {
                        stack[++Top]= 'B';
                        goto q0;
                }
                else if(Input[i]=='C' && (stack[Top]== 'A'||stack[Top]== 'B'))
                {
                        goto q1;
                }
                else
                {
                        goto Invalid;
                }
        q1:
                i++;
                if(Input[i]=='0' && stack[Top]== 'A')
                {
                        Top--;
                        goto q1;
                }
                else if(Input[i]=='1' && stack[Top]== 'B')
                {
                        Top--;
                        goto q1;
                }
                else if(Input[i]=='\0' && stack[Top]== 'Z')
                {
                        goto Valid;
                }
                else
                {
                        goto Invalid;
                }
        Valid:
                cout<<"\n Output: Valid String";
                goto exit;
```

```
        Invalid:
                cout<<"\n Output: Invalid String";
                goto exit;
        exit:
                getch();
}
```

Practical - 07

AIM: Design a Turing Machine that calculate 2's complement of given binary string.

Discussion:

As per the AIM, Input can be any binary string and we have to design a turing machine that can calculate its two's complement.
For example (as shown in Figure 1) if the input binary string is 1011010000 then its two's complement will be 0100110000. For constructing a turing machine if we look in the logic then if we read the input string from right to left then the output string will remain exactly the same until the first 1 is found and as the first 1 encounter, complement of rest string appear in the output.

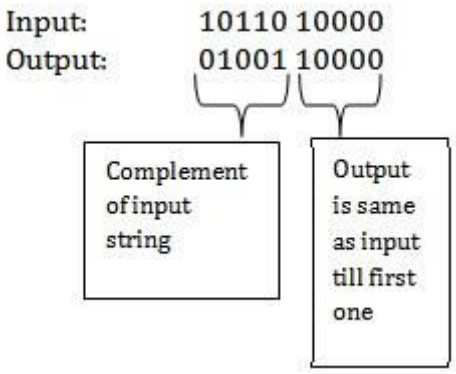


Figure 1: Input outpur relationship of 2's complement of binary string while reading right to left.

Let M be the Turing Machine for above AIM, hence it can be define as M(Q, Σ, Γ, δ, q0, B, F) where
Q: set of states: {q0, q1, q2}
Σ: set of input symbols: {0, 1}
Γ: Set of Tape symbols: {0, 1, B}
q0: initial state (q0)
B: Blank Symbol (B)
F: set of Final states: { } [Note: Here, set of final states is null as here we have to design turing machine as enumerator.]
δ: Transition Function:

State	Input		
-	0	1	B
q0	(q0, 0, R)	(q0, 1, R)	(q1, B, L)
q1	(q1, 0, L)	(q2, 1, L)	
q2	(q2, 1, L)	(q2, 0, L)	(q3, B, R)
q3			

Rules for implementing turing machine for a given language

Initial Setup: Load the input string in input buffer and consider the machine in at initial state q0.
Rules:

1. **Move the input head at right direction, till it reaches to B. If B encounter then change state from q0 to q1.**
2. **Now, start reading input symbol right to left one by one.**
3. **If the input symbol is 0, move the input head left and do the same for all 0 till 1 encounter.**
4. **If the input symbol is 1, move the input head left and change its state from q1 to q2.**
5. **Now, if the input symbol is 0, make it 1 and if input symbol is 1 make it 0 till reach to the starting point.**

Code in C++

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
void main()
{
        char Input[100];
        clrscr();
        cout<<"Enter input binary string\n";
        gets(Input);
        int i=0;
        q0:
                if(Input[i]=='0'|| Input[i]=='1')
                {
                        i++;
                        goto q0;
                }
                else if(Input[i]=='\0')
                {
                        i--;
                        goto q1;
                }
                else
                {
                        goto Invalid;
                }
```

```
        q1:
                if(Input[i]=='0')
                {
                        i--;
                        goto q1;
                }
                else if(Input[i]=='1')
                {
                        i--;
                        goto q2;
                }
                else
                {
                        goto Invalid;
                }
        q2:
                if(Input[i]=='0')
                {
                        Input[i]='1';
                        i--;
                        goto q2;
                }
                else if(Input[i]=='1')
                {
                        Input[i]='0';
                        i--;
                        goto q2;
                }
                else if(i== -1)
                {
                        goto q3;
                }
                else
                {
                        goto Invalid;
                }
        q3:
                cout<<"\nOutput: Two's complement is";
                puts(Input);
                goto exit;
        Invalid:
                cout<<"\n You have entered some invalid string.";
                goto exit;
        exit:
                getch();
}
```

Practical- 8

Design a Program which will increment the given binary number by one.

Program-

```
// Design a Program which will increment the given binary number by one.
#include<iostream>
#include<string>
using namespace std;


string addBinary(string a, string b)
{
	string result = "";
	int s = 0;


	int i = a.size() - 1, j = b.size() - 1;
	while (i>= 0 || j >= 0 || s == 1)
	{

		s += ((i>= 0) ? a[i] - '0' : 0);
		s += ((j >= 0) ? b[j] - '0' : 0);


		result = char(s % 2 + '0') + result;


		s /= 2;


		i--; j--;
	}
	return result;
}


int main()
{
	string a , b = "1";
	cout<< "Enter a binary number ";
	cin>> a;
	cout<<addBinary(a, b) <<endl;
	return 0;
}
```

Output-

```
Enter a binary number 1101101
Incremented binary number-
1101110
```