Experiment-4:

Implementing the prolog programs for example Bitwise AND, Bitwise OR, Bitwise X-OR, Bitwise Left-shift, Bitwise Right shift, Bitwise Complement.

Objective:

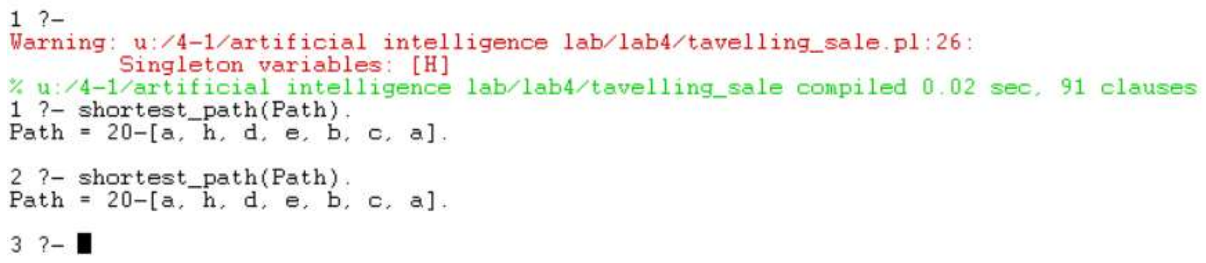
We have to create some prolog codes of Bitwise AND, Bitwise OR, Bitwise X-OR, Bitwise Left-shift, Bitwise Right shift, Bitwise Complement and then checking the validation of these codes.

Traveling Salesman Problem:

```
edge(a, b, 3).
edge(a, c, 4).
edge(a, d, 2).
edge(a, e, 7).
edge(b, c, 4).
edge(b, d, 6).
edge(b, e, 3).
edge(c, d, 5).
edge(c, e, 8).
edge(d, e, 6).
edge(b, a, 3).
edge(c, a, 4).
edge(d, a, 2).
edge(e, a, 7).
edge(c, b, 4).
edge(d, b, 6).
edge(e, b, 3).
edge(d, c, 5).
edge(e, c, 8).
edge(e, d, 6).
edge(a, h, 2).
edge(h, d, 1).


len([], 0).
len([H|T], N):- len(T, X), N is X+1 .


best_path(Visited, Total):- path(a, a, Visited, Total).
```

```
path(Start, Fin, Visited, Total) :- path(Start, Fin, [Start], Visited, 0, Total).


path(Start, Fin, CurrentLoc, Visited, Costn, Total) :-
    edge(Start, StopLoc, Distance), NewCostn is Costn + Distance, \+
member(StopLoc, CurrentLoc),
    path(StopLoc, Fin, [StopLoc|CurrentLoc], Visited, NewCostn, Total).

path(Start, Fin, CurrentLoc, Visited, Costn, Total) :-
    edge(Start, Fin, Distance), reverse([Fin|CurrentLoc], Visited), len(Visited, Q),
    (Q\=7 -> Total is 100000; Total is Costn + Distance).


shortest_path(Path):-setof(Cost-Path, best_path(Path,Cost),
Holder),pick(Holder,Path).


best(Cost-Holder,Bcost-_,Cost-Holder):- Cost<Bcost,!.
best(_,X,X).

pick([Cost-Holder|R],X):- pick(R,Bcost-Bholder),best(Cost-Holder,Bcost-Bholder,X),!.
pick([X],X).
```

Output:



```
1 ?-
Warning: u:/4-1/artificial intelligence lab/lab4/tavelling_sale.pl:26:
        Singleton variables: [H]
% u:/4-1/artificial intelligence lab/lab4/tavelling_sale compiled 0.02 sec, 91 clauses
1 ?- shortest_path(Path).
Path = 20-[a, h, d, e, b, c, a].

2 ?- shortest_path(Path).
Path = 20-[a, h, d, e, b, c, a].

3 ?- █
```

Bitwise AND:

```
bitwiseAnd(X,Y) :-
       R is X /\ Y,
      write(' Result of Bitwise AND is : '), write(R),!.
```

Bitwise OR:

```
bitwiseOR(X,Y) :-
       R is X \/ Y,
      write(' Result of Bitwise OR is : '), write(R),!.
```

Bitwise X-OR:

```
bitwiseXOR(X,Y) :-
       R is X xor Y,
      write(' Result of Bitwise XOR is : '), write(R),!.
```

Bitwise Right-shift:

```
bitwiseRight(X,Y) :-
       R is X >> Y,
      write(' Result of Bitwise Right is : '), write(R),!.
```

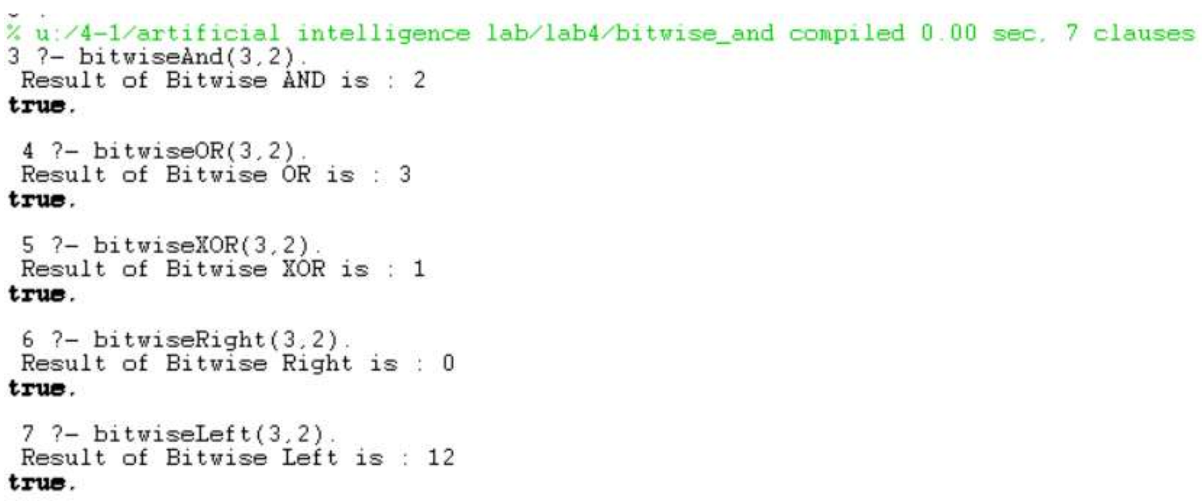
Bitwise Left-shift:

```
bitwiseLeft(X,Y) :-
       R is X << Y,
      write(' Result of Bitwise Left is : '), write(R),!.
```

Bitwise Complement:

```
bitwiseCoplement(X) :-
       R is \X,
      write(' Result of Bitwise Complement is : '), write(R),!.
```

Output:



```
% u:/4-1/artificial intelligence lab/lab4/bitwise_and compiled 0.00 sec, 7 clauses
3 ?- bitwiseAnd(3,2).
 Result of Bitwise AND is : 2
true.

 4 ?- bitwiseOR(3,2).
 Result of Bitwise OR is : 3
true.

 5 ?- bitwiseXOR(3,2).
 Result of Bitwise XOR is : 1
true.

 6 ?- bitwiseRight(3,2).
 Result of Bitwise Right is : 0
true.

 7 ?- bitwiseLeft(3,2).
 Result of Bitwise Left is : 12
true.
```