Experiment 1

logistic-regression

September 30, 2024

1 importing Dependence

[1]:
```
import numpy as np
import pandas as pd
```

2 importing dataset

[3]:
```
dataset = pd.read_csv(r'dataset.csv')
dataset.head()
```

[3]:

	temp_c	condition	wind_kph	pressure_mb	humidity	feelslike_c	heatindex_c
0	24.5	Clear	4.7	1004.0	41.0	25.1	25.1
1	24.2	Clear	4.7	1004.0	41.0	24.9	24.9
2	23.8	Clear	4.7	1004.0	41.0	24.8	24.8
3	23.5	Clear	4.7	1004.0	42.0	24.6	24.6
4	23.2	Clear	4.3	1004.0	43.0	24.6	24.6

3 Train Test Split

[5]:
```
x = dataset.drop('condition' , axis = 1)
y = dataset['condition']
```

4 Over-Sampling the data

[7]:
```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=20 , k_neighbors = 2)
x , y = sm.fit_resample(x , y)
```

5 Encoding our output variable

[9]:
```
from sklearn.preprocessing import LabelEncoder
condition_encoder = LabelEncoder()
dataset['condition'] = condition_encoder.fit_transform(dataset['condition'])
dataset.head()
```

[9]:

	temp_c	condition	wind_kph	pressure_mb	humidity	feelslike_c	heatindex_c
0	24.5	0	4.7	1004.0	41.0	25.1	25.1
1	24.2	0	4.7	1004.0	41.0	24.9	24.9
2	23.8	0	4.7	1004.0	41.0	24.8	24.8
3	23.5	0	4.7	1004.0	42.0	24.6	24.6
4	23.2	0	4.3	1004.0	43.0	24.6	24.6

[11]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x , y , test_size=.3 ,␣
 ↪random_state=42)
```

6 model training

[13]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train,y_train)
```

```
C:\Users\ayush\anaconda3\Lib\site-
packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[13]:
```
LogisticRegression()
```

7 model evaluation

[15]:
```
from sklearn.metrics import accuracy_score,classification_report
y_pred = model.predict(X_test)
report = classification_report(y_test , y_pred)
score = accuracy_score(y_test , y_pred)
```

[17]:
```
print(report)
print("Score : ",score)
```

```
                              precision    recall  f1-score   support

                       Clear       0.76      0.73      0.75       104
                      Cloudy       0.53      0.50      0.52       101
                         Fog       0.84      0.87      0.85       107
               Light drizzle       0.57      0.65      0.61        96
                  Light rain       0.63      0.66      0.64       103
         Light rain shower         0.18      0.06      0.09       107
                        Mist       0.42      0.77      0.55        90
 Moderate or heavy rain shower       0.19      0.21      0.20        89
              Moderate rain         0.34      0.35      0.35        92
                   Overcast         0.56      0.34      0.42       109
             Partly cloudy          0.39      0.36      0.38        96
Patchy light rain with thunder      0.55      1.00      0.71        88
          Patchy rain possible      0.25      0.32      0.28       101
                      Sunny         0.72      0.63      0.68       115
   Thundery outbreaks possible      0.77      0.31      0.45       105

                     accuracy                           0.51      1503
                    macro avg       0.51      0.52      0.50      1503
                 weighted avg       0.52      0.51      0.50      1503

Score :  0.5149700598802395
```

Experiment 2

decision-tree

October 4, 2024

1 Importing Dependencies and Dataset

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
dataset = pd.read_csv(r'dataset.csv')
dataset.head()
```

[2]:

	temp_c	condition	wind_kph	pressure_mb	humidity	feelslike_c	heatindex_c
0	24.5	Clear	4.7	1004.0	41.0	25.1	25.1
1	24.2	Clear	4.7	1004.0	41.0	24.9	24.9
2	23.8	Clear	4.7	1004.0	41.0	24.8	24.8
3	23.5	Clear	4.7	1004.0	42.0	24.6	24.6
4	23.2	Clear	4.3	1004.0	43.0	24.6	24.6

[3]:
```
x = dataset.drop('condition' , axis = 1)
y = dataset['condition']
```

2 Over-Sampling the data

[5]:
```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=20 , k_neighbors = 2)
x , y = sm.fit_resample(x , y)
```

3 Encoding our output variable

[7]:
```
from sklearn.preprocessing import LabelEncoder
condition_encoder = LabelEncoder()
dataset['condition'] = condition_encoder.fit_transform(dataset['condition'])
dataset.head()
```

[7]:

	temp_c	condition	wind_kph	pressure_mb	humidity	feelslike_c	\
0	24.5	0	4.7	1004.0	41.0	25.1	
1	24.2	0	4.7	1004.0	41.0	24.9	
2	23.8	0	4.7	1004.0	41.0	24.8	

```
3    23.5          0       4.7       1004.0      42.0         24.6
4    23.2          0       4.3       1004.0      43.0         24.6

   heatindex_c
0         25.1
1         24.9
2         24.8
3         24.6
4         24.6
```

4 Train Test Split

[9]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x , y , test_size=.3 ,␣
 ↪random_state=42)
```

5 model training

[11]:
```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(X_train,y_train)
```

[11]:
```
DecisionTreeClassifier()
```

6 model evaluation

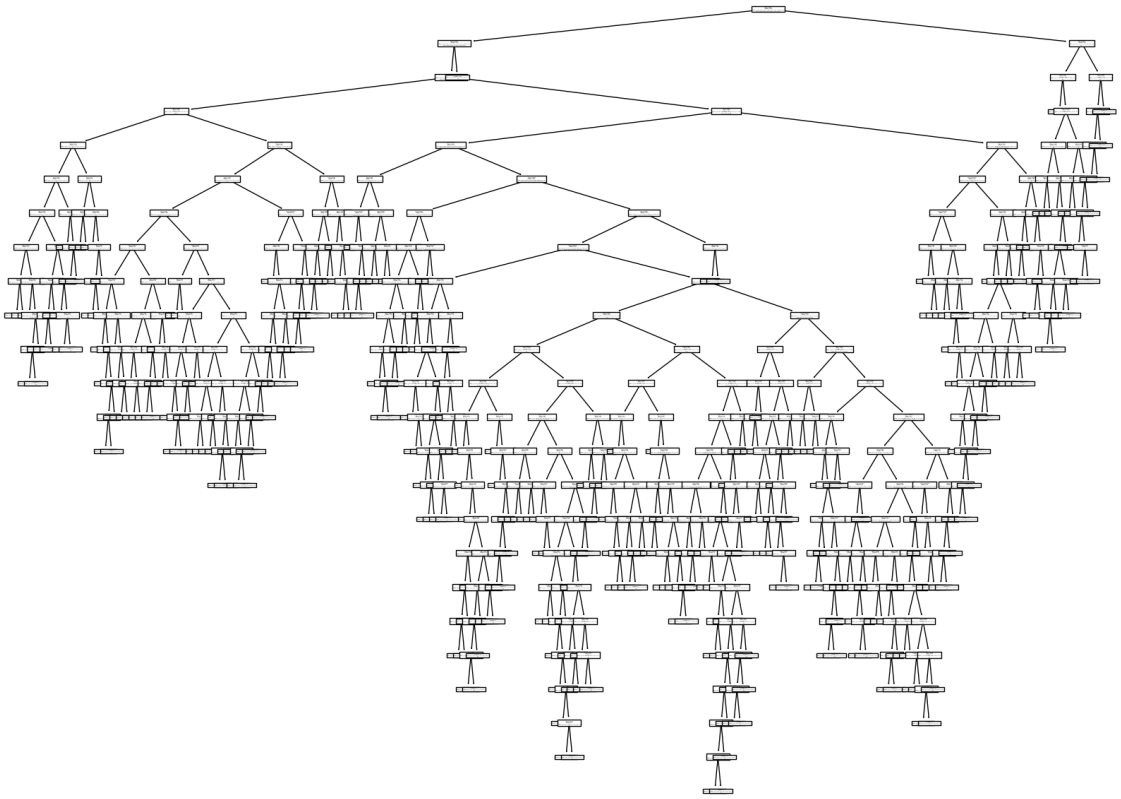
[13]:
```
from sklearn.metrics import accuracy_score,classification_report
y_pred = model.predict(X_test)
report = classification_report(y_test , y_pred)
score = accuracy_score(y_test , y_pred)
print(report)
print("Score : ",score)
```

```
                              precision    recall  f1-score   support

                       Clear       0.86      0.84      0.85       104
                      Cloudy       0.95      0.91      0.93       101
                         Fog       0.95      0.96      0.96       107
               Light drizzle       0.98      1.00      0.99        96
                  Light rain       1.00      1.00      1.00       103
         Light rain shower         0.89      0.80      0.84       107
                        Mist       0.96      0.98      0.97        90
Moderate or heavy rain shower       0.98      0.94      0.96        89
               Moderate rain        1.00      1.00      1.00        92
```

Overcast	0.83	0.96	0.89	109
Partly cloudy	0.80	0.81	0.81	96
Patchy light rain with thunder	0.99	1.00	0.99	88
Patchy rain possible	0.85	0.86	0.86	101
Sunny	0.90	0.80	0.85	115
Thundery outbreaks possible	0.92	0.99	0.95	105
accuracy			0.92	1503
macro avg	0.92	0.92	0.92	1503
weighted avg	0.92	0.92	0.92	1503

Score : 0.9214903526280772

[26]:
```
from sklearn import tree
plt.figure(figsize=(20 , 15))
tree.plot_tree(model)
plt.show()
```



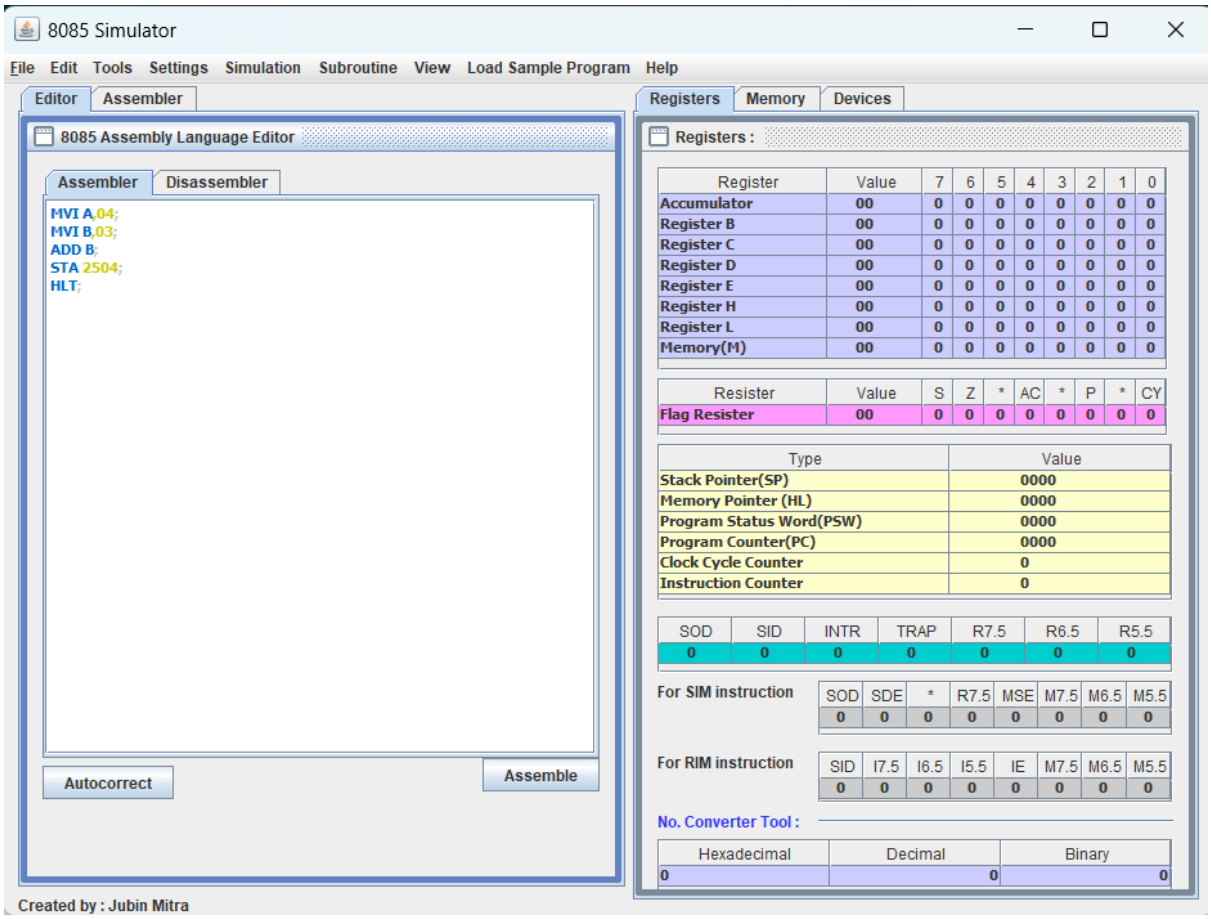
Experiment – 04

Aim –

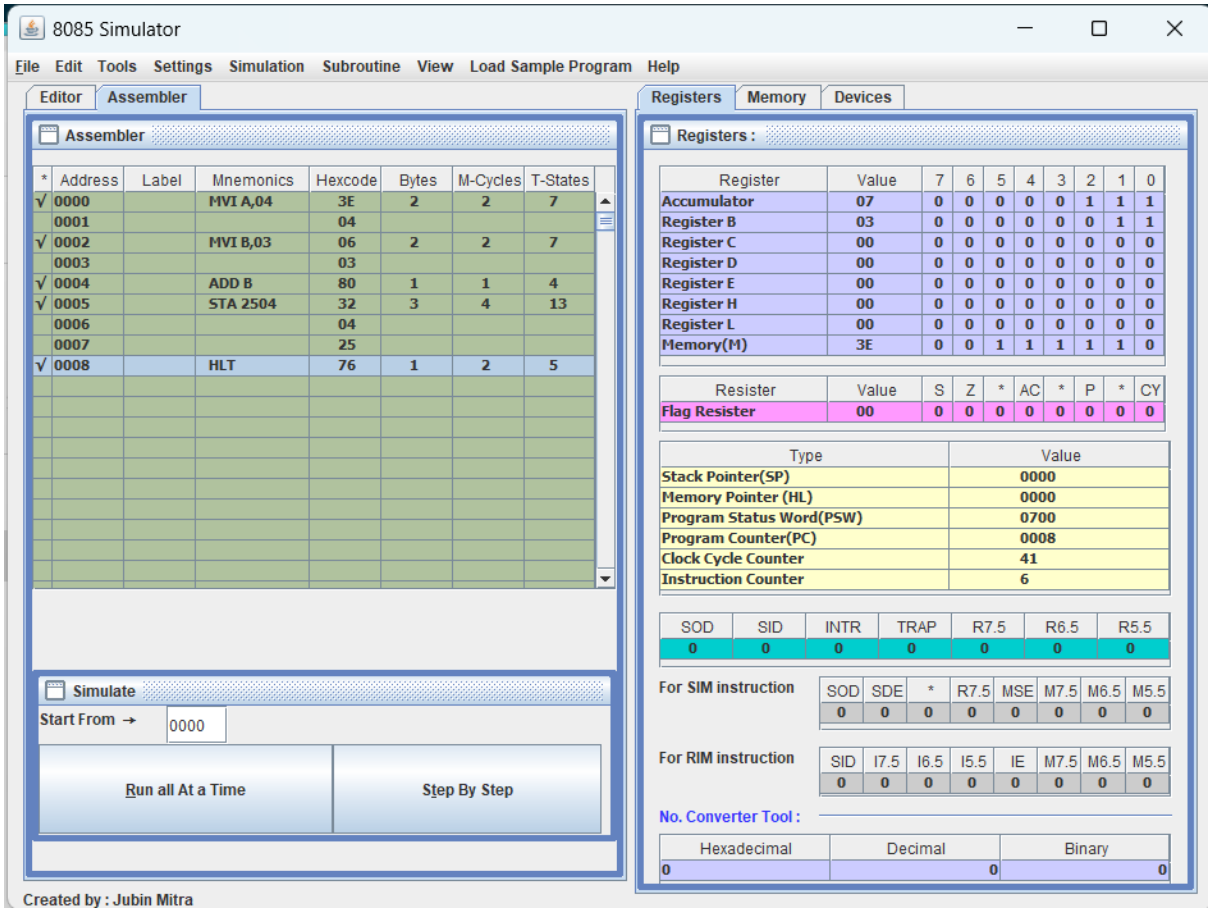
(A) Write a program to perform addition of two 8-bit numbers in 8085.
(B) Write a program to perform subtraction of two 8-bit numbers in 8085.

Software Used – Jubin Simulator 8085.

Program for Addition –

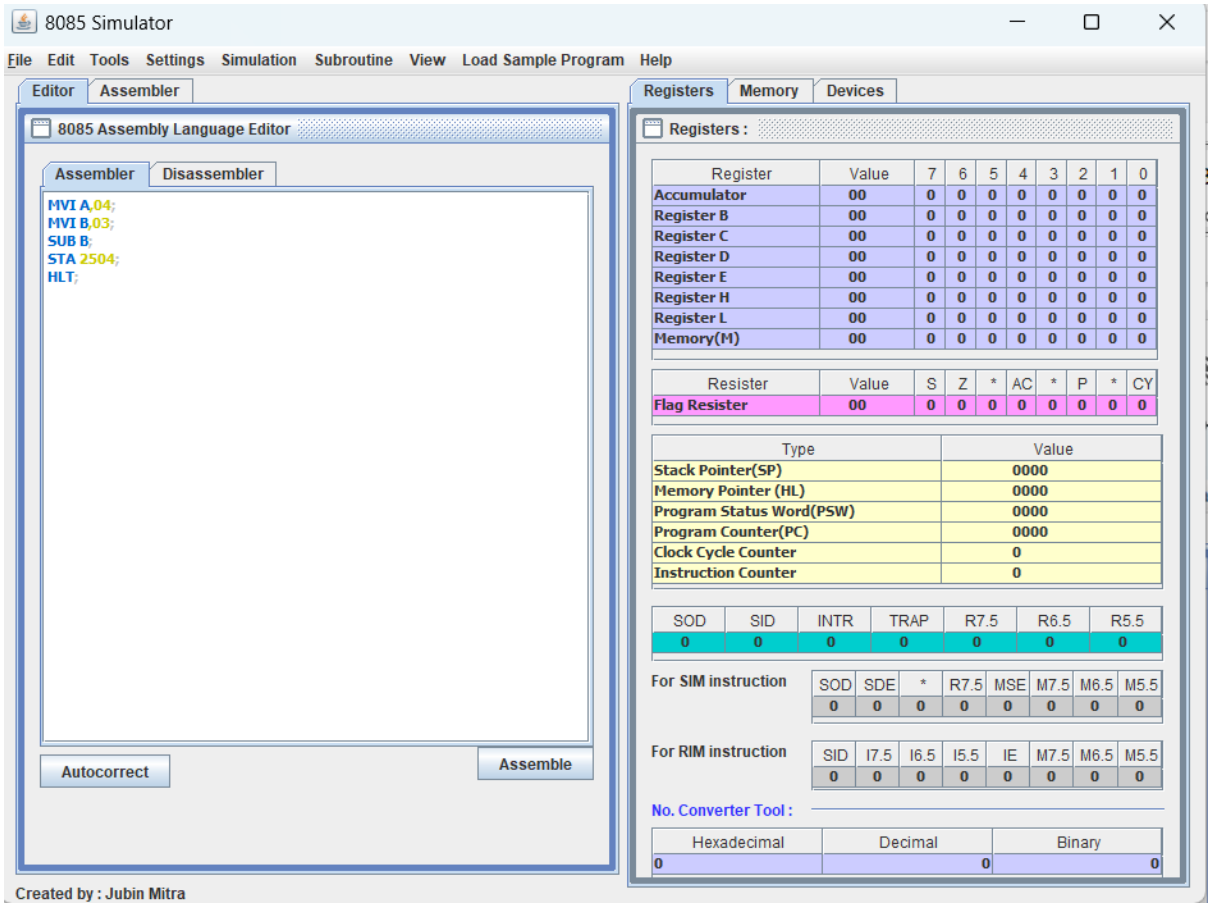


Output –

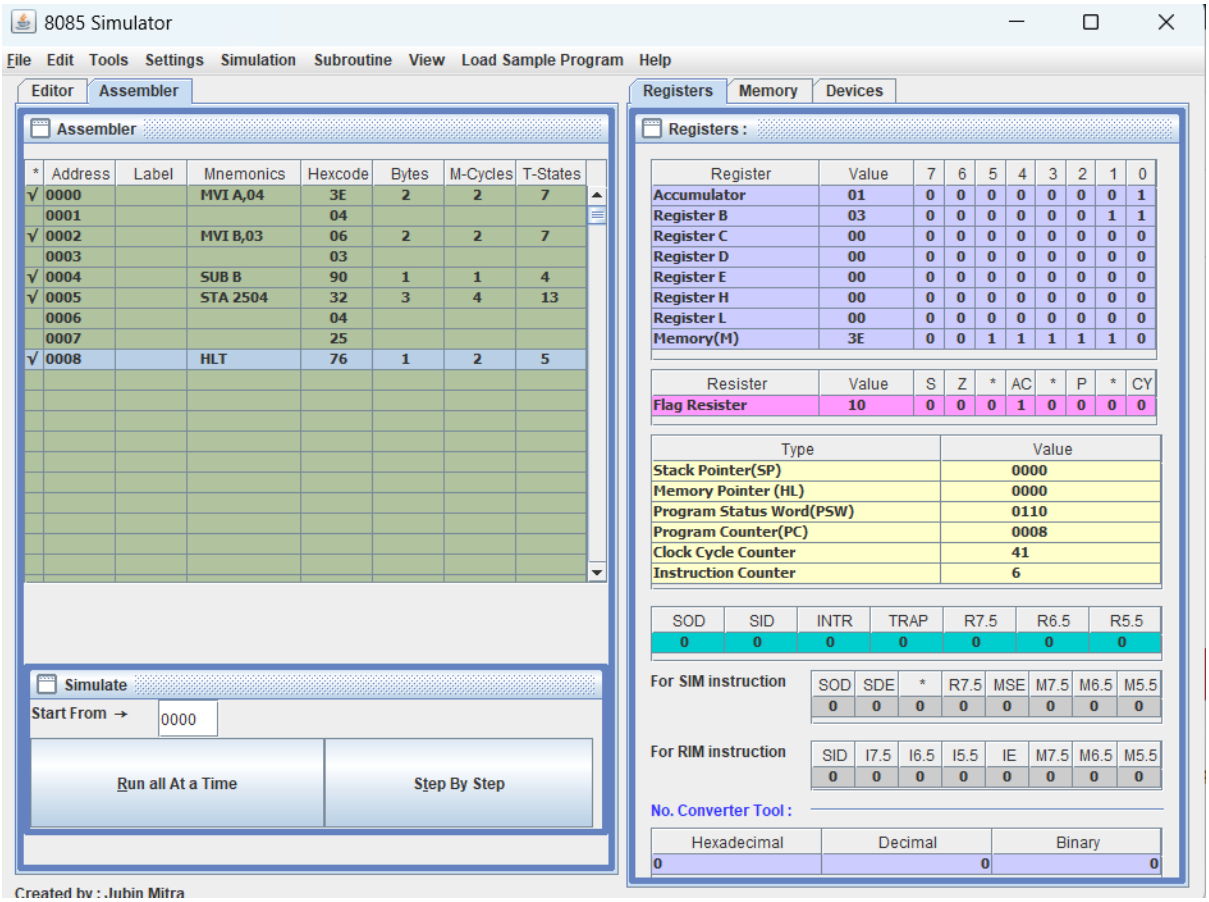


Result :- Hence, the Addition of two 8-bit numbers is completed.

Program for Subtraction –



Output –



*	Address	Label	Mnemonics	Hexcode	Bytes	M-Cycles	T-States
√	0000		MVI A,04	3E	2	2	7
	0001			04			
√	0002		MVI B,03	06	2	2	7
	0003			03			
√	0004		SUB B	90	1	1	4
√	0005		STA 2504	32	3	4	13
	0006			04			
	0007			25			
√	0008		HLT	76	1	2	5

Register	Value	7	6	5	4	3	2	1	0
Accumulator	01	0	0	0	0	0	0	0	1
Register B	03	0	0	0	0	0	0	1	1
Register C	00	0	0	0	0	0	0	0	0
Register D	00	0	0	0	0	0	0	0	0
Register E	00	0	0	0	0	0	0	0	0
Register H	00	0	0	0	0	0	0	0	0
Register L	00	0	0	0	0	0	0	0	0
Memory(M)	3E	0	0	1	1	1	1	1	0

Resister	Value	S	Z	*	AC	*	P	*	CY
Flag Resister	10	0	0	0	1	0	0	0	0

Type	Value
Stack Pointer(SP)	0000
Memory Pointer (HL)	0000
Program Status Word(PSW)	0110
Program Counter(PC)	0008
Clock Cycle Counter	41
Instruction Counter	6

SOD	SID	INTR	TRAP	R7.5	R6.5	R5.5
0	0	0	0	0	0	0

SOD	SDE	*	R7.5	MSE	M7.5	M6.5	M5.5
0	0	0	0	0	0	0	0

SID	I7.5	I6.5	I5.5	IE	M7.5	M6.5	M5.5
0	0	0	0	0	0	0	0

Hexadecimal	Decimal	Binary
0	0	0

Result :- Hence, the Subtraction of two 8-bit numbers is completed.

practical-2

October 2, 2024

1 0176CD221037

2 Data Preprocessing

3 importing libraries

[1]:
```
import numpy as np
import pandas as pd
```

4 set data display limit

[2]:
```
pd.set_option('display.max_rows' , 60)
pd.set_option('display.max_columns' , 60)
```

5 load data and preprocessed

[3]:
```
dataset = pd.read_csv(r'Uttar Pradesh.csv')
```

[4]:
```
drop = [ 'will_it_snow' , 'chance_of_snow' , 'vis_km' , 'vis_miles' , 'gust_mph'
        , 'gust_kph' , 'chance_of_rain' , 'state' , 'city']
dataset.drop(drop , axis = 'columns' , inplace = True)
```

[5]:
```
dataset
```

[5]:

	temp_c	condition	wind_kph	\
0	24.5	{'text': 'Clear', 'icon': '//cdn.weatherapi.co…	4.7	
1	24.2	{'text': 'Clear', 'icon': '//cdn.weatherapi.co…	4.7	
2	23.8	{'text': 'Clear', 'icon': '//cdn.weatherapi.co…	4.7	
3	23.5	{'text': 'Clear', 'icon': '//cdn.weatherapi.co…	4.7	
4	23.2	{'text': 'Clear', 'icon': '//cdn.weatherapi.co…	4.3	
…	…	…	…	
1147	26.0	{'text': 'Clear', 'icon': '//cdn.weatherapi.co…	11.5	
1148	25.1	{'text': 'Clear', 'icon': '//cdn.weatherapi.co…	11.5	
1149	24.1	{'text': 'Clear', 'icon': '//cdn.weatherapi.co…	11.5	
1150	23.7	{'text': 'Clear', 'icon': '//cdn.weatherapi.co…	9.7	

```
1151    23.3  {'text': 'Clear', 'icon': '//cdn.weatherapi.co…       7.9

      pressure_mb  humidity  feelslike_c  windchill_c  heatindex_c
0          1004.0      41.0         25.1         24.5         25.1
1          1004.0      41.0         24.9         24.2         24.9
2          1004.0      41.0         24.8         23.8         24.8
3          1004.0      42.0         24.6         23.5         24.6
4          1004.0      43.0         24.6         23.2         24.6
…             …         …            …            …            …
1147       1010.0      57.0         27.2         26.0         27.2
1148       1010.0      59.0         26.3         25.1         26.3
1149       1011.0      60.0         25.5         24.1         25.5
1150       1011.0      61.0         25.3         23.7         25.3
1151       1011.0      62.0         25.1         23.3         25.1

[1152 rows x 8 columns]
```

[6]:
```
cond = dataset['condition']
cond.dtype
```

[6]:
```
dtype('O')
```

[7]:
```
# to convert string dictionary to dictionary
type(eval(cond[0]))
```

[7]:
```
dict
```

[8]:
```
# this functions convert all the string dictionary values to dictionary value
# and only get the relevant value that we need
def string_to_dict(condition):
    new_cond = []
    for i in range(len(condition)):
        data = condition[i]
        data = eval(data)
        text = data['text']
        new_cond.append(text)
    return new_cond
```

[9]:
```
new_cond = string_to_dict(cond)
```

[10]:
```
dataset['condition'] = new_cond
```

[11]:
```
dataset
```

[11]:
```
      temp_c condition  wind_kph  pressure_mb  humidity  feelslike_c  \
0       24.5     Clear       4.7       1004.0      41.0         25.1
1       24.2     Clear       4.7       1004.0      41.0         24.9
```

```
2       23.8     Clear       4.7       1004.0      41.0          24.8
3       23.5     Clear       4.7       1004.0      42.0          24.6
4       23.2     Clear       4.3       1004.0      43.0          24.6
…       …        …           …         …           …             …
1147    26.0     Clear      11.5       1010.0      57.0          27.2
1148    25.1     Clear      11.5       1010.0      59.0          26.3
1149    24.1     Clear      11.5       1011.0      60.0          25.5
1150    23.7     Clear       9.7       1011.0      61.0          25.3
1151    23.3     Clear       7.9       1011.0      62.0          25.1

      windchill_c  heatindex_c
0            24.5         25.1
1            24.2         24.9
2            23.8         24.8
3            23.5         24.6
4            23.2         24.6
…             …            …
1147         26.0         27.2
1148         25.1         26.3
1149         24.1         25.5
1150         23.7         25.3
1151         23.3         25.1

[1152 rows x 8 columns]
```

6 data visulization

[13]:
```
data_visualize = dataset.drop('condition' , axis= 'columns')
```

[14]:
```
data_visualize
```

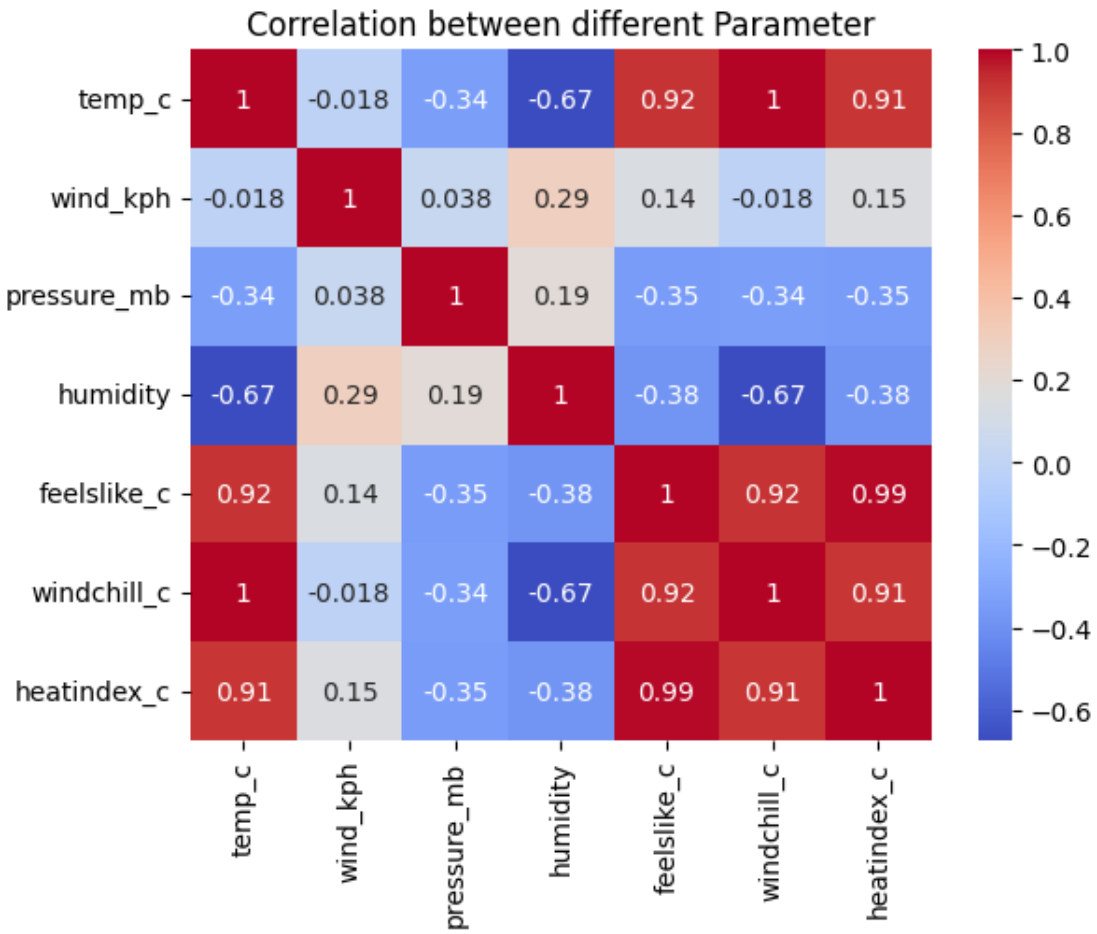
[14]:
```
      temp_c  wind_kph  pressure_mb  humidity  feelslike_c  windchill_c  \
0       24.5       4.7       1004.0      41.0         25.1         24.5
1       24.2       4.7       1004.0      41.0         24.9         24.2
2       23.8       4.7       1004.0      41.0         24.8         23.8
3       23.5       4.7       1004.0      42.0         24.6         23.5
4       23.2       4.3       1004.0      43.0         24.6         23.2
…       …          …         …           …            …            …
1147    26.0      11.5       1010.0      57.0         27.2         26.0
1148    25.1      11.5       1010.0      59.0         26.3         25.1
1149    24.1      11.5       1011.0      60.0         25.5         24.1
1150    23.7       9.7       1011.0      61.0         25.3         23.7
1151    23.3       7.9       1011.0      62.0         25.1         23.3

      heatindex_c
0            25.1
```

```
1             24.9
2             24.8
3             24.6
4             24.6
…              …
1147          27.2
1148          26.3
1149          25.5
1150          25.3
1151          25.1

[1152 rows x 7 columns]
```

[15]:
```
# to find correlation between different parameters
corr = data_visualize.corr()
```

[16]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[17]:
```
sns.heatmap(corr , annot=True ,cmap = 'coolwarm')
plt.title('Correlation between different Parameter')
# save this figure so that we can visualize it
plt.savefig('CorrelationHeatMap.jpg')
plt.show()
```



[19]:
```
dataset.drop('windchill_c' , axis = 'columns' , inplace= True)
```

[20]:
```
# we save our pre processed dataset for further use in machine learning␣
  ↪algorithm deployment
dataset.to_csv('dataset.csv' , index = False)
```

practical-4

October 2, 2024

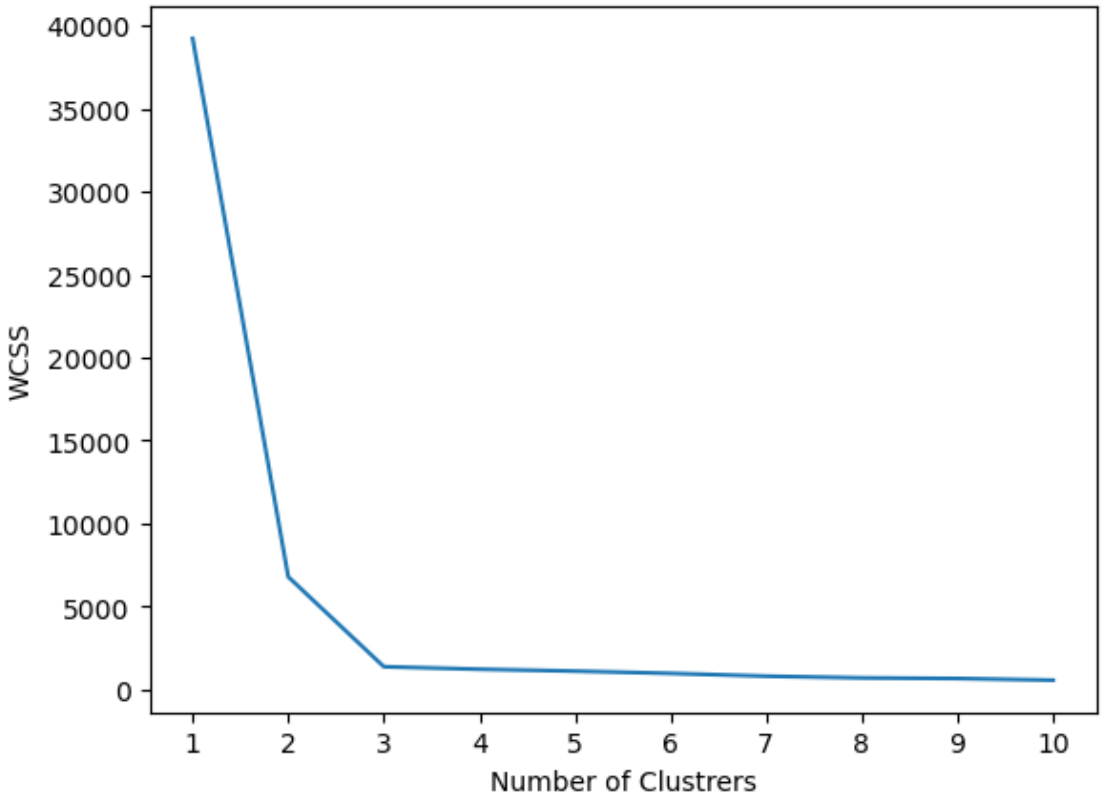
1 0176CD221037

2 Kmeans and Kmedoids clustering

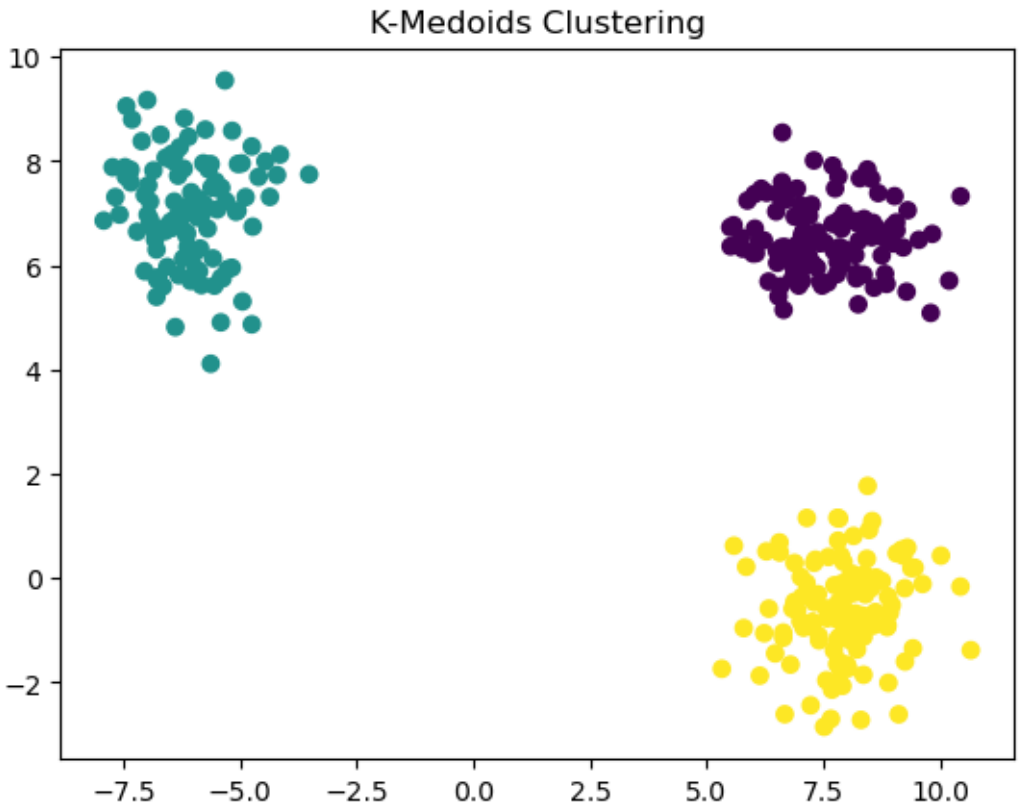
[2]:
```
# importing dependencies
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
import pandas as pd
import numpy as np
%matplotlib inline
```

Kmeans clustering

[4]:
```
X,y=make_blobs(n_samples=1000,centers=3,n_features=2)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
 ↪random_state=42)

import warnings
warnings.filterwarnings("ignore")

from sklearn.cluster import KMeans
## Elbow method To select K Value
wcss=[]
for k in range(1,11):
    kmeans=KMeans(n_clusters=k,init="k-means++")
    kmeans.fit(X_train)
    wcss.append(kmeans.inertia_)
## plot elbow curve
plt.plot(range(1,11),wcss)
plt.xticks(range(1,11))
plt.xlabel("Number of Clustrers")
plt.ylabel("WCSS")
plt.show()
```



[5]:
```
# Here we take k value as 3
kmeans=KMeans(n_clusters=3,init="k-means++")
kmeans.fit_predict(X_train)
y_pred=kmeans.predict(X_test)

plt.scatter(X_test[:,0],X_test[:,1],c=y_pred)
plt.title('K-Medoids Clustering')
plt.show()
```



[6]:
```
## the output we get is right as we select 3 centers and here we get 3 clusters
```

Kmedoids Clustering

[8]:
```
import numpy as np
from sklearn.metrics import pairwise_distances
from random import sample

# Function to compute total cost (sum of distances) for a set of medoids
def compute_cost(X, medoids, clusters):
    cost = 0
    for medoid, cluster in zip(medoids, clusters):
        cost += np.sum(pairwise_distances(X[cluster], X[medoid].reshape(1, -1)))
    return cost

# K-medoids clustering using Partitioning Around Medoids (PAM)
def k_medoids(X, k, max_iter=300):
    m, n = X.shape
    # Randomly initialize medoids
    medoids = sample(range(m), k)
    for iteration in range(max_iter):
```

```
        clusters = [[] for _ in range(k)]

        # Assign each point to the nearest medoid
        for idx, point in enumerate(X):
            distances = [np.linalg.norm(point - X[medoid]) for medoid in␣
 ↪medoids]
            closest_medoid = np.argmin(distances)
            clusters[closest_medoid].append(idx)

        new_medoids = []
        # Update medoids for each cluster
        for cluster in clusters:
            if len(cluster) == 0:
                continue
            distances_sum = np.sum(pairwise_distances(X[cluster], X[cluster]),␣
 ↪axis=1)
            new_medoid = cluster[np.argmin(distances_sum)]
            new_medoids.append(new_medoid)
        # Check for convergence
        if set(medoids) == set(new_medoids):
            break
        medoids = new_medoids
    # Final cluster assignment
    final_clusters = [[] for _ in range(k)]
    for idx, point in enumerate(X):
        distances = [np.linalg.norm(point - X[medoid]) for medoid in medoids]
        closest_medoid = np.argmin(distances)
        final_clusters[closest_medoid].append(idx)
    # Compute final cost
    final_cost = compute_cost(X, medoids, final_clusters)
    return medoids, final_clusters, final_cost

# Example usage:
if __name__ == "__main__":
    from sklearn.datasets import make_blobs
    import matplotlib.pyplot as plt
    # Create sample data
    X, y = make_blobs(n_samples=300, centers=4, random_state=42)
    # Perform K-medoids clustering
    k = 4
    medoids, clusters, cost = k_medoids(X, k)
    # Plot the clusters and medoids
    for i, cluster in enumerate(clusters):
        plt.scatter(X[cluster, 0], X[cluster, 1], label=f'Cluster {i+1}')
    plt.scatter(X[medoids, 0], X[medoids, 1], s=200, c='red', label='Medoids',␣
 ↪marker='x')
    plt.legend()
```

```
    plt.title('K-Medoids Clustering')
    plt.show()
```