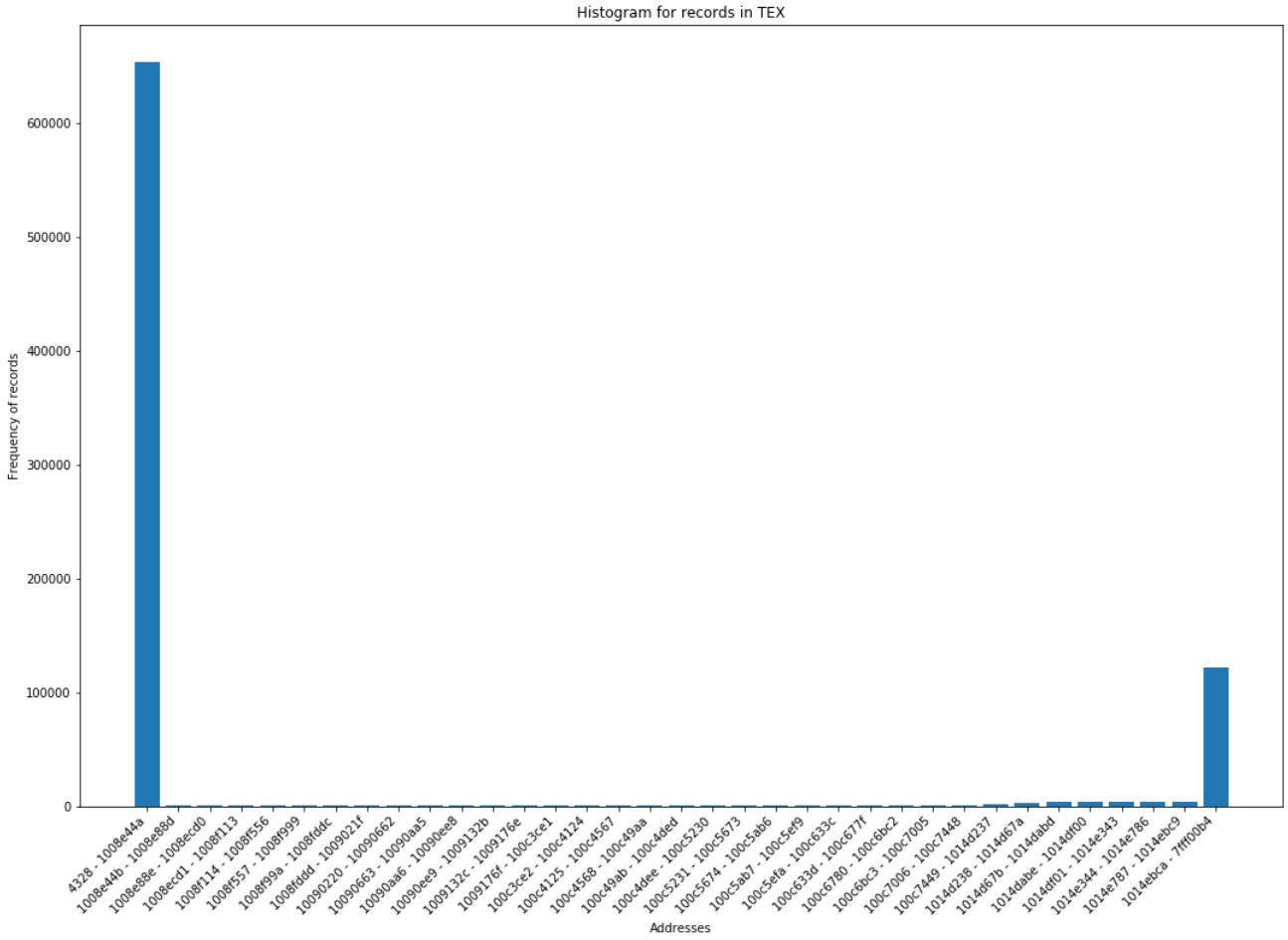
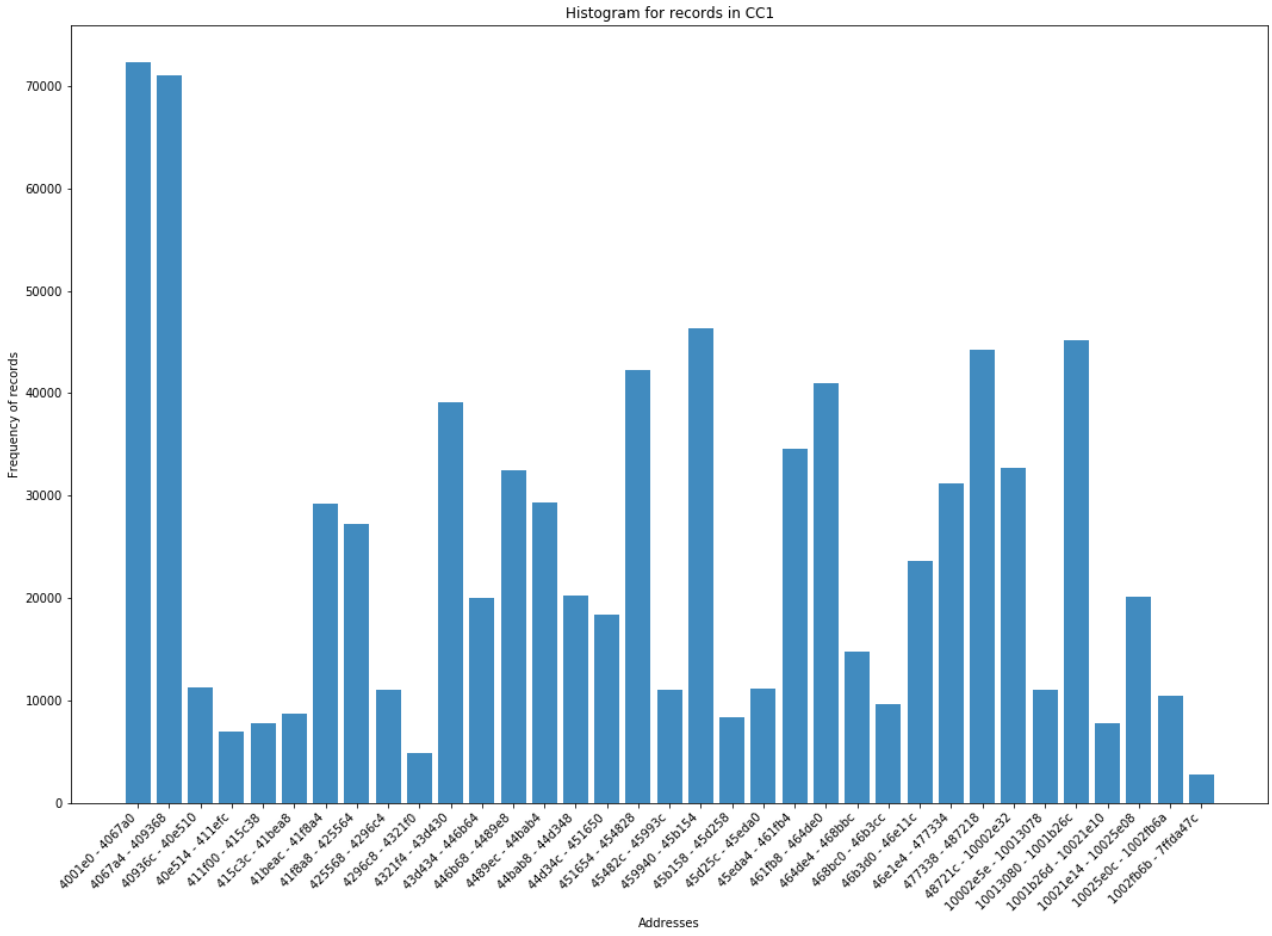
Q1: (a) Using trace files, i.e. files that contain addresses issued by some CPU to execute some application(s), draw the histogram of address distribution for each of them (2x20 points)

Ans: The files TEX.DIN and CC1.DIN have been saved as text files in the working directory for ease of operation. For the current implementation, the trace files are imported as pandas data-frame. The hexadecimal records are sorted and then frequency of occurrences is calculated in ranges. Below are the two histograms generated corresponding to the two trace files. The implementation is done in python(v3.6) using jupyter notebooks. The detailed implementation steps can be found in the attached file **HW1__Q1.ipynb**



Plot 1 : Histogram from records in TEX file.

The OX axis shows 35 bins with records in range 1091 in each bins. The OY axis shows the frequency of records in that range. E.g.: For the first bin(Address Range) [4328 - 1008e44a] the sum of all the record call is 654258. This means records in this range have been called 654258 times which is considerably higher than most of the bin frequencies.



Plot 2 : Histogram from records in CC1 file.

For CC1 file, we have total number of unique records: 43051 which is a prime number.

However, 43050 has factor pairs (35, 1230). So, let's take 34 ranges with 1230 elements each with the 35th range having an extra record: 1231

Comments on the Histograms:

For the TEX file the addresses frequencies are unevenly distributed compared to the cc1 file. For the TEX plot, most of the records are in the first and last bin of the Histogram: this means the addresses in range `4328 - 1008e44a` and `1014ebca - 7fff00b4` have been used for Read, Write and Instruction fetch operations much more as compared to the other addresses.

These two ranges have frequencies of 654258 and 122680 respectively while most of the remaining record ranges have frequencies of about 1k with a few closings near 1.5k

In case of the CC1 file, however, the distribution is comparatively balanced with first two ranges having large frequencies.

Steps taken for generating the histograms:

1. Load file as `pandas` data frame and drop the operation column. (We only need 'records' column for the plot.)

```
# Load the tex file as a pandas dataframe
df1 = pd.read_csv('tex.txt', sep=" ", header=None)
df1.columns = ['operation', 'records']
df1.head(2)
```

	operation	records
0	2	430d70
1	2	430d74

```
df_op1 = df1
df1 = df1.drop(df1.columns[0], axis=1)
df1['records_int'] = df1['records'].apply(int, base=16)
df1.shape
```

```
(832477, 2)
```

2. Convert hexadecimal values to int and sort them.
3. Sort the records(rows) for '**records_int**' column, add another column to store the frequency of each address's occurrence

```
df_rec_sorted = df1.sort_values('records_int')
df_rec_sorted['freq'] = df_rec_sorted.groupby('records')['records'].transform('count')
df_rec_sorted =df_rec_sorted.drop_duplicates()
df_rec_sorted = df_rec_sorted.reset_index(drop=True)
df_rec_sorted.shape
```

```
(38185, 3)
```

4. As we can see from **df_rec_sorted.shape** we have 38185 rows containing unique address records. To plot a histogram, we need to plot frequency of occurrence on the OY axis. However, as we have 38K records it is not possible to plot each of the datapoint individually. Instead it makes sense to split 38185 in equal divisions. Factors of 38185 are: [1, 5, 7, 35, 1091, 5455, 7637, 38185]. Now if we take each of the 1091 records as one BAR the we have to plot 35 such BARs which is suitable to plot.

```
#Create dataframe df_plot where we will store the AddressRanges and Frequencies of all occurances in those ranges
df_plot = pd.DataFrame(index=range(0,35),columns=['Address Range', 'Frequency'])

#For each of 35 ranges lets store the frequency against the ranges in df_plot as below:
for i in range(0, 35):
    if i==34:
        df_plot['Address Range'].iloc[i] = str(df_rec_sorted['records'].iloc[1091* (i)]) + " - " + \
                                                                                    str(df_rec_sorted['records'].iloc[(-1) ])
        df_plot['Frequency'].iloc[i] =0
        for j in range(0, 1091):
                df_plot['Frequency'].iloc[i] += (df_rec_sorted['freq'].iloc[(1091 * i) + j])
    else:
        df_plot['Address Range'].iloc[i] = str(df_rec_sorted['records'].iloc[1091* (i)]) + " - " + \
                                                                        str(df_rec_sorted['records'].iloc[(1091*(i+1))-1])
        df_plot['Frequency'].iloc[i] =0
        for j in range(0, 1091):
                df_plot['Frequency'].iloc[i] += (df_rec_sorted['freq'].iloc[(1091 * i) + j])
```

```
# Plot the Histogram
plt.figure(figsize=(18, 12))
plt.bar(df_plot["Address Range"], df_plot["Frequency"], align='center', alpha=1)
plt.xticks(range(len(df_plot["Address Range"])), rotation = 45, ha="right")
plt.xlabel('Addresses')
plt.ylabel('Frequency of records')
plt.title("Histogram for records in TEX", loc='center')
plt.savefig("hist_tex.png")
plt.show()
```

Q1. (b) What is the frequency of writes (5)? What is the frequency of reads (5)? Please comment on these results (5).

Ans:

```
The frequency of WRITE in the TEX file is: 12.554460964086696%
The frequency of WRITE in the CC1 file is: 8.302983394033213 %
Average frequency of Write = 10.43%

The frequency of READ in the TEX file is: 15.694727902392499%
The frequency of READ in the CC1 file is: 15.963068073863852 %
Average frequency of Read = 15.83%
```

Observations on the results:

1. The frequency of read is clearly more that write for both of the files.
2. Read and Write operations together constitute of ~28% for TEX and ~24% for CC1 out of all the three operations. This means INSTRUCTION FETCH operation has the most frequency, more than TWICE than READ and WRITE combined together.
3. Frequency of WRITE in both of the trace files are almost same: 15.69% and 15.96% respectively for TEX and CC1 files.

Git Link: https://bitbucket.org/ArunabhSaikia/cs402/src/master/HW1/

Files:

- Histogram for TEX file: Files/**hist_tex.png**
- Histogram for CC1 file: Files/**hist_cc1.png**
- Implementation for Q1: Files/**HW1__Q1.ipynb** (*The HW1__Q1.HTML report also shows the same implementations. It appears that the file cannot be opened online directly in Bitbucket. Here is the alternate GitHub link which supports opening .ipynb online:* **HW1__Q1.ipynb**)

Q2: (a) Write a program, using your favorite programming language, that multiplies two rectangular matrices.

Ans:

I have used python to implement the matrix multiplications: **algo1.py** and **algo2.py** (Attached and also uploaded in bitbucket). The programs do not need to be compiled and can be run directly from terminal as: "python algo1.py"

"algo1.py" contains two functions:

`matrix_multiply_int()` which multiplies two randomly generated **integer** matrices of size (200,300) and (300, 400) with values ranging from (0, 99)

`matrix_multiply_double()` which multiplies two randomly generated matrices with **double** values of size (200,300) and (300, 400) with values ranging from (0, 99.9)

For the two matrices *matrix_A* and *matrix_B* the algorithm iterates through each row of *matrix_A.* For each such iterations it then goes into each column of *matrix_B* and further iterates through each row of *matrix_B.*

The algorithm **algo1.py** is run in two different systems whose configurations are as given below:

System 1: Personal PC (Laptop)

Operating System	Windows 10
Manufacturer	HP
Processor:	Intel(R) Core(TM) i7-3610QM CPU @ 2.30GHz 2.30 GHz
Installed memory (RAM):	6.00 GB (5.90 GB usable)

System 2: MTCC PC (Desktop)

Operating System	Windows 10
Manufacturer	Dell
Processor:	Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz 3.60 GHz
Installed memory (RAM):	16.0 GB

Q. Measure the time it takes each program to complete (2x5) and then compare the performance of the two systems (5).

Execution No.	algo1.py			
	System1		**System2**	
	Integer	Double	Integer	Double
1	27.45	25.30	18.57	17.10
2	27.31	25.42	18.29	17.10
3	27.71	25.44	18.37	17.07
4	27.47	25.44	18.31	17.07
5	27.41	25.33	18.73	17.35
6	27.42	25.36	20.21	17.49
7	27.14	25.31	18.24	17.06
8	27.37	25.21	18.38	17.17
9	27.50	25.39	18.38	17.10
10	27.48	25.24	18.34	17.10
Average	27.43	25.34	18.58	17.16

Table 1: Time taken by algo1.py for the Matrix Multiplications

Q: Is the performance ratio the same as the clock rate ratio of the two systems (5)? Explain. Based on the retail price of the two systems, which one is more cost effective (5)?

Ans: CPU Clock rate ratio of the two systems = $\frac{(Clock\ Rate)_{System2}}{(Clock\ Rate)_{System1}}$

$$= \frac{2.30}{3.60} = 1.565 \sim 1.5$$

<u>Now let's calculate the performance ratio of the two systems:</u>

For algo1.py:

For Integer Multiplication: $\frac{(Time\ Taken)_{System1}}{(Time\ Taken)_{System2}}$

$$= \frac{27.43}{18.58} = 1.476 \sim 1.5$$

For Double Multiplication: $\frac{(Time\ Taken)_{System1}}{(Time\ Taken)_{System2}}$

$$= \frac{25.34}{17.16} = 1.476 \sim 1.5$$

Is the performance ratio the same as the clock rate ratio of the two systems (5)?

System 2 is faster than system 1 in terms of clock speed and also does calculations faster than system1. As we can see from the above calculation: Clock rate ratio ~ 1.5 and Performance ratio ~ 1.5. Thus, we can conclude that performance rate ratio is same as the clock rate ratio of the two systems.

Based on the retail price of the two systems, which one is more cost effective (5)?

Retail price of System 1 is 650 USD and System 2 is 800 USD. If time is our main concern, which in most of the computational cases is valuable, then for 150$ we are getting around 50% increased performance Thus system 2 is more cost effective.

Q2(b). Change your multiplication algorithm and repeat the steps above

Ans: For **algo2.py** let us iterate by column of *matrix_B* first and the by row of *matrix_A.*

Execution No.	Algo2.py			
	System1		**System2**	
	Integer	Double	Integer	Double
1	27.08	24.92	19.48	18.20
2	27.21	25.15	19.52	18.40
3	27.22	25.03	20.00	17.57
4	27.42	25.01	18.23	17.09
5	27.24	25.08	18.31	17.07
6	27.47	25.03	18.32	17.04
7	27.25	25.22	18.20	17.03
8	27.12	26.37	18.28	17.03
9	27.22	29.50	18.29	17.03
10	27.25	24.96	18.31	17.06
Average	27.25	25.63	18.69	17.35

Table 2: Time taken by algo2.py for the Matrix Multiplications

For algo2.py:

For Integer Multiplication: $\frac{(Time\ Taken)_{System1}}{(Time\ Taken)_{System2}}$

$= \frac{27.25}{18.69} = 1.457 \sim 1.5$

For Double Multiplication: $\frac{(Time\ Taken)_{System1}}{(Time\ Taken)_{System2}}$

$= \frac{25.63}{17.35} = 1.477 \sim 1.5$

Is the performance ratio the same as the clock rate ratio of the two systems (5)?

Similar to what we saw before, the performance ratio is same as the clock rate ratio(almost).

Based on the retail price of the two systems, which one is more cost effective (5)?

Similar to our previous algorithm System 2 seems more cost effective as it gives great performance improvement for considerably small increase in retail price.

Git Link: https://bitbucket.org/ArunabhSaikia/cs402/src/master/HW1/

Files:

- Code Q2: Implementations are available in attached: Files/**algo1.py** and Files/**algo2.py** files.
- Execution screenshots are available as: Files/algo1_system1.png, algo1_system2.png, algo2_system1.png and algo2_system2.png Which shows Executions of the two algorithms in the two systems.