Project II - Memory Hierarchy Simulator

CSCE 2303 - 01

Merna Wael Abdelbadie - 900203731

Maya Hussein - 900201198

Farah Fathy - 900201155

The American University in Cairo

Implementation Description:

First, we imported the math library and created four global variables, which are hit, miss, noOfCycles, and noOfAccesses and initialized all of them with zero.

Main Function:

To be able to create a GUI (using pyqt5):-

A Qmessage is initialized if the number of cycles is less than 1 or greater than 10, and the user is asked to re-input the number of cycles. Otherwise, two dictionaries are created: memoryDivison and cache. Then the user is asked to enter the size of the cache (s) and the line size (l). The number of lines (c) is then computed by s/l. Offset size, index size, and tag size are then computed using the following equations:

offsetSize = $\log_2(l)$

indexSize = $\log_2(c)$

tagSize = 32 - (offsetSize + indexSize)

readFromFile is then called, and offsetSize, indexSize, tagSize, memoryDivision, and cache are passed to it.

ReadFromFile:

The function receives offsetSize, indexSize, tagSize, memoryDivision, cache, l and c. It reads the access sequence text file and creates the index, offset, and tag of each memory address. It opens the file and reads it line by line. For each line, it computes the memory address in binary (32 bits). Then, it divides the memory address into index, offset, and tag. The tag would take the digits matching its size starting from the leftmost digit. The index would take the following digits matching its size. Lastly, the offset would take the rest of the digits which should be matching its size.

createCache:

This function receives offsetSize, indexSize, cache, and memoryDivision. It gets the total size by calculating two to the power of the offsetSize + indexSize. Then it gets the numOfBits using offsetSize + indexSize. After that, it loops over the total size to get all possible combinations of index and offset while ensuring all are represented in the same amount of bits which is numOfBits. It then loops over allCombinations and gets the substring of the index and the offset, then appends to the cache the index, tag, vBit (initially all zero as the cache is assumed to be empty at the beginning), and tag((initially empty string as the cache is assumed to be empty at the beginning).

cachePlacement:

This function receives cache and memoryDivision. It loops over the memoryDivison and then loops over the cache to check whether the element in the memoryDivision exists in the cache. It does that by first checking whether the index and offset cache match that of the memoryDivision. If yes, then it checks the vBit in the cache; if zero, it increments the miss and the noOfAccesses, then sets the tag in the corresponding place in the cache to the tag of the memoryDivison, and the vBit in the cache of the corresponding place changes to 1. If the vBit is one, then it checks whether the tag in the cache matches that of the memoryDivison. If it matches, then the hit gets incremented, if they are different, then the miss and noOfAccesses get incremented, and the tag in the corresponding place in the cache gets updated with that of the memoryDivision. After each memory access, the following are printed: tag, vBit, hit ratio (hit/noOfAccesses), and miss ratio (1 - hit/noOfAccesses). The function then prints the final cache in the GUI and in the terminal it prints every change that happens in the cache.

window:

The window function basically sets the standard layout for the GUI, along with a window title and the designated geometry.

Any Bugs or issues in the simulator:

No bugs in the simulator.

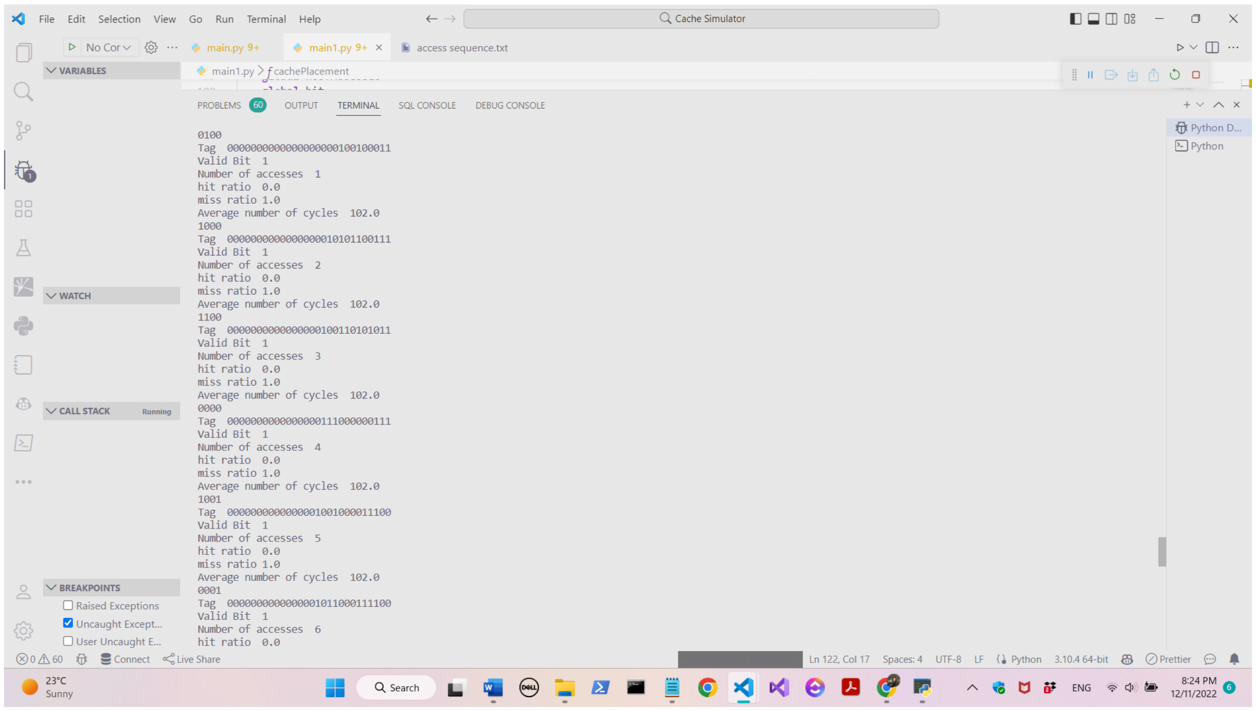
User Guide:

To run the simulator:

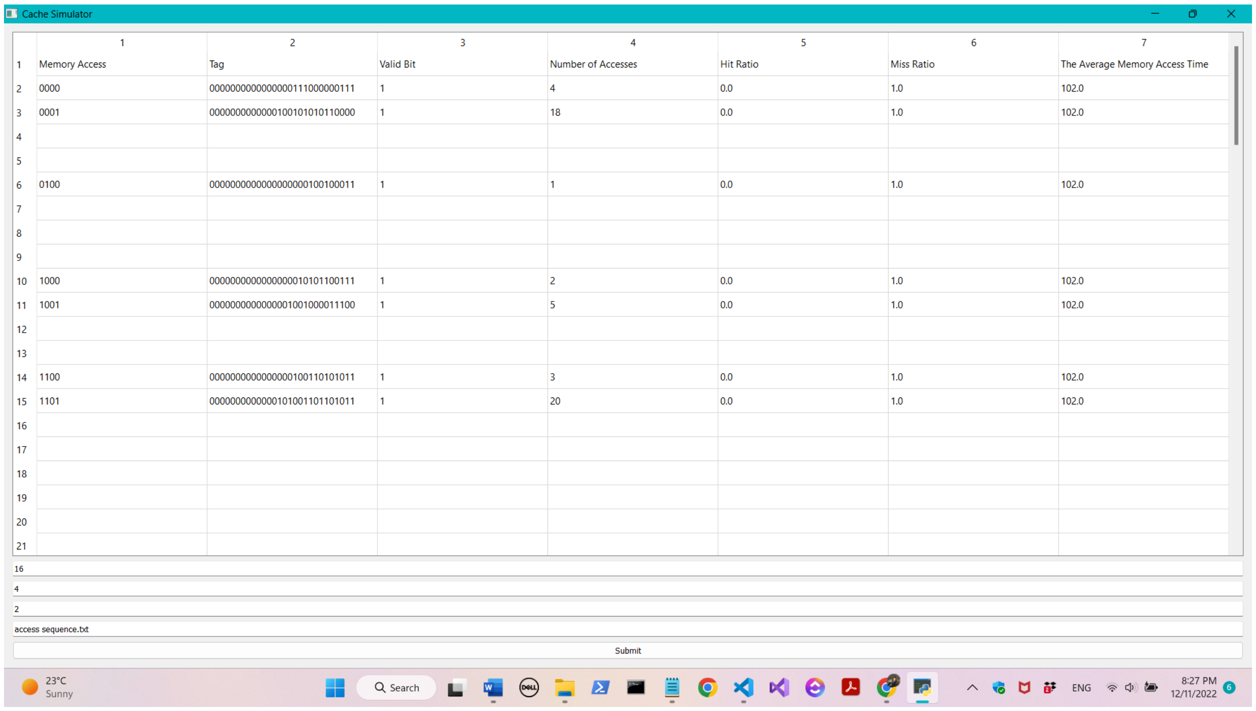
1. Simply open the python file (main1.py) and run the program
2. The GUI will ask the user to input the cache size along with the line size and the number of cycles.
    a. Note that the file path is also requested. It is favorable if the text file is in the same directory as the main1.py
    b. Note that if the number of cycles is inputted > 10 or <1, an error pop-up is displayed, and the user is asked to input the appropriate number of cycles.

List of sequences simulated: in Test Case Folder

Test Case 1:



```
0100
Tag  0000000000000000000100100011
Valid Bit  1
Number of accesses  1
hit ratio  0.0
miss ratio 1.0
Average number of cycles  102.0
1000
Tag  0000000000000000010101100111
Valid Bit  1
Number of accesses  2
hit ratio  0.0
miss ratio 1.0
Average number of cycles  102.0
1100
Tag  0000000000000000100110101011
Valid Bit  1
Number of accesses  3
hit ratio  0.0
miss ratio 1.0
Average number of cycles  102.0
0000
Tag  0000000000000000111000000111
Valid Bit  1
Number of accesses  4
hit ratio  0.0
miss ratio 1.0
Average number of cycles  102.0
1001
Tag  0000000000000001001000011100
Valid Bit  1
Number of accesses  5
hit ratio  0.0
miss ratio 1.0
Average number of cycles  102.0
0001
Tag  0000000000000001011000111100
Valid Bit  1
Number of accesses  6
hit ratio  0.0
```



Cache Simulator

	1	2	3	4	5	6	7
1	Memory Access	Tag	Valid Bit	Number of Accesses	Hit Ratio	Miss Ratio	The Average Memory Access Time
2	0000	0000000000000000111000000111	1	4	0.0	1.0	102.0
3	0001	0000000000000100101010110000	1	18	0.0	1.0	102.0
4							
5							
6	0100	0000000000000000000100100011	1	1	0.0	1.0	102.0
7							
8							
9							
10	1000	0000000000000000010101100111	1	2	0.0	1.0	102.0
11	1001	0000000000000001001000011100	1	5	0.0	1.0	102.0
12							
13							
14	1100	0000000000000000100110101011	1	3	0.0	1.0	102.0
15	1101	0000000000000101001101101011	1	20	0.0	1.0	102.0
16							
17							
18							
19							
20							
21							

16

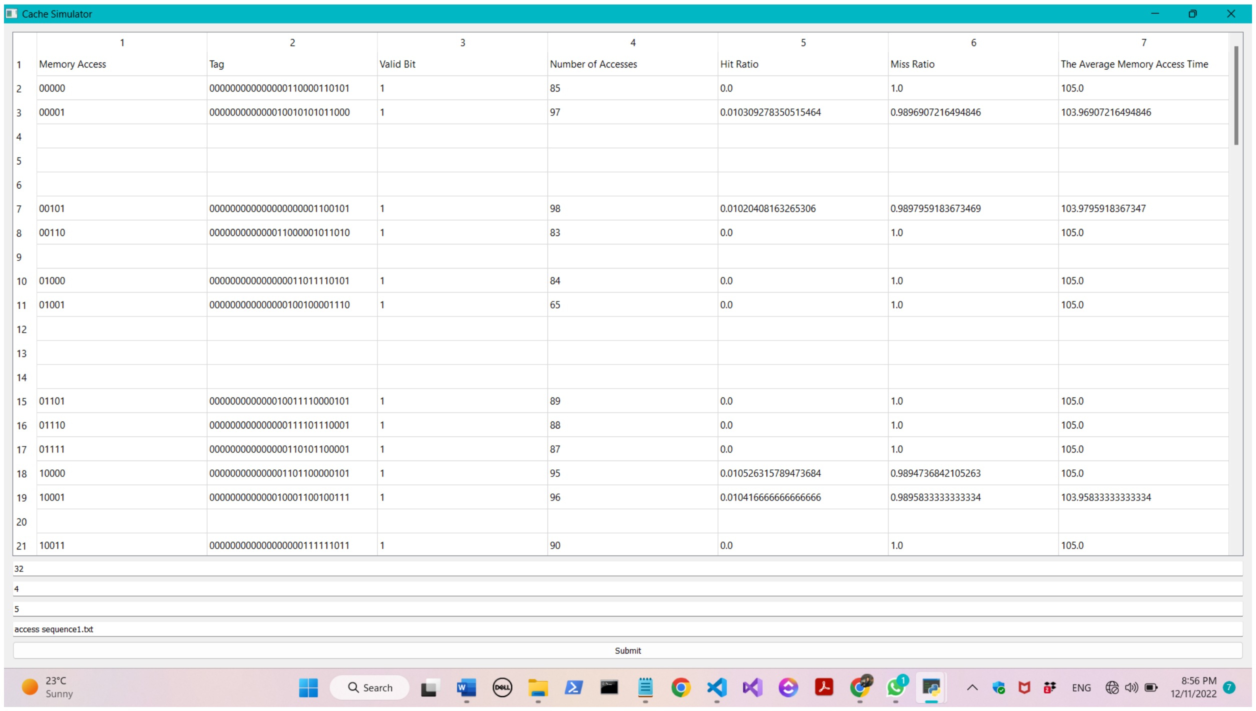
4

2

access sequence.txt

Submit

Test Case 2:



Cache Simulator

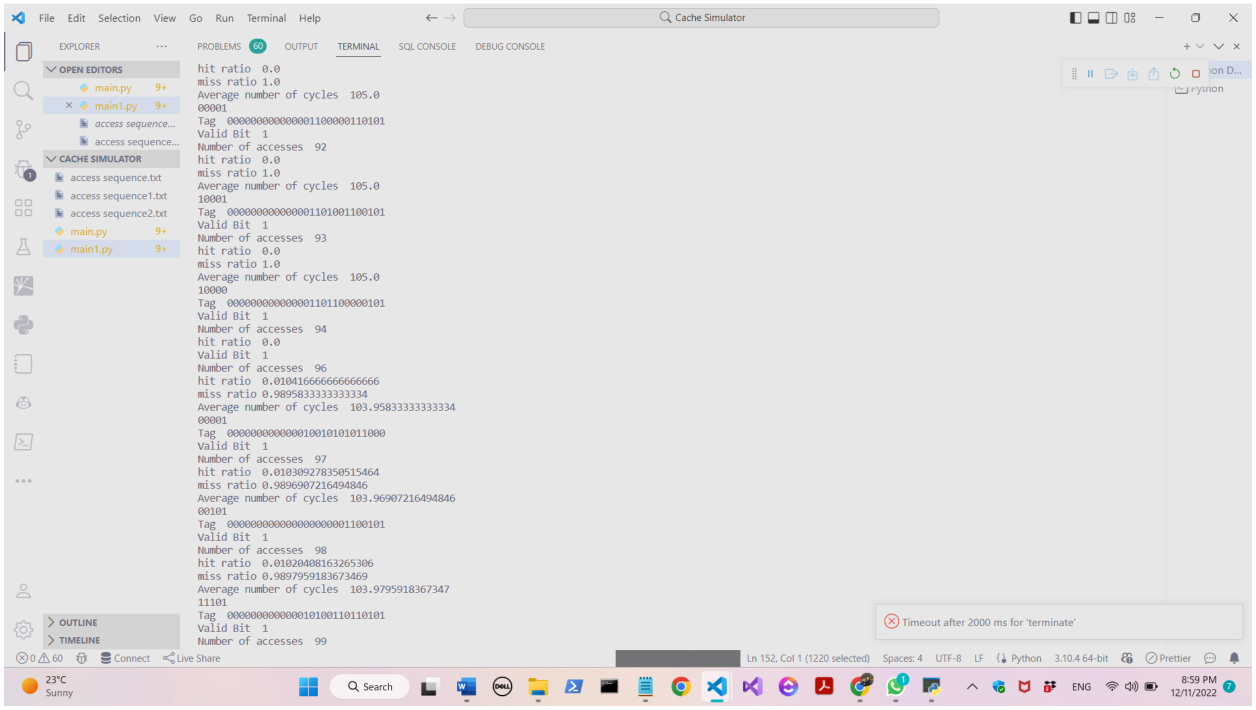
	1	2	3	4	5	6	7
1	Memory Access	Tag	Valid Bit	Number of Accesses	Hit Ratio	Miss Ratio	The Average Memory Access Time
2	00000	000000000000000110000110101	1	85	0.0	1.0	105.0
3	00001	000000000000010010101011000	1	97	0.010309278350515464	0.9896907216494846	103.96907216494846
4							
5							
6							
7	00101	000000000000000000001100101	1	98	0.01020408163265306	0.9897959183673469	103.9795918367347
8	00110	000000000000011000001011010	1	83	0.0	1.0	105.0
9							
10	01000	000000000000000011011110101	1	84	0.0	1.0	105.0
11	01001	000000000000000100100001110	1	65	0.0	1.0	105.0
12							
13							
14							
15	01101	000000000000010011110000101	1	89	0.0	1.0	105.0
16	01110	000000000000000111101110001	1	88	0.0	1.0	105.0
17	01111	000000000000000110101100001	1	87	0.0	1.0	105.0
18	10000	000000000000001101100000101	1	95	0.010526315789473684	0.9894736842105263	105.0
19	10001	000000000000010001100100111	1	96	0.010416666666666666	0.9895833333333334	103.95833333333334
20							
21	10011	000000000000000000111111011	1	90	0.0	1.0	105.0

32

4

5

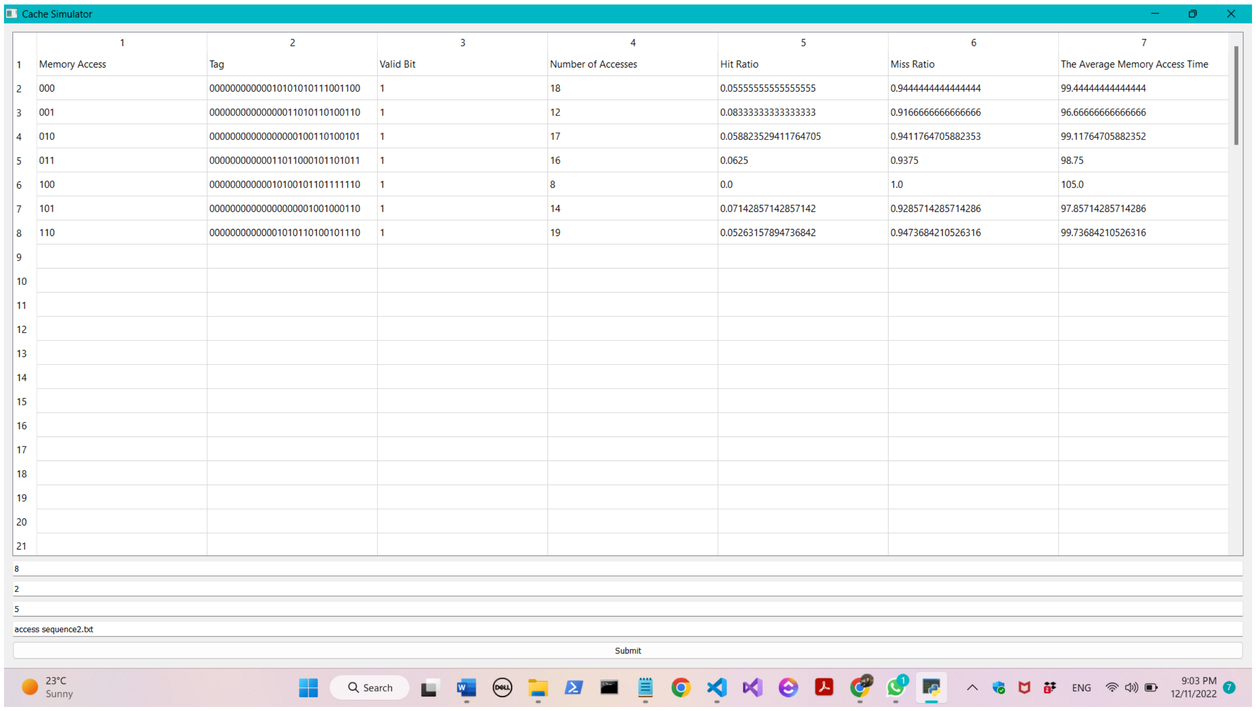
access sequence1.txt



```
hit ratio  0.0
miss ratio 1.0
Average number of cycles  105.0
00001
Tag  000000000000001100000110101
Valid Bit  1
Number of accesses  92
hit ratio  0.0
miss ratio 1.0
Average number of cycles  105.0
10001
Tag  000000000000001101001100101
Valid Bit  1
Number of accesses  93
hit ratio  0.0
miss ratio 1.0
Average number of cycles  105.0
10000
Tag  000000000000001101100000101
Valid Bit  1
Number of accesses  94
hit ratio  0.0
Valid Bit  1
Number of accesses  96
hit ratio  0.010416666666666666
miss ratio 0.9895833333333334
Average number of cycles  103.95833333333334
00001
Tag  000000000000010010101011000
Valid Bit  1
Number of accesses  97
hit ratio  0.010309278350515464
miss ratio 0.9896907216494846
Average number of cycles  103.96907216494846
00101
Tag  000000000000000000001100101
Valid Bit  1
Number of accesses  98
hit ratio  0.01020408163265306
miss ratio 0.9897959183673469
Average number of cycles  103.9795918367347
11101
Tag  000000000000010100110110101
Valid Bit  1
Number of accesses  99
```

Timeout after 2000 ms for 'terminate'

Test Case 3:



Cache Simulator

	1	2	3	4	5	6	7
1	Memory Access	Tag	Valid Bit	Number of Accesses	Hit Ratio	Miss Ratio	The Average Memory Access Time
2	000	00000000000010101010111001100	1	18	0.0555555555555555	0.9444444444444444	99.4444444444444
3	001	00000000000000011010110100110	1	12	0.0833333333333333	0.9166666666666666	96.6666666666666
4	010	00000000000000000100110100101	1	17	0.058823529411764705	0.9411764705882353	99.11764705882352
5	011	00000000000011011000101101011	1	16	0.0625	0.9375	98.75
6	100	00000000000010100101101111110	1	8	0.0	1.0	105.0
7	101	00000000000000000001001000110	1	14	0.07142857142857142	0.9285714285714286	97.85714285714286
8	110	0000000000001010110100101110	1	19	0.05263157894736842	0.9473684210526316	99.73684210526316
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							

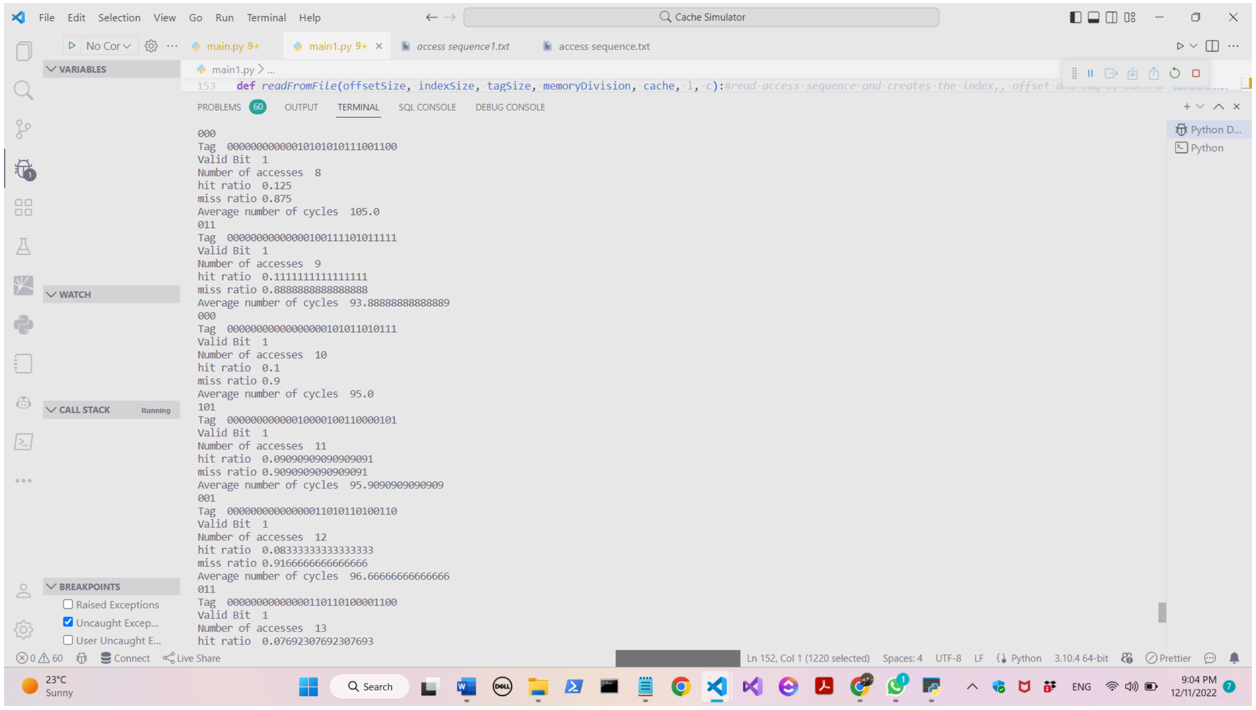
8

2

5

access sequence2.txt

Submit



```
def readFromFile(offsetSize, indexSize, tagSize, memoryDivision, cache, l, c):#read access sequence and creates the index, offset
```

```
000
Tag  00000000000010101010111001100
Valid Bit  1
Number of accesses  8
hit ratio  0.125
miss ratio 0.875
Average number of cycles  105.0
011
Tag  00000000000000100111101011111
Valid Bit  1
Number of accesses  9
hit ratio  0.1111111111111111
miss ratio 0.8888888888888888
Average number of cycles  93.88888888888889
000
Tag  00000000000000000101011010111
Valid Bit  1
Number of accesses  10
hit ratio  0.1
miss ratio 0.9
Average number of cycles  95.0
101
Tag  00000000000010000100110000101
Valid Bit  1
Number of accesses  11
hit ratio  0.09090909090909091
miss ratio 0.9090909090909091
Average number of cycles  95.9090909090909
001
Tag  00000000000000011010110100110
Valid Bit  1
Number of accesses  12
hit ratio  0.08333333333333333
miss ratio 0.9166666666666666
Average number of cycles  96.66666666666666
011
Tag  00000000000000110110100001100
Valid Bit  1
Number of accesses  13
hit ratio  0.07692307692307693
```