Experiment 4

Date of Performance: 11th October 2021

Date of Submission: 12th October 2021

Aim:

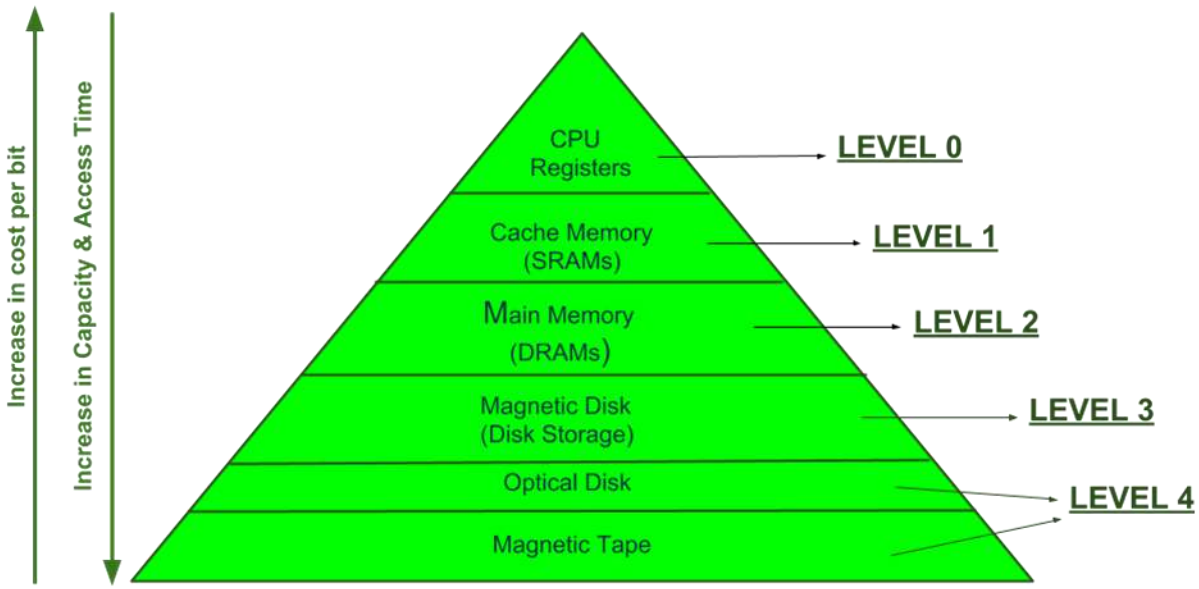
Implement Sequential Memory Organization with given details: Processor can access one word at a time (Word accessible memory), L1 cache can store max 32 words, L2 cache can store 128 words, main memory has the capacity of 2048 bytes. Consider TL1=20 ns, TL2=60ns, TMM=120ns. Create output table containing

Processor Request (Sp. Word)	Location of Hit	Average Memory Access Time

Theory:

Computer hardware consists of the following memory structure. The cache memory also consists of 2 parts: L1 cache and L2 cache.

A CPU cache is a hardware cache used by the central processing unit (CPU) of a computer to reduce the average cost (time) of accessing data from the main memory. A cache is a smaller, faster memory, located closer to a processor core, which stores copies of the data from frequently used main memory locations. Most CPUs have a hierarchy of multiple cache levels (L1, L2, often L3, and rarely even L4), with separate instruction-specific and data-specific caches at level 1.



MEMORY HIERARCHY DESIGN

A "cache hit" occurs when a file is requested from a cache and the cache is able to fulfil that request.
Sequential access is a method where for every word request from the processor, we first search the L1 cache, and if the requested word is not found, we then search for it in L2 and so on till we reach the main memory.

Formula to calculate sequential average access time:
Avg -Time = (H1* T1) + ((1- H1) * H2) *(T2+ T1) ((1- H1) *(1- H2) *(T1+ T2+ T3))
H1: hit ratio for L1
T1: time for accessing L1
H2: hit ratio for L2
T2: time for accessing L2
T3: time for accessing main memory

Code:

```
import random

L1_cache=[]
L2_cache=[]
l1p=0
l2p=0
low=0
high=511
L1_cache_hits=0
L2_cache_hits=0
l1_cache_time=20
l2_cache_time=60
main_memory_time=120

print('Word \t\t Location \t\t Time')
for i in range(0,100):
    word=random.randint(low,high)
    if(word in L1_cache):
        L1_cache_hits+=1
        avg1=int(((L1_cache_hits*1.0/(i+1))*l1_cache_time)
+((1-(L1_cache_hits*1.0/(i+1)))*(L2_cache_hits*1.0/(i+1))*
(l2_cache_time+l1_cache_time)*1.0)
         + ( (1-(L1_cache_hits*1.0/(i+1))) * (1-
(L2_cache_hits*1.0/(i+1))) *
(l2_cache_time+l1_cache_time+main_memory_time)*1.0 ) )
        print(str(word)+"\t\t in L1 \t\t"+str(avg1))
        continue
    elif(word in L2_cache):
        L2_cache_hits+=1
```

```
        avg2=int(((L1_cache_hits*1.0/(i+1))*l1_cache_time)
+((1-
(L1_cache_hits*1.0/(i+1)))*(L2_cache_hits*1.0/(i+1))*(l2_cache
_time+l1_cache_time)*1.0 )
        + ( (1-(L1_cache_hits*1.0/(i+1))) * (1-
(L2_cache_hits*1.0/(i+1))) *
(l2_cache_time+l1_cache_time+main_memory_time)*1.0 ) )
        print(str(word)+"\t\t in L1 \t\t"+str(avg2))
        if(len(L1_cache)<32):
            L1_cache.append(word)
        else:
            l1p=(l1p+1)%32
            L1_cache.insert(l1p,word)
        continue
    else:

avg3=int(((L1_cache_hits*1.0/(i+1))*l1_cache_time)+((1-
(L1_cache_hits*1.0/(i+1)))*(L2_cache_hits*1.0/(i+1))*(l2_cache
_time+l1_cache_time)*1.0 )
        + ( (1-(L1_cache_hits*1.0/(i+1))) * (1-
(L2_cache_hits*1.0/(i+1))) *
(l2_cache_time+l1_cache_time+main_memory_time)*1.0 ) )
        print(str(word)+"\t\t in MM \t\t"+str(avg3))
        if(len(L1_cache)<32):
            L1_cache.append(word)
        else:
            L1_cache.insert(l1p,word)
            l1p=(l1p+1)%32
        if(len(L2_cache)<128):
            L2_cache.insert(l2p,word)
            L2_cache.append(word+1)
        else:
            L2_cache.insert(l2p,word-1)
            L2_cache.insert(l2p+1,word)
            l2p = (l2p+2) % 64
```

Output:

Word	Location	Time
488	in MM	200
164	in MM	200
429	in MM	200
175	in MM	200
175	in L1	164
146	in MM	170

379 in MM 174
484 in MM 177
335 in MM 180
120 in MM 182
349 in MM 183
430 in L2 175
173 in MM 177
80 in MM 179
359 in MM 180
172 in MM 181
438 in MM 182
43 in MM 183
62 in MM 184
415 in MM 185
487 in MM 185
253 in MM 186
335 in L1 179
173 in L1 173
8 in MM 174
94 in MM 175
111 in MM 176
1 in MM 176
298 in MM 177
206 in MM 178
128 in MM 179
446 in MM 179
32 in MM 180
287 in MM 180
260 in MM 181
116 in MM 181
18 in MM 182
399 in MM 182
56 in MM 183
467 in MM 183
278 in MM 184
84 in MM 184
132 in MM 184
100 in MM 185
240 in MM 185
14 in MM 185
244 in MM 186
368 in MM 186
113 in MM 186
495 in MM 186

161	in MM	187
378	in MM	187
291	in MM	187
364	in MM	187
75	in MM	188
75	in L1	185
270	in MM	185
446	in L1	182
326	in MM	182
502	in MM	183
295	in MM	183
374	in MM	183
504	in MM	183
156	in MM	184
63	in L2	182
13	in MM	183
91	in MM	183
471	in MM	183
421	in MM	183
469	in MM	183
171	in MM	184
128	in L1	181
59	in MM	182
186	in MM	182
97	in MM	182
506	in MM	182
406	in MM	183
387	in MM	183
2	in L2	182
390	in MM	182
113	in L1	180
71	in MM	180
441	in MM	180
26	in MM	181
128	in L1	179
251	in MM	179
402	in MM	179
10	in MM	179
492	in MM	180
142	in MM	180
188	in MM	180
17	in MM	180
295	in L1	179
126	in MM	179

392	in MM	179
138	in MM	179
361	in MM	179
442	in MM	180
354	in MM	180
12	in MM	180

Conclusion:

Sequential access is a method where for every word request from the processor, we search the L1 cache first and in case of a cache miss, we try searching in L2 and so on until we reach the main memory. Sequential memory access is used time and again in our computers. The caches are used to reduce the access time by saving the words/digits which are requested by the processor most frequently. In this experiment, we have implemented sequential memory interfacing in python. We are given a setup where we have L1 cache, L2 cache and Main Memory, having access times of 20ns, 60ns and 120ns respectively.