Question 1:

```
#define ARRAY_SIZE 1024000
int main(){
    int a[ARRAY_SIZE];
    register int b = 1;
    int i = 0;
    for (; i<ARRAY_SIZE; i++){
        a[i] = b;
    }
    return 0;
}
```

Above is the microbenchmark that we used to verify the correctness of the next line prefetcher. In the microbenchmark we are accessing sequential data AKA next line data in the array and initializing the values inside of the array. This verifies the design because with a program that specifically does sequential accesses, we would expect the performance of the next line prefetcher to be perfect.

Config	**L1 Miss Rate**	**L2 Miss Rate**
No prefetcher	1.25%	50.17%
Next Line	0%	0.31%

This is what we verify. When running this benchmark with a large array_size the miss rate was 0% which was expected.

We used the configuration file provided for the next line prefetcher.

Question 2:

```
#define ARRAY_SIZE 10240000
#define STRIDE 64
int main(){
    int a[ARRAY_SIZE];
    register int b = 1;
    register int i = 0;
    for (; i<ARRAY_SIZE; i+=STRIDE){
        a[i] = b;
    }
    return 0;
}
```

Above is mbq2.c which was used to microbenchmark the stride prefetcher. This program was chosen to have a steady stride and the strides exceeded one block size. Adding an additional increment in the for loop with a defined stride had the effect. We expect that this will cause the

next line prefetcher to perform poorly, similar to not having a prefetcher since the benchmark is not accessing any sequential information. It is also expected that stride will be able to have 0% miss rate.

Config	**L1 Miss Rate**	**L2 Miss Rate**
No prefetcher	97.81%	50.06%
Next Line	97.7%	50.03%
Stride	0%	0.12%

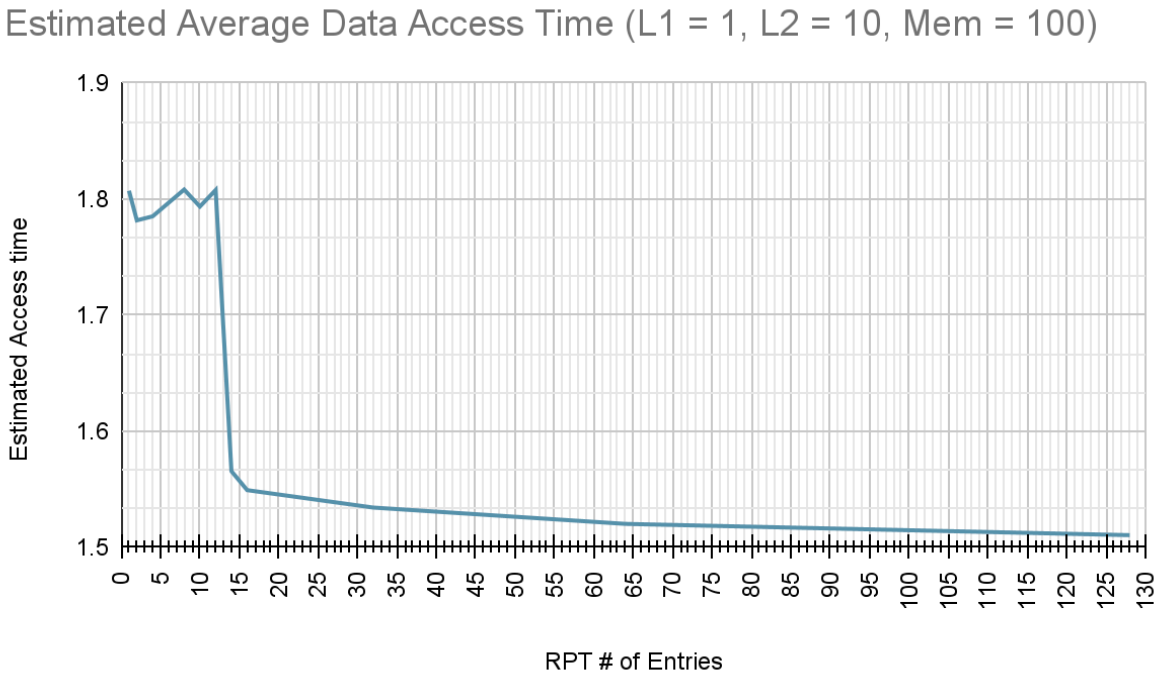
The result shows that stride performs perfectly while next line and not having a prefetcher are mostly equivalent. This verifies the design.

The configuration file that we used was the provided one for the stride prefetcher.

Question 3:
Access time = (1 - L1 miss rate)(L1 access time) + (L1 miss rate)((1 - L2 miss rate)(L2 access time) + (L2 miss rate)(Memory access time))

Config	**L1 Miss Rate**	**L2 Miss Rate**	**Avg Access Time**
No prefetch	4.16%	11.40%	1.8012
Next Line	4.16%	8.41%	1.6893
Stride	3.87%	5.77%	1.5493

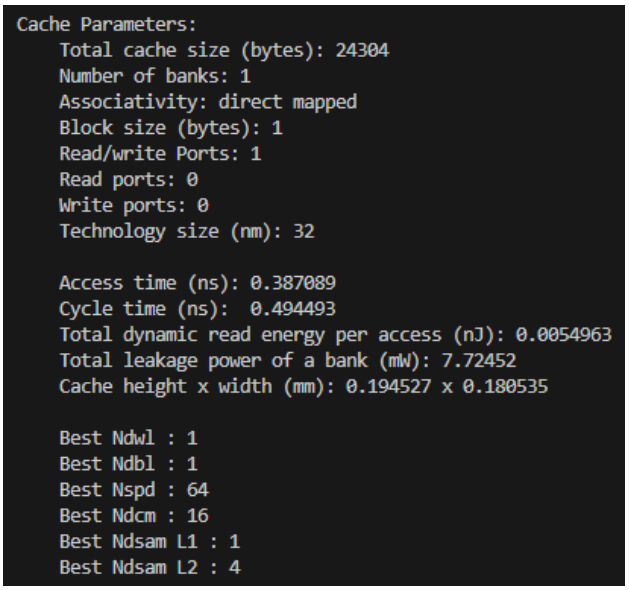
Question 4:
For the metric of prefetcher performance, we decided to use the estimated data access time with the access times provided in question 3. There is a steep improvement to data access times as the RPT entry count crossed 14 entries. This was probably around the time where repeated data access addresses stopped aliasing for compress.eio. There are minor improvements to the performance as the entry count increases, but we don't see another major improvement to performance, most likely because all the other misses are due to unpredictable strides.

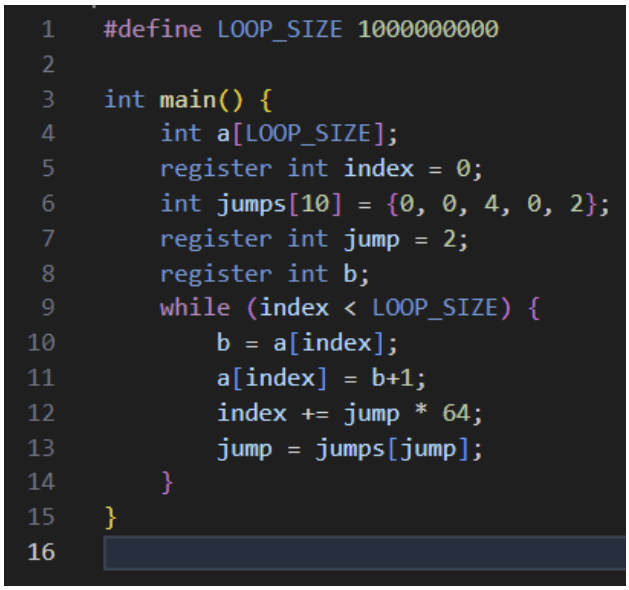


Question 5:
Early and late prefetch count:
This statistic would be the total number of prefetches that were evicted before they were used and prefetches that were fetched after it had already cache missed. For this current lab, we assume that prefetches happen instantly, but for prefetchers in general, timeliness is an important factor. Therefore, I think this would be useful to be able to gauge the aggressiveness of prefetchers and if they are fetching too early or too late.

Question 6:
The open ended prefetcher is a Delta-Correlating Prediction Tables (DCPT) prefetcher. This prefetcher is similar to the stride prefetcher but for each entry in the RPT, there is a linked list to keep track of the previous 19 strides (deltas). During prefetch, after a matching entry has been found, the previous delta (d1) and the previous previous delta (d2) are searched for in the deltas linked list. If a match has been found, then all deltas in the future of the match are sequentially

added to the current address and prefetched. This means that the DCPT prefetcher can perfectly predict any delta patterns with a period of 2, and provide between 1 to 17 prefetches into the future.





There are 98 entries at most in our RPT table, and each entry can have at most 19 linked list elements. The total number of bytes that the DCPT prefetcher uses is at most 24kB. From CACTI, the overhead total physical size is 0.035118mm^2 and the access time is about 0.387ns. For an SOC with 400mm^2 per tile, our prefetcher physically would only be 0.0088% of the total area. The access time means that the prefetcher is able to keep up with a 2.58GHz processor even if it were accessing memory every single cycle. This means that our prefetcher should be small and fast enough to be used in a real processor.

Here is the microbenchmark used to test the open ended prefetcher. This main function accesses an array with varying jumps to the index. The idea is to cause the stride prefetcher to be unable to be able to find a steady state stride. Furthermore, both jumps are also large enough that the next line prefetcher would fail to fetch any useful data. The configuration used is the default open prefetcher configuration, and the result is as follows:

Config	**L1 Miss Rate**	**L2 Miss Rate**
No prefetcher	33%	50%
Next Line	33%	50%
Stride	33%	50%
Open (DCPT)	0%	0%

The DCPT prefetcher was able to predict the jumps perfectly since it is able to capture any pattern with a period of 2 or less.