My cache data structure: 2D structure array

```
block **cache;
cache = malloc(set_number*sizeof(block *));
for (int i = 0; i < set_number; i=i+1)
{
   cache[i] = malloc (block_per_set*sizeof(block));
}
```

A summary of your function to parse the address:
Get size of offset, index and tag according to blocksize, cache size and way. Then use shift operation to get each element. (since we don't need offset in assignment, I just skip offset part.)

```
typedef struct
{
   int C;
   int S;
   int B;
   int blocks;
   int caches;
   int wayss;
}info;

inf = malloc(sizeof(info));
inf->C=(log(cachesize)/log(2));
inf->S=(log(ways)/log(2));
inf->B=(log(blocksize)/log(2));

addr_t offsetbit(info *inf){
   return inf->B;
}

addr_t indexbit(info *inf){
   return inf->C-inf->B-inf->S;
}

addr_t tagbit(info *inf){
   return 64-offsetbit(inf)-indexbit(inf);
}

addr_t getindex(addr_t address,info *inf){
   return  (address<<(tagbit(inf)))>> (tagbit(inf)+offsetbit(inf));
}
addr_t gettag(addr_t address,info *inf){
return address>>(offsetbit(inf)+indexbit(inf));
```

}

A summary of how you implement your cache LRU stack:
Actually I do not use stack for LRU, I use a globe counter. Each time I access a block, I will increase the counter and assign the counter value to my block's LRU value.

Result discussion

```
lawn-128-61-121-91:p4 fly$ ./cachesim trace.bubble 16 65536 4
6322343, 6288550, 33793, 11833
lawn-128-61-121-91:p4 fly$ ./cachesim trace.bubble 16 65536 8
6322343, 6289037, 33306, 11649
lawn-128-61-121-91:p4 fly$ ./cachesim trace.bubble 16 65536 16
6322343, 6289353, 32990, 11589
```

From 4 block per line to 8 block per line, we have 1.5% miss rate decrease.
From 8 blocks per line to 8 blocks per line, we have 0.5% miss rate improve.

I will choose to have 8 blocks per line

For trace.bubble:
(Block size)/(cache size)/(blocks per line)

```
Flys-MacBook-Air:p4 fly$ ./cachesim trace.bubble 16 65536 8
6322343, 6289037, 33306, 11649
```

(accesses, hits, misses, writebacks).

```
Flys-MacBook-Air:p4 fly$ ./cachesim trace.bubble 32 65536 8
6322343, 6303240, 19103, 6438

Flys-MacBook-Air:p4 fly$ ./cachesim trace.bubble 64 65536 8
6322343, 6310453, 11890, 3820
```

```
Flys-MacBook-Air:p4 fly$ ./cachesim trace.bubble 128 65536 8
6322343, 6314008, 8335, 2494

Flys-MacBook-Air:p4 fly$ ./cachesim trace.bubble 256 65536 8
6322343, 6315555, 6788, 1831

Flys-MacBook-Air:p4 fly$ ./cachesim trace.bubble 512 65536 8
6322343, 6315717, 6626, 1587

Flys-MacBook-Air:p4 fly$ ./cachesim trace.bubble 1024 65536 8
6322343, 6313998, 8345, 1781
```

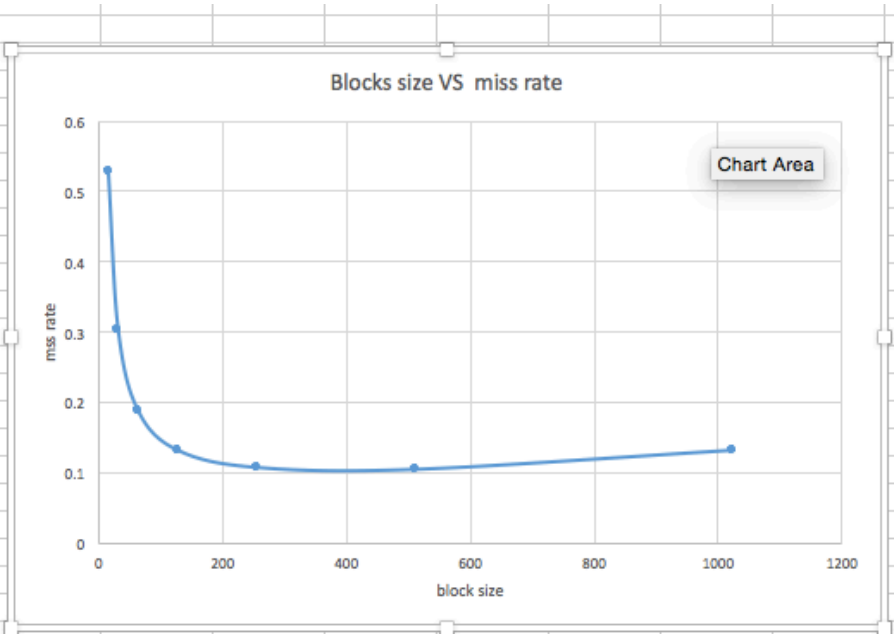
For trace.merge:

```
Flys-MacBook-Air:p4 fly$ ./cachesim trace.merge 16 65536 8
7678430, 7626173, 52257, 23308

Flys-MacBook-Air:p4 fly$ ./cachesim trace.merge 32 65536 8
7678430, 7648970, 29460, 12618

Flys-MacBook-Air:p4 fly$ ./cachesim trace.merge 64 65536 8
7678430, 7660543, 17887, 7191

Flys-MacBook-Air:p4 fly$ ./cachesim trace.merge 128 65536 8
7678430, 7666333, 12097, 4450

Flys-MacBook-Air:p4 fly$ ./cachesim trace.merge 256 65536 8
7678430, 7669051, 9379, 2996

Flys-MacBook-Air:p4 fly$ ./cachesim trace.merge 512 65536 8
7678430, 7669718, 8712, 2324

Flys-MacBook-Air:p4 fly$ ./cachesim trace.merge 1024 65536 8
7678430, 7668391, 10039, 2266
```

The best configuration we have is 512 65536 8

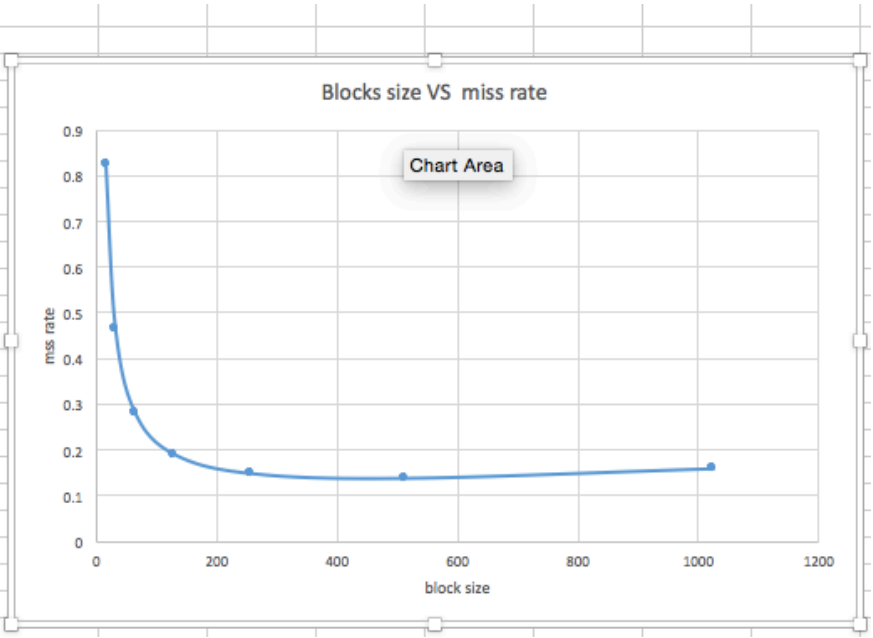
Which mean block size is 512, cache is 65536 and 8 block per line.
For trace.bubble, plot for miss rate VS block size

16	0.52679837
32	0.30215064
64	0.18806319
128	0.13183404
256	0.10736526
512	0.10480292
1024	0.13199221



For trace.merge:

size	miss
16	0.82654484
32	0.46596649
64	0.28291727
128	0.19133729
256	0.1483469
512	0.13779702
1024	0.15878607



Compute the overall miss rate, the read miss rate, and the write miss rate. For example, the read miss rate is the ratio of the total number of read misses to the total number of read operations.

Trace.bubble:
Cache: 512,65536,8
Overall missrate: 0.104%;
Read miss rate:0.09%;
Write miss rate:0.014%;

```
lawn-128-61-121-91:p4 fly$ ./cachesim trace.bubble 512 65536 8
6322343, 6315717, 6626,1587,6008(read miss), 618(write miss)
```

Compute the volume of write-back traffic in bytes:

```
Block size * write back = 512*1587 = 812544
```

Compute the total memory access volume. This is the total number of bytes fetched from memory. Compare this against the total number of memory references. How much memory access volume (in bytes) did the cache save?

```
Total memory access:

Overall miss* block size = 6626*512 = 3392512

Overall memory reference: 6322343*4 = 25289372

Volume cache save: hit * 4 = 6315717*4 = 25262868
```

Trace.merge:
Cache: 512,65536,8
Overall missrate: 0.1378%;
Read miss rate:0.1 %;
Write miss rate:0.01378 %;

```
lawn-128-61-121-91:p4 fly$ ./cachesim trace.merge 512 65536 8
7678430, 7669718, 8712,2324,7686(read miss), 1026(write miss)
```

Compute the volume of write-back traffic in bytes:

```
Block size * write back = 2324*512 = 1189888;
```

Compute the total memory access volume. This is the total number of bytes fetched from memory. Compare this against the total number of memory references. How much memory access volume (in bytes) did the cache save?

```
Total memory access:

Overall miss* block size = 8712*512=4460544

Overall memory reference: 7678430*4 = 30713720

Volume cache save: hit*4 = 7669718 * 4 = 30678872
```