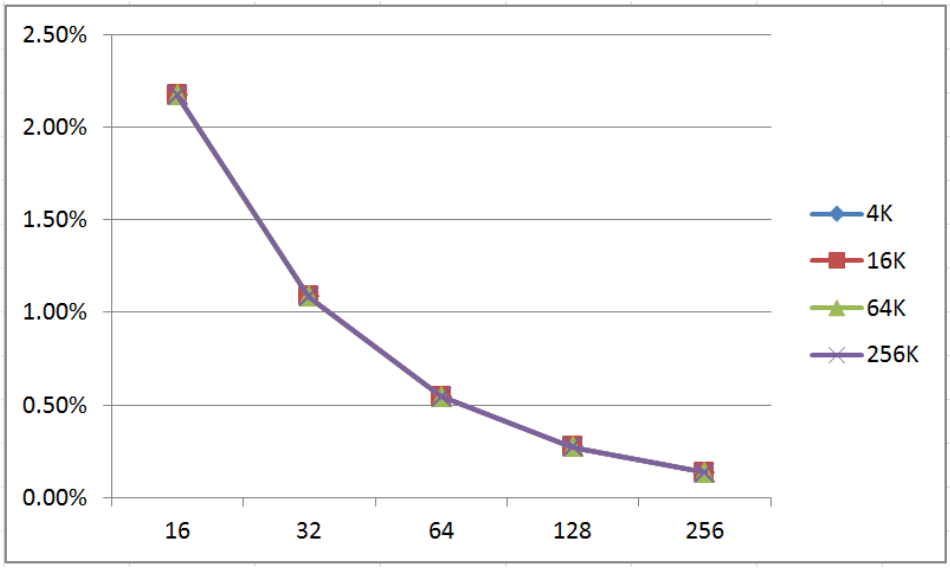
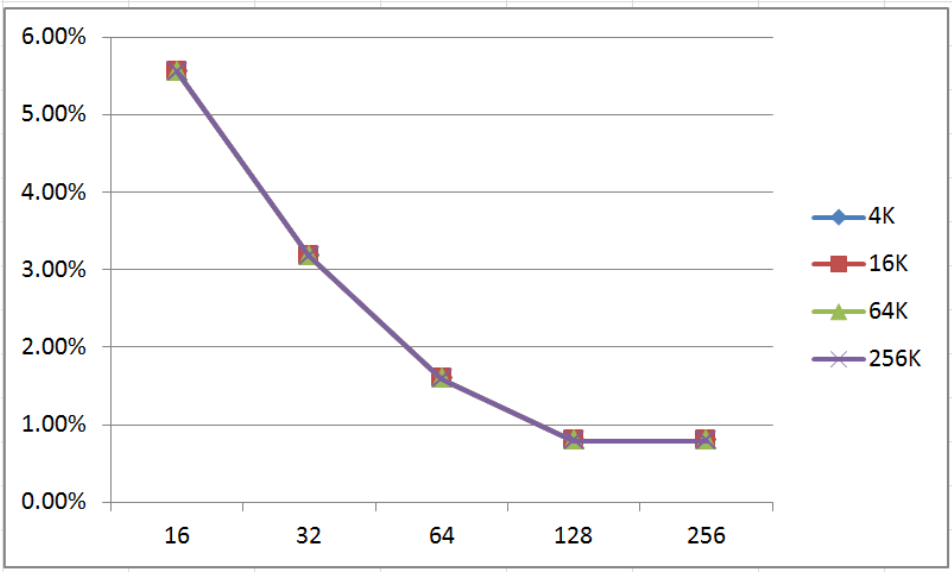
ICACHE



	16	32	64	128	256
4K	2.17096	1.08548	0.542741	0.27137	0.135685
16K	2.17096	1.08548	0.542741	0.27137	0.135685
64K	2.17096	1.08548	0.542741	0.27137	0.135685
256K	2.17096	1.08548	0.542741	0.27137	0.135685

當 block size 增大，會因為 spatial locality 的特性降低 miss rate.
但一旦 cache size 是固定的,又無節制的提高 block size，會發生 pollution，提高 miss penalty.

DCACHE

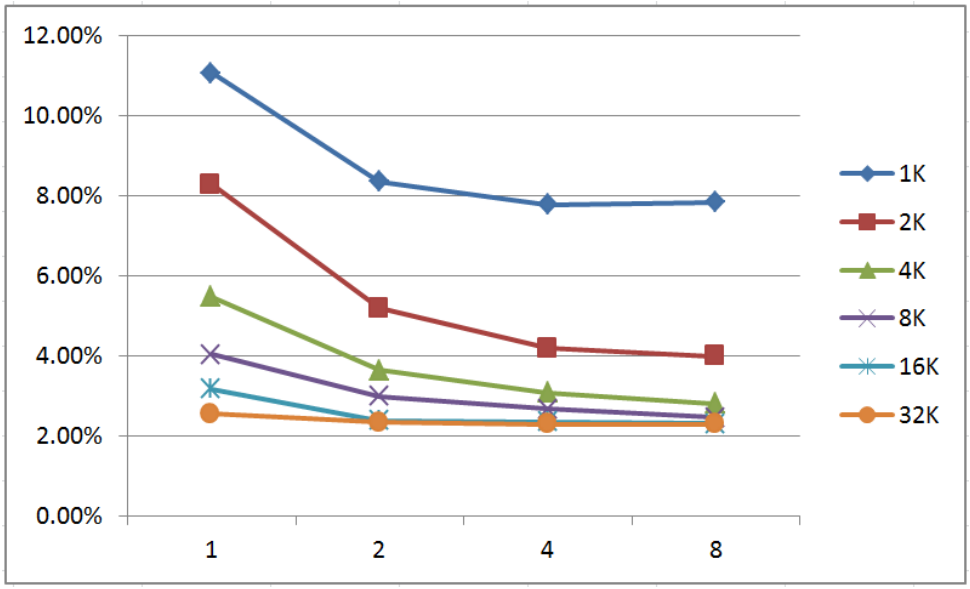


	16	32	64	128	256
4K	5.55556(%)	3.1746(%)	1.5873(%)	0.793651(%)	0.793651(%)
16K	5.55556(%)	3.1746(%)	1.5873(%)	0.793651(%)	0.793651(%)
64K	5.55556(%)	3.1746(%)	1.5873(%)	0.793651(%)	0.793651(%)
256K	5.55556(%)	3.1746(%)	1.5873(%)	0.793651(%)	0.793651(%)

當 block size 增大，會因為 spatial locality 的特性降低 miss rate.
但一旦 cache size 是固定的,又無節制的提高 block size，會發生 pollution，提高 miss penalty.

total bits	1	2	4	8
1K	8560	8576	8592	8608
2K	17088	17120	17152	17184
4K	34112	34176	34240	34304
8K	68096	68224	68352	68480
16K	135936	136192	136448	136704
32K	271360	271872	272384	272896

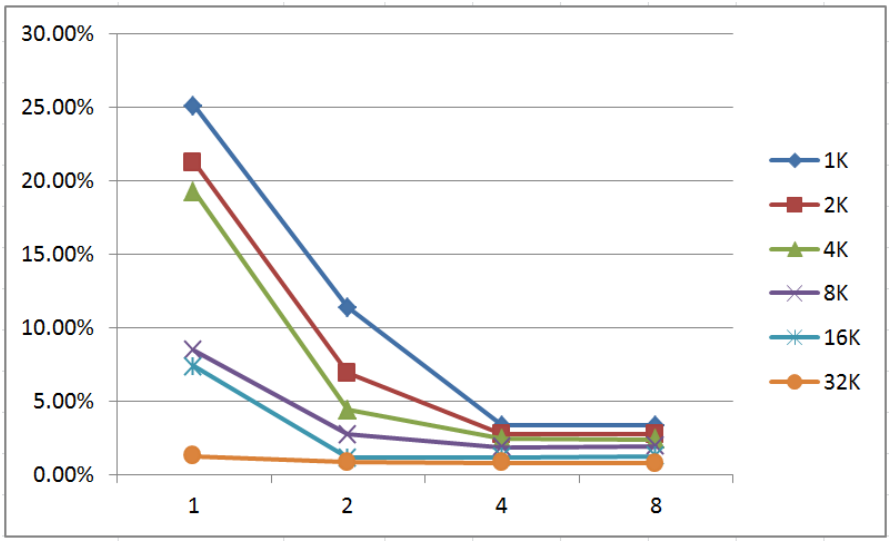
LU



	1	2	4	8
1K	11.07%	8.36%	7.78%	7.83%
2K	8.28%	5.18%	4.19%	3.98%
4K	5.47%	3.63%	3.07%	2.81%
8K	4.03%	2.98%	2.67%	2.45%
16K	3.16%	2.37%	2.34%	2.29%
32K	2.54%	2.33%	2.28%	2.28%

當 block size 增大，會因為 spatial locality 的特性降低 miss rate.
但一旦 cache size 是固定的,又無節制的提高 block size，會發生 pollution，提高 miss penalty.
(圖中的 1K 即為例子)

RADIX



	1	2	4	8
1K	25.09%	11.37%	3.35%	3.33%
2K	21.22%	6.87%	2.73%	2.70%
4K	19.26%	4.37%	2.40%	2.37%
8K	8.43%	2.71%	1.86%	1.93%
16K	7.36%	1.12%	1.17%	1.24%
32K	1.23%	0.81%	0.78%	0.76%

當 block size 增大，會因為 spatial locality 的特性降低 miss rate.
但一旦 cache size 是固定的,又無節制的提高 block size，會發生 pollution，提高 miss penalty.
(圖中的 8K & 16K 即為例子)

Description:

lab5 使用了 lab3 的 CPU
為了配合 TestBench.v，作了以下修改
- 修改 Simple_Single_CPU.v 為 CPU.v
- 修改 CPU.v 的 port, rst_i 為 start_i
- 用了助教給的 Instruction_Memory.v，所以 CPU 裡面模組名稱有換
- 修改 CPU.v 裡面 Data_Memory Data_Memory 為 Data_Memory DM

cpp code 裡面的 function, double log2(double n)有誤，所以有更改

Implementation:

direct_mapped_cache.cpp,

只要加上 miss_count 跟 hit_count，就能印出 miss rate.

direct_mapped_cache_lru.cpp,

因為是 n-way set，所以 tag 要變成一維 array，去分別記錄 n 個 tag.
以及 lru 是指說把最久以前才被使用到的那筆 record 給 replace 掉.
所以也是用一個 array 去紀錄被 access 到的時間,
因為兩者可以連在一起紀錄，所以我就多創了一個 struct P.
然後 valid bit 也是一個 block 一個，也是用 array 去記，所以三個可以連在一起觀察.
然後如果 miss 發生的話，再用 for loop 掃，決定出哪筆 record 最久以前才被用到，再 replace 掉.