计算机组织与体系结构实习报告 Lab3.1

学号：1600012830

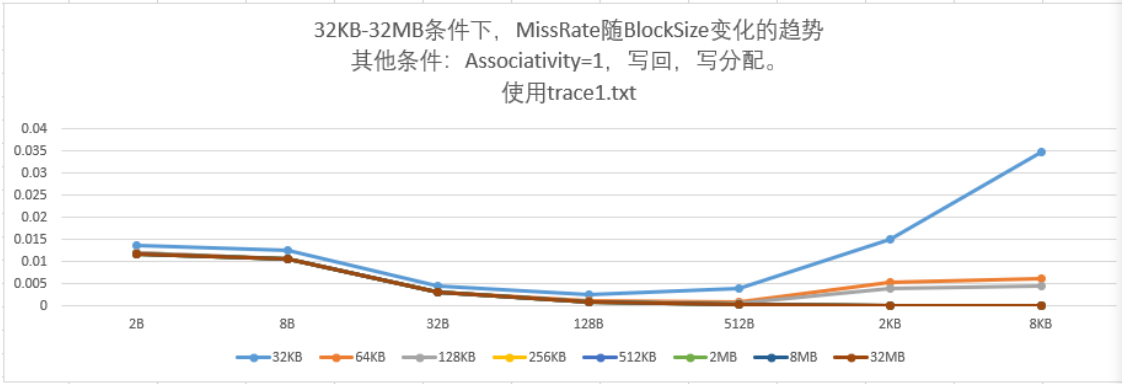
姓名：盛朱恒

PART 1、单层Cache模拟（100分）

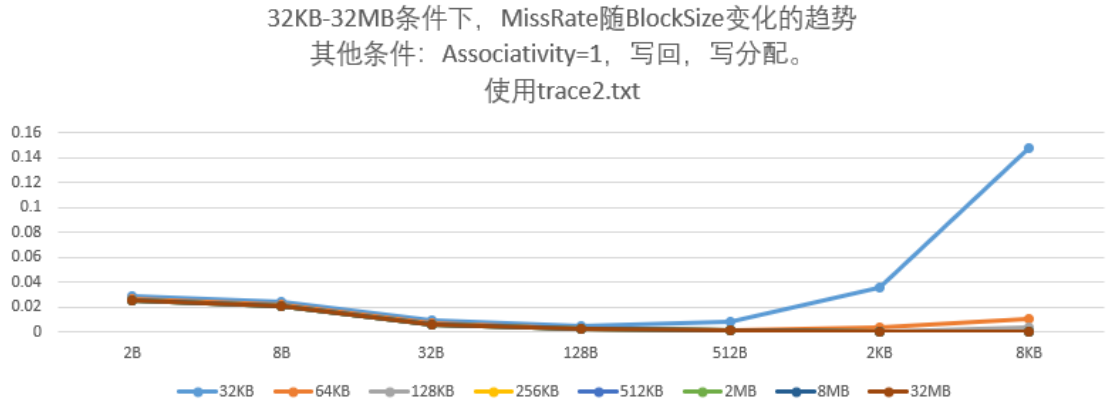
在单层Cache模拟的实现过程中，从不进行Bypass和Prefetch。

使用附件中所给模拟器框架，使用给定的测试trace，完成单层cache的模拟。要求：

1. 在不同的 Cache Size （ 32KB ~ 32MB） 的条件下， Miss Rate 随 Block Size变化的趋势，收集数据并绘制折线图。并说明变化原因。至少有4个不同的size大小对应的折线图。（40分）

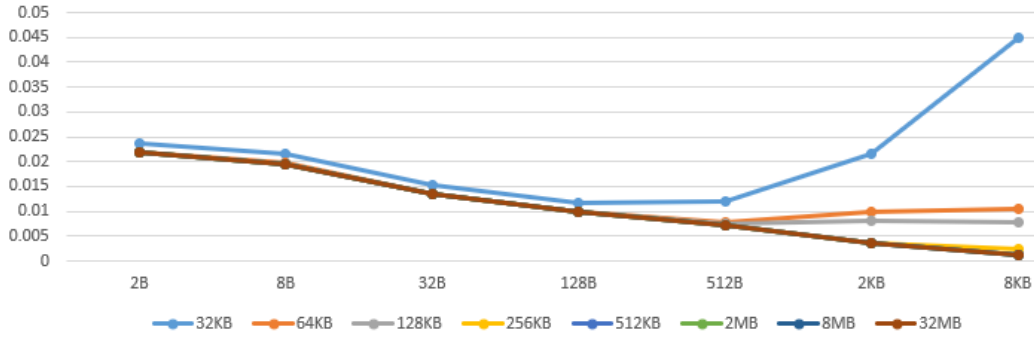


	32KB	64KB	128KB	256KB	512KB	2MB	8MB	32MB
2B	0.013714	0.011928	0.011916	0.011776	0.011773	0.011773	0.011773	0.011773
8B	0.012529	0.010703	0.010689	0.010546	0.010544	0.010544	0.010544	0.010544
32B	0.004333	0.003104	0.003089	0.002987	0.002985	0.002985	0.002985	0.002985
128B	0.002443	0.001004	0.000985	0.000879	0.000877	0.000877	0.000877	0.000877
512B	0.004031	0.000739	0.000713	0.000308	0.000301	0.000301	0.000301	0.000301
2KB	0.014945	0.005237	0.004029	0.000132	0.000121	0.000121	0.000121	0.000121
8KB	0.034628	0.006029	0.004483	0.000089	0.000053	0.000053	0.000053	0.000053



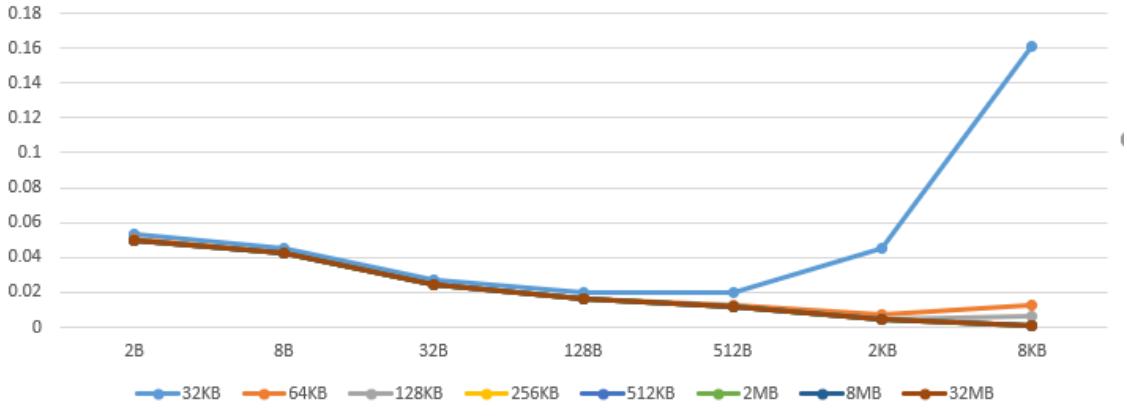
	32KB	64KB	128KB	256KB	512KB	2MB	8MB	32MB
2B	0.028589	0.026101	0.025166	0.025166	0.025166	0.025166	0.025166	0.025166
8B	0.023601	0.021231	0.020409	0.020409	0.020409	0.020409	0.020409	0.020409
32B	0.008682	0.006604	0.006216	0.00621	0.00621	0.00621	0.00621	0.00621
128B	0.005202	0.002275	0.001892	0.00188	0.00188	0.00188	0.00188	0.00188
512B	0.007544	0.001289	0.000704	0.000692	0.000687	0.000687	0.000687	0.000687
2KB	0.035723	0.003023	0.000428	0.000321	0.000304	0.000298	0.000298	0.000298
8KB	0.147964	0.010472	0.00375	0.000327	0.000158	0.000146	0.000146	0.000146

32KB-32MB条件下，MissRate随BlockSize变化的趋势
其他条件：Associativity=1，写穿透，写不分配。
使用trace1.txt



	32KB	64KB	128KB	256KB	512KB	2MB	8MB	32MB
2B	0.023674	0.02191	0.021897	0.021857	0.021855	0.021855	0.021855	0.021855
8B	0.021443	0.019641	0.019626	0.019583	0.019581	0.019581	0.019581	0.019581
32B	0.015187	0.013603	0.013584	0.013527	0.013525	0.013525	0.013525	0.013525
128B	0.011733	0.009944	0.009905	0.009825	0.009823	0.009823	0.009823	0.009823
512B	0.011856	0.007659	0.007623	0.007171	0.007167	0.007167	0.007167	0.007167
2KB	0.021668	0.009975	0.00819	0.00349	0.003475	0.003475	0.003475	0.003475
8KB	0.045015	0.010455	0.007744	0.002395	0.001331	0.001331	0.001331	0.001331

32KB-32MB条件下，MissRate随BlockSize变化的趋势
其他条件：Associativity=1，写穿透，写不分配。
使用trace2.txt

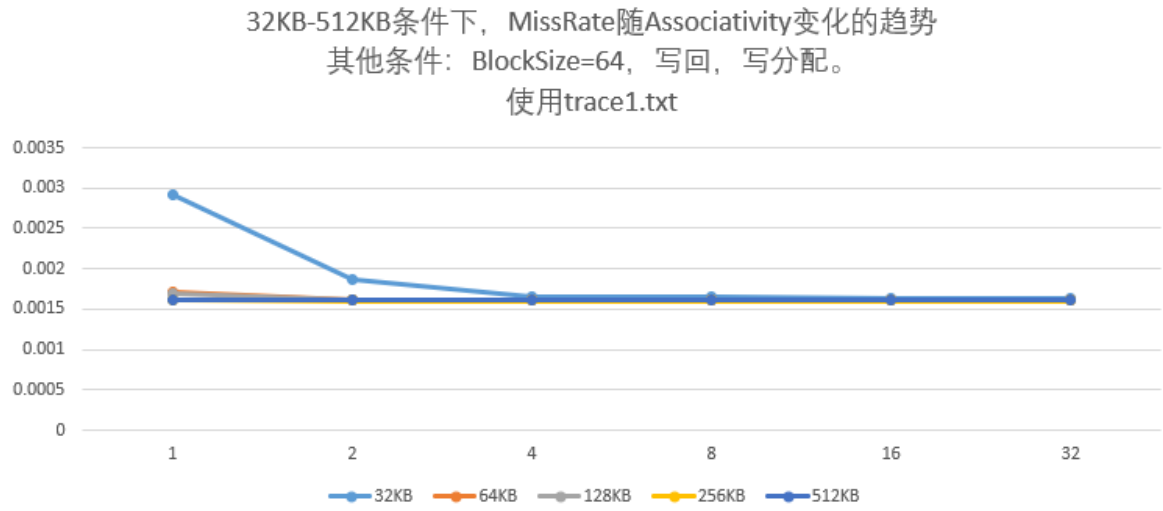


	32KB	64KB	128KB	256KB	512KB	2MB	8MB	32MB
2B	0.053238	0.050642	0.049905	0.049905	0.049905	0.049905	0.049905	0.049905
8B	0.045378	0.042895	0.042192	0.042192	0.042192	0.042192	0.042192	0.042192
32B	0.02721	0.024564	0.024215	0.024215	0.024215	0.024215	0.024215	0.024215
128B	0.019992	0.016333	0.015933	0.015927	0.015927	0.015927	0.015927	0.015927
512B	0.019643	0.012335	0.01175	0.011744	0.011739	0.011739	0.011739	0.011739
2KB	0.045395	0.007702	0.004983	0.004887	0.00487	0.004864	0.004864	0.004864
8KB	0.160862	0.012628	0.005951	0.00125	0.001081	0.00107	0.00107	0.00107

根据以上四种不同的情况下的结果，可以得出结论：Miss Rate随着Block Size的变化是先减小，再增大。并且这一变化点随着Cache Size的增大而增大。（如果Cache Size足够大，那么这一转折点在图中会无法表现出来。）

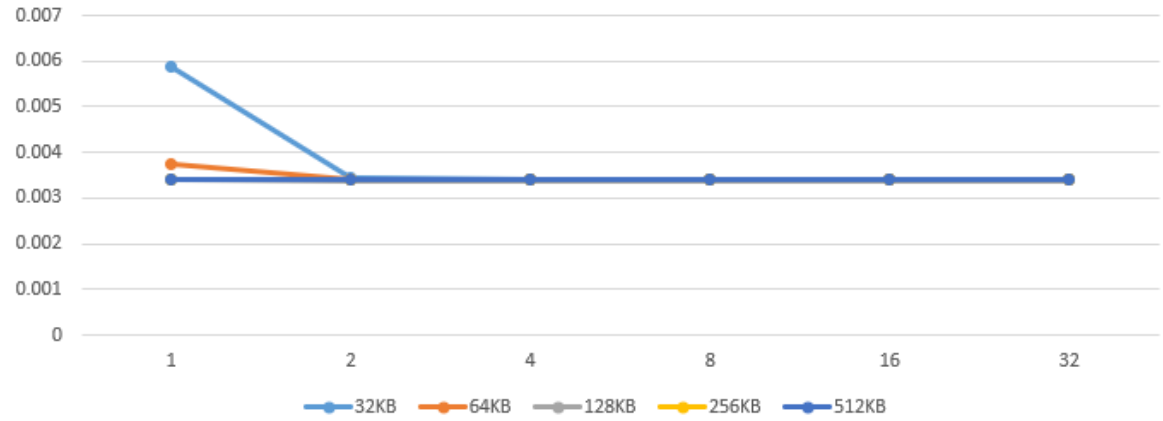
这一变化出现的原因是因为，在Cache Size固定的情况下，Block Size增大会导致Line数量的减少。也就是说，虽然每一个Line中所能够缓存的字节数增大了，但是Line的总数减少了。每一个Line中所能够缓存的字节数增大可以带来收益，但是Line的总数减少会造成损失。当超过零界点后，损失大于收益，那么Miss Rate会上升。

2. 在给定 Cache Size 的条件下，如128KB和512KB， Miss Rate 随 Associativity（ 1-32） 变化的趋势，收集数据并绘制折线图。并说明变化原因。至少有2、4、8、16、32对应的折线图。（40分）



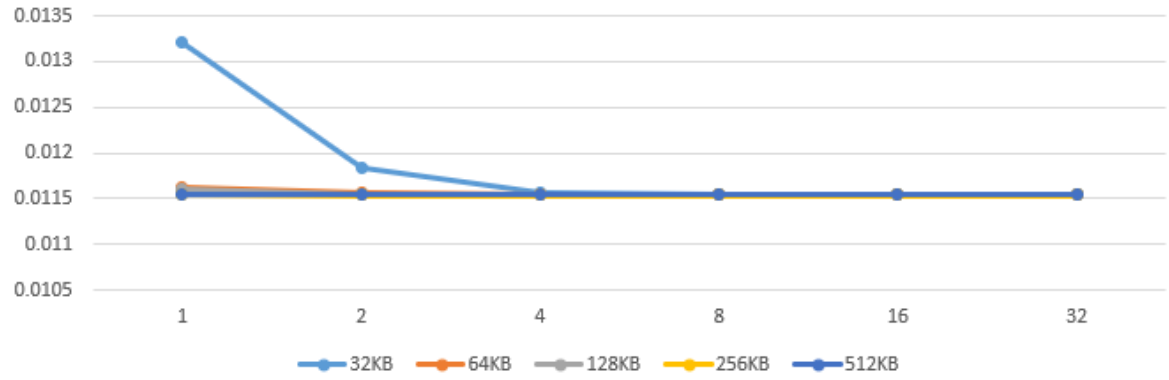
	32KB	64KB	128KB	256KB	512KB
1	0.002927	0.001713	0.001698	0.001611	0.001609
2	0.001868	0.001611	0.001609	0.001609	0.001609
4	0.00166	0.001609	0.001609	0.001609	0.001609
8	0.001645	0.001609	0.001609	0.001609	0.001609
16	0.00163	0.001609	0.001609	0.001609	0.001609
32	0.001635	0.001609	0.001609	0.001609	0.001609

32KB-512KB条件下，MissRate随Associativity变化的趋势
其他条件：BlockSize=64，写回，写分配。
使用trace2.txt

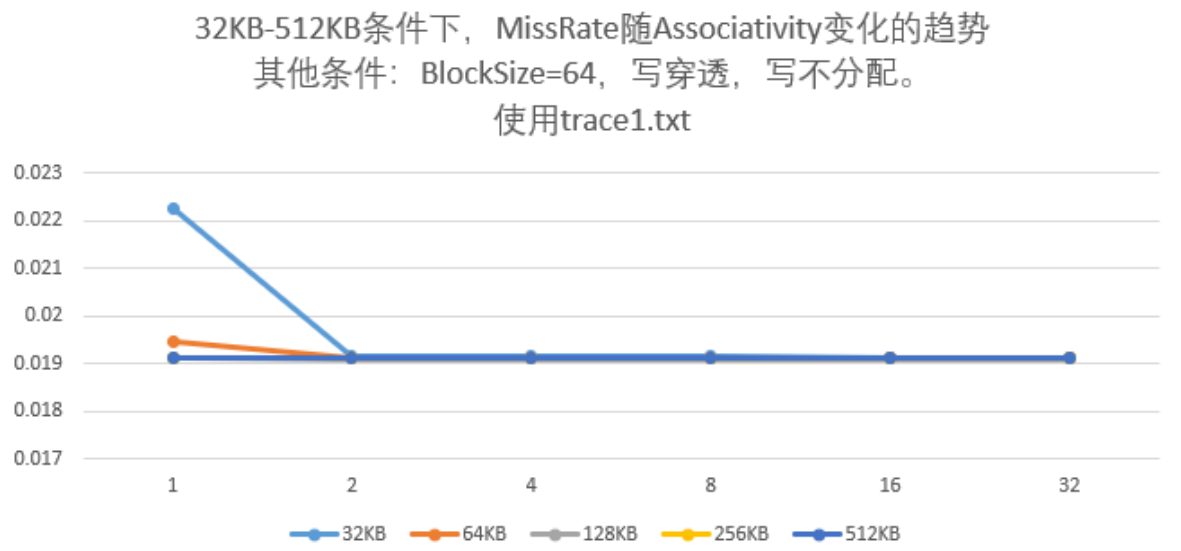


	32KB	64KB	128KB	256KB	512KB
1	0.005895	0.003755	0.003406	0.003395	0.003395
2	0.003451	0.003406	0.003395	0.003395	0.003395
4	0.003417	0.003401	0.003395	0.003395	0.003395
8	0.003417	0.003395	0.003395	0.003395	0.003395
16	0.003412	0.003395	0.003395	0.003395	0.003395
32	0.003401	0.003395	0.003395	0.003395	0.003395

32KB-512KB条件下，MissRate随Associativity变化的趋势
其他条件：BlockSize=64，写穿透，写不分配。
使用trace1.txt



	32KB	64KB	128KB	256KB	512KB
1	0.013215	0.011625	0.011601	0.011546	0.011544
2	0.011829	0.011557	0.011544	0.011544	0.011544
4	0.011557	0.011544	0.011544	0.011544	0.011544
8	0.011553	0.011544	0.011544	0.011544	0.011544
16	0.011551	0.011544	0.011544	0.011544	0.011544
32	0.011548	0.011544	0.011544	0.011544	0.011544



	32KB	64KB	128KB	256KB	512KB
1	0.022244	0.019446	0.019114	0.019108	0.019108
2	0.019159	0.019108	0.019108	0.019108	0.019108
4	0.019159	0.019108	0.019108	0.019108	0.019108
8	0.019153	0.019108	0.019108	0.019108	0.019108
16	0.019108	0.019108	0.019108	0.019108	0.019108
32	0.019108	0.019108	0.019108	0.019108	0.019108

根据以上四种不同的情况下的结果，可以得出结论：Miss Rate随着 Associativity的增大会减小，并且趋向与一个定值（这一定值与Block Size以及运行的trace有关）。同时，Cache Size越大，Miss Rate的初始值会越小。

原因，在Cache Size和Block Size固定的情况下，Associativity的增加会导致 Set数量减少，但是Line数量不变。因为trace中的地址是比较连续的，并且是在一个较小范围内的变化，因此它对于Associativity比较敏感，对于Set的数量不敏感。Associativity的增加减少了因为set_index相同数据被覆盖的机会。因为64位地址是tag---set_index---block_offset这样的排布，trace中的地址变化主要在于set_index和block_offset，并且set_index的变化较小（所以对Set数量不敏感）。因此Miss Rate随着 Associativity的增大会减小。但是这种减小又收到Block Size的限制，所以会趋向于一个值。

3. 比较 Write Through 和 Write Back、 Write Allocate 和 No-write allocate 的总访问延时的差异。（20分）

	32KB	64KB	128KB	256KB	512KB	2MB	8MB	32MB
2B	1028895	1026381	1026381	1026381	1026381	1026381	1026381	1026381
	20525005	20524702	20524702	20524702	20524702	20524702	20524702	20524702
8B	971281	968466	968366	968366	968366	968366	968366	968366
	20502482	20502179	20502179	20502179	20502179	20502179	20502179	20502179
32B	617336	612015	611915	611915	611915	611915	611915	611915
	20258365	20258062	20258062	20258062	20258062	20258062	20258062	20258062
128B	517498	512877	512477	512477	512477	512477	512477	512477
	20184736	20183726	20183726	20183726	20183726	20183726	20183726	20183726
512B	491253	485723	485322	485322	485322	485322	485322	485322
	20162920	20160597	20160496	20160496	20160496	20160496	20160496	20160496
Associativity = 4, trace1.txt								
	32KB	64KB	128KB	256KB	512KB	2MB	8MB	32MB
2B	627114	625313	625313	625313	625313	625313	625313	625313
	6943711	6943610	6943610	6943610	6943610	6943610	6943610	6943610
8B	542619	540718	540718	540718	540718	540718	540718	540718
	6896241	6896140	6896140	6896140	6896140	6896140	6896140	6896140
32B	290973	288271	288171	288171	288171	288171	288171	288171
	6760598	6760396	6760396	6760396	6760396	6760396	6760396	6760396
128B	213236	211531	211130	211130	211130	211130	211130	211130
	6710300	6710199	6710098	6710098	6710098	6710098	6710098	6710098
512B	193492	190480	189877	189877	189877	189877	189877	189877
	6694443	6693635	6693332	6693332	6693332	6693332	6693332	6693332
Associativity = 4, trace2.txt								

一对结果中，上方的是写回+写分配，下方的是写穿透+写不分配。可以明显看出，写回+写分配的周期远少于写穿透+写不分配。

原因在于以下几点：

1、写穿透在写命中时需要访问内存，而由之前的结果可以看出，命中的概率并不低，也就说，需要由大量的访问内存。而写回在写命中时只需要改dirty位，并且只有在由dirty标记的Line被替换时才需要将里面的内容写回内存，发生的次数相比而言不多。

2、写分配的方式虽然实现比较复杂，在写不命中的时候需要访问内存后再访问Cache并进行替换等步骤，但是它充分利用了局部性原理，可以在后续的过程中节省时间。写不分配虽然在写不命中时，虽然只需要访问内存，但是没有利用局部性原理。在显示情况下，对一片内存的连续写是会普遍发生的，写不分配明显不占优势。

3、写穿透+写不分配的方法存在使用Buffer进行优化的操作。即在cache和memory中间加入Buffer，在需要写内存时，写入Buffer，然后CPU就可以不用继续等待了，由Buffer进行写入。这样可以节省时间。但是这里的实现中难以加入Buffer，并且Buffer难以真实地模拟。因此写穿透+写不分配的时间有一定的偏大。这是导致差距如此悬殊的原因。

PART 2. 与lab2中的处理器性能模拟器联调。

与Lab 2中的流水线模拟器联调，运行测试程序。
该测试中cache的配置如下：

Level	Capacity	Associativity	Line size(Bytes)	WriteUp Polity	Hit Latency	Bus Latency
L1	32 KB	8 ways	64	write Back	1 cpu cycle	0 cpu cycle
L2	256 KB	8 ways	64	write Back	8 cpu cycle	6 cpu cycle
LLC	8 MB	8 ways	64	write Back	20 cpu cycle	20 cpu cycle

在cache的实现过程中，没有进行Bypass和PreFetch，同时没有进行指令Cache和数据Cache分开，两者通过同一接口进行访问L1Cache。

1、测试程序自选，测试结果正确，并打印动态执行的指令数和CPI。（40分）

- 能够与流水线模拟器协同工作。（10分）
- 能够模拟多层次cache。（10分）
- 测试程序运行正确，并打印出动态执行指令数，以及相应的CPI。（4分*5）

运行lab2-2中的测试程序。

```
|        | test1 | test2 | test3 | test4 | test5 | test6 | test7 | test8 | test9
| test10 |
| ------ | ----- | ----- | ----- | ----- | ----- | ----- | ----- | ----- | -----
| ------ |
| insts  | 28    | 28    | 117   | 162   | 118   | 68    | 132   | 60    | 33
| 23     |
| cycles | 1602  | 1602  | 1167  | 2186  | 1483  | 1075  | 2126  | 2302  | 1606
| 1282   |
| CPI    | 57.21 | 57.21 | 9.974 | 13.49 | 12.56 | 15.80 | 16.10 | 38.36 | 48.66
| 55.73  |

运行ack，快速排序，矩阵乘法。

（其中实现了系统调用，可以输入整数，输出整数、字符、字符串，计时（time函数），但是由于Cache的加入，输出字符串部分存在bug，输出字符串只能从memory中直接读取，尚未得出简便的解决办法。）

示例图：
```

```
szh@ubuntu:~/Documents/lab3-1/pipeline_cache$ ./pipeline_cache_rv64isim
filename:
../ack
Input file:../ack
print_inst?(1/0)
0
onestep?(1/0)
0
loops:100
result:
61
time:2seconds
Over! enter e to exit. enter c to check regs and memory
e
Instruct Num: 7296742
Cycles:13758252
CPI:1.88553
data_hazard:4734477
control_hazard:854121
```

结果统计：

	加入Cach后CPI	原本的CPI
ack(loop=100)	1.88553	1.88296
qsort(loop=1000)	2.00158	1.99637
matrix(loop=1000)	2.30579	2.30428

2、对比lab2中的流水线模拟器，分析CPI的变化原因。（10分）

lab2-2中提供的测试程序的CPI出现了明显的增加，而ack,qsort,matrix这三个程序的CPI相比之下增加非常微小。

主要原因在于，之前对取指和读写内存都是默认命中L1Cache的，忽视了访问内存会带来巨大的延迟。所以现在的CPI大于之前所得到的CPI。

lab2-2中的测试程序所包含的指令数都比较少，整个的执行流程比较短，无法充分利用到局部性，平均下来的CPI会很大。而且从这十个测试程序的结果中也可以看出，随着执行的指令数的增加，CPI会明显下降，这是利用到局部性的结果。

ack,qsort,matrix这三个程序的结果更为明显，由于他们内部有大量的循环，因此局部性非常好，所以他们的CPI只有极少量的增加，因为大部分的访问都在L1Cache命中了。