Lab07 report　PB22051078 王珏

实验目的与内容

在此前的实验中，同学们已经实现了一个简单的流水线 CPU。在流水线 CPU 中，我们读取指令和访问内存都是通过直接访问内存来实现的。然而，直接访问内存的效率并不高，因为内存的访问速度远远低于 CPU 的执行速度。为了提高 CPU 的性能，我们通常会在 CPU 和内存之间加入一个高速缓存 Cache。在本次实验中，我们将学习高速缓存 Cache 的组织结构和工作机理。

逻辑设计

部分核心代码：

二路组相连 Cache（LRU 策略）：

```
always @(*) begin
    if(!data_from_mem) begin
        if(hit == 1) begin
            age[age_addr] = 1;
            age[r_index*2 + hit_way[0]] = 0;
        end
    end
end
```

给每个 Cache Line 设置一个年龄（用 age 数组来记录），每当一个 Cache Line 被访问时，其他 Cache Line 的年龄都会加一，而被访问的 Cache Line 的年龄会被清零，每次替换年龄最大的那一行 Cache Line。

二路组相连 Cache（FIFO 策略）：

```
always @(*) begin
    if(!data_from_mem) begin
        if(hit == 1) begin
            age[r_index*2 + hit_way[1]] = 1;
            age[r_index*2 + hit_way[0]] = 0;
        end
    end
end
```

基于先进先出的原则，即最先进入 Cache 的数据最先被替换出去。每当一个 Cache Line 被访问时。将其年龄设为 1，而最先进入的值为 0，替换时将年龄最小的 Cache Line 替换掉。

二路组相连 Cache（RAND 策略）：

```
always @(*) begin
    if(!data_from_mem) begin
        if(hit == 1) begin
            age[r_index*2 + 1] = {$random(1)}%2;
            age[r_index*2 + 0] = (age[r_index*2 + 1]>0)?0:1;
        end
    end
end
```

随机选择一个 way 进行替换，替换时将年龄最小的 Cache Line 进行更换。

N 路组相连：

```
integer t;
generate
        always @(*) begin
            small_num = age[r_index*2];
            for(t = 0; t < WAY_NUM ;t = t +1) begin
                if(age[r_index*2+t] == 0) begin
                    small_addr = t;
                    small_num = age[r_index*2+t];
                end
                else if(small_num>age[r_index*2+t]) begin
                    small_num = age[r_index*2+t];
                    small_addr = t;
                end
            end
        end
endgenerate
```

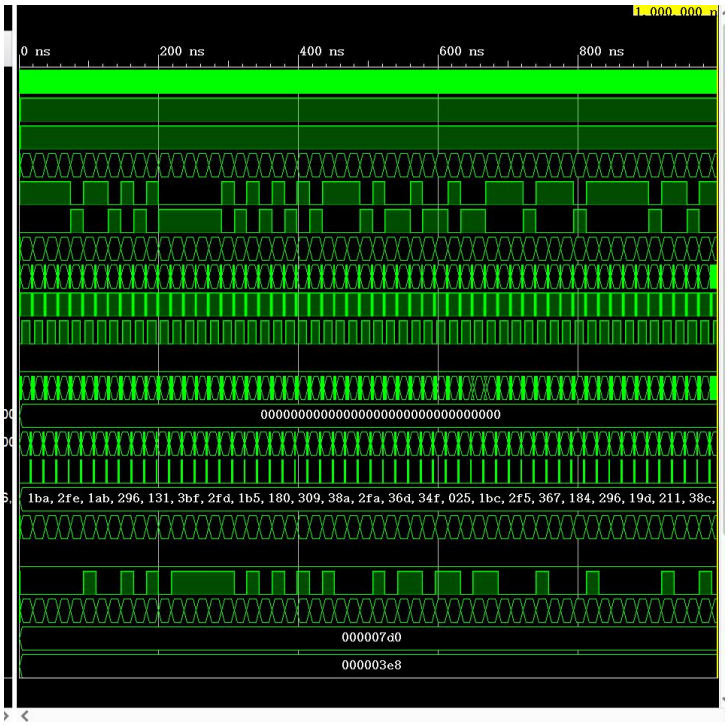
N 路组相连的 way_num 由 N 来控制，使用 small_num 和 small_addr 来记录年龄最小的 Cache Line 的年龄和地址，替换时将年龄最小的 Cache Line 进行更换。

测试结果与分析：

首先测试正确性，此时 MODE = 0 ，会生成随机读写序列

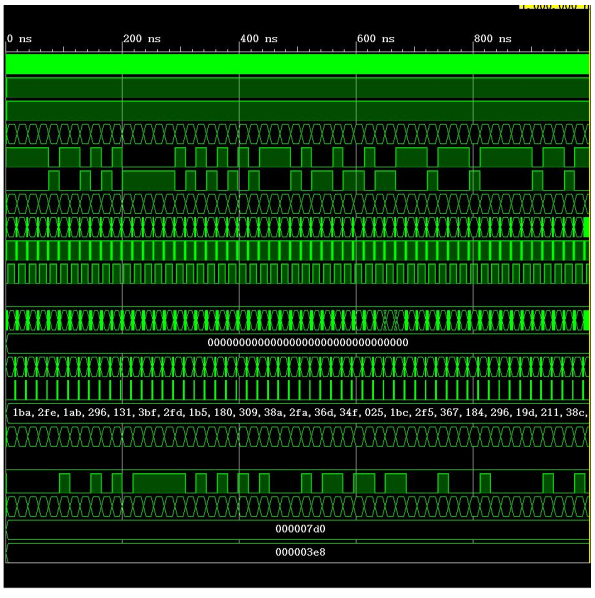
二路组相连 Cache（LRU 策略）：

```
# run 1000ns
INFO: [USF-XSim-96] XSim completed. Design snapshot 'cache_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:00 ; elapsed = 00:00:13 . Memory (MB): peak = 812.883 ; gain = 11.926
```



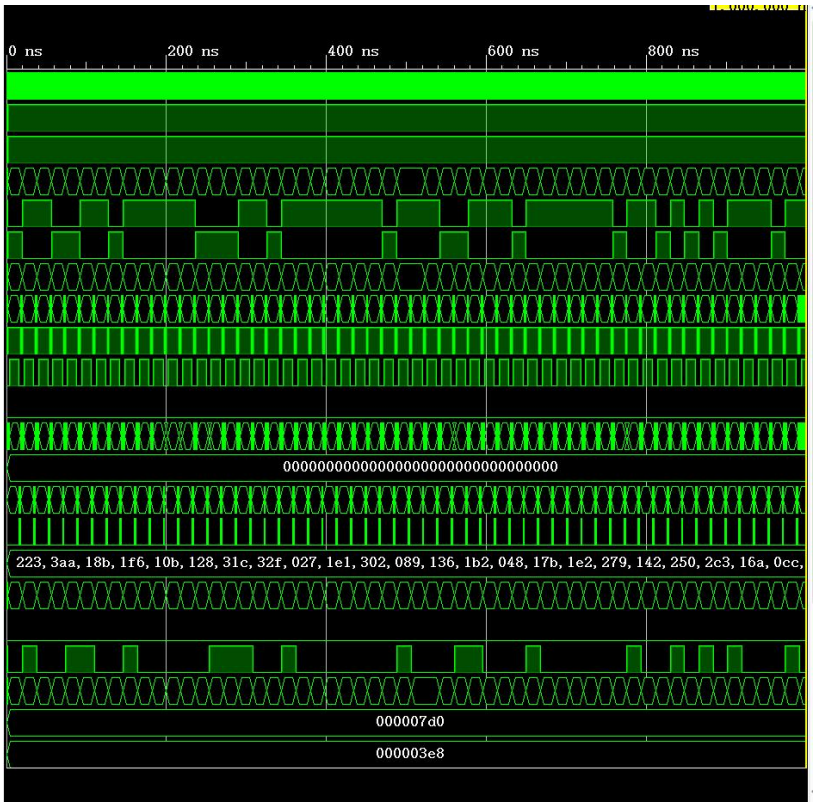
二路组相连 Cache（FIFO 策略）：

```
WARNING: [Wavedata 42-489] Can't add object "/cache_tb/test_data" to the wave window because
# run 1000ns
INFO: [USF-XSim-96] XSim completed. Design snapshot 'cache_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```



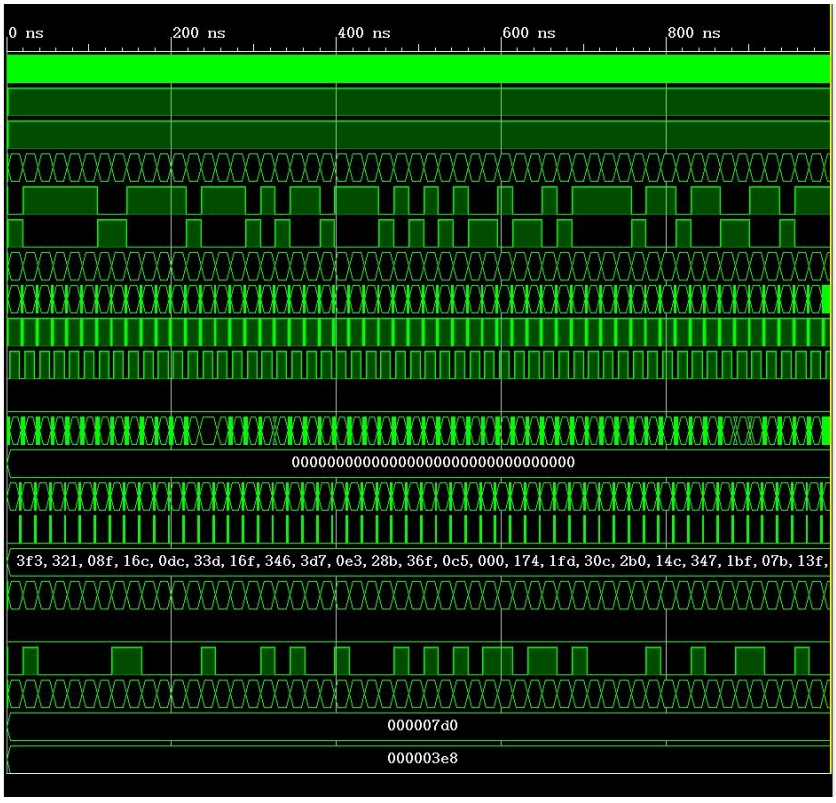
二路组相连 Cache（RAND 策略）：

```
# run 1000ns
INFO: [USF-XSim-96] XSim completed. Design snapshot 'cache_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```



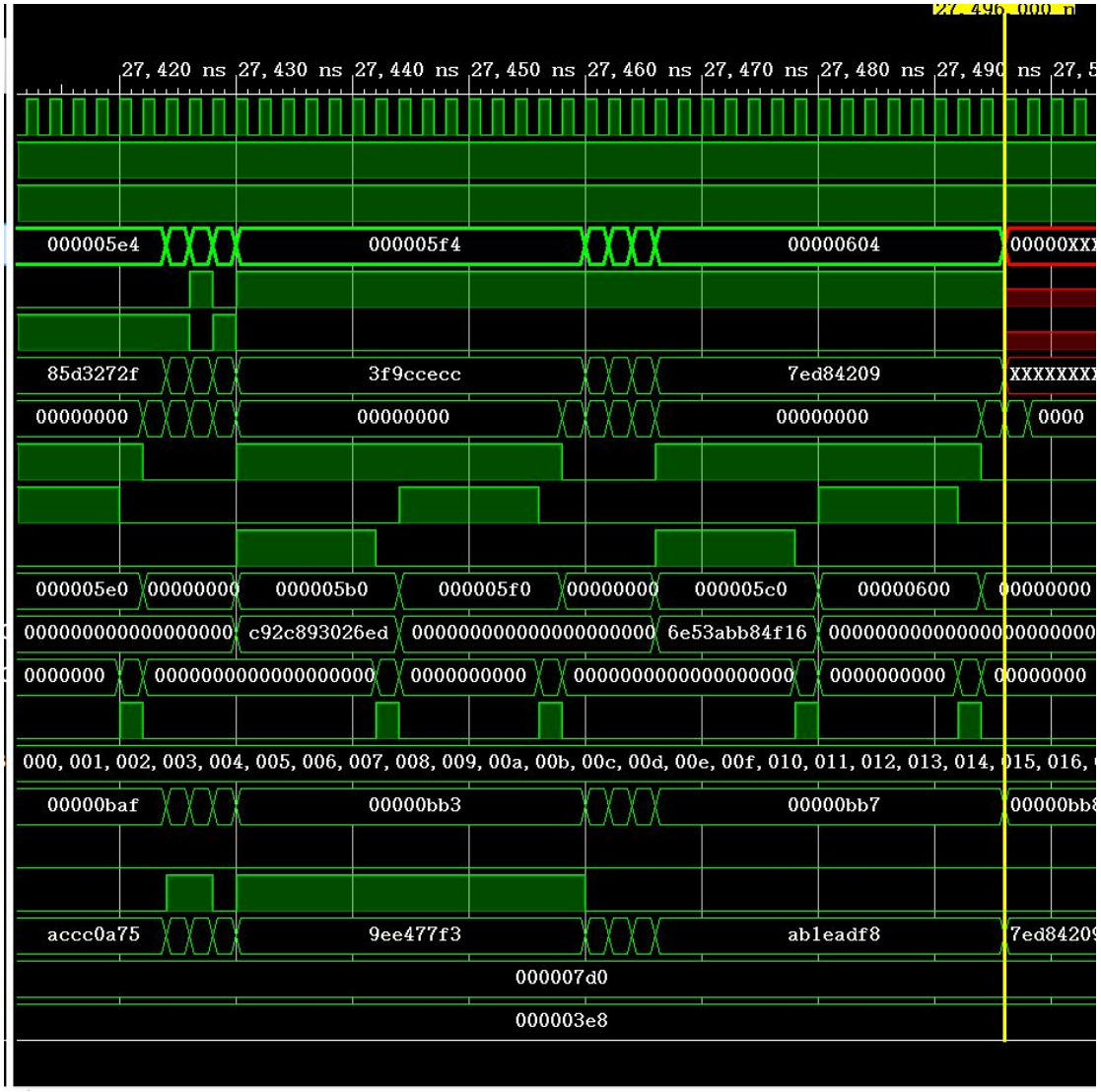
N 路组相连：（此处 N 为 8）

```
# run 1000ns
INFO: [USF-XSim-96] XSim completed. Design snapshot 'cache_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

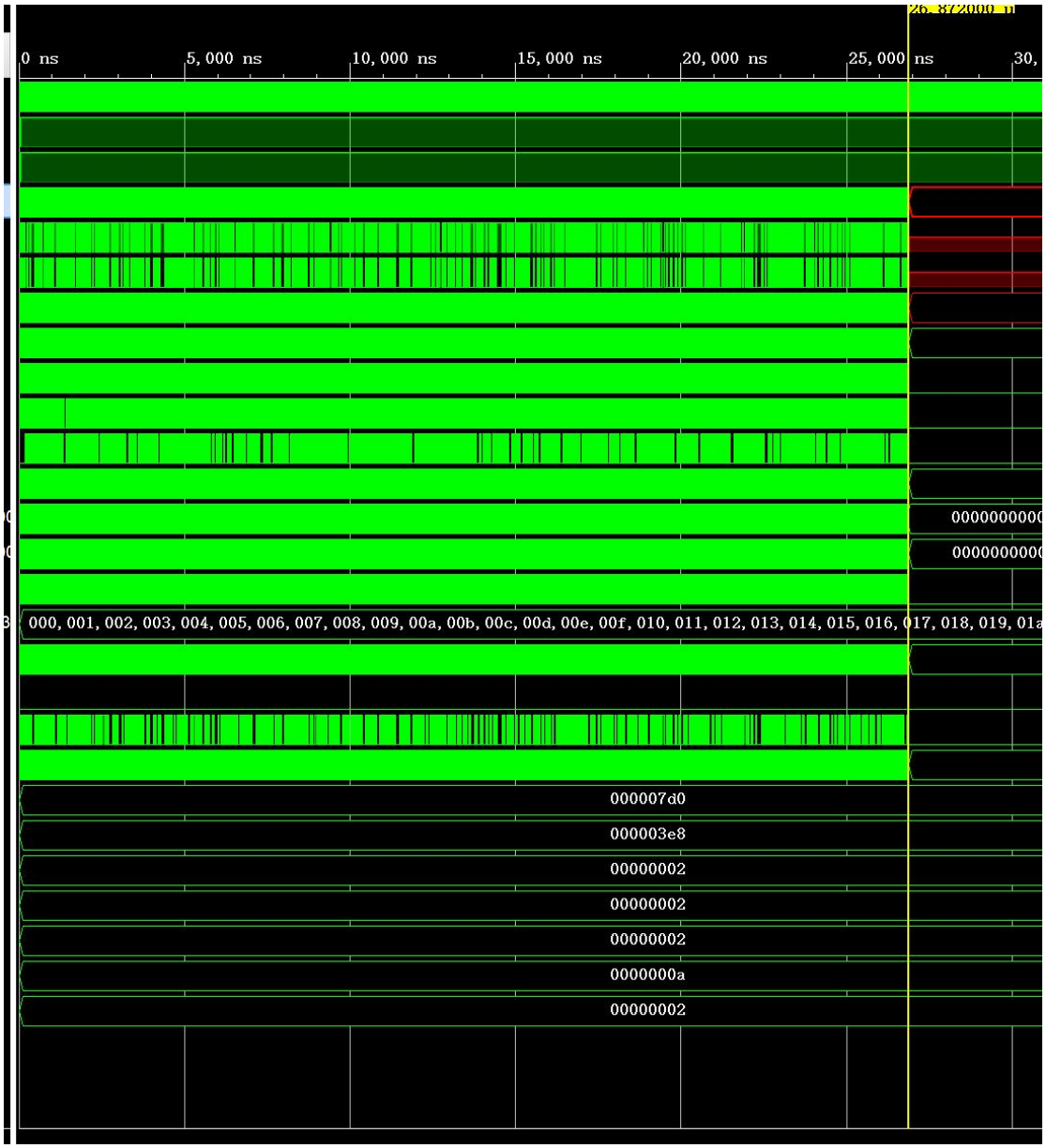


下面以二路组相连为例来测试不同策略的优劣，此时 MODE = 1 ，会生成伪顺序读写序列，适用于测试 Cache 的性能。

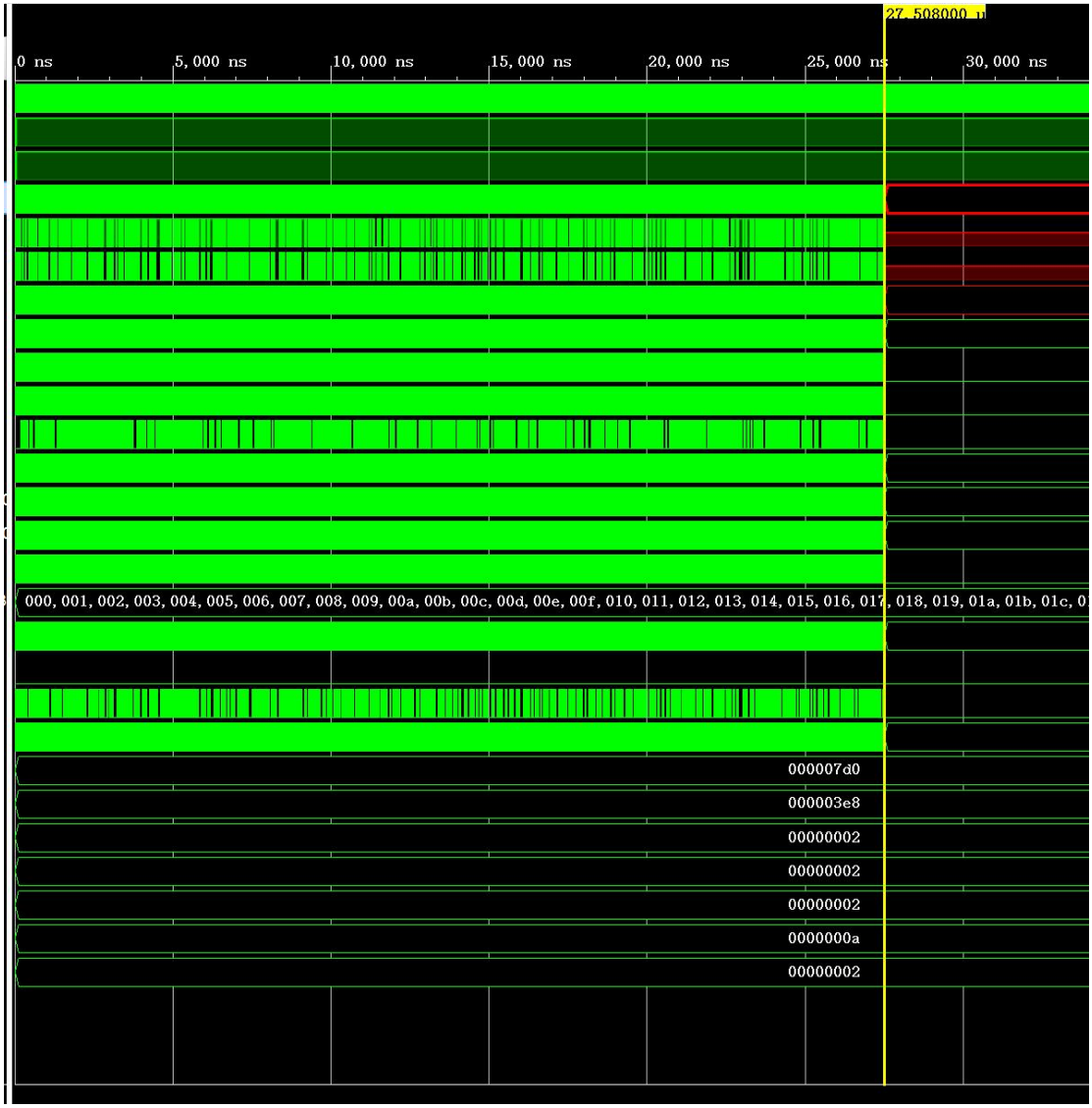
LRU:27496ns 完成



FIFO:26872ns 完成



RAND：27508ns 完成



由上面的数据可知，从优到劣：LRU>FIFO>RANDOM。LRU 较好的反应了程序的局部性，FIFO 利用了历史信息但未必能对应真正的局部性，RANDOM 完全没有考虑程序的随机性。

思考与总结

通过这次实验，我实现了一个高速缓存 cache，对 cache 的各种替换策略有了更深的理解。通过这一系列的实验，我对一个完整的 CPU 的内部实现有了深刻的理解，为我这门课的学习提供了很大的帮助。