RISC-V 流水线 CPU La3 实验报告

PB16060130　　顾健鑫

一、实验目的

本次实验需实现一个组相联的基于写回法的 cache。在实现 cache 模块的过程中，需要权衡选择合适的组相联度来平衡命中率收益与存储资源、电路面积的开销。

二、实验环境和工具

本次实验使用的软件环境为 Vivado 2017.4 的版本。在创建工程项目时，直接导入提供好的模块框架文件并在完成代码的编写后，导入测试文件用于测试，并用 Vivado 的波形仿真结果作为结果检验依据。

三、Cache 的具体实现

本次实验中的 cache 实现均额外使用了一个 age 数组，age 数组记录所有组中每一个 Line 的新旧情况，FIFO 和 LRU 替换策略将分别采用不同的方式对 age 数组进行更新与赋值，已达到替换效果。

a. FIFO

每一组中的换出块的选择策略：选择当前所有块中 age 数值最大的块作为换出块。

age 数组的更新策略：在初始化时，全部初始化为最大值，即全部位均为 1。在新块换入成功后，即在 SWAP_IN_OK 状态，把换入块的 age 置 0，把所有其他有效块（VALID 值为 1）的 age 加一。

通过上述初始化与更新策略即可实现 FIFO。

b. LRU

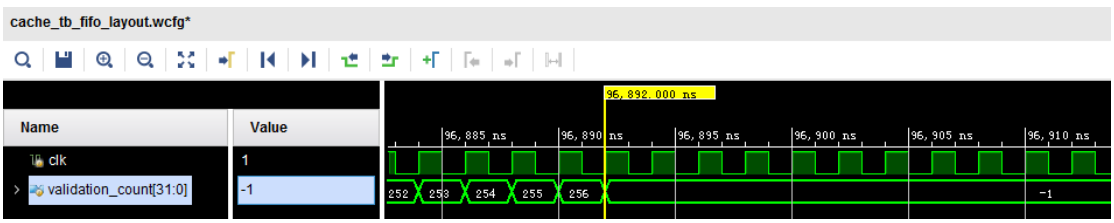
每一组中换出块的选择策略：选择当前所有块中 age 数值最小的块作为换出块。

age 数组更新策略：在初始化时，将 age 全部初始化为 0。在 IDLE 状态会做 cache 命中与缺失的判断，若命中，则将此块的 age 加一；若缺失，则将此块的 age 置 0。依次策略，age 数组中值最小的块即为当前最少使用的块，即为换出块。

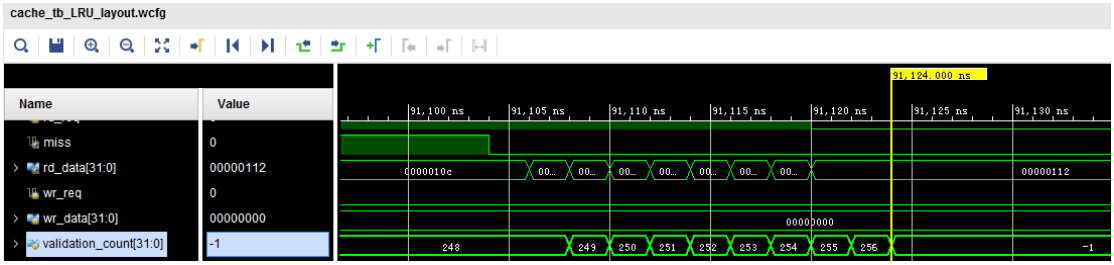
四、Cache 独立测试

在独立测试中，使用提供的 cache_tb 文件，并用 python 程序生成 256 的顺序读写测试程序，FIFO 和 LRU 的运行结果如下图所示，经过 256 次顺序读测试后，计数变量变为-1，表示测试通过

FIFO：



LRU：



五、CPU+Cache 联合测试

性能分析

在联合测试中，快排的数组长度为 512，矩阵大小为 16*16，得到如下结果

组数	组相连度	LRU 快排缺失率	FIFO 快排缺失率	LRU 矩阵乘法缺失率	FIFO 矩阵乘法缺失率
3	1	6.51%	6.51%	60.23%	60.23%
	2	5.04%	2.94%	57.36%	57.20%
	4	3.68%	1.71%	39.51%	20.22%
	6	2.61%	1.24%	20.29%	1.89%
	8	1.08%	0.92%	4.94%	1.52%
	16	0.63%	0.63%	1.14%	1.14%

在规模较小的测试中，横向对比的结果出乎意料，相对于较复杂的 LRU 替换策略，较简单的 FIFO 策略甚至有更低的缺失率。在纵向对比上，随着组相连度的增加，cache 的缺失率有着明显的降低，从直接相连的 60.23%至 1.14%，对于访存时间的提升相当明显。

为了做进一步验证，我增大了快排数组的规模，再次验证在组数为 8 的情况下，两替换策略在组相连度为 4,8,16 时的缺失率，统计结果如下

组数	组相连度	LRU 快排缺失率	FIFO 快排缺失率
3	4	7.22%	6.48%
	8	5.15%	5.77%
	16	0.17%	0.17%

在此可以看到，随着数组规模的增大以及组相连度的提升，LRU 表现出了优于 FIFO 的缺失率，因此两者均有很高的实用价值但实际应用中的表现还会与具体任务的数据局部相关性以及多种条件有关。

除了单纯的改变组相连度，本次实验还对组数与组相连度的关系做了考察，使用的矩阵大小设置为 16*16，在维持组数*组相连度不变的情况下，对矩阵乘法做了三组对比测试，结果如下

组数	组相连度	LRU 矩阵乘法缺失率	FIFO 矩阵乘法缺失率
2	12	18.73%	2.27%

3	8	4.94%	1.52%
4	6	1.14%	1.14%

从结果可以看出，尽管总数不变，但随着组相连度的增加，缺失率会增大。即在保持 cache 总量不变的情况下，为了降低程序运行的缺失率，适当增加组数会有更好的效果。

运行时间测试

针对上述出现的测试，记录了相应的程序运行的时间，结果列表如下，时间单位为 us（微秒）

组数	组相连度	LRU 快排运行时间	LRU 矩阵乘法运行时间	FIFO 快排运行时间	FIFO 矩阵乘法运行时间
3	1	623.283	1407.234	623.284	1407.234
	2	560.552	1354.756	458.372	1351.724
	4	466.376	1025.96	386.28	676.948
	8	379.689	372.148	352.524	286.232
	16	347.471	275.98	347.472	275.992

在保持 cache 的大小不变的情况下，也对运行时间进行记录，列表如下

组数	组相连度	LRU 矩阵乘法运行时间	FIFO 矩阵乘法运行时间
2	12	636.695	301.704
3	8	372.148	286.232
4	6	275.992	275.992

根据测试结果分析如下：

程序的运行时间与 cahce 的缺失率有直接的关系，缺失率越低运行时间越短；在本次 cache 实现中，程序运行时间的变化未与替换策略有明显的相关性，其变化幅度基本依赖于该参数设置下 cache 的缺失率。

资源分析

将 WB 段寄存器设置为 top 模块，对其进行综合，统计使用的 LUT 和 FF 的数量，列表如下，

组数	组相连度	LUT(LRU)	FF(LRU)	LUT(FIFO)	FF(FIFO)
3	2	2273	5543	2888	5543
	4	4764	10055	5744	10055
	8	8115	19080	9471	19079
	16	16305	37136	19654	37127

从表格中的数据可得出结论，随着组相连度的增加，LUT 和 FF 的使用数量会有明显的增加，这也很符合常理。横向对比下，FIFO 与 LRU 并未有明显的差别，这可能是由于在具体实现中，FIFO 和 LRU 都使用了 age 数组来进行信息记录，但 FIFO 的实现可以做更多简化，用单纯的计数变量来实现，因此理论上 FIFO 会比 LRU 使用更少的硬件资源。

综合对比而言，FIFO 在硬件资源使用上会比 LRU 占用的更少，但具体的缺失率会随着程序的具体不同而有所改变，因此不能单纯 LRU 是比 FIFO 更好的替换策略，两者各有优劣，不能一概而论。

六、写回法的优劣分析

写回法的优点在于速度快并且可以有效减少访问内存的次数，内存中数据是否被修改由dirty标志位决定。但其明显的问题就在于数据一致性的维护。在本次实验中，虽然无需考虑这种一致性。然而，在实际的计算机结构中，存在不经过CPU直接与内存进行数据读写的模块，如DMA。因此在实际情况下，写回法的实现要更为复杂，需要添加额外的标志信息来帮助换出块的决策。

与写回法对应的是写直达法，其易于实现，且容易保持不同存储层次间的一致性。但相对的写直达法会频繁访问内存，导致速度相较于写回法很慢，不利于计算机性能的提升。

七、总结

通过本次实验，我实现了基于FIFO与LRU替换策略的组相连cache，对cache的结构有了更加深入的了解和深刻的认知。同时通过联合测试，也对cache的性能以及其各项参数之间的关系有了更加深刻的认知。