trifling-mips：NSCSCC2020 设计报告

武汉大学一队

更新：August 5, 2020

摘　　要

本文系 2020 年“龙芯杯”武汉大学一队的设计报告。其内容主要包括，trifling-mips 的 cache 部分设计与 cpu 部分设计。

关键词： trifling-mips, 龙芯杯

1　cache

1.1　icache

我们设计的 icache 如图1所示，为 3-pipeline icache。在 stage 1，由外围电路通过 ibus 提供所必须的读指令信号。在 pipeline 1 会在记录读指令请求的同时，读取出对应的 tag 和下一个 cacheline 的 tag。如果 tag 命中，则流水线不中断，同时依据下一个 cacheline 的 tag 是否 hit 决定是否发起预取，否则会检查 prefetch 模块是否 hit：如果 hit 则将预取的 cacheline 放到 icache 里面，这时只会 stall 一个周期；如果没有 hit，则进入 ICACHE_FETCH 状态，向 prefetch 模块发送请求，得到数据后，便放到 icache 里面，状态机恢复 ICACHE_IDLE，继续执行后续操作。在 pipeline 2 会记录命中信息，并且读取出对应的 data。在 stage 3 便使用这些信息多路选择出最后的 rddata，设置 rddata_vld。在 pipeline 3 进行 sync 处理，便交给 ibus 送出结果。icache 在测试的时候尚未对 inv 和 flush 进行完整的测试，除此外的部分皆已通过 ramdom 测试，测试结果如表1所示，针对 sequ_rand 的情形，maxsequ 在 25 以上添加 prefetch 有更好的效果。

```
interface cpu_ibus_if();
// control signals
logic ready;
// indicate that corresponding stage shall be directly terminated.
// flush_1 will be '1' whenever flush_2 is '1'.
// stall shall be '0' whenever flush_2 is '1'.
logic stall;    // stall signal from icache
logic flush_1, flush_2, flush_3;
```

```
// read signals
logic read;     // whether start read req
logic rddata_vld; // whether rddata is valid
phys_t addr;    // read addr
uint32_t rddata;  // read data

modport master (
input ready, stall, rddata_vld, rddata,
output flush_1, flush_2, flush_3, read, addr
);

modport slave (
output ready, stall, rddata_vld, rddata,
input flush_1, flush_2, flush_3, read, addr
);

endinterface
```

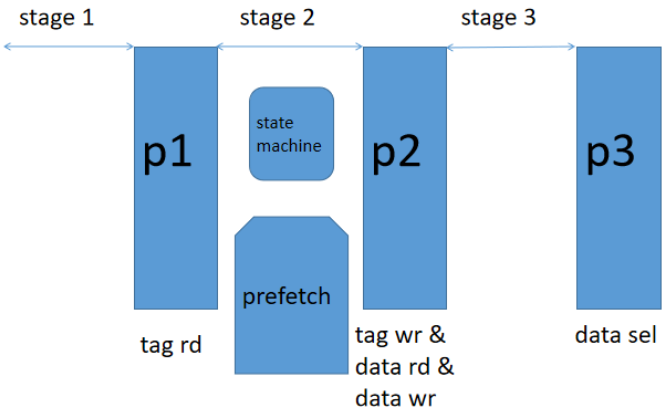

图 1: ICACHE 设计

1.2 dcache

我们设计的 dcache 如图2所示，为 3-pipeline dcache。在 stage 1，由外围电路通过 dbus 提供所必须的 lsu_req 信号。在 pipeline 1 会在记录 req 的同时，读取出对应的 tag 和下一个 cacheline 的 tag。如果 tag 命中，则流水线不中断，同时依据下一个 cacheline 的 tag 是否 hit 决定是否发起预取，否则会检查 write_buffer 和 prefetch 模块是否 hit：如果 write_buffer hit 且是正在写回或者正在 pop 则将预取

表 1: icache expr result

n_req	with prefetch	sequential(cycle)	random(cycle)	sequ_rand(cycle)	max_sequ
100	true	327	1320	656	-
100	false	387	1320	606	-
1000	true	1762	10562	4625	10
1000	false	2506	10393	3991	10
1000	true	1762	10497	2485	50
1000	false	2506	10294	2869	50
1000	true	1762	10500	3519	20
1000	false	2506	10305	3419	20
1000	true	1762	10568	2798	30
1000	false	2506	10360	3001	30
1000	true	1762	10380	2859	25
1000	false	2506	10195	3045	25

的 cacheline 放到 dcache 里面，这时只会 stall 一个周期，其他情况则依据 req 直接操作 write_buffer；如果 write_buffer 没有 hit，则检查 prefetch 是否 hit，如果 hit 则进入 DCACHE_PREFETCH_LOAD 状态，这时也只会 stall 一个周期；如果 prefetch 没有 hit，则进入 DCACHE_FETCH 状态，向 prefetch 模块发送请求，得到数据后，便放到 dcache 里面，状态机恢复 DCACHE_IDLE，继续执行后续操作。在 pipeline 2 会记录 req 和读取 dcache 中的 data，这些将在 stage 3 中使用。stage 3 会依据具体的 pipeline 2 中记录的 req 决定是写还是读，读操作和 icache 差不多，写操作会将最新的 data 与 wrdata 进行 mux，将得到的数据写入到 dcache。实验部分 (尚未与清华去年的 dcache 进行对比) 如表2，我们可以发现在官方测试数据集上执行的总周期数 cycl 和请求数 (n_req) 差别不大，说明 4-way lru 16KB 的 dcache 具有较好的性能。

```
typedef struct packed {
logic [$clog2(`N_RESV_LSU):0] lsu_idx;
phys_t addr;    // aligned in 4-bytes
// byteenable[i] corresponds to wrdata[(i + 1) * 8 - 1 : i * 8]
logic [$bits(uint32_t) / $bits(uint8_t) - 1:0] be;
uint32_t wrdata;
logic read, write, uncached;
} lsu_req;
typedef struct packed {
logic [$clog2(`N_RESV_LSU):0] lsu_idx;
uint32_t rddata;
```

```
logic rddata_vld;
} lsu_resp;
interface cpu_dbus_if();
// control signals
// for D$
logic stall, inv_dcache;
// for I$
logic inv_icache;
// lsu_req
lsu_req lsu_req;
lsu_resp lsu_resp, lsu_uncached_resp;

modport master (
output inv_dcache, inv_icache, lsu_req,
input stall, lsu_resp, lsu_uncached_resp
);

modport slave (
input inv_dcache, inv_icache, lsu_req,
output stall, lsu_resp, lsu_uncached_resp
);

endinterface
```

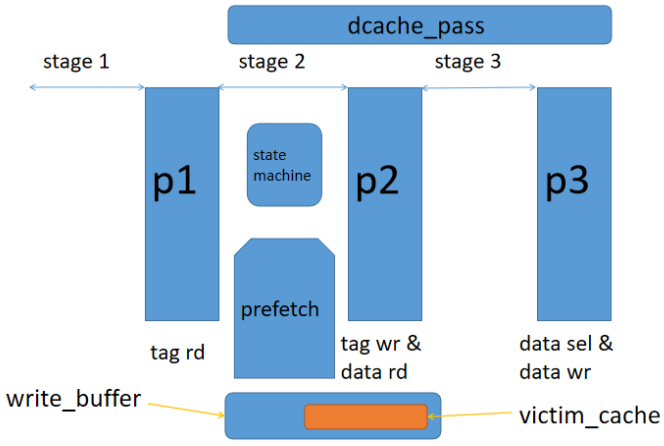

图 2: DCACHE 设计

表 2: dcache expr result

name	stall_cnt	rd_cnt	wr_cnt	n_req	cycle
test_inv	7	2	1	6	171
mem_bitcount	47	34	0	3800	4166
mem_bubble_sort	174	118	0	61613	62456
mem_dc_coremark	142	109	0	82967	83677
mem_quick_sort	778	525	0	38517	41704
mem_select_sort	174	118	0	21594	22437
mem_stream_copy	165	91	0	39924	40315
mem_string_search	303	579	0	33101	34525
random.2	20297	6150	3932	50000	161979
random.be	20548	6198	3951	50000	162786
random	20805	6257	4005	50000	164583
sequential	1024	513	0	32768	38531
simple	2	2	0	10	152

2 cpu

2.1 实现的指令集

实现了初赛要求的 57 条指令。

2.2 架构介绍

单发射五级流水，配备 AXI 接口。