目录

图示总表

摘要

我们团队参加的是 la 赛道团队赛，提交作品：“**8 级顺序双发射流水 CPU**”。

流水级划分：是:PreIF,IF1,IF2,ID,Launch(issue),EXE,MEM1,MEM2,WB 一共 8 级流水(如果计算 preif 则为 9 级流水)采用前后端分离。

CPU 频率：支持 80mhz。

发射宽度：顺序双发射。

发射队列长度：设置了长度为 16 的发射队列。

预测器：使用 pht+btb 分支预测器，和 mmu 翻译的 uncache 属性的一位饱和预测器。

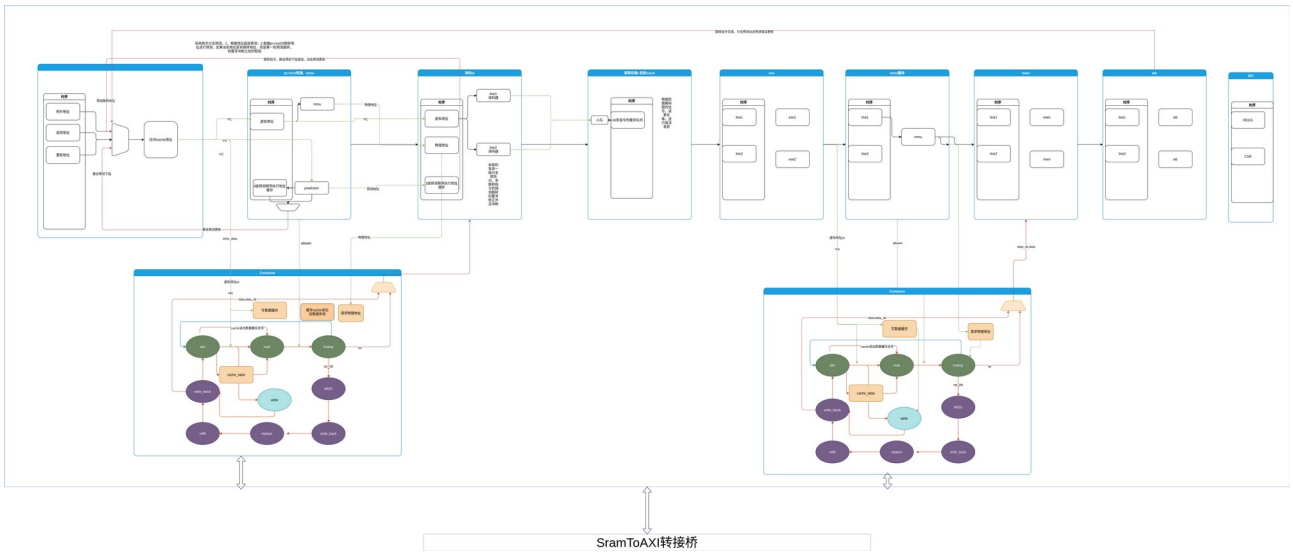
CACHE:装载了两路组相连的 vipt 的 icache 和 dcache，cache 是两级流水结构，cache 的大小为 4KB,其中 icache 支持最高同时取出 2 条指令,dcache 支持 storebuffer。

乘法器和除法器：采用 3 级流水线的华莱士树结构的乘法器，和 34 周期的一位 booth 除法器。

csr 和 regs:使用的是双端口的 csr 寄存器堆和 4 端口读和双端口写的 Regs 寄存器堆。

MMU：一级 mmu,tlb 项数为 32 个。

CPU 总体结构图如所示图示 1（由于图片过于庞大，所以在 xx 文件夹中有完整图片）。



图示 1: CPU 总体结构图

1 前端设计

1.1 取指令部分

前端一共分 3 级分别是 PreIF, IF1，IF2, ID。如图示 2

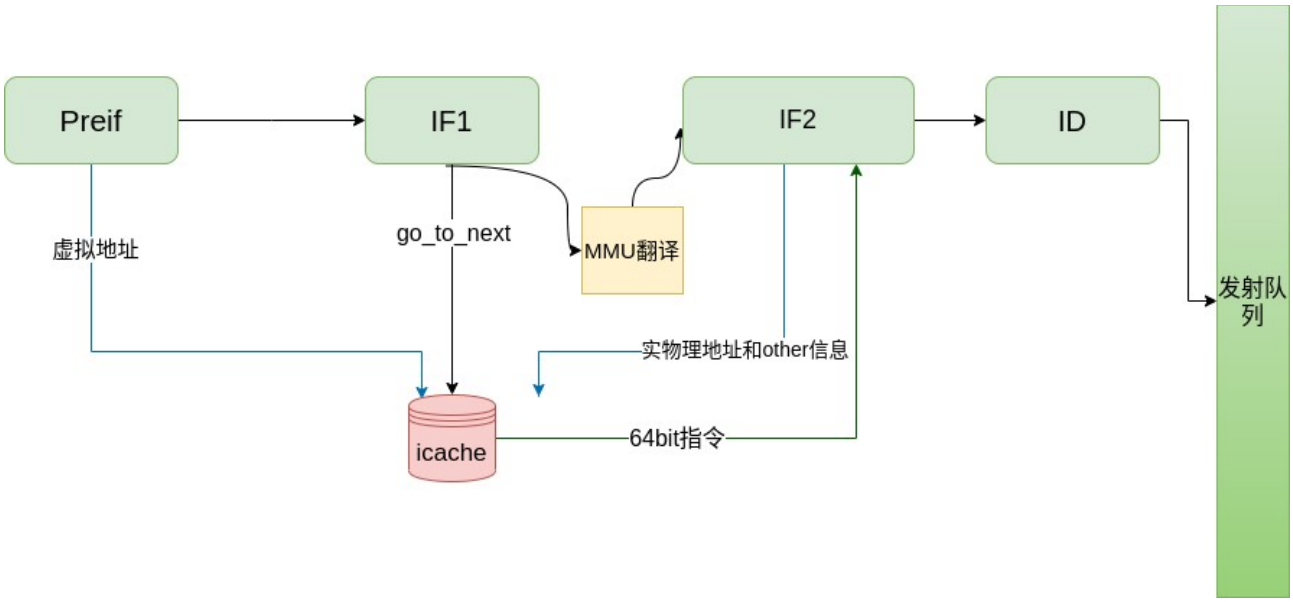
preif：计算当前访问 pc，并发请求到 cache

IF1: 进行 mmu 翻译，将其将要流往 if2 的标志信息发 go_to_next 给 *icache*,用于和 *icache* 中的请求同步。

IF2：将实地址和 uncache 属性信息发给 icache,并获得获取指令

ID：将取出的取出的指令进行译码，同时对气泡进行压缩，使气泡不会被存放到发射队列。

前端指令能对 64bit 对齐的取指令，所有可能存在非 64 对齐的起始 pc，就会导致无法取出 64 字节的指令，而是 32 字节的指令，所有会产生气泡，在译码级 id 到发射队期间会对这个气泡压缩挤出流水，保证发射队列中不会存在气泡，因为存在气泡将会及其浪费发射队列的空间。



图示 2: 取指令

1.2 分支预测：

本处采用 3 级预测更新，第一级 IF2 使用静态预测当前指令下指，第二级 IF2 更新对 uncache 预测错误的，第三级 ID：用动态预测更更新静态预测的跳转指令。最后在 wb 对动态预测错误进行更新。具体结构图见图示 3:

使用 3 个预测部件，pht,btb,和 pcu，分别预测是否跳转，跳转地址，以及指令对应的 uncache 属性,

1.pht:2 位饱和计数器，用于预测是否跳转，大小：2 的 10 次×4bit，即 1024 项，每项 4bit**,**访问地址为 10 位，由于本项目使用的是双发射，所以截取 PC[12:3]作为访问地址,存储介质 bram

2.btb：历史跳转地址，用于预测跳转指令的下指,大小：2 的 6 次方×55bit,即 128 项，每项 55bit,存储介质 bram

项={valid 位，tag 位，预测 pc}

valid:1bit,

tag:22bit

预测 pc：32bit

{tag,btb 读地址}=PC[31:3]

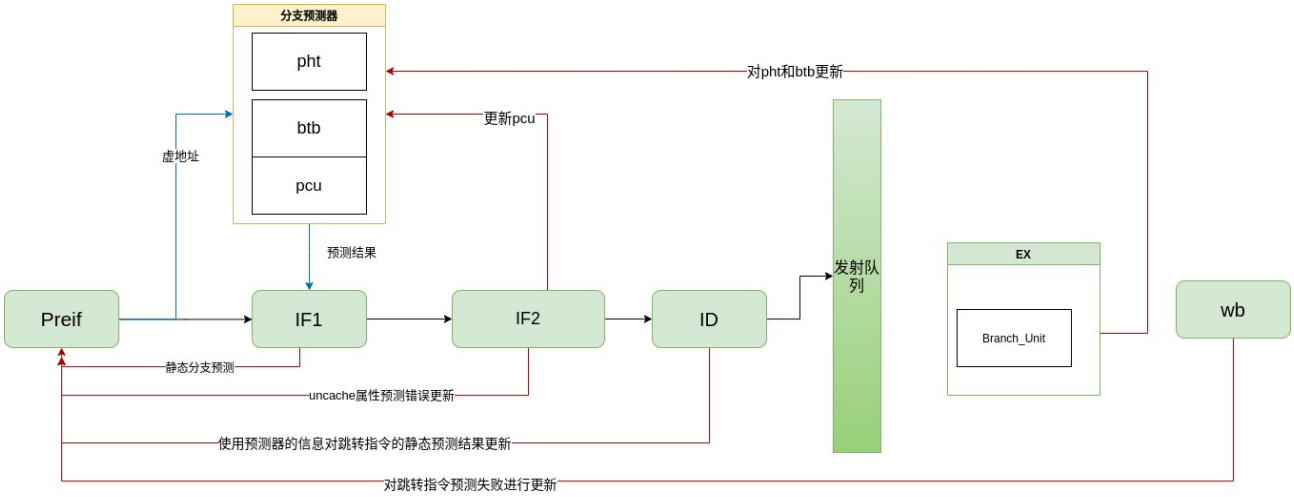
3.pcu：用于预测当前指令的 uncache 属性，一共 32 项，每项 1bit,存储介质 dram 。

pcu 的访问地址 raddr=pc[16:12],因为页大小为 4kb,所以起始地址从 pc 的 12 位开始。

为什么需要设置 uncache 预测器?

因为需要通过 uncache 属性判断当前 PC 是否能取出两条指令，这个将会影响下指，能则静态预测下指是：当前 pc+8,不能则是当前 当前 pc+4.

又由于直接使用 mmu 输出的 uncache 将会变成关键路径，所以选择预测的方式


分支预测器
pht
btb
pcu
对pht和btb更新
更新pcu
虚地址
预测结果
Preif
IF1
IF2
ID
发射队列
EX
Branch_Unit
wb
静态分支预测
uncache属性预测错误更新
使用预测器的信息对跳转指令的静态预测结果更新
对跳转指令预测失败进行更新

图示 3: 分支预测

2 后端设计：

一共包含 5 级：Launch,EXE,MEM1,MEM2,WB，结构如图示 4

Launch:负责从发射队列中顺序取出两条指令，发 4 个读地址到 4 端口读的寄存器堆中，同时对两条指令进行检测，判断是否能同发射。将寄存器读出数据和后面流水级的数据 forward 的数据传递到 DR 模块。**发射级是非对称的，**不支持两个条访存指令并行执行，两条除法指令并行指令，其他类型指令都可以并行指令，支持分支指令并行指令。

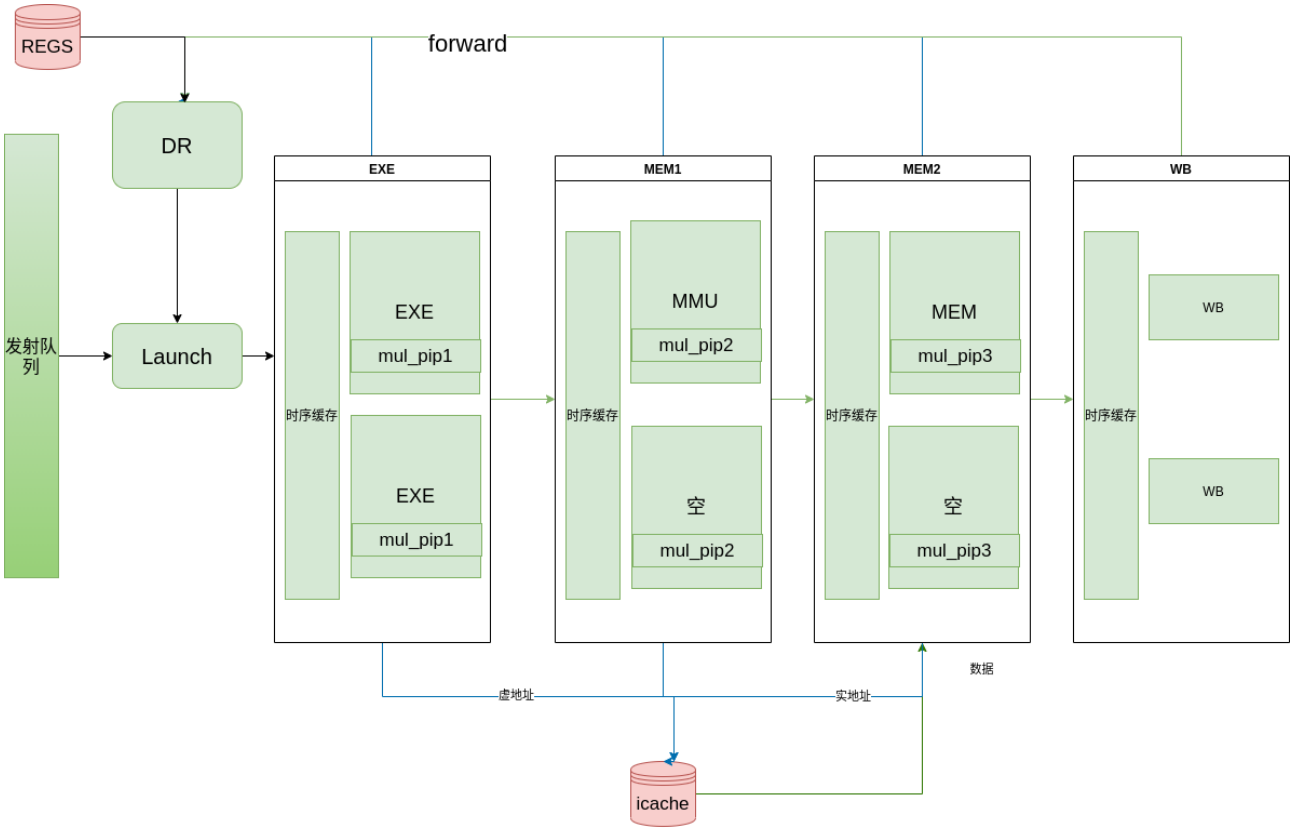
DR 负责检测这些 Regs 读出的数据是不是最新数据，不是就更新为 forward 的数据。将更新后的数据发给发射级。

EXE: 主要进行 ALU 和 DIV 运算,一级乘法运算，同时发出访存地址

MEM1:主要进行 load,store 的虚实地址翻译， 二级乘法运算

MEM2:取得 load 操作数据， 三级乘法运算（完成乘法）

WB：两条指令的提交写寄存器堆 Regs,


REGS
forward
DR
发射队列
Launch
EXE
MEM1
MEM2
WB
时序缓存
EXE
mul_pip1
EXE
mul_pip1
MMU
mul_pip2
空
mul_pip2
MEM
mul_pip3
空
mul_pip3
WB
WB
虚地址
实地址
数据
icache

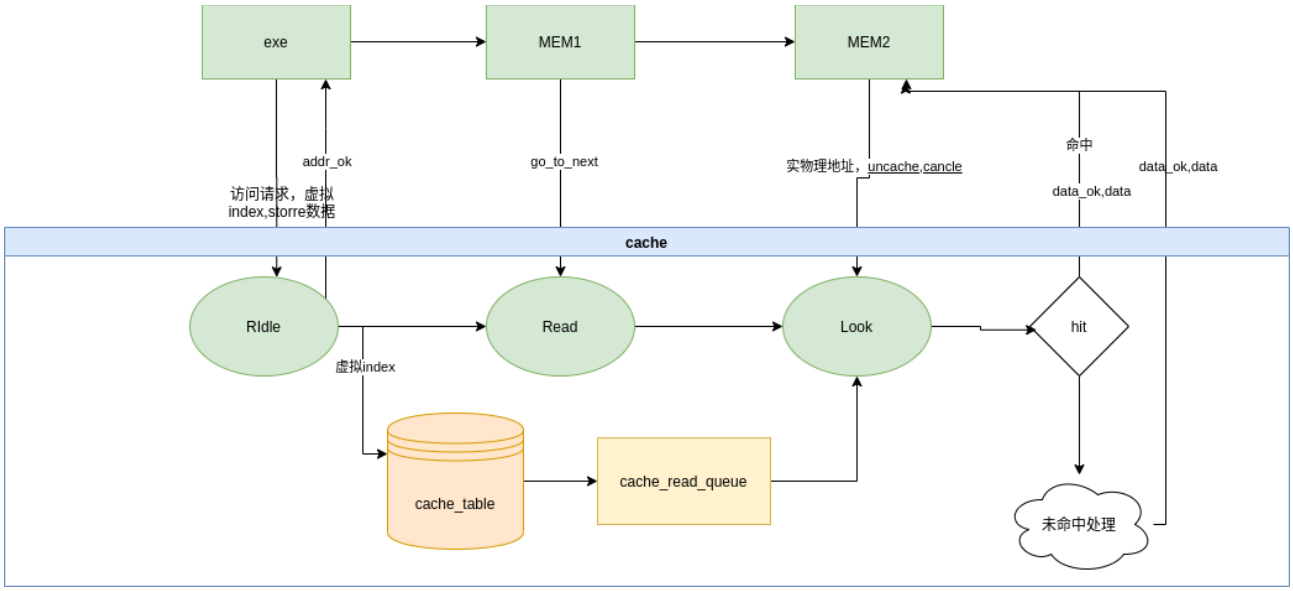
图示 4: 后端结构

3 cache 设计

3.1 cache 与 cpu 交互

cache 采用的 2 级流水的结构，第一级响应接收请求，并进行查找 cache_table,第二级处理：即在 lookup 状态，根据接收请求信息和 cache 查找出的信息判断是否命中，1、未命中则进行缺失、写回、重填处理，2、命中则返回数据。为了保证 cache 处理的请求和 cpu 处理的指令是同步的，这里引入一个 go_to_next 信号，由 cpu 发出，因为 cache 在此阶段不会阻塞，但是 CPU 可能会阻塞，所以需要 CPU 传入一个信号 go_to_next 表示它处理完了，用于同步 CPU 和 cache。如图示 5

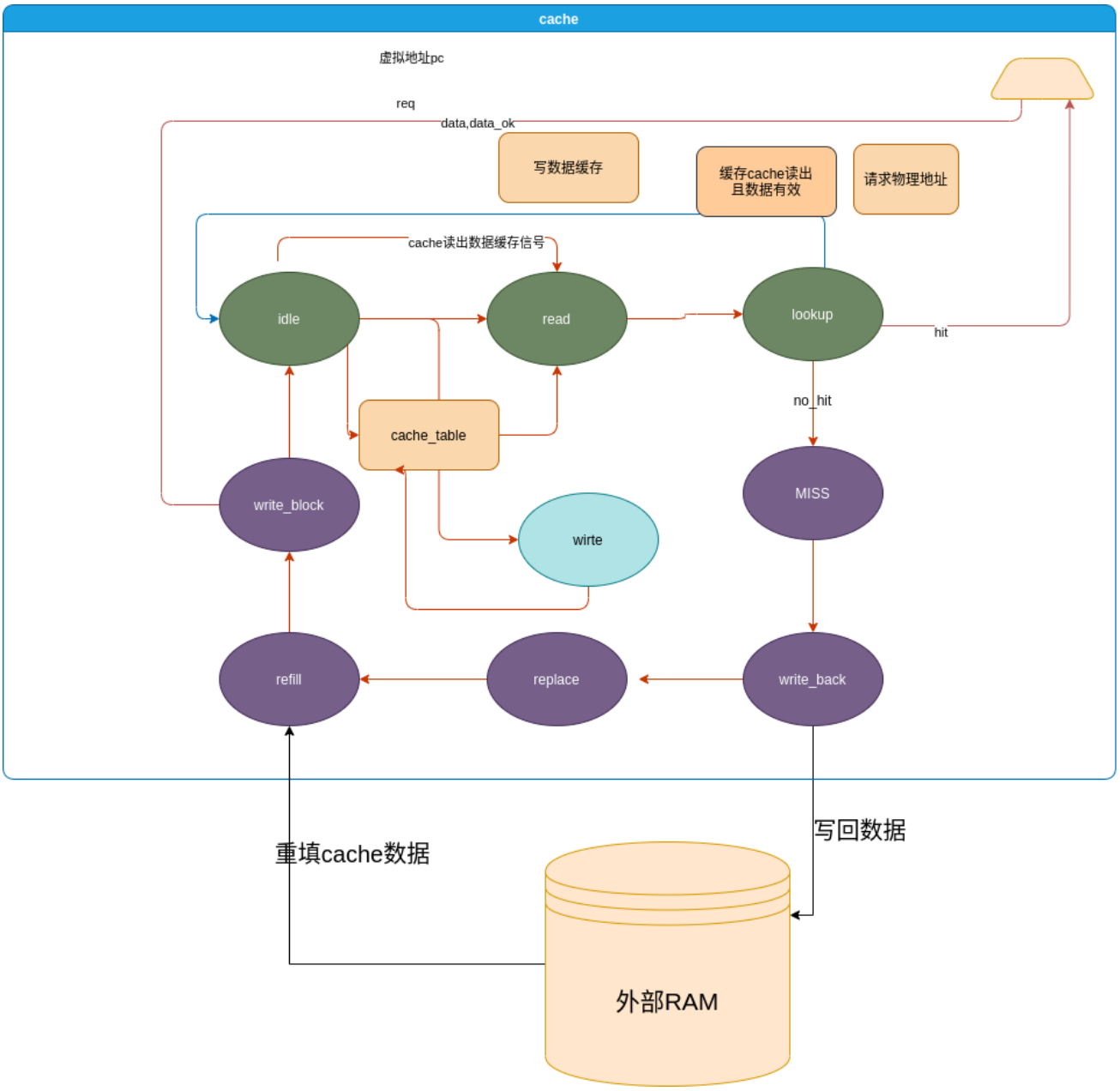
本 cache 支持 storebuffer,



图示 5: cpu 和 cache 交互

3.2 cache 状态机

本处使用一个状态机进行控制 cache,一共 9 个状态。参考了《CPU 设计实战》结构如：图示 6。



图示 6: cache 状态机

4 参考文献

《CPU 设计实战》