哈尔滨工业大学（深圳）2 队

UltraMIPS 设计报告

李程浩　刘定邦　宫浩辰　任翔宇

2020 年 8 月

目录

第一章　项目概览

1.1　总述

本项目是一个基于 MIPS 32 指令集设计并实现的 CPU，支持 MIPS 32 指令集的一个子集。CPU 采用顺序双发射的六级流水线结构，具有一级指令 Cache 和数据 Cache，通过 AXI 接口与外设通信。

1.2　指令集

我们实现了大赛要求的所有指令。除此之外，还加入了陷阱指令、非对齐访存指令、链接加载指令和条件加载指令。

1.3　结构

我们实现了六级流水线的顺序双发射 CPU。在经典五级流水的基础上，为了减少取指造成的流水线堵塞，将取指切分为两段，一段用于发送取指请求，一段用于预取指令。为了最大限度保证预取指令的正确性，我们进行了分支预测，减少因跳转导致 InstBuffer 的频繁清空。并且，由于引入了 InstBuffer，这样译码发射阶段若不能双发指令，就可以将无法发射的指令暂存，使得 PC 更新的逻辑变得简单。

1.4　CP0 和异常处理

我们实现了大赛要求的所有 CP0 寄存器，包括系统测试所需要的 EBase 寄存器，同时增加了一些别的 CP0 寄存器，能够检测并处理大赛所规定的异常。

1.5　分支预测

我们实现了一个动态分支预测器，可以预测给定 PC 值处的指令是否需要跳转，以及跳转的目的地址。其较高的命中率可以消除绝大多数分支指令带来的惩罚，为连续取指带来可能。

1.6　Cache

我们实现了 Cache，充分利用程序局部性原理，提高读写内存的效率。其中，指令 Cache 采用 8KB 的二路组相联流水化设计，数据 Cache 采用 8KB 的二路组相联非流水化状态机设计。

第二章　CPU 设计

2.1　指令集

我们实现了大赛要求的所有指令，在此基础上增加了一些额外的指令。下按功能分类列出。

逻辑运算指令 OR, AND, XOR, NOR, ORI, ANDI, XORI, LUI

移位指令 SLL, SRL, SRA, SLLV, SRLV, SRAV

数据移动指令 MFHI, MFLO, MTHI, MTLO, MOVN, MOVZ

算术运算指令 ADD, ADDU, SUB, SUBU, SLT, SLTU, ADDI, ADDIU, SLTI, SLTIU, MULT, MULTU, MUL, DIV, DIVU

跳转/分支指令 J, JAL, JR, JALR, BEQ, BNE, BGTZ, BLEZ, BGEZ, BGEZAL, BLTZ, BLTZAL

加载/存储指令 LB, LBU, LH, LHU, LW, SB, SH, SW, LWL, LWR, SWL, SWR, LL, SC

特权指令 MFC0, MTC0, ERET

自陷指令 BREAK, SYSCALL, TEQ, TNE, TGE, TGEU, TLT, TLTU, TEQI, TNEI, TGEI, TGEIU, TLTI, TLTIU

2.2　结构

2.2.1　总体架构

我们的 CPU 采用了顺序双发射的六级流水线结构。各部分功能简述如下：

取指请求阶段 计算当前 PC，并向指令 Cache 发出取指请求。同时，PC 值送入分支预测器，进行分支预测

取指缓存阶段 指令 Cache 将取出的指令连同它的 PC 值放入 InstBuffer 中暂存，以供后续阶段使用。

译码发射阶段 从 FIFO 中取出指令，进行译码，决定是否发射以及发射条数，并从寄存器中取出操作数。

执行阶段 执行指令，计算出结果，决定是否跳转，并生成数据 Cache 的驱动信号。

访存阶段 取出数据 Cache 送出的数据，并判断异常。

提交阶段 将指令做出的修改提交给通用寄存器以及其他寄存器。

2.2.2　结构图

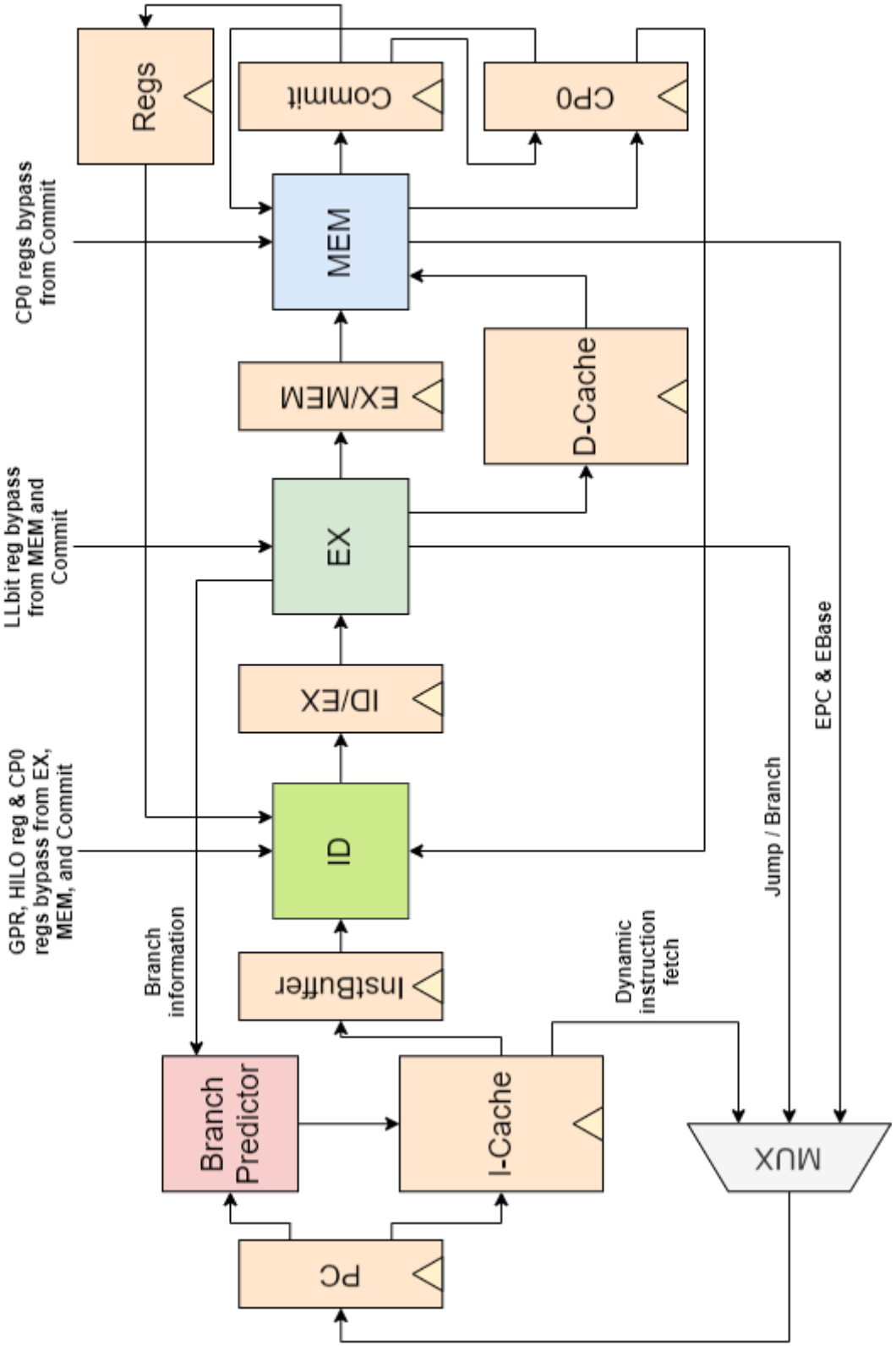

图 2.1: CPU 结构图

2.2.3 各阶段设计说明

取指请求阶段

此阶段驱动取指请求信号，并计算 PC 值。在这个阶段，我们结合了动态分支预测，采用了依据分支预测的动态取指方法，对于取指进行精准的操控，大大减少指令的浪费和流水线堵塞的时间，同时也加快了取指的速度。对于分支预测，我们采用动态的设计架构，对 PC 进行存储、匹配、预测。对于所有指令都预测不跳转。

根据流水线的实际

运行，可将 PC 寄存器的更新分为以下几种情况：

- 复位，PC 更新为入口地址 0xbfc00000。
- 异常，PC 更新为中断向量地址。
- 异常返回，PC 更新为 EPC 的值。
- 分支预测失败，PC 更新为实际的跳转地址。
- InstBuffer 满，PC 保持不变。
- 其他情况，PC 更新为动态取指给出的值。

有关分支预测的细节将在后面单独详述。

取指缓存阶段

此阶段读取指令 Cache 中的指令，将其送入 InstBuffer 中。InstBuffer 可以预取指令，使得后续流水线不因指令 Cache 读缺失而堵塞，使得取指部分相对独立于流水线。

考虑到当取出的指令不会被执行的情况下，只能清空 InstBuffer。为了最大程度发挥取指的优势，我们进一步控制 PC，充分与分支预测器进行结合，尽可能减少取指错误被清空的可能性。于是我们设计了关键性的动态取指方案，在取值请求阶段以非常高的准确率对 PC 和指令 Cache 进行控制，使得尽量少的浪费指令取出时间，保证流水线的畅通。

同时，我们对于整个流水线的暂停，取指并不关心。换句话说，只要是取出来的指令不会造成错误，那么取指就会一直执行下去，当然，为了防止取指到短期内不必要的指令，我们将 InstBuffer 的大小设定在了合适的范围。

InstBuffer 在使用 InstBuffer 之前，由于块地址的不对齐问题，会出现指令 Cache 强制单发问题，这样会降低双发率，将本可以双发的指令变为单发。同时，我们还要忍受各种暂停所浪费的时间。InstBuffer 将取指进一步剥离访存通路，使得指令更加无形化。在面对多种流水线暂停的条件时，能够继续读取指令。同时，为了能够处理延迟槽指令，我们将设定取指底线。为防止 InstBuffer 过满，我们也设置了取指的限度。值得注意的是，在分支预测和 InstBuffer 的操作下，我们实际上已经实现了指令预取，并且结合了分支预测的结果，我们将其称作动态取指。

动态取指 由于当前的 PC 的控制能力已经减弱至仅仅为取指，指令 Cache 和 InstBuffer 将进一步控制取指阶段的操作，因此，分支预测也就不得不接入到整个取指体系。同时，我们会遇到由此相关的 PC 取指问题，因此在某些情况下，PC 的值需要由动态取指给出，这表明 PC 已经大部分成为了指令 Cache 控制器的一部分。分支预测能够判断该 PC 是否为跳转指令，并且是否跳转，跳转的地址是多少，指令 Cache 将按照该预测进行取指，以保证在跳转的情况下，延迟槽以及跳转之后的指令能够被正确取出。当然对于不跳转也要正确处理。但是实际上，由于 PC 并非单独仅仅按照分支预测来进行处理，还要考虑指令 Cache 单发限定、InstBuffer 过满，Uncached 和 Cached 段取指的切换，Cache 不命中和命中等等状态，因此指令预取将考虑到以上所有的情况，并且同时还能支持分支预测错误情况的取指调整等措施，以动态的方式进行指令获取，大大提高取指的准确率。

译码发射阶段

此阶段接收 FIFO 中的指令和指令地址数据，对指令进行译码，取操作数，并决定发射条数。译码段由两个译码子部件和一个发射部件组成[1]。发射条数与指令类型、是否为延迟槽指令以及两条指令是否存在数据相关有关。具体如下：

- 若两条指令中存在乘除、访存、特权或自陷指令，则本次仅发射第一条指令。
- 若第一条指令是延迟槽指令，或者第二条指令是跳转/分支指令，则本次仅发射第一条指令。
- 为了避免 EX 段两个子部件互锁造成过高的组合逻辑延迟，存在“读后写”（RAW）相关的两条指令不同时发射。

若指令没准备好，则本次不发射指令，并发出流水线堵塞请求。

对于取操作数，由于其数据通路级数较多，我们采用以下方式，使之与译码阶段并行：取消寄存器堆的读使能信号，即让寄存器堆永远可读。之后，我们将指令的 rs 和 rt 字段直接发送给寄存器堆，得到操作数，再根据译码结果从操作数、旁路或立即数中选择所需要的最终操作数。

此阶段会检测异常。译码阶段可以检测的异常有指令地址不对齐异常、断点异常和系统调用异常。

执行阶段

此阶段执行指令，计算写回的结果。对于访存指令，还会生成数据 Cache 以及 LLBit 寄存器的驱动信号。单周期指令可在一周期内得到结果。多周期指令则需要若干个周期，在计算完成前需要发出流水线堵塞请求。多周期指令包括乘除法指令。除法采用试商法，共需要 32 个周期。乘法采用 Wallace 树型乘法器，先通过改进的 Booth 编码生成部分积，再通过多层进位保留加法器将若干个部分积压缩为 2 个，最终用 64 位先行进位加法器将部分积相加，得到乘法结果。由于乘法电路时延较大，会影响 CPU 主频，考虑到乘法指令在实际程序中占比较少，我们将乘法电路切分为两级，每个周期计算一级，共需 2 个周期得到乘法结果。

[1] 译码子部件设计文件为 id_sub.v，而发射部件以及它与译码子部件的连线在译码部件的设计文件 id.v 中

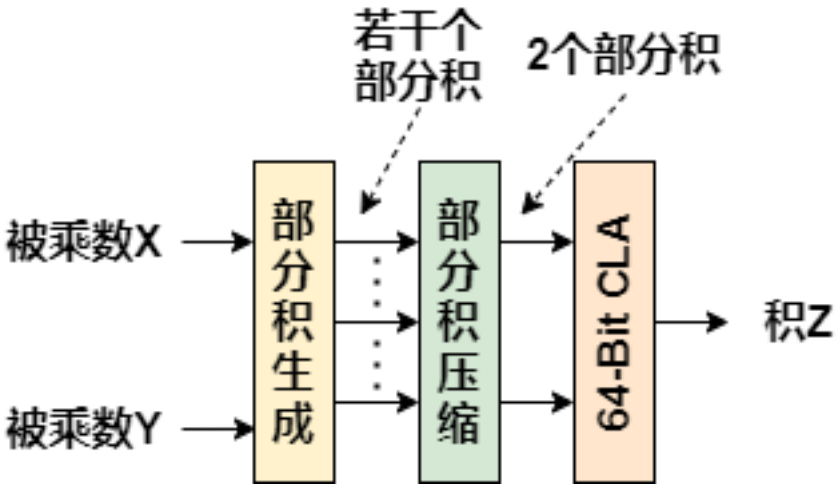

图 2.2: Wallace 树型乘法器简单示意

同时，如果当前执行的为访存指令，我们会在此阶段发出访存指令，如果命中，则会在下一个周期返回数据。值得注意的是，该情况不会暂停流水线，数据将会传到访存阶段，与指令对应起来。当遇到读缺失或写缺失时，同样需要堵塞流水线。由于时序设计特点，它在接收到访存请求的当拍不会发出堵塞请求，所以访存指令无论命中与否一定会被送到访存阶段，因此当访存缺失时，CPU 会延迟一拍堵塞访存及之前的流水线。

此阶段会检测异常。执行阶段可以检测的异常有溢出异常、自陷异常和访存地址不对齐异常。当执行阶段的指令的异常标志位显示有异常，或者访存阶段发出异常信号时，执行阶段发给数据 Cache 的读写请求将全部取消。

访存阶段

此阶段取出数据 Cache 送出的数据，根据指令类型，确定最终的写回数据。另外，此阶段将检测指令的异常信息，如发现确有异常，则会将异常信息告知流水线控制模块和 CP0 寄存器，发出流水线清除信号，并进入异常处理程序。

提交阶段

此阶段将需要写回的数据提交给各寄存器，包括通用寄存器、HILO 寄存器、LLBit 寄存器和 CP0 寄存器。

2.3 CP0

CP0 寄存器是 MIPS 规范中一系列特殊而又重要的寄存器，包含了许多 CPU 的状态信息，为操作系统的运行和管理提供了支持。我们实现的寄存器见下表。

表 2.1: 实现了的 CP0 寄存器

选择	编号	名称	功能简述	读写性
0	8	BadVAddr	最近发生异常的地址	只读
0	9	Count	计数器	读/写
0	11	Compare	计数器中断控制器	读/写
0	12	Status	状态寄存器	部分可写
0	13	Cause	最近一次异常原因	部分可写
0	14	EPC	最近发生异常的 PC	读/写
0	15	PRId	处理器标识符和版本	只读
1	15	EBase	中断向量基地址寄存器	读/写
0	16	Config	处理器配置寄存器	只读

2.4 异常

处理器在运行过程中会遇到一些特殊情况，需要打断原有的执行过程，这就是异常。

2.4.1 异常处理过程

MIPS 中的异常是“精确异常”。对于实现“精确异常”的处理器，若有一条发生异常的指令，则在这条指令之前的指令都要正常执行，而这条指令以及之后的指令都将被取消。为了实现“精确异常”，在流水线上执行的指令若发生了异常，并不会立刻处理它，而是把它记录下来，送到一个指定的阶段统一处理。在我们的处理器中，每条指令发生的异常都会在 MEM 阶段统一处理。

访存阶段检测到异常后，将按照优先级顺序向 CP0 寄存器报告异常类型，并向流水线控制部件告知发生了异常。MEM 及前面流水段的指令将全部清除，异常向量基地址会被送入 PC 寄存器，Cause 寄存器记录异常编号以及是不是延迟槽指令发生异常，Status 寄存器的 EXL 位置 1，表示进入异常处理阶段。这时，中断被屏蔽，即不会处理外部中断。对于地址错类型的异常，除了执行上述操作，还会将错误的地址写入 BadVAddr 寄存器。中断处理例程完成后，一般会执行异常返回指令。在我们的处理器中，异常返回被视为一种特殊的异常，处理机制类似于其他异常，只不过处理过程中送入 PC 寄存器的是 EPC 的值，Status 寄存器的 EXL 位置 0，表示异常处理完成，Status 寄存器的其他位以及其他 CP0 寄存器不会作任何修改。

2.4.2 支持的异常类型

我们的处理器实现了对大赛要求的所有异常的支持，下表列出了这些异常。

表 2.2: 支持的异常类型

异常类型	助记符	ExcCode 编码	异常描述
中断	Int	0x00	检测到了中断
读地址错	AdEL	0x04	指令地址或读数据地址不对齐
写地址错	AdES	0x05	写数据地址不对齐
系统调用	Sys	0x08	执行了系统调用指令
断点	Bp	0x09	执行断点指令
保留指令	RI	0x0a	执行了未实现的指令
算术溢出	Ov	0x0c	定点数加减法溢出
陷阱	Tr	0x0d	陷阱指令条件为真

2.5 分支预测

UltraMIPS 的架构在分支指令时会遇到至少 5 个周期的惩罚，导致其无法发挥出真正的性能。因此需要根据 PC 进行分支预测，提前将分支指令是否跳转的判断以及跳转地址发送至 Cache，以进行指令预取。程序中的大量分支指令以循环结构的形式出现，这类分支指令往往会在一段时间内反复跳转至同一地址。还有一类分支指令会在不同情况下跳向不同的地址，这类分支通常出现在 JAL 指令和 JR 指令配合使用的情况下，因此也是可预测的。

为保证预测的准确性，UltraMIPS 采用的动态分支预测。其实际上是根据历史分支信息判断，因此需要几个存放历史信息的查找表，存放分支指令的类型、跳转情况以及跳转地址。预测器需要在一周期内给出预测结果，考虑到组合逻辑时延，逻辑应当尽可能的简化。因为程序中分支指令并不很多，所以我们决定采用直接映射的 Cache 来存放分支信息。这样设计可以将 PC 直接作为索引，做到 PC 和分支信息的精准匹配，确保存入查找表的是分支指令，避免因压缩 PC 导致信息损失而造成误判，同时可以简化逻辑，为后期主频提升留足空间。

2.5.1 方向预测

方向预测即预测指令跳或不跳，UltraMIPS 采用两位状态机（饱和计数器）描述这一信息，四种状态如下：

SNT(strongly not taken): 指令有强烈的不跳转倾向

WNT(weakly not taken): 指令有一般的不跳转倾向

WT(weakly taken): 指令有一般的跳转倾向

ST(strongly taken): 指令有很强的跳转倾向

查找表中的每条指令都具备这一信息，该状态机在执行阶段传回准确的分支信息时更新。如果一条指令发生跳转，那么它的跳转倾向将加强。考虑到效率，我们没有采用基于局部历史或者全局历史的预测方式，因此只有当一条指令连续连续发生跳转的情况下，才会判断其未来将会跳转。这样做的好处是，不论对于循环类型的分支结构，还是调用类型的分支结构，都可以拥有极快的训练速度，从而达到极高的预测精度。缺点是，对于时而跳转，时而不跳转的程序，无法做到特别精确的预测，但这是极少数的情况，不影响预测器的整体表现。

2.5.2 目标预测

对于循环类型的分支结构（通常出现在 BEQ、BNE 等指令），UltraMIPS 采用缓存直接存储近期出现的分支跳转地址和分支指令类型，这些信息由执行阶段提供。对于调用类型的分支结构，预测器中实现了一个返回地址栈。此时，JAL 指令继续采用缓存预测跳转地址，因为缓存中存放了跳转类型，那么在对 JAL 指令进行预测后可以将其对应的 PC 加 8 后压入栈，在下遇到 JR 指令时直接从栈中获取跳转地址。这样的设计在对递归有很好的预测效果。

2.5.3 结构图

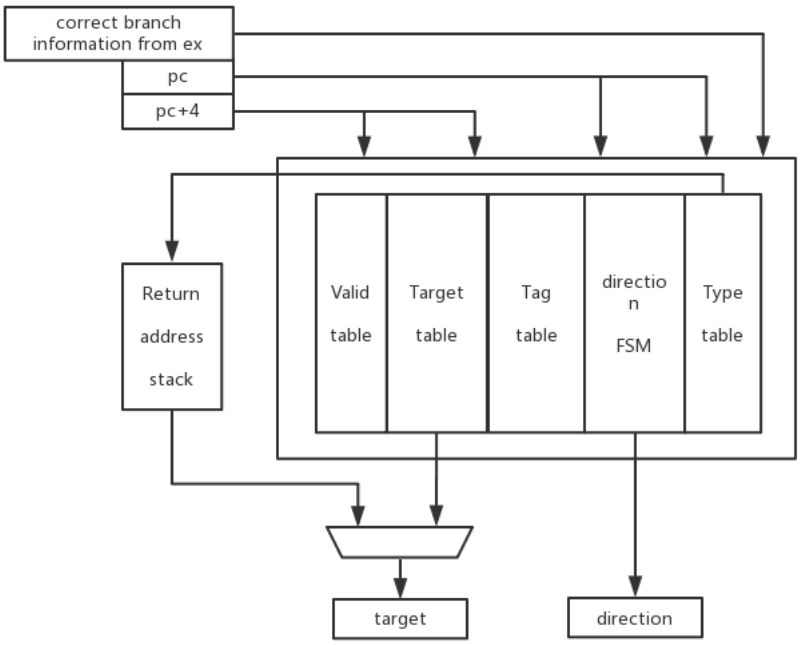

图 2.3: BPU 结构图

2.6 Cache

Cache 在整个 UltraMIPs 的架构之中是至关重要的存在，UltraMIPS 在设计 Cache 的时候更是将其发挥的淋漓尽致。Cache 深度融合于 CPU 之中，将数据的处理通路于 CPU 来说几乎隐形，只留下一个 Cache 的暂停信号，这其中的访存暂停机制都由 Cache 准备妥当，当数据准备充分，暂停信号便会自动撤销。在实际的设计之中，我们将整个 Cache 打包作为指令 Cache 和数据 Cache 的顶层模块。为了充分减少不必要的功能，我们将指令 Cache 和数据 Cache 均设计成为以块作为处理单元的模块，减少与主存进行交互的逻辑设计，在其之外我们设立了专门负责 Cache 和 AXI 交互的模块，称为 CacheAXI_Interface。该模块能够仲裁各路信号进行主存交互，并能够处理以块为单位的读写。

同时，Cache 作为其顶层模块，我们在其中加入了控制器[2]，用来控制 Uncached 和 Cached 处理，对

[2]实际上，为了方便进行动态取指和对指令 Cache 执行相应的动态指令预取，我们将动态取指的控制器模块也放在了 Cache 顶层模块，用来动

不同的取指核访存请求进行相应判断和处理。

指令 Cache 和数据 Cache 均为 8KB 的二路组相联，块大小为 32 个字节，即八个字。两者的替换策略都是伪 LRU 替换策略，替换的时候选择最近没被使用过的块。DCache 采用的主存写策略为写回方式，在遇到写不命中的情况，会先从主存读数据保存到 Cache 再进行写入。

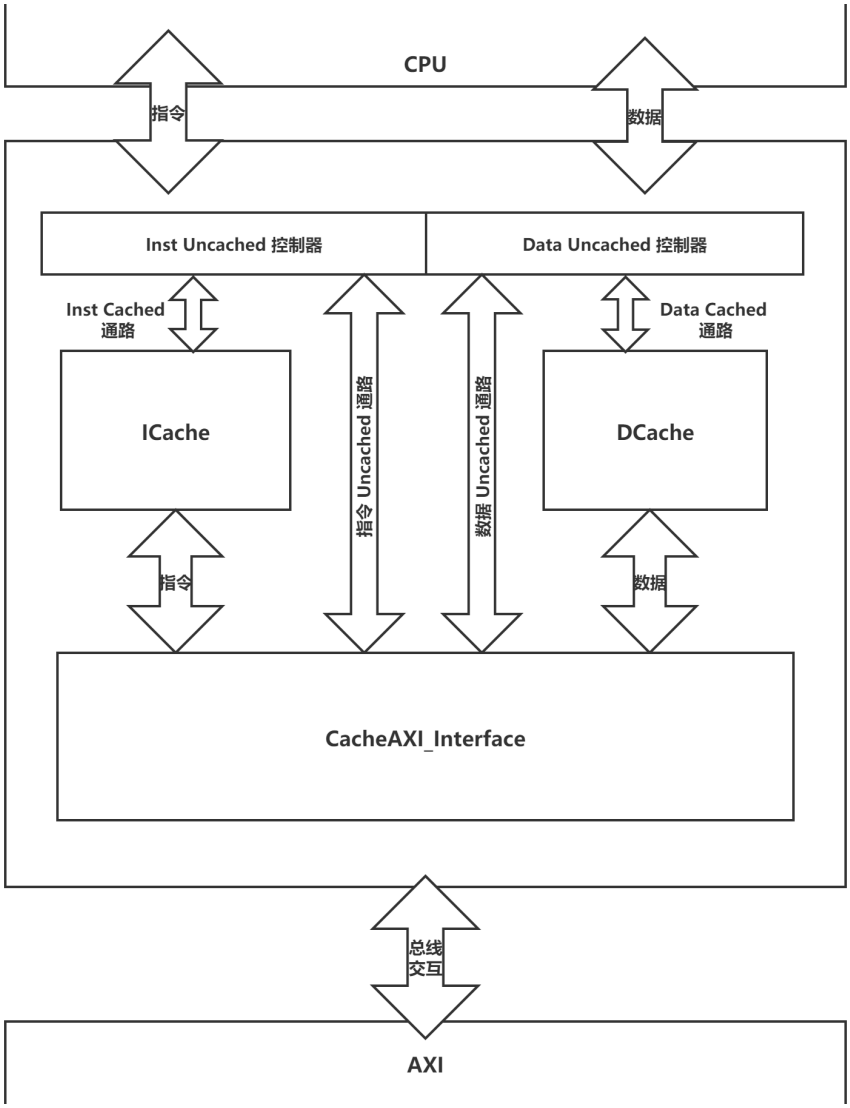

图 2.4: Cache 结构

指令 Cache

UltraMIPS 采用了流水指令 Cache，这样我们就能够在命中的时候连续的从 CPU 的取指阶段接受读指令信号，同时在译码阶段不停的送回指令数据。从设计架构来说，分成三个部分，但是实际上只有两级参与流水，

- LOOK_UP（流水）：将数据赋值于 ram 取数据，同时这个阶段要处理不命中返回 ram 的写入操作（该操作会造成数据相关问题）。

态的控制指令 Cache 的取指以及 PC 的变化。但是在之后结构图中我们去除了该部件，将其看作 CPU 中的一部分。

- SCAN_CACHE（流水）：对读出的数据进行处理，判断是否命中，命中则输出，不命中则送入不命中额外处理。
- HIT_FAIL（不命中额外处理）：访存，在访存结束的时候输出数据，并且将读出块写回 ram。

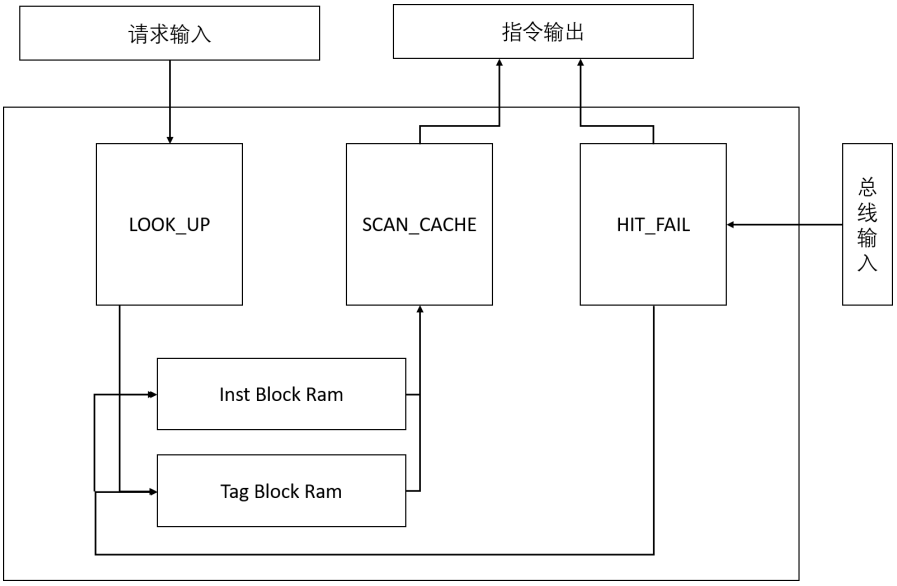

图 2.5: 指令 Cache 结构

数据 Cache

UltraMIPS 采用了非流水状态机数据 Cache，在 CPU 的执行阶段接受访存信号，在访存阶段送回数据。

数据 Cache 中的块被替换的时候采用的为伪 LRU 替换策略。Cache 写入主存中更新其数据的时候，采用的方式是写回方式，需要有脏数据的标记。当执行主存写指令的时候，若其数据存于 Cache 之中，则将 Cache 中的数据进行更改，并且同时将其所在块标记为“脏”。等到进行 Cache 数据替换脏块的时候，将脏块写入进内存中。为了保证器件复用，写不命中的时候不会直接写入主存，而是取出对应的数据将其填入数据 Cache，并做写入数据合并。数据 Cache 的具体状态机设计以 CPU 两级时序为参照，设计了对应两个阶段的状态如下。

- LOOK_UP：将数据赋值于 ram 取数据。
- SCAN_CACHE：对读出的数据进行处理，判断是否命中，命中则输出，不命中则送入不命中额外处理，对于写还需要将对应的写入数据进行拼接。

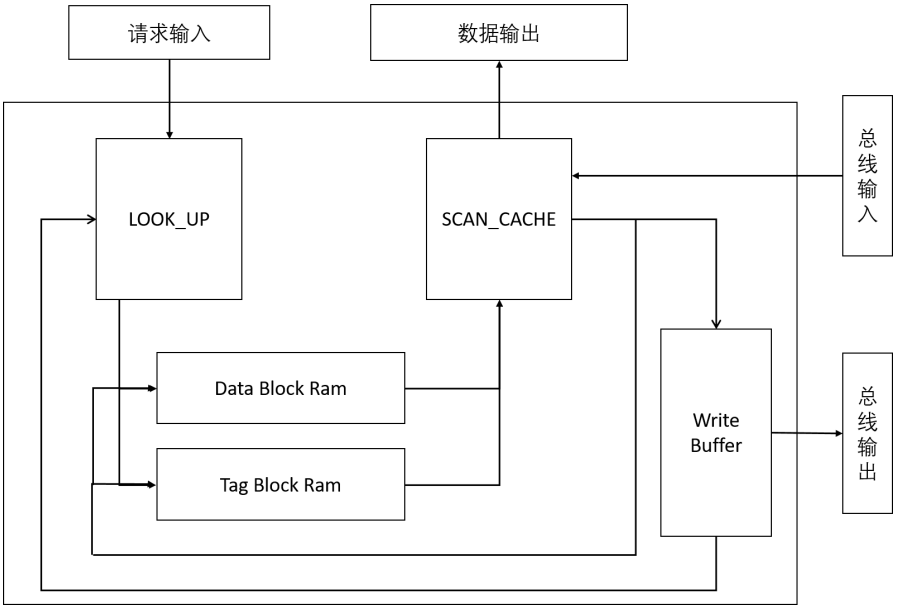

图 2.6: 数据 Cache 结构

具体来说,在基于器件尽量复用的角度下,UltraMIPS 中的数据 Cache 在第一个状态（LOOK_UP）均进行虚实转换经过地址寻址（VIPT）,交付给 cache_ram 取出对应的数据。在第二个状态（SCAN_CACHE）则有了一些区别，具体如下处理读写命中与不命中的情况如下

主存读命令：命中 Cache 将 Tag 和第零路和第一路以及 WriteBuffer 进行对比确定是否命中，命中则返回相应数据。

主存读命令：未命中 Cache 如果发现 Cache 和 WriteBuffer 都未命中，暂停 cpu，同时进行访存，直到收到数据返回信号。判断被替换的块数据是否为脏块，若是则压入 WriteBuffer（具体原理稍后介绍）进行写入。相应地址使能等交付给总线，之后开始等待。数据返回后送出，同时将数据送给 ram 进行写回。

主存写命令：命中 Cache 将命中数据与写入的数据进行组合重写进 Cache。

主存写命令：未命中 Cache 如果发现 Cache 和 WriteBuffer 都未命中，同时进行访存，直到收到数据返回信号。判断被替换的块数据是否为脏块，若是则压入 WriteBuffer 进行写入。相应地址使能等交付给总线，之后开始等待。数据返回后送出，同时将数据拼接写入的数据送给 ram 进行写回。这一阶段值得注意的是：写不命中时，若没有发生新的访存请求，DCache 不会发出暂停信号，让 cpu 继续运行。

WriteBuffer

该部件实质上是一个写入主存的等待队列，用来与总线进行数据交互，作为写缓存，WriteBuffer 避免了流水线和 cache 的堵塞。但是随之而来，我们会遇到一些数据逻辑上的问题。

- 读冲突：当读取地址在 WriteBuffer 中，我们需要将其从 WriteBuffer 中取出，并且表示为命中。
- 写冲突：当写入地址在 WriteBuffer 中，我们需要将其覆盖，保持最新数据。但是，如果需要被覆盖的数据正在被写入主存，我们就需要将其覆盖，并且标志之后需要继续重写。

Uncached

指令和数据地址在 0xA000000 至 0xBFFFFFFF 段将不经过 Cache，直接和内存进行交互。Uncached 最大挑战是如何让 CPU 能够感知不到其存在。为了解决这个问题，我们在 Cache 顶层模块加入了控制器，分别控制指令和数据的 Uncached 处理，并且在暂停和时序表现上都与 Cached 表现一致。

具体来说，首先我们先进行地址映射，进行虚实地址转换，以及相应的 Uncached 和 Cached 判断，并且进行相应的屏蔽，使得访存只在 Cache 和绕开 Cache 直接和 AXI 接口（指 CacheAXI_Interface）通信。我们对于指令 Uncached 和数据 Uncached 都进行了处理，并且设立了状态机和数据保持器，使得其能够和未命中 Cache 的表现情况相同。

CacheAXI_Interface

该模块兼任去耦合的任务，将整个 Cache 的访存通信进行包装，处理六路仲裁（指令读 Cached 与 Uncached，数据读写 Cached 和 Uncached）。同时，与之前提到的去耦合相对应，我们还要处理 Cache 以块为大小的访存问题，解决相应总线猝发和块拆解以及组合问题。

第三章　项目文件说明

以下为压缩包文件结构及部分文件的说明。

> HITSZ_2_lichenghao

- > bit
  - > func_test
    - > axi_mem_game_test 存放记忆游戏比特流文件
    - > soc_axi_func 存放 AXI 接口的功能测试比特流文件
    - > soc_sram_func 完成了 AXI 接口的 CPU，所以此文件夹为空
  - > perf_test 存放性能测试比特流文件
  - > system_test 存放系统测试比特流文件
- > src
  - > mycpu CPU 设计文件
  - * perf_clk_pll.xci 性能测试时钟 IP 核
- * design.pdf 设计文档
- * score.xls 分数报告

附录 A　参考设计及文献

A.1　参考设计说明

我们的 CPU 设计参考雷思磊的五级流水线 OpenMIPS，沿用其部分命名格式以及数据通路。在此基础上，为了实现双发射结构，并提高性能，我们进行了比较大的修改。

A.2　参考文献

- David A. Patterson, John L. Hennessy. 计算机组成与设计：硬件/软件接口（第 5 版）. 王党辉等译. 机械工业出版社
- John L. Hennessy, David A. Patterson. 计算机体系结构：量化方法（第 5 版）. 机械工业出版社
- D. Sweetman. See MIPS Run Linux (2nd Edition). 屈建勤译
- 雷思磊. 自己动手写 CPU. 电子工业出版社
- 姚永斌. 超标量处理器设计. 清华大学出版社
- CSDN 博客. https://blog.csdn.net/wxwd14388/article/details/82972947. 访问时间：2020 年 6 月 19 日