秋季学期实验指导书

实验版

课程教学用

2023 年秋季

《计算机体系结构》课程实验指导书

实验要求与注意事项

1. 实验分组进行，三个选题，其中浮点课题为 2 人一组，最多 3 组；TLB+cache 部分 3 人一组，最多 3 组；多发射、乱序执行以及猜测转移部分为 5 人，组数不限，实验评分包括现场验收和实验报告两部分。
2. 实验时间从实验任务布置日开始至本学期末结束，考核时间暂定为十二月底。
3. 现场验收由助教根据现场硬件情况核定，按照小组进行验收（要求小组所有成员必须在场），现场验收成绩占个人课设总成绩 50%。
4. 实验报告内容包括：实验目的，设计方案（原理说明及框图），关键代码及文件清单，仿真结果及分析，综合情况（面积和时序性能），硬件调试情况，成员分工，实验收获等。实验报告占个人课设总成绩 50%。注：需要在实验报告对应部分标明完成人姓名。
5. 实验报告提交方式：实验报告（pdf）和设计代码打包后发送至学生助教邮箱，文件名按照“2023 体系结构课程设计+实验小组编号”命名。
6. 硬件实验板将在实验开始后，由院里发放到各实验小组，实验验收后上缴。
7. 同学可在实验室完成实验，也可在寝室或其他地方自行完成，没有考勤限制，现场验收必须在指定时间段内的教室或机房进行。
8. 根据综合结果，在流水线设计功能正确的所有小组中，具有最高时钟频率的前 5 个小组将可获得 5-10%的加分，申请加分的小组需要单独提交申请，并需另提交一份代码和设计说明。
9. 实验严禁抄袭，有抄袭嫌疑（实验报告或者设计代码出现雷同、回答问题反映出相关工作明显非本人完成等）的现场实验或者实验报告按零分处理。

秋季学期实验：体系结构课程铁人三项实验设计

实验名称：

计算机体系结构铁人三项实验设计-二期

实验目的：

培养学生的计算机基本功：

1、 熟悉现代处理器的基本工作原理；掌握流水线、浮点、缓存、多发射、乱序执行、分支预测等处理器的设计方法。

2、 针对开源指令集编译器前端与操作系统设计，形成处理器、操作系统以及编译器的完备体。

实验工具：

HDL：Verilog；

IDE：Vivado；

FPGA 开发板。

实验一：数据通路设计

实验介绍

数据通路设计是微架构设计的核心，其决定了通用处理器性能。传统五级流水线单一架构，不能满足高性能任务处理需求，故在本实验中，我们要设计一个具备多发射，支持乱序执行和分支预测功能的处理器核，这些功能是一款现代处理器的基本要求。

多发射指的是一个周期内能够同时取多条指令，执行多条指令，这实现了 Instruction-level Parallelism(ILP)，可以显著提升单核的性能，当前处理器核普遍做到了 4 发射以上。在本实验中，需实现最基本的 2 发射结构（每周期至少支持取 2 条指令，发射 2 条指令到运行部件，提交 2 条指令）。

执行乱序执行的数据通路是一个大的框架，实验目的即实现框架内的各个模块，各模块之间需要定义好接口，以便不同模块之间可以同时开发，提升开发效率。

分支预测对处理器速度提升具有重要意义，该技术主要包含几个个方面，第一，跳转判定，一般由 Branch Predictor 实现；第二，跳转目标锁定，一般由 Branch Target Buffer 记录完成；第三，要考虑分支预测失败之后现场恢复的问题。

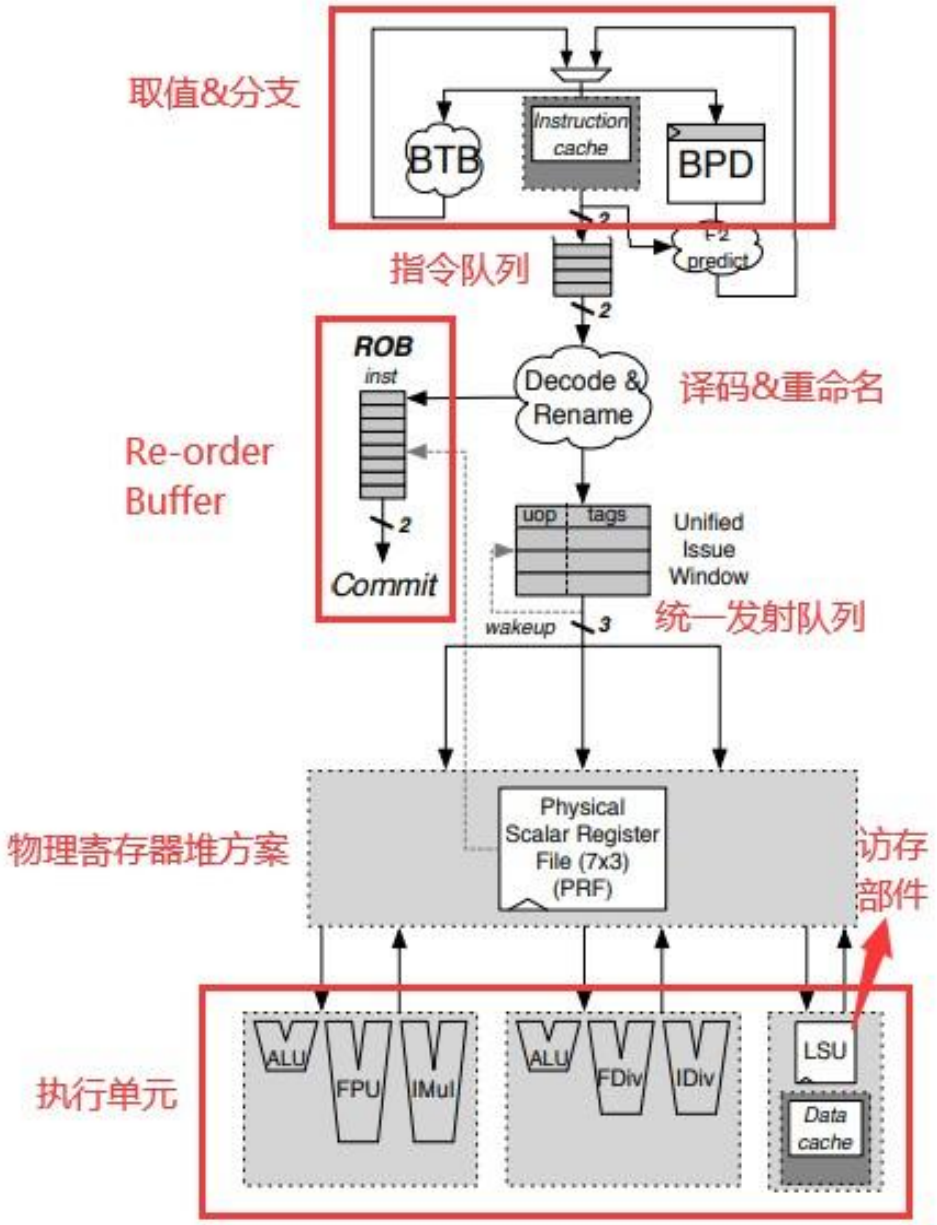
实现路径

乱序执行

乱序执行目的在与同步执行无真相关（RAW）指令，由于通用处理器内部寄存器资源有限，实际的程序中会存在假相关(WAW，WAR)，乱序执行需消除假相关，使无真相关的指令直接执行。

数据通路可分为前端和后端两部分，前端包括取指，译码，转移预测，寄存器重命名等，后端负责执行各条指令的详细操作，例如整数运算，浮点运算，访存操作等，指令执行结束后还要进行提交。前后端之间通过发射队列(Issue Queue)实现彼此的隔离，前端把译码好的指令发送到发射队列中，后端则从发射队列中读取执行。

具体的乱序执行流水线有很多种，可以参考课程讲解，指南以 BOOM v1 为例讲解一下现代处理器常采用的方案。



BOOM v1 是 UC.Berkeley 设计的一款乱序执行多发射 RISCV 处理器，其一个周期内，可以从 cache 中取两条指令，发射 3 条到执行单元，提交两条指令到 ROB。其采用了物理寄存器堆的重命名算法实现了数据依赖的消除，使用统一发射队列进行发射，不同执行部件的指令会被缓存到同一个发射队列中。

指令首先会被取到指令队列中，在取指的过程中，遇到 branch 指令，直接分支预测到新的位置进行后续的取值，到指令队列中的指令是分支预测后的结果。

指令队列中的指令被顺序发射到译码&重命名单元中进行下一步处理，在这里对逻辑寄存器号重新分配一个物理寄存器号，同时指令会进入 ROB。

译码重命名后的结果送入统一发射队列中，发射队列会记录指令实际使用的物理寄存器号，如果该寄存器可用了，直接发射到执行单元，在发射中会读取寄存器的值，相较于经典的保留站方案，该方式能少读一次寄存器。统一发射队列会将不同执行单元的指令都放到同一个队列中，资源利用率高，intel 常使用这种设计。

发射队列需要有唤醒电路和仲裁电路，唤醒电路负责识别那些指令的寄存器已经准备好了，仲裁电路则在所有准备好的指令中选择几条发送到执行单元中。执行单元的周期可以是不确定的，执行结束后，值要写到对应的寄存器中，还要将该指令提交到 ROB 中。分支指令还需要将分支结果和跳转值发送给分支预测单元，验证预测是否正确，并更新分支预测表，如果有中断，把中断信号带到最后，提交 ROB 时进行处理。

在设计数据通路时，要首先从整体层次上对各个部件的功能进行划分，定义好每个模块的输入输出和功能以及时序，降低各个模块之间的耦合，建议以文档约定好后，小组内部每个人实现不同的模块，进行分工、协作与对接。

本实验并不对乱序实现的方案做出太多具体的要求，可从保留站方案，物理寄存器堆重命名等方案中选择，但需要最后能实现相关功能。在验证阶段，需要使用 FPGA 进行测试，因此在编写寄存器堆时需要关注 FPGA 的特点，使代码好综合。

多发射

多发射让一个核能在一个周期内执行多条指令，一般我们用整个流水线中最窄宽度描述发射宽度。本实验中我们要求实现一个双发射的处理器，取值宽度，issue 宽度(指从 issue queue 到 execution unit)，提交宽度最低为 2，一般 issue 可以宽一些，因为执行单元可以多放几个。

多发射的难点在于前端译码和重命名步骤，在重命名阶段，除了找出当前指令与之前指令的冲突外，还需要判断与同时发射指令之间的冲突，并修正两者之间的冲突。此外，各个部件的端口数也要相应进行拓宽，支持多条指令同时读写。issue queue 的设计也是一个难点，需要设计好唤醒电路和仲裁电路。

分支预测

分支预测主要位于前端，在取指阶段对分支指令进行预测并猜测跳转地址。可将分支预测分为两部分工作，一部分是预测是否进行跳转，这个由 BPU(Brach Prediction Unit)负责，目前有较多经典算法可以参考，如 Gselect，Gshare，TAGE 等等，可以自行选择实现并调整其参数；另一部分是分支地址预测，设计一个 BTB(Branch Target Buffer)缓存数据，实现方式可参考 CAM（Content Address Memory），用 pc 进行索引，内容存储的是预测的跳转地址。

此外针对最常见的程序调用和返回，很多处理器会单独设计一个 RAS(Return Address Stack)来缓存程序返回时的地址。因为程序调用和返回满足栈的规则，因此可以在硬件中设计一个栈来记录 PC 信息，方便返回时预测地址。

分支预测的一个问题是预测错误时要回撤，这要求每次预测时要把预测的地址记下来，等待真实执行后进行比较。如果预测错误，需消除错误路径上的指令，此时可以选择等待的策略，等着分支指令被提交，然后处理错误路径，也可以通过 checkpoint 等方式直接消除错误路径的影响。

分支预测器需要具备更新功能的，因此还需要考虑分支预测器的更新工作，每次分支指令提交时，可把正确分支结果和分支地址送到 BPU 和 BTB 中，进行更新。

实验建议

- 利用 Verilog 编写硬件开发，建议基于上一期铁人三项的框架进行开发；
- 适度参考已有开源工作设计原理，例如 BOOM v1，BOOM v2，naxriscv，玄铁 C910 等等，主要了解其数据通路，时序安排，但不能完全照抄，最终要设计一个自己的架构。
- 本实验协作性质较强，设计前期需有较好的规划，在代码实现之前需规划好各个模块及其接口，准备共享开发文档，而后开始协作开发，每个成员可负责若干模块。开发遇到问题时应及时更新文档，尽量保证模块间低耦合，避免一个模块更改需要牵连多个模块的改动。
- 了解流水线的设计基本原则，模块之间注意建立 Valid/Payload 握手机制等。
- 认识到测试的重要性，在编写代码前可以先把测试代码写好。

实验要求

- 架构基础文档
  - 数据通路框图(参考上文 BOOM v1 图例)，标清楚各个模块，模块之间的连接关系，通路上的发射宽度
  - 各个模块之间的接口文档，说明 input/output 的类型和功能
  - 每个模块内部的架构框图，RTL 级别，大致介绍运作逻辑
  - 验收时需介绍架构的数据通路
- 实现乱序执行功能

    - 需要有多个执行单元(例如 3 个 ALU)
    - 需要有 ROB
    - 需要有发射队列
    - 能够实现精确中断，中断信息在 commit 时进行处理
    - 进阶内容：
        - 使用统一发射队列发射指令
- 实现多发射功能
    - 发射宽度至少为 2
        - 取指为 2，提交为 2，发射到执行单元的宽度可适当增加
    - 解码阶段能正确实现 2 条指令间冲突的消除
- 实现分支预测功能
    - 需要设计 BPU
        - 不要求具体分支预测算法，可用 Gshare，Gselect，TAGE 等等，不能用静态的(固定预测一边的)
        - 要求准确率 70%以上。(大多数算法都能保证 90%以上的准确率)
    - 实现 BTB
- 进阶内容：
    - RAS
    - 实现 checkpoints 恢复流水线

最终评定会首先评估实验完成情况，在基本要求都完成的情况下，会依据 benchmark 程序比较 IPC，并根据 IPC 进行排名。

参考资料

BOOM github

BOOM v1

BOOM v2

NaxRiscv

Alpha 21264 BPU

玄铁 C910

TAGE 分支预测器

Onur Lecture

动态分支预测-罗切斯特理工

CPU 的分支預測器是怎樣工作的？ - 知乎 (zhihu.com)

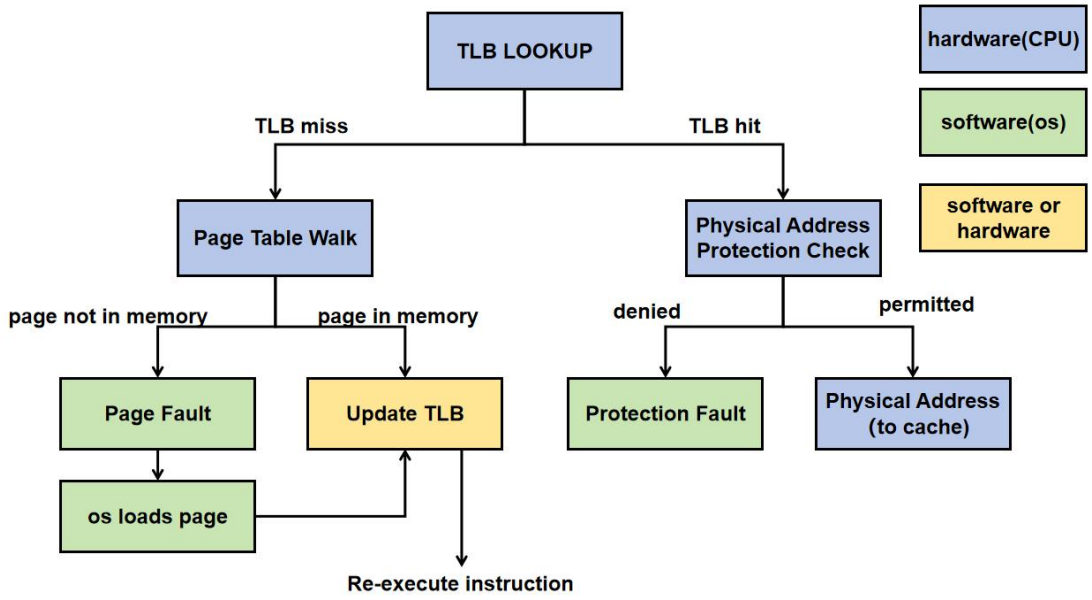
超标量处理器设计 姚永斌 清华大学出版社

实验二：缓存设计

TLB 设计

CPU 发出的访存地址是虚拟地址，经页表转换成物理地址，才能访问内存数据。页表一般存储在内存中，这意味着每次访问内存数据，实际上至少都要访问两次内存，即第一次访问页表获得物理地址，第二次才是访问数据。RISC-V 的分页方案比较典型的有 Sv32 和 Sv39，两种方案的页表项布局具体内容可以参考特权指令文档。为了减少访存次数从而提高访存性能，处理器中增加了一个特殊的 Cache，即 TLB（Translation Look-aside Buffer），用于缓存经常使用的页表项。TLB 由若干条 entry 组成，每条 entry 记录了一个内存页的虚实页号映射关系以及其他必要的控制位（TLB 表项的状态）。RISC-V 中用 satp（Supervisor Address Translation and Protection）的 S 模式控制寄存器来管理分页系统，satp 控制寄存器有三个域，MODE 域可以开启分页并选择页表级数，设置为 0 则关闭，MMU 设置为非 0 则开 MMU，ASID 域是可选的，用于进程的上下文切换，PPN 域存储了页目录表基址所在页面的物理地址。

TLB 通常设计为全相联映射。给定一个虚拟地址，CPU 检查 TLB（TLB lookup）以确定正在访问的页面的虚拟页码（VPN）是否在 TLB 中。若找到 TLB 条目（TLB hit），返回对应的物理页码，用于计算目标物理地址，并检查访问地址的合法性。若是未找到 TLB（TLB miss），则 CPU 从页表中寻找对应的条目（Page Table Walk）。找到当前虚拟页码（VPN）对应的物理页码（PPN）后，需要将对应的条目（VPN:FPN）添加到 TLB。如果在页表中找不到对应的物理页地址，应该触发页表异常（page fault），将控制权转交到操作系统，由操作系统将相应的页放入内存，更新页表，并返回到触发异常的指令继续执行。



Cache 设计

内存访问延时、功效开销大，频繁访问带来巨大的性能和功耗损失；另一方面，大多数程序都不会均衡的访问所有代码和数据，访问往往在时间、空间上具有局部性。根据这两点，往往在 CPU 和内存间设置高速缓存（cache），其由访问速度更快、容量相对更小的 SRAM 实现，用于暂存最近常访问的内存数据副本，减少对内存的频繁访问。

cache 设计主要关注几个方面的问题：1）结构组成，2）地址映射方式，3）写策略，4）置换算法，这也囊括了对 cache 部分实验设置的基本要求。

cache 的最基本结构是标签（tag）和数据（data）阵列以及控制逻辑。其中 tag 是地址的一部分，用于标识 cache 行；cache 容量小于内存，其中暂存的数据是部分内存的副本，这意味着多个内存中的块可能映射到 cache 的同一个行上，于是需要 tag 字段来对内存块进行标识。

cache 的地址映射方式主要有三种：直接相联映射，组相联映射，全相联映射，三种映射方式的含义不再赘述。图 1 是以 4 路组相联为例画的 cache 基本逻辑示意图（不包含置换算法的实现逻辑）。首先根据 cache 地址映射方式，根据其中的 set（组/行号）字段找到对应的 cache 组（或者直接相联里的 cache 行，全相联相当于只有一个组的组相联，没有 set 字段）；各组内各路（way）cache

行内的 tag 位与地址里的高位 tag 字段进行对比，并结合每个 cache 行的有效位（valid）判断是否命中，命中了哪一路（如果未命中，则需要置换算法决定替换哪一路）；在读取 tag 阵列对应 set 的多路内容进行对比的同时，读取 data 阵列对应 set 的内容，并在 tag 对比结果得出后，选择所得路的数据内容。

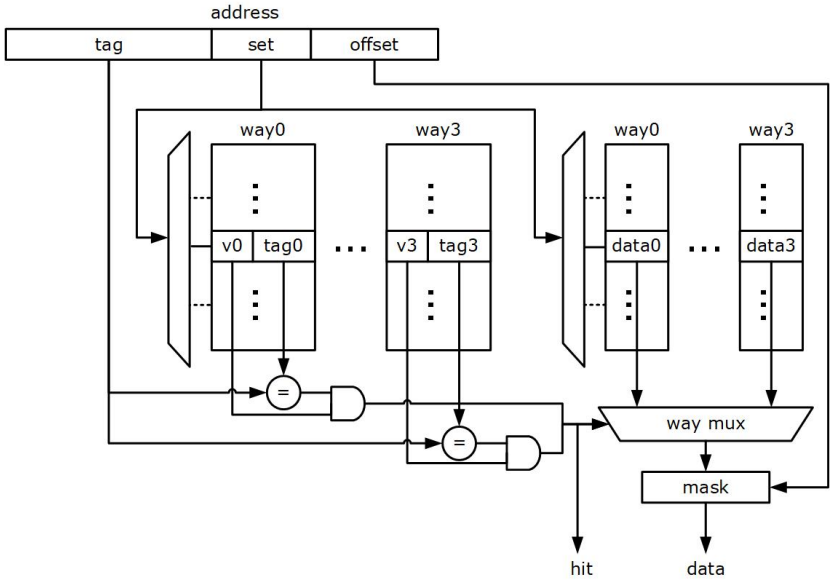


图 2.2 路组相联 cache 示意图

图是 tag 和 data 阵列并行访问设计，也存在 tag 和 data 阵列串行访问的设计，即在读取 tag 阵列内容并且得到对比结果之后，才根据得出的命中和路信息访问 data 阵列指定 set 和 way 的内容。后者可以减少暂存 data 阵列读取内容寄存器的开销，但是在时序优化方面没有优势。

cache 的写策略主要分为两部分内容，涉及到要被写的地址是否在 cache 中，这部分也不再赘述，只说明写策略有两种组合，一种是直写和写不分配的组合，一种是写回和写分配的组合。考虑到减少内存频繁访问情况的目的，写回和写分配的组合具有更多的优势。

对于组相联和全相联映射，当要访问的 cache 组已满，而需要访问的数据内容又不在该组中（cache miss），则需要用要访问的新数据把其中一路替换掉，这就需要使用置换算法。常见的置换算法有以下几种：1）先进先出，即先进入该组的路优先被替换，该算法由队列实现；2）随机算法，即随机替换一路，由一个内置的时钟计数器实现，需要替换时根据计数器内容决定置换哪路；3）最近最少使用（least-recently used，LRU），替换最近最早访问的路，实现方式

是每个 cache 行对应一个 log2(路数量 way)位的计数器，取值 0~(way-1)，数值小的表示最近被访问的，数值最大的优先被替换，计数器值变动方式是当访问某一路 cache 行时，该行对应计数器置 0，其余计数器都加一；4）最近最不经常用（least-frequently used），计数器记录访问次数，优先替换访问次数最小的。

还存在其他的置换算法，如伪 LRU 策略（pseudo-LRU，PLRU），该策略的出现是为了减小 LRU 策略带来的较大的硬件开销问题。该策略采用树的方式记录各路数据的访问情况，树上的每个枝干节点 1 位，取值 0 或 1，表示是右边的最近被访问还是左边；way 个叶节点正对应了一组里的所有 cache 行。相比 LRU，PLRU 策略每个组只需要 way-1 位，而 LRU 每组需要 way x log2（way）位，PLRU 大幅降低硬件开销；但 PLRU 也有缺陷，它并不总是选择最不常用的 cache 行进行替换，这也是为什么它被称为伪 LRU 算法。

除此之外还有许多其他 cache 置换算法，欢迎各位同学自行调研取舍。

除上述 cache 的基本构成外，还有一些从性能角度出发对于 cache 的优化手段，下面主要列举比较常见的几种：

1. 多级缓存；
2. 流水线设计；
3. 无阻塞缓存：阻塞缓存在 cache 未命中时不能接受新的访存指令，而非阻塞缓存可以暂存未命中的指令信息，在等待其处理完成的过程中接收来自 CPU 的更多访问指令。这是通过在 cache 里加入多个缺失状态寄存器（Miss Status Holding Registers, MSHRs）来实现的，MSHR 负责暂存 cache 未命中的指令信息，并等待给内存的请求的结果返回。每个 MSHR 对应一个 cache 行的访问指令，多个访问同个 cache 行未命中的指令可以合并到一个 MSHR 里。当有新的 cache 行未命中，且 MSHR 分配满时，则发生资源冲突。
4. 采用多种缓存以提高缓存带宽：即将缓存划分为几个相互独立、支持同时访问的缓存组。从地址的角度看，划分 bank 个组即从地址中选择连续的 log2(bank)位，取值从 0~(bank-1)标识各个 bank（取位需要高于 offset 字段）。

5. 合并写。

其他优化方法还包括路预测、关键字优先和提前重启动、编译器优化、预取、编译器控制预取等。

实验要求：

Cache 部分：

实现一个高速缓存，**必须满足的设计要求**如下：

1. 采用多路组相联设计；
2. tag 和 data 阵列分开，且并行访问；
3. 写策略为写回和写分配的组合；
4. 实现一种 cache 置换算法。

进阶要求：

选择并实现至少一种 cache 优化手段，可以不在上述列举的优化手段中。

TLB 部分：

CPU：

在实验给定的处理器核上，设计一个支持 Sv32 或者 Sv39 页表项（PTE）的 Memory Management Unit （MMU）模块，并采用 TLB 加速虚实地址的转换。其中指令 TLB（ITLB）和数据 TLB（DTLB）分开设计，为全相联映射。TLB 表项的设计方案可以参考 cva6 架构设计书 。

需要实现虚实地址的翻译，TLB 表项的插入和替换。其中 TLB 的替换算法可以应用简单的替换算法如 LRU，也可以阅读更多文献选择其他高效的替换算法。

OS：

操作系统中添加对应的 page fault 处理函数，用特权指令将页表项（PTE）加载进 TLB。

评估指标：

主要包括访问延时、带宽、吞吐量、cache 命中率、硬件开销。将根据各指标表现进行综合评定。

参考资料：

https://github.com/airin711/Verilog-caches
https://github.com/IObundle/iob-cache
https://github.com/xdesigns/4way-cache
https://github.com/rajshadow/4-way-set-associative-cache-verilog
https://github.com/psnjk/SimpleCache
参考书目：
1. 计算机体系结构—量化研究方法
2. 计算机组成与设计硬件软件接口

cva6 架构设计书 ：
https://docs.openhwgroup.org/projects/cva6-user-manual/04_cv32a6_design/source/cv32a6_execute.html#load-store-unit-lsu

实验三：浮点功能部分

浮点运算是现代计算机系统中不可或缺的一部分，尤其在科学计算、图形处理和深度学习等领域。RISC-V 架构支持多种浮点指令集，为了更好地理解和实践浮点运算的原理和设计，本实验需完成处理器浮点部件设计。

实验目标

- 理解 IEEE 754 浮点数表示法。与其他架构的处理器一样，浮点部件的实现需要遵循 IEEE 754 的浮点数乘法标准。IEEE 754 是一个国际标准，定义了浮点数的表示方法和运算过程。它确保了不同的处理器和系统在进行浮点运算时能得到一致的结果。与其他架构的处理器一样，浮点部件的实现需要遵循 IEEE 754 的浮点数乘法标准。这意味着，无论你在哪里，无论你使用的是哪种处理器，只要遵循这个标准，都可预期相同的浮点运算结果。

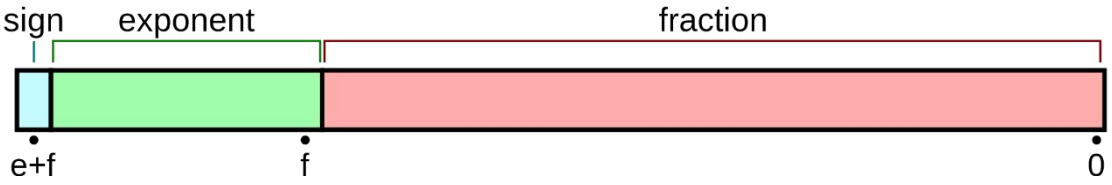
- 设计并实现浮点加法、乘法和除法运算。浮点运算并不像整数运算那么直观，例如，浮点数的加法涉及到对齐、加法、规格化和舍入等步骤，而乘法和除法更为复杂。本实验要求实现基本的浮点运算部件，需理解浮点数背后数学原理。

- 了解浮点部件的优化技术。作为加分项，了解浮点部件的优化技术，比如流水线化，并行处理，精度调整，特殊指令优化等。

浮点数基础

IEEE 754-2019 浮点算数标准是计算机中浮点运算的黄金标准。它确保了不同的处理器和系统在进行浮点运算时能得到一致的结果。这不仅仅是一些数学公式，而是一个确保计算机中数值计算准确性和一致性的关键工具。

- IEEE 754 标准:

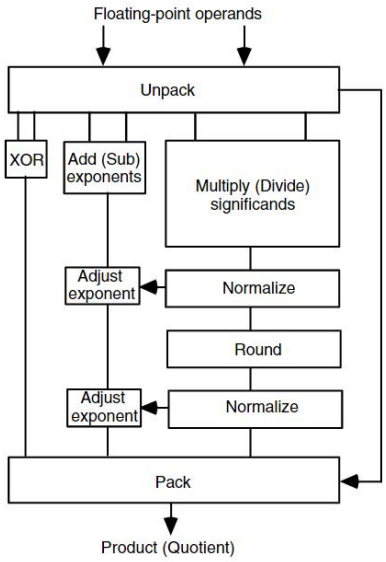


格式和操作：此标准定义了浮点数在计算机中的存储格式和运算规则。它涵盖了二进制和十进制浮点数据的格式，包括加法、减法、乘法、除法等基本操作。

异常处理：定义了浮点运算中可能出现的异常条件，并指定了这些条件的处理方法。这确保了浮点运算的鲁棒性和可靠性。

舍入模式：舍入是浮点运算中的一个关键步骤，它确保了浮点数的精确表示。IEEE 754 标准定义了多种舍入模式，包括向最近偶数、向零、向正无穷和向负无穷。这些模式提供了不同的舍入策略，可以根据具体的应用需求进行选择。

浮点乘法器的设计参考



浮点乘法器设计示意框图

浮点乘法的实现思路主要包括以下几个部分：

1. 解析 2 个操作数（unpack），把符号位，指数和尾数区分开。并分辨特殊的数据类型。（无穷，NaN 等）

2. 两个操作数的指数部分求和并减去一倍指数的偏移量。

3. 操作数的尾数部分按照定点乘法的规则相乘，生成新的尾数部分。

4. 做规格化和舍入的操作。把数据重新打包，输出最后的结果。

除此之外，浮点乘法器应当同样支持对特殊数字（NaN，无穷等）的处理，并按照 IEEE754 标准输出期望中的结果。

乘法器的设计中，比较重要的是乘法算法的选择。尾数部分的乘法可以参考定点的乘法器设计，思路是把乘法转化成加法计算。此处主要的优化思路有两个方向，一是减少部分和的数量，二是减少部分和的累加过程中的延迟。可以选择使用基 4-Booth 乘法/华莱士树/4:2 进位压缩器或者其他算法来优化定点乘法的实现。

基 4-Booth 乘法简介：

booth 乘法是基于数据的补码表示中每一位的权重设计的算法，具体的数学推导此处不再赘述，可以去阅读参考链接中的文章。其优化方向是减少部分积的数量，根据被乘数 A 的每 3 位编码，可以直接得出对乘数 B 需要做的操作以生成部分积。基 4-booth 乘法可以减少定点乘法中一半的部分积。下图展示了基 4-booth 乘法的编码原理。

跟基2的算法一样，假设A和B是乘数和被乘数，且有：

$$A = (a_{2n+1}a_{2n})a_{2n-1}a_{2n-2}\dots a_1a_0(a_{-1}) \quad (1)$$
$$B = b_{2n-1}b_{2n-2}\dots b_1b_0 \quad (2)$$

其中，a_{-1}是末尾补的0，a_{2n}, a_{2n+1}是扩展的两位符号位。可以将乘数A表示为：

$$A = (-1 \cdot a_{2n-1})2^{2n-1} + a_{2n-2} \cdot 2^{2n-2} + \dots + a_1 \cdot 2 + a_0$$

同样可以将两数的积表示为：

$$\begin{aligned} AB &= (a_{-1} + a_0 - 2a_1) \times B \times 2^0 + (a_1 + a_2 - 2a_3) \times B \times 2^2 \\ &\quad + (a_3 + a_4 - a_5) \times B \times 2^4 + \dots \\ &\quad + (a_{2n-1} + a_{2n} - 2a_{2n+1}) \times B \times 2^{2n} \\ &= B \times [\sum_{k=0}^{n}(a_{2k-1} + a_{2k} - 2a_{2k+1}) \cdot 2^{2k}] \\ &= B \times Val(A) \end{aligned}$$

基 4-booth 乘法的编码原理

华莱士树简介：

华莱士树是基于 3:2 进位压缩器（全加器）设计的一种加法树。它可以通过 3:2 进位压缩器逐级地把 3n 个数据压缩为 2n 个，并且减少累加过程中不同组数据之间的进位依赖。华莱士树的基本思想就是利用 n 个全加器把 3 个 n 位的数字相加转换为 2 个 n+1 位的树相加，在具体实现中，可以用 n 个全加器把 m 个 n 位的部分积相加转换成 2m/3 个 n+1 位的数相加，依次类推，最后转换成 2 个数相加。

优化思路：此处的关键路径会在尾数的乘法部分。可以通过把这部分使用多级流水实现以优化性能。

实验内容

1. 浮点加法器设计
    - 输入：两个浮点数。
    - 输出：加法结果。
    - 主要步骤：对齐、加法、规格化、舍入。
2. 浮点乘法器设计
    - 输入：两个浮点数。
    - 输出：乘法结果。
    - 主要步骤：乘法、规格化、舍入。
3. 浮点除法器设计
    - 输入：两个浮点数。
    - 输出：除法结果。

- 主要步骤：除法、规格化、舍入。

实验要求

- 基本要求：

  1. 设计并实现浮点加法器、乘法器和除法器。
  2. 验证设计的正确性，使用给定的测试数据。

- 进阶要求：

  1. 对浮点部件进行优化，如流水线设计或并行计算。
  2. 分析优化前后的性能差异。

评估指标

- 功能正确性：浮点部件的输出是否与预期一致。
- 性能：浮点部件的吞吐量和延迟。
- 资源使用：使用的逻辑单元、寄存器等硬件资源数量。

参考资料：

1. IEEE 754 - 2019 浮点算数标准 - 知乎 (zhihu.com)
2. IEEE SA - IEEE 754-2019
3. He-Liu-ooo/Ibex-RISCV-floating-point-instruction-set-extensions: This is a floating-point instruction set extension based on Ibex RISCV CPU (github.com)
4. julianamitie/RISCV_FloatingPoint (github.com)
5. 《COMPUTER ARITHMETIC : Algorithms and Hardware Designs》

6. 《计算机体系结构》 胡伟武 等
7. 【硬件算法笔记 18】浮点运算器设计 - 知乎 (zhihu.com)
8. 【HDL 系列】乘法器(6)——Radix-4 Booth 乘法器 - 知乎 (zhihu.com)
9. Jung-Yup Kang and J. . -L. Gaudiot, "A Simple High-Speed Multiplier Design," in IEEE Transactions on Computers, vol. 55, no. 10, pp. 1253-1258, Oct. 2006, doi: 10.1109/TC.2006.156.

指南编写小组成员：

王新宇、中科院计算所
卢美璇、中科院计算所
吴瑜萍、中科院计算所
孙琰斌、中科院计算所
王磊、中科院计算所
闵丰、中科院计算所