n00b 设计概要

版本：α

更新：January 1, 2022

本文档为 n00b CPU 的设计概要文档。

1 流水线设计

乱序 CPU 上指令的执行在概念上可以分为如下 10 个阶段：取指，译码，寄存器重命名，读寄存器，调度，发射，执行，读写内存，写回以及提交。在实现的过程中部分阶段可以进行合并，部分阶段需要进行拆分，最终得到如下几个流水级：

1. **取指** 为后续指令集提供不间断的指令流。
2. **译码** 将指令转换为处理器内部的信号，并将之放入 ROB。
3. **寄存器重命名** 为目标寄存器分配新的物理寄存器，同时替换源寄存器为相应的物理寄存器。
4. **调度** 将指令送入各个保留站中等待执行。
5. **发射** 各个功能单元从保留站中选出一条已经准备好的指令准备执行。同时没准备好的指令监听结果总线以获取需要的数据。
6. **读寄存器** 从寄存器堆中读取相应的操作数。
7. **执行** 进行运算。
8. **写回** 将运算结果写回到寄存器。
9. **提交** 按顺序提交最多 4 条已经完成的指令。异常以及 CP0 相关指令在此处理。

2 取指单元

为了尽可能的提高主频，n00b 的取指单元共分为 5 个阶段 $F_1 \sim F_5$。

1. F_1 更新 PC。
2. F_2 从指令缓存中获取最多 8 条指令，同时在此对分支/跳转指令进行预测。
3. F_3 从指令缓存返回的 4 组指令中选择正确的结果。
4. F_4 对获取到的指令进行预译码，根据分支预测器的结果确定跳转的目标地址，以此修正 PC。并将之放入指令队列中等待执行。
5. F_5 从指令队列中获取 1 组指令送入后续指令级开始解码等操作。

ITLB 是 TLB 的子集，为 16 项全相联结构。当 ITLB 查询失败时，会到 TLB 中查找结果并回填到 ITLB 内，如果仍然查找出错，则产生一个 TLB 异常。

我们拟实现的分支预测器为混合式的分支预测器。对于条件转移指令，我们使用 GShare 分支预测机制；对于间接跳转我们则通过一个转移目标表预测目标地址。

我们拟实现的 GShare 分支预测机制包含一个 9 位的历史寄存器以及 512 项的转移历史表 (BHT)。该表的每一项都是 2 位饱和计数器，每当分支预测正确时计数器加一，预测失败计数器减一。当计数器的最高位为 1 时预测跳转成功。

对于间接跳转，我们拟实现一个 16 项的转移目标表 (BTB) 来记录转移指令的地址以及目标地址。当需要进行替换时，根据被查找的频次进行替换。

MIPS 中的函数调用通常通过转移链接指令以及 $jr\ r31$ 指令来完成。我们可以利用这一特性，借助返回地址栈 (RAS) 精准预测返回指令的地址。当出现转移链接指令时，我们将其 $PC + 8$ 压入 RAS，当出现 $jr\ r31$ 指令，则将 RAS 的顶作为目标地址送入到下一流水级。

当前的设计会导致执行循环体较小的程序时候效率过低，在后续可以考虑复用指令队列实现一个循环缓冲 (loop buffer) 提高循环的效率。

3 寄存器重命名单元

n00b 使用物理寄存器堆的方法实现寄存器重命名，物理寄存器堆的大小为 64 + 5(HiLo) 项，物理寄存器和体系结构寄存器之间的关系被保存在一个 32 + 2 项的寄存器重命名表 (RAT) 里面。RAT 分为两组，其中一组在重命名时即时被修改 (current RAT, CRAT)，另一组则在指令提交后才进行修改 (retirement RAT, RRAT)。进行指令重命名时，处理器首先从表中读出各个操作数寄存器对应的物理寄存器，之后为目标寄存器从闲位表 (free list) 中分配物理寄存器，并将之写入到 CRAT 内。RRAT 只有在提交阶段才被更新，一旦提交阶段出现异常/转移预测失误，我们可以直接将 RRAT 复制到 CRAT 内来完成命名表的回滚。

闲位表记录了当前寄存器的分配情况，其被实现为一个 64 位的寄存器以及两对相向的优先编码器，以此来为最多四个目标位置分配物理寄存器。

忙位表 (busy table) 记录了各个物理寄存器内的值是否已经准备好，该表内寄存器对应的值在分配时被置位，在写回时清零。当发生异常/转移预测失误时，将该表清零即可。

HiLo 寄存器使用更为简单的方式重命名，在此不再赘述。

在指令分别被重命名后，还需要处理同一拍内各个指令间的数据依赖关系，如果后面的指令的源寄存器中包含了前方指令的目标寄存器，则需要对源寄存器进行替换。

重命名结束后，各个指令按照类别的不同被送往不同的保留站。

4 指令调度

重命名后的指令被送到保留站等待执行，n00b 具有两个独立的保留站，每个 16 项：访存指令被送入访存保留站，其余指令被送入运算保留站。

在寄存器重命名时，各个操作数是否已经准备好可以通过查忙位表来确定。对于没有准备好的操作数，指令在保留站中根据监听结果总线以及前推总线的结果来确定源操作数是否准备好，结果总线中的数据来自四个功能部件。

保留站每一拍最多可以同时发射四条指令到四个功能部件 (3 个 ALU, 1 个访存部件)，当多个指令可以发射时，优先选择最先进入保留站的指令发射。

从保留站中发射的指令首先需要读物理寄存器堆，n00b 的寄存器堆大小为 $64 \times 32 + 5 \times 32$ 位，有 9 个读端口，4 个写端口。ALU_1 和 ALU_2 各占用 2 个读端口 1 个写端口，ALU_3 占用 3 个读端口和 1 个写端口，访存部件占用 2 个读端口和 1 个写端口。

5 指令执行

保留站中的指令最后会被发往功能部件执行。n00b 拟实现 3 个定点运算模块，1 个访存模块。

ALU_1 和 ALU_2 用于执行加减运算，逻辑运算，移位，比较，Trap 以及转移指令。这些操作延迟均为 1 拍。

ALU_3 除去执行加减运算，逻辑运算，移位，比较这些延迟为一拍的操作外，还需要执行乘法和除法运算。乘法使用 Xilinx 提供的 IP 核实现，为全流水操作，延迟为 6 拍。除法使用不恢复余数除算法，延迟为 32 拍，非流水操作。

访存单元的实现与 MIPS R10k 中的访存单元类似。

6 指令提交

为了实现精确异常，在 n00b 中指令乱序执行但顺序提交。重定序缓冲 (ROB) 负责指令的顺序提交。ROB 被实现为一个 16 项的 FIFO，每一项最多可以保存 4 条指令的结果。只有当一项内的指令全部执行完毕才会进行提交操作。提交时，忙位表、闲位表以及 RRAT 的相应位置被修改，跳转指令的结果被送回到分支预测器以优化预测器。

当异常/分支预测错误被触发后，例外原因以及 PC 数值被记录在 ROB 对应的项内。当提交到该项时，流水线刷新，新的 PC 值被送入寄存器，同时 ROB 中后续全部项被标记为已完成但无效。对于已完成但无效的指令，其目标物理寄存器会被回收到对应的闲位表和忙位表中。

CP0 的读写操作在此进行，当同一项内在对应指令前面的指令确定执行完毕后，CP0 的读写操作开始执行，延迟为 1 个周期。通知存储单元进行写操作的方式与之类似。

7 总结

目前 n00b 的设计可以笼统的分为两部分，取指部分负责从指令缓存中获取指令流并将之转为后端处理的 ‘‘指令包’’，后端则以指令包为单位进行执行并写回。如此设计可以确保取指逻辑与执行逻辑分离，为后续优化留下空间。