The Just-In-Time (JIT) Processor

李增钰
lizengyu21@mails.tsinghua.edu.cn

岳章乔
yuezq21@mails.tsinghua.edu.cn

张作远
zhangzuo21@mails.tsinghua.edu.cn

赵涵远
zhaohy22@mails.tsinghua.edu.cn

In *Qualifier Submission of National Student Computer System Capability Challenge (NSCSCC'2024).* By Tsinghua University, 3 pages.

1 Introduction

本项目是第八届“龙芯杯”全国大学生计算机系统能力培养大赛 (NSCSCC 2024) 的参赛作品。我们基于 LoongArch 32 Reduced 指令集架构实现了乱序多发射 CPU： JIT (Just-In-Time)，并在比赛提供的 Chiplab 与 FPGA 实验平台上完成了完整的 CPU 微架构设计和验证。经过反复检验，该 CPU 能够通过全部功能测试和性能测试。

2 Methodology

为了提高开发效率，我们选用了在工业界已有部分应用的开源硬件描述语言 SpinalHDL，通过 SpinalHDL 生成 Verilog 代码，并结合龙芯提供的 Chiplab 进行仿真和综合实现。

我们使用 mill 进行项目管理，并搭建了基于 Chiplab 的敏捷开发范式。设计模式上，我们参考了 SpinalHDL 社区成熟项目 VexRiscv，复用了其对流水段（Stage）、流水线（Pipeline）和流水段逻辑（Plugin）的定义。在架构设计上，我们采用了基于 Tomasulo + ROB 算法的乱序多发射架构。该架构下，CPU 由前端和后端两部分组成。前端包括四阶段取指（IF1 / IF2 / IF3 / IF4）、译码（DECODE）、重命名（RENAME）和分发（DISPATCH）；后端则根据不同的流水线类型，划分为不同的流水段，主要包括发射（ISS）、读寄存器（RD）、执行（EXE）和写回（WB）阶段。最终，所有的指令都必须通过统一的提交口在 ROB 中进行提交（retire）。

硬件平台方面，我们使用比赛提供的基于龙芯架构的教学实验箱（Artix-7）作为目标平台，使用龙芯提供的基于 nemu 的 difftest 框架对 CPU 进行调试，使用基于 Chiplab 的框架配置 ci，通过 Xilinx™ Vivado 2023.2 进行综合实现。

在功能实现上，我们实现了异常、中断、地址转换等功能，并且由于流水线级数较深，大量采用了分支预测、缓存和旁路等技术进行加速。

3 Design

在基于 Tomasulo + ROB 算法的乱序多发射架构下，我们的微架构分为前端与后端两部分。下面将根据具体的单元进行介绍。

3.1 取指流水线

取指流水线的功能部件，可以分为以下三类：

3.1.1 指令读取与分发.

- **I-Cache Controller**

  对读取的指令块暂存，按需访问总线获取执行指令码，存储单元实现为基于 Block RAM 的存储块，缓存大小 8KB，组织为二路组相联，必要时以 LRU 机制对缓存行替换。

- **FetchPacketFIFO**

  把取指部分生成的指令流，结合有关的分支预测结果和预测时快照，放入 FIFO 内，由译码部分获取。

3.1.2 地址转换.

- **IMMU**

  实现由 ISA 定义的直接地址映射，和页表映射模式的地址转换，生成实体地址，交给 I-Cache 对 Tag 匹配，选取数据。

3.1.3 分支预测.

- **PCMux**

  综合多级分支预测器的预测结果，以及后端重定向地址，生成下一个指令（包）的基地址。

- **GHR**

  跳转记录，作为分支预测器的辅助信息。

- **BranchPredictor**

  即集成 (ensemble) 分支预测器，集成了规模极小、逻辑简单的下一行预测器 (NLP)，精确度较低的、决策过程比较简单的相关预测器，以及精确度较高的、决策过程较为复杂的 Tage 预测器。与此同时，在预测器的末端，还实现了一个 RAS，对调用返回对进行捕获；较精确的预测器在取指段较为靠后的部分，把较前部分的预测结果覆盖，如果失配则冲刷较前段。

由于分支预测器的功能比较复杂，加上乱序超变量处理器中，预测错误对性能的影响比较明显，本项目特别的对该部分功能做出优化，下面介绍当前实现的预测器中包含的子模块。

3.1.4 跳转选取信息的维护.

1. **L0 PHT** 作为 NLP 的一部分，其原理同下方的相关预测器，规模很小，基于寄存器 Bank 实现，在 IF1 即可生成跳转选择结果；
2. **相关预测器**
   该预测器结合全局信息和取指 PC，得到一个散列值，该散列值对一个二位计数器表索引，该计数器的高位表示是否跳转，按跳转结果增减预测时计数器值；
3. **TAGE 预测器** 该预测器维护多个表，对于不同的表，以不同长度（成几何关系）的跳转历史结合 PC 对表索引；倾向选取历史长度较大者的有效输出；
   表上还有一个 Useful 计数器字段，这个字段用以辅助新记录的分配过程中，Useful 计数器还会定时清空，保证表上信息的时效性，原理上类似 Clock 替换算法。

3.1.5 跳转目标的维护.

1. **L0 BTB** 作为 NLP 的一部分，对立即数偏移目标的跳转指令（条件跳转、B, BL），缓存其跳转目标。该缓冲区大小为 8，基于寄存器 Bank 实现，在 IF1 即可生成跳转目标；
2. **L1 BTB** 可以对基于寄存器的间接跳转以外的绝大部分跳转指令，进行精确的跳转目标的预测；
3. **RAS** 捕捉调用–返回对，遇调用指令时把地址入栈，遇返回指令则把地址退栈。

3.2 解码流水线

解码流水线由 DECODE, RENAME 和 DISPATCH 三个阶段组成，各阶段的作用如下：

- **DECODE 阶段**：完成指令的解码以及立即数的扩展，将指令拆解成各个功能部件所需的控制信号和数据，生成微码（uOP), 以便后续阶段使用。
- **RENAME 阶段**：进行寄存器的重命名，在我们的设计中，使用两个寄存器重命名表 aRAT 和 sRAT 来维护寄存器映射状态，其中 sRAT 由 RENAME 阶段维护，aRAT 由 COMMIT 阶段维护，当需要恢复处理器状态时，将 aRAT 拷贝到 sRAT 中即可完成映射关系的恢复。
- **DISPATCH 阶段**：指令分发的逻辑实际实现在各个发射队列插件中，各发射队列根据指令类型将重命名后的指令放入发射队列。由于某些整数指令只能在特定的整数发射队列处理，因此需要处理可能产生的阻塞。

3.3 整数流水线

整数流水线由 ISS，RD，EXE，WB 四个阶段组成，在我们的微架构中共设置三条整数流水线，每个整数流水线对应一个整数发射队列，为了减少片上面积，某些出现频率较低的整数指令只会在特定的整数流水线处理。以下是各个阶段的功能：

- **ISS 阶段**：从发射队列中取出指令，同时进行指令的唤醒。
- **RD 阶段**：从物理寄存器堆中读取寄存器。
- **EXE 阶段**：进行算术运算和分支跳转等计算，并且在该阶段需要进行数据旁路的计算。
- **WB 阶段**：将计算结果写回寄存器堆以及 ROB。

注：算术运算指令、分支跳转指令、CSR 访问指令、RDTIME 指令、INVTLB 指令均从整数流水线提交。

3.4 乘除流水线

乘除流水线包括以下阶段：ISS、RD、EXE 和 WB。其中 EXE 阶段是多周期实现。乘法器使用的是 IP multiplier 核，而除法器则调用了 SpinalHDL 中的 Divider.scala 库

3.5 访存流水线

访存流水线包含以下阶段：ISS、RD、MEMADDR、MEM1、MEM2 以及 WB。其中 MEMADDR 和 MEM1 产生访存地址，MEM2 完成访存。其主要组件如下：

- **DCache**：使用 VIPT 技术，LRU 替换策略，缓存行大小为 64 字节，共有 64 组，每组 2 路。
- **StoreBuffer**：处理 Uncached Load 和 All Store 类指令。其效果只有在提交阶段才同步生效。

3.6 其它逻辑部件

这里是其他逻辑部件的简单介绍。所有相关的 CSR 都在对应的插件中实现。

- **ROBFIFOPlugin**：实现重排序缓冲（ROB）。
- **CommitPlugin**：实现指令提交逻辑，并向前端传递更新信号。
- **ExceptionHandlerPlugin / InterruptHandlerPlugin**：处理所有异常和中断，包括时钟中断。
- **PhysicalRegisterFilePlugin**：物理寄存器堆（PRF），在这里实现指令唤醒的广播。
- **Timer64Plugin**：64 位计数器。
- **MMUPlugin**：进行地址翻译。

4 References

在本项目的设计与开发过程中, 我们参考了以下资料:

1. Seznec, A., & Michaud, P. (2006). A case for (partially) TAgged GEometric history length branch prediction. The Journal of Instruction-Level Parallelism, 我们参考了其中的方法实现分支预测器。

2. VexRiscv. https://github.com/SpinalHDL/VexRiscv. 我们参考了其中的 builder 模块的设计哲学, 即主要是关于逻辑插件 (Plugin) 的定义.

3. 超标量处理器设计. 姚永斌. 我们参考了其中关于乱序多发射超标量流水线的实现.

4. SpinalHDL Documentations

5. RISC-V BOOM 和 XiangShan 关于模块化预测器的设计理念

6. SpinalCrypto https://github.com/SpinalHDL/SpinalCrypto 参考该库实现的随机数生成器 (Fibonacci LFSR), 用于 TAGE 表的分配

7. NOP-processor. 我们感谢往届的参赛学长对我们在最初阶段的指导.