

# AzureMIPS——顺序双发射 11 级流水处理器

## NSCSCC2022 决赛答辩

李睿潇、朱元依、张政镒、李少群

复旦大学

2022 年 8 月 20 日

- ① 工作介绍
- ② 处理器介绍
- ③ 系统设计



- ① 工作介绍
- ② 处理器介绍
- ③ 系统设计



# 概述

本次参赛所实现的处理器（包括高速缓存部分）全部使用以 Scala 为基础的新锐硬件开发语言 SpinalHDL 进行开发设计。通过将 SpinalHDL 的源码转换为 Verilog 在 FPGA 上进行综合、实现与验证。

利用 SpinalHDL，我们实现了硬件设计的模块**参数化**，同时得以其语言特性，模块之间的连接也有相当高的**灵活性**。由于 SpinalHDL 不需要像 Chisel 一样通过中间层转换，其获得的 Verilog 代码可读性更高，给后续 Debug 带来了便利。

我们工程的所有源代码已经在[GitHub](#)上开源。



图 1: SpinalHDL

## ① 工作介绍

## ② 处理器介绍

处理器整体架构  
流水线设计  
分支预测设计  
高速缓存设计

## ③ 系统设计



## ① 工作介绍

## ② 处理器介绍

处理器整体架构

流水线设计

分支预测设计

高速缓存设计

## ③ 系统设计

# 前端架构

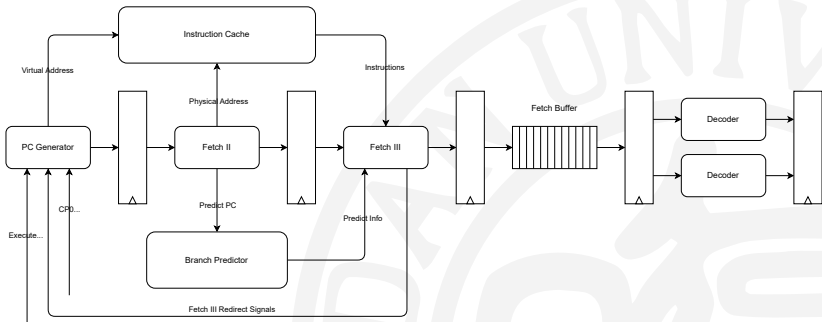


图 2: 前端架构设计简图

# 后端架构

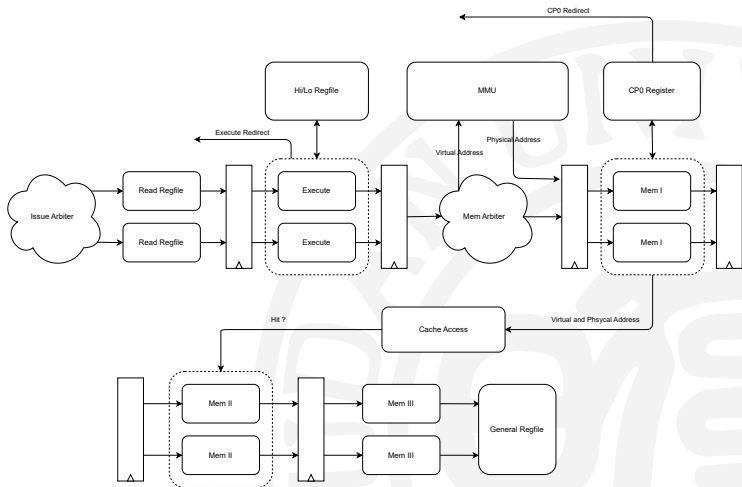


图 3: 后端架构设计简图



## ① 工作介绍

## ② 处理器介绍

处理器整体架构

流水线设计

分支预测设计

高速缓存设计

## ③ 系统设计

# 取指

- 取指流水线
  - Fetch 1: 仲裁 PC, 发送虚拟地址至 ICache
  - Fetch 2: 发送通过 MMU 获得的物理地址, 若有异常, 则作标记
  - Fetch 3: 对 ICache 返回的指令作快速译码, 判断出分支指令, 必要时进行 PC 重定向
- 取指缓冲
  - 16-Entries 指针 FIFO
  - 每周期最多 “4 进 2 出”
  - 跳过非分支延迟槽中的空指令
- 分支预测
  - RAS 分支预测
  - Bi-Mode 分支预测器
  - 256-Entries BTB

# 取指缓冲对双访存的优化

```
1  sw t8, 0(a3)
2  lw t7, 8(t0)
3  nop
4  sw t7, 8(a3)
5  lw t6, 12(a0)
6  nop
7  sw t6, 12(a3)
8  lw t5, 16(a0)
9  nop
10 sw t4, 20(a3)
11 lw t3, 24(t0)
12 nop
13 ...
```

```
1  sw t8, 0(a3)
2  lw t7, 8(t0)
3
4  sw t7, 8(a3)
5  lw t6, 12(a0)
6
7  sw t6, 12(a3)
8  lw t5, 16(a0)
9
10 sw t4, 20(a3)
11 lw t3, 24(t0)
12
13 ...
```

# 译码、发射与读寄存器

- 译码
  - 译码模块单独占一个流水级
  - 将指令译成对应的  $uOp$ ，并译出必要信息
  - 判断是否为保留指令或会触发协处理器不可用的指令，并作标记
- 发射
  - 组合逻辑，与读寄存器共占一级流水级
  - 进行单发射仲裁
- 读寄存器
  - 后续流水级旁路和通用寄存器堆间的数据仲裁
  - 当所需数据在流水线中还没被计算出来时，暂停操作
  - **提前计算跳转地址**供执行阶段进行分支预测正确性检验

# 执行

- 执行
  - 对于普通运算指令，执行阶段直接计算结果并放入流水线中
  - 对于访存指令，执行阶段计算其访存地址
    - 在初赛提交中，本周期还会进行访存地址冲突检查
    - 在决赛提交中，访存地址冲突检查移到单独一级流水
  - 对于多周期指令，执行单元请求流水线暂停，直到其执行完毕
  - 执行单元直接和 Hi/Lo 寄存器交互，故不需要对 Hi/Lo 寄存器作旁路转发
  - 如果指令需要跳转，执行单元会发出跳转信号

# 执行

通过参数化标记,上下两个执行单元有着**不一样的功能,但用着同一份代码**。只有上执行单元拥有处理分支指令的能力,去除了无用的冗余,也不引来额外的麻烦。

```
1 class SingleExecute(  
2     advanced : Boolean = true  
3 ) extends Component {  
4     ...  
5 }  
6  
7 class Execute extends Component {  
8     ...  
9     val units = Seq(  
10         new SingleExecute(true),  
11         new SingleExecute(false)  
12     )  
13     ...  
14 }
```

# 访存

**访存冲突与虚实转换流水级：**该级为决赛提交中拥有的一级流水级，原因在于决赛提交版本中含有 TLB，访问需要较长的时间，因此我们对流水线再次进行了切分，其主要功能如下

- 地址冲突检查
  - 若出现同时写同一个字对齐后的地址，则进行单发调度
  - 若出现使用不同数据通路（DCache 与 Uncache）的地址，则进行单发调度
- 虚实地址转换
  - 访问 TLB，获得 VPN 对应的 PFN
  - 若出现 TLB 异常，则对指令作标记，在下一周期中处理

# 访存

- 访存第一阶段
  - 向 CacheAccess 发送虚拟地址与物理地址，拉起访存请求
  - 将指令的异常信息交付异常处理单元
- 访存第二阶段
  - 等待 CacheAccess 返回 Hit 信号
- 访存第三阶段
  - 收到返回的数据，写回通用寄存器堆



## ① 工作介绍

## ② 处理器介绍

处理器整体架构  
流水线设计  
分支预测设计  
高速缓存设计

## ③ 系统设计

## Bi-Mode 分支预测器

在设计上, 我们借鉴了开源的 RISC-V 处理器玄铁 910 的分支预测部分, 采用了设计简单、性能优秀的 Bi-Mode 分支预测器。这种分支预测方法融合了局部分支预测和全局分支预测, 充分利用了历史分支信息, 同时还将倾向于跳转和倾向于不跳转的分支指令分两个表进行存储, 有效降低了混叠干扰。

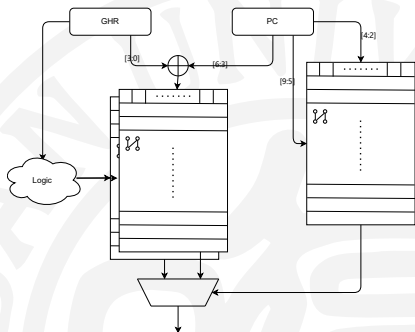


图 4: 分支预测图示

# 分支预测准确率

表 1: 性能测试中的分支预测表现

程序	准确率	程序	准确率
bubble sort	70.1%	coremark	77.3%
crc32	95.3%	dhrystone	96.0%
quick sort	75.5%	select sort	92.1%
sha	97.6%	stream copy	97.1%
stringsearch	88.4%		

## 1 工作介绍

## 2 处理器介绍

处理器整体架构

流水线设计

分支预测设计

高速缓存设计

指令高速缓存

数据高速缓存

## 3 系统设计

# 指令高速缓存

- 4 路组相联
- 4 Banks, 基于  $xpm\_memory$  128bit 4
- 取指合并
- 所需的 4 条指令不在同一个缓存行中 → 1 CacheLine
- 支持 Cache 指令

# 数据高速缓存

- 4 路组相联
- 基于  $xpm\_memory$  **真双端口**  
    《
- 支持 Cache 指令

## ① 工作介绍

## ② 处理器介绍

## ③ 系统设计

PMON

uCore

Linux

## ① 工作介绍

## ② 处理器介绍

## ③ 系统设计

PMON

uCore

Linux



## 成功启动 PMON，并能正确运行 PMON 的全部指令



## ① 工作介绍

## ② 处理器介绍

## ③ 系统设计

PMON

uCore

Linux



## ① 工作介绍

## ② 处理器介绍

## ③ 系统设计

PMON

uCore

Linux