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. 外设
4. 系统设计

1 工作介绍

2 处理器介绍

3 外设

4 系统设计

开发语言

(a) Scala (b) SpinalHDL

图 1: Scala 与 SpinalHDL

参数化、灵活、可读性高。

SpinalHDL 相比于 Chisel 更有独特的优势

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 外设

4 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 外设

4 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 外设

4 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**分支快速译码**，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

```
sw t8, 0(a3)
lw t7, 8(t0)
nop
sw t7, 8(a3)
lw t6, 12(a0)
nop
sw t6, 12(a3)
lw t5, 16(a0)
nop
sw t4, 20(a3)
lw t3, 24(t0)
nop
...
```

```
sw t8, 0(a3)
lw t7, 8(t0)

sw t7, 8(a3)
lw t6, 12(a0)

sw t6, 12(a3)
lw t5, 16(a0)

sw t4, 20(a3)
lw t3, 24(t0)

...
```

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
  - 普通运算
    - 交ALU执行
  - 访存
    - 计算地址
    - 生成Mask/Size等AXI参数
    - 地址冲突检查 初赛
  - 多周期
    - stall
    - 直接与HiLo交互
  - 跳转 advance
    - 发出跳转信号
    - 计算跳转地址

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。

```
class SingleExecute(
  advanced : Boolean = true
) extends Component {
  ...
}

class Execute extends Component {
  ...
  val units = Seq(
    new SingleExecute(true),
    new SingleExecute(false)
  )
  ...
}
```

访存

预访存流水级：该级为决赛提交中拥有的一级流水级，其主要功能如下

- 地址冲突检查
- 虚实地址转换

图 4: TLB 转换示意图

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 外设

4 系统设计

Bi–Mode 分支预测器

- 借鉴于**玄铁 910** 的分支预测设计
- **融全局与局部**为一体
- 分表存储，降低混叠

图 5: 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 外设

4 系统设计

指令高速缓存

- 4 路组相联，4 Banks，基于 xpm_memory
- 每周期可返回 128 bit 数据（4 条指令）
- 取指合并
  - 所需的 4 条指令不在同一个缓存行中 →1 次请求访问两条 Cache Line，将访问结果合并后输出
- 支持 Cache 指令

图 6: 指令高速缓存结构示意图

数据高速缓存

- 4 路组相联
- 基于 xpm_memory
- **真双端口**，一周期两条访存指令，收益 ≫ 成本！
- 支持 Cache 指令

图 7: 数据高速缓存结构示意图

1 工作介绍

2 处理器介绍

3 外设

4 系统设计

VGA

(a) VGA：显示条形图

(b) VGA：显示图片

飞机大战游戏演示

1 工作介绍

2 处理器介绍

3 外设

4 系统设计
- PMON
- uCore
- Linux

1 工作介绍

2 处理器介绍

3 外设

4 系统设计
- PMON
- uCore
- Linux

PMON

- 成功启动 PMON
- 正确运行 PMON 的全部指令

1 工作介绍

2 处理器介绍

3 外设

4 系统设计
- PMON
- uCore
- Linux

uCore

- 成功启动 uCore
- 能正确运行 uCore 的指令
- 实现了自定义算法指令

1 工作介绍

2 处理器介绍

3 外设

4 系统设计
- PMON
- uCore
- Linux

Linux

- 完成 Linux 的内核启动流程，进入 `kernel_init` 函数

致谢

谢谢！

我们的代码已经开源
https://github.com/0xtaruhi/AzureMIPS