gem5 O3CPU 代码解读实验报告

O3CPU 结构

O3CPU（Out-of-Order CPU）的工作分为几个阶段，每个阶段在每个时钟周期都会执行一次特定的工作。整个流程包括：取指（Fetch）、解码（Decode）、重命名（Rename）、发射（Dispatch）、发射（Issue）、执行（Execute）、写回（Writeback）和提交（Commit）。

各阶段的工作过程

一、Fetch（取指阶段）

目标：从内存中取指令

1. **检查信号：**

1.1 检查是否有来自后续阶段的信号阻塞（block）或取消（squash）当前的操作。

1.1.1 信号阻塞（block）

阻塞信号（block）通常由 Decode 阶段发出，用于控制 Fetch 阶段暂停指令提取，以协调流水线操作和资源使用。

发生情况与处理流程：

检测 block 信号：Fetch 阶段会通过 checkSignalsAndUpdate() 函数检测来自 Decode 阶段的 block 信号。

暂停提取：如果检测到 block 信号，Fetch 阶段会暂停从指令缓存中提取新的指令。

保持状态：在 block 信号被清除之前，Fetch 阶段保持当前状态，不会更新 pc（程序计数器）或 fetchQueue（提取队列）。

1.1.2 信号取消（squash）

取消信号（squash）通常由 Commit 阶段发出，用于处理分支预测错误或其他需要重新开始提取的情况。

发生情况与处理流程：

检测 squash 信号：Fetch 阶段会通过 checkSignalsAndUpdate() 函数检测来自 Commit 阶段的 squash 信号。

清空提取队列：如果检测到 squash 信号，Fetch 阶段会清空 fetchQueue，以确保之前提取的指令被丢弃。

重定向程序计数器：Fetch 阶段会根据 squash 信号提供的新 PC 值（程序计数器）重新开始指令提取。

恢复分支预测器：Fetch 阶段会回退或重置分支预测器的状态，以反映新的执行路径。

1.2 更新分支预测器的状态。

更新分支预测器的状态，根据来自 Commit 阶段的 `doneSeqNum` 信号。

2. **取指令：**

2.1 从内存中取指令并放入一个缓冲区（fetchBuffer）检查 Fetch 阶段的当前状态 fetchStatus

2.1.1 如果 `fetchStatus` 为 Running 并且当前块已失效：

- 设置 `fetchStatus` 为 ItlbWait（等待地址翻译完成），构造 `FetchTranslation` 事件，调用 MMU 的 `translateTiming()` 进行地址翻译。

- 地址翻译完成后调用 `finishTranslation()`，向 `icachePort` 发送请求，更新 `fetchStatus` 为 IcacheWaitResponse(等待指令缓存的响应)。

- 接收到内容后调用 `recvTimingResp()`，复制数据到 `fetchBuffer`，更新 `fetchStatus` 为 IcacheAccessComplete(指令缓存访问完成)。

2.1.2 如果 `fetchStatus` 为 IcacheAccessComplete，更新为 Running。

2.1.3 如果 `fetchStatus` 为其他状态，不做任何事。

2.2 如果遇到分支指令，使用分支预测器来猜测下一个将要执行的指令地址。

3. **处理状态：**

3.1 若上述步骤发生了 status_change，调用 updateFetchStatus()

3.2 对于所有线程，若 issuePipelinedIfetch 被设置，则调用 pipelineIcacheAccesses()：取 this_pc+fetchOffset 所在的行

4. **准备下一步：**

将指令从 fetchQueue 移到 toDecode，准备传递给下一个阶段（Decode）。

二、Decode（解码阶段）

目标：将取到的指令解码为具体的操作

1. **检查信号并更新状态：**

调用 checkSignalsAndUpdate()函数检查所有信号：

1.1 检查来自 Commit 阶段的 `squash` 信号：如果收到 `squash` 信号，清空 `fromFetch`、`insts queue`、`skidBuffer`。

1.2 如果被 Rename 阶段 block，调用 `block()`：将指令从 `insts queue` 移入 `skidBuffer`，给 `toFetch` 设置 block 信号。

1.3 解除上述 squash 或是 block 信号

2. **解码指令：**

若 decodeStatus 为 Running、Idle、Unblocking，调用 decodeInsts()进行解码操作

2.1 decode Status 若为 Unblocking 状态，则它刚从阻塞状态恢复过来，从 skidBuffer 中读取指令进行操作，否则则从 insts queue 中读取指令进行后续操作

3.准备下一步

对于至多 decodeWidth 条指令，传递到下一个阶段进行操作

三、Rename（重命名阶段）

目标：消除寄存器之间的依赖关系

1. **检查信号并更新状态：**

调用 checkSignalsAndUpdate()：

readStallSignals()：来自 Dispatch 的 block/unblock 信号。

来自 Commit 的 squash 信号：清空 fromDecode、insts 队列、skidBuffer。

如果被 Dispatch 阻塞，调用 block()：将指令从 insts 队列移到 skidBuffer，设置 toDecode 为 block。

解除 Blocked 或 Squashing 状态。

2. **重命名寄存器：**

若 renameStatus 为 Running、Idle、Unblocking，调用 renameInsts() 进行 rename 操作，将指令中的逻辑寄存器（例如 R1, R2）重命名为物理寄存器（例如 P1, P2），避免不同指令之间的寄存器冲突。

3. **更新映射表：**

记录每个逻辑寄存器对应的物理寄存器。

4.**准备下一步**

将重命名后的指令传递给下一个阶段（IEW）。

四、IEW 阶段

（一）、Dispatch（发射阶段）

目标：将指令分发到合适的执行单元

1. 信号检查：

调用 checkSignalsAndUpdate()检查信号：

readStallSignals()：检查来自 Issue 的 block/unblock 信号。

若有来自 Commit 的 squash 信号：清空 fromRename、insts 队列、skidBuffer。

如果被 Issue 阻塞，调用 block()：将指令从 insts 队列移到 skidBuffer，设置 toRename 为 block。

解除 Blocked 或 Squashing 状态。

2.分派指令

如果 dispatchStatus 为 Running、Idle、Unblocking，调用 dispatchInsts()：

根据 dispatchStatus 是否为 Unblocking，若不是则从 insts 队列读取指令，否则从 skidBuffer 读取指令。

对于最多 dispatchWidth 条指令：

检查目的寄存器是否已分配物理寄存器。

将指令移动到相应的功能单元（例如整数 ALU、浮点单元等）的队列中。

更新调度队列状态。

（二）、Issue（发射阶段）

目标：准备好执行指令

1.信号检查

checkSignalsAndUpdate()：

readStallSignals()：来自 Execute 的 block/unblock 信号。

来自 Commit 的 squash 信号：清空 fromDispatch、insts 队列、skidBuffer。

如果被 Execute 阻塞，调用 block()：将指令从 insts 队列移到 skidBuffer，设置 toDispatch 为 block。

解除 Blocked 或 Squashing 状态。

2.指令发出：

如果 issueStatus 为 Running、Idle、Unblocking，调用 issueInsts()：

根据 issueStatus 是否为 Unblocking，从 insts 队列或 skidBuffer 读取指令。

对于最多 issueWidth 条指令：

检查操作数是否准备好，如果准备好则发射指令到执行单元。

更新操作数状态和功能单元状态。

如果所有指令均已处理完毕，设置 issueStatus 为 Idle。

（三）、Execute（执行阶段）

目标：执行具体的操作

1. 信号检查：

checkSignalsAndUpdate()：

readStallSignals()：来自 Commit 的 block/unblock 信号。

来自 Commit 的 squash 信号：清空 fromIssue、insts 队列、skidBuffer。

如果被 Commit 阻塞，调用 block()：将指令从 insts 队列移到 skidBuffer，设置 toIssue 为 block。

解除 Blocked 或 Squashing 状态。

2. 指令执行：

如果 executeStatus 为 Running、Idle、Unblocking，调用 executeInsts()：

根据 executeStatus 是否为Unblocking,从 insts 队列或 skidBuffer 读取指令。

对于最多 executeWidth 条指令：

执行指令，包括 ALU 操作、加载/存储操作等。

更新目的寄存器的状态。

如果所有指令均已处理完毕，设置 executeStatus 为 Idle。

（四）、Writeback（写回阶段）

目标：将执行结果写回寄存器文件

1.结果写回：

将执行完成的指令结果写回到物理寄存器文件中，或更新内存（对于存储指令）。

2.释放资源：回收执行过程中使用的功能单元和寄存器资源，以便后续指令使用。

五、Commit（提交阶段）

目标：确认指令执行完毕并更新状态

1.信号检查

checkSignalsAndUpdate()：

readStallSignals()：来自其他阶段的 block/unblock 信号。

检查所有阶段的 squash 信号，执行必要的清理操作。

2.确定指令结果并更新资源

1.1 从 Retirement Queue（ROB）中选择已完成执行且所有数据依赖已解决的指令。

1.2 对于选中的指令，更新架构状态，包括修改寄存器文件、内存、以及状态标志（如零标志、进位标志等）。

1.3 之后，从 ROB 中移除这些指令，释放它们占用的条目和关联资源。这包括清除对物理寄存器和内存的预订，以及释放任何因指令执行而预留的其他资源。

3.处理异常和中断

处理分支预测错误和异常情况，如果有误预测或异常，进行恢复。

运行截图

build/X86/gem5.opt ./configs/learning_gem5/part1/two_level.py

使用 DerivO3CPU