IF端口

Inst_fifo模块

1. 时钟与复位

端口名	方向	位宽/类型	描述
`clk`	Input	`Clock()`	主时钟信号
`rst`	Input	`Bool()`	**同步复位**（高有效），清空所有寄存器和状态
`fifo_rst`	Input	`Bool()`	**异步复位**（高有效），用于分支预测错误时立即清空FIFO

---

2. 写入端口（输入）

端口名	方向	位宽	描述
`write_en1`	Input	`Bool()`	第一条指令写入使能
`write_addr1`	Input	`UInt(32)`	第一条指令的PC地址
`write_data1`	Input	`UInt(32)`	第一条指令的指令数据
`write_en2`	Input	`Bool()`	第二条指令写入使能
`write_addr2`	Input	`UInt(32)`	第二条指令的PC地址
`write_data2`	Input	`UInt(32)`	第二条指令的指令数据

---

3. 读取端口（输出）

端口名	方向	位宽	描述

`read_en1`	Input	`Bool()`	第一条指令读取使能
`read_data1`	Output	`UInt(32)`	输出的第一条指令数据（`read_en1` 有效且FIFO非空时更新）
`read_addr1`	Output	`UInt(32)`	输出的第一条指令PC地址
`read_en2`	Input	`Bool()`	第二条指令读取使能
`read_data2`	Output	`UInt(32)`	输出的第二条指令数据（需 `read_en2` 有效且FIFO至少有2条指令）
`read_addr2`	Output	`UInt(32)`	输出的第二条指令PC地址

---

4. 流控信号（输出）

端口名	方向	位宽	描述
`empty`	Output	`Bool()`	**高有效**，FIFO为空时置1
`almost_empty`	Output	`Bool()`	**高有效**，FIFO仅剩1条指令时置1
`full`	Output	`Bool()`	**高有效**，FIFO满时置1（用于反压上游取指单元）

---

5. 流水线控制（输入）

端口名	方向	位宽	描述
`i_stall`	Input	`Bool()`	流水线暂停信号（高有效），可能用于阻塞FIFO的读写操作

关键设计注明

1. **双端口操作**

   支持同时写入/读取两条指令，适应双发射流水线需求。

   写入时需确保 `write_en1` 先于 `write_en2` （`write_en2` 隐含依赖 `write_en1`）。

2. **状态信号**

   `full` 信号用于反压 `PCReg`，防止FIFO溢出。

   `almost_empty` 可优化下游模块的流水线阻塞逻辑。

3. **复位优先级**

   `fifo_rst` > `rst` > 正常操作，确保分支预测错误时能立即清空，对ID模块影响最小。

4. **地址循环**

   使用 `+%` 运算符自动处理指针回绕（如 `write_ptr +% 1.U`）。

5. **输出行为**

   读取端口为组合逻辑输出，数据随指针变化立即更新（无寄存器延迟）。

Pc_reg模块

1. 时钟与复位

端口名	方向	位宽/类型	描述
`clk`	Input	`Clock()`	主时钟信号
`rst`	Input	`Bool()`	**同步复位**（高有效），将PC重置为 `0xbfc00000`（启动地址）

2. PC 输入控制（输入）

端口名	方向	位宽	描述
`pc_en`	Input	`Bool()`	PC更新使能（高有效），若为低则冻结PC
`D_fifo_full`	Input	`Bool()`	指令FIFO满信号（高有效），反压时暂停PC更新
`F_inst_data_ok1`	Input	`Bool()`	第一条指令取指成功标志

`F_inst_data_ok2`	Input	`Bool()`	第二条指令取指成功标志

3. 跳转与异常处理（输入）

端口名	方向	位宽	描述
`E_bj`	Input	`Bool()`	**执行阶段**实际分支跳转信号（高有效）
`E_bj_target`	Input	`UInt(32)`	执行阶段计算出的分支目标地址
`D_bj`	Input	`Bool()`	**译码阶段**分支预测跳转信号（高有效）
`D_bj_target`	Input	`UInt(32)`	分支预测的目标地址
`M_except`	Input	`Bool()`	异常触发信号（高有效）
`M_except_addr`	Input	`UInt(32)`	异常处理入口地址
`M_flush_all`	Input	`Bool()`	流水线冲刷信号（高有效，如分支预测错误）
`M_flush_all_addr`	Input	`UInt(32)`	冲刷后重新取指的地址

4. PC 输出

端口名	方向	位宽	描述
`pc_curr`	Output	`UInt(32)`	当前周期PC值（寄存器的直接输出）
`pc_next`	Output	`UInt(32)`	下一周期PC值（组合逻辑计算出的预更新值）

端口优先级与更新逻辑

PC更新按以下优先级顺序（从高到低）：

1. **复位**（`rst`）→ `0xbfc00000`
2. **异常**（`M_except`）→ `M_except_addr`
3. **流水线冲刷**（`M_flush_all`）→ `M_flush_all_addr`
4. **执行阶段分支**（`E_bj`）→ `E_bj_target`（修正预测错误）

5. **译码阶段预测**（`D_bj`）→ `D_bj_target`
6. **FIFO反压**（`D_fifo_full`）→ 保持当前PC
7. **正常取指**：
   - 双指令成功（`F_inst_data_ok1 && F_inst_data_ok2`）→ `pc_curr + 8`
   - 单指令成功（`F_inst_data_ok1`）→ `pc_curr + 4`
   - 默认 → 保持当前PC

关键设计注明

1. **分支预测与修正**

   `D_bj` 来自BTB预测，`E_bj` 来自执行单元实际结果，后者优先级更高。

   预测错误时通过 `M_flush_all` 冲刷流水线，并更新PC至正确地址。

2. **取指流控**

   `D_fifo_full` 为高时，PC冻结以避免FIFO溢出。

   双指令取指成功时PC+8，单指令成功时PC+4。

3. **输出特性**

   `pc_curr` 为当前周期PC值（寄存器输出），`pc_next` 为预计算的下一周期值（组合逻辑）。

4. **复位地址**

   固定为 `0xbfc00000`（典型MIPS架构启动地址），不可配置。