2024 年春季学期计算机组成原理课程

32 位 LoongArch流水线CPU

设计报告

# 何颜彤 李昀樵 黄铿岚 赵严昊天

班级：2022 级计算机二班 2024 年 5月29 日

一、 项目简述

## 项目环境与级别

设计语言：Verilog 硬件描述语言

仿真环境：Vivado 2023.2版本

## 任务目标

设计目标：基于32位LoongArch精简指令集的5级流水线CPU

指令集： 32位LoongArch精简指令集

设计目标：

1、实现兼容32RISC-V指令集的流水线CPU

2、实现的指令需覆盖所有对应MIPS指令中R型、I型计算类、I型存数类、I型取数类、I型条件判断类、J型中的指令。

3、能够运行简单的小程序，且充分利用所实现的指令。

## 参考资料

1、龙芯架构32位精简版参考手册

2、课程PPT（指导老师：何安平）

## 项目文件夹结构

流水线cpu设计

|——架构图.vsdx

|——设计报告.docx

|——<src>

|——cpu.zip #工程文件

二、项目内容

1. 指令集选取

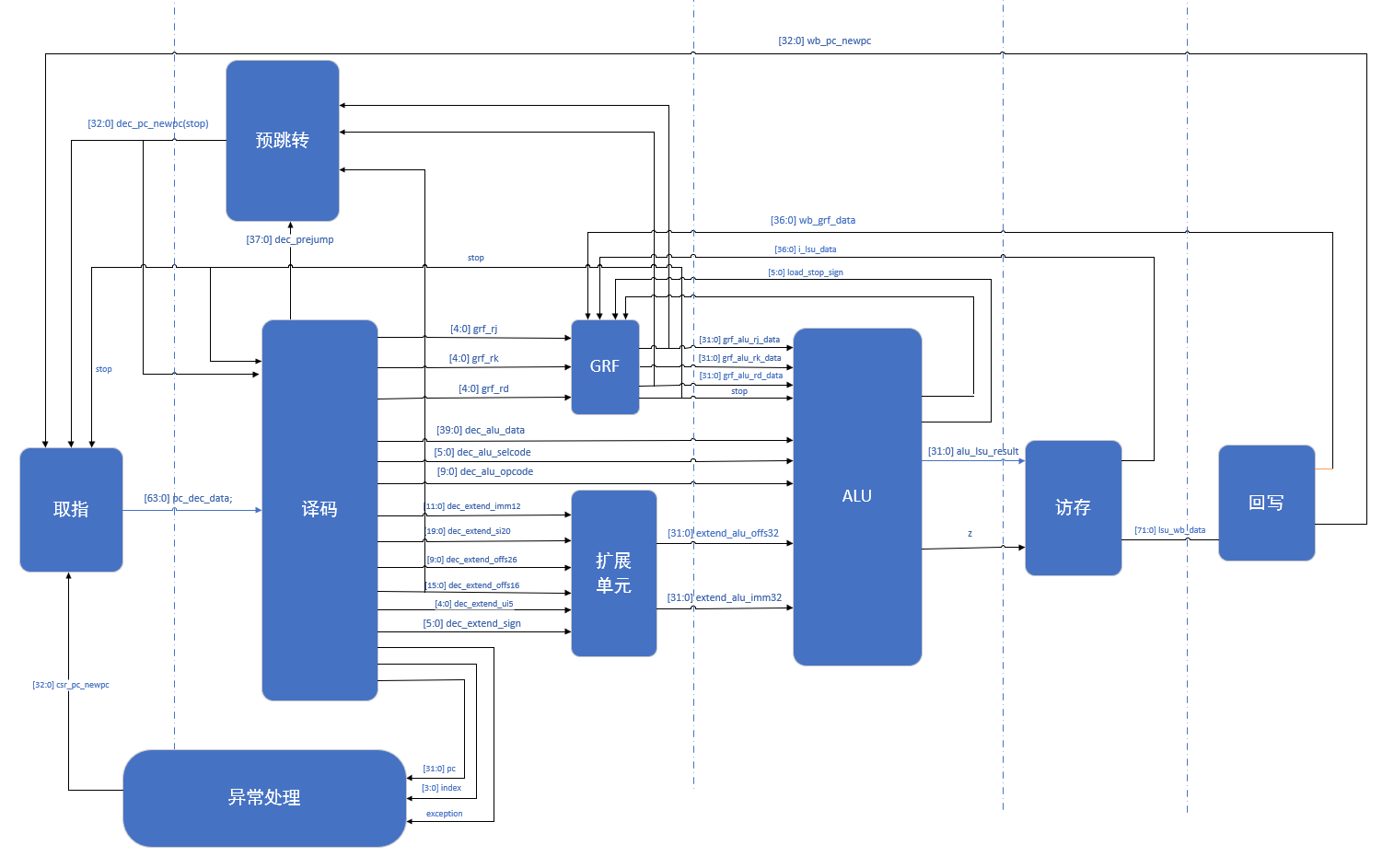
本次设计实现了LoongArch 32位精简指令集中基础整数指令的子项：算数运算指令、移位运算指令、转移指令和普通访存指令（仅LD.W和ST.W）

本次设计共实现35条指令，具体所实现的指令如下图所示：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **助记符** | **指令格式** | | | | **操作及其意义** |
|  | 31 15 | 14 10 | 9 5 | 4 0 |  |
| ADD.W | 00000000000100000 | rk | rj | rd | GR[rd] = GR[rj] + GR[rk] |
| SUB.W | 00000000000100010 | rk | rj | rd | GR[rd] = GR[rj] - GR[rk] |
| SLL.W | 00000000000101110 | rk | rj | rd | GR[rd] = SLL(GR[rj], GR[rk][4:0]) |
| SRL.W | 00000000000101111 | rk | rj | rd | GR[rd] = SRL(GR[rj], GR[rk][4:0]) |
| SRA.W | 00000000000110000 | rk | rj | rd | GR[rd] = SRA(GR[rj], GR[rk][4:0]) |
| AND | 00000000000101001 | rk | rj | rd | GR[rd] = GR[rj] & GR[rk] |
| OR | 00000000000101010 | rk | rj | rd | GR[rd] = GR[rj] | GR[rk] |
| XOR | 00000000000101011 | rk | rj | rd | GR[rd] = GR[rj] ^ GR[rk] |
| SLT | 00000000000100100 | rk | rj | rd | GR[rd] = (signed(GR[rj]) < signed(GR[rk])) ? 1 : 0 |
| SLTU | 00000000000100101 | rk | rj | rd | GR[rd] = (unsigned(GR[rj]) < unsigned(GR[rk])) ? 1 : 0 |
| SLLI.W | 00000000010000001 | ui5 | rj | rd | GR[rd] = SLL(GR[rj], ui5) |
| SRLI.W | 00000000010001001 | ui5 | rj | rd | GR[rd]= SRL(GR[rj], ui5) |
| SRAI.W | 00000000010010001 | ui5 | rj | rd | GR[rd]= SRA(GR[rj], ui5) |
| SYSCALL | 00000000001010110 | 00000 | 00000 | 00000 | 无条件的触发系统调用例外 |
| BREAK | 00000000001010100 | 00000 | 00000 | 00000 | 无条件的触发断点例外 |

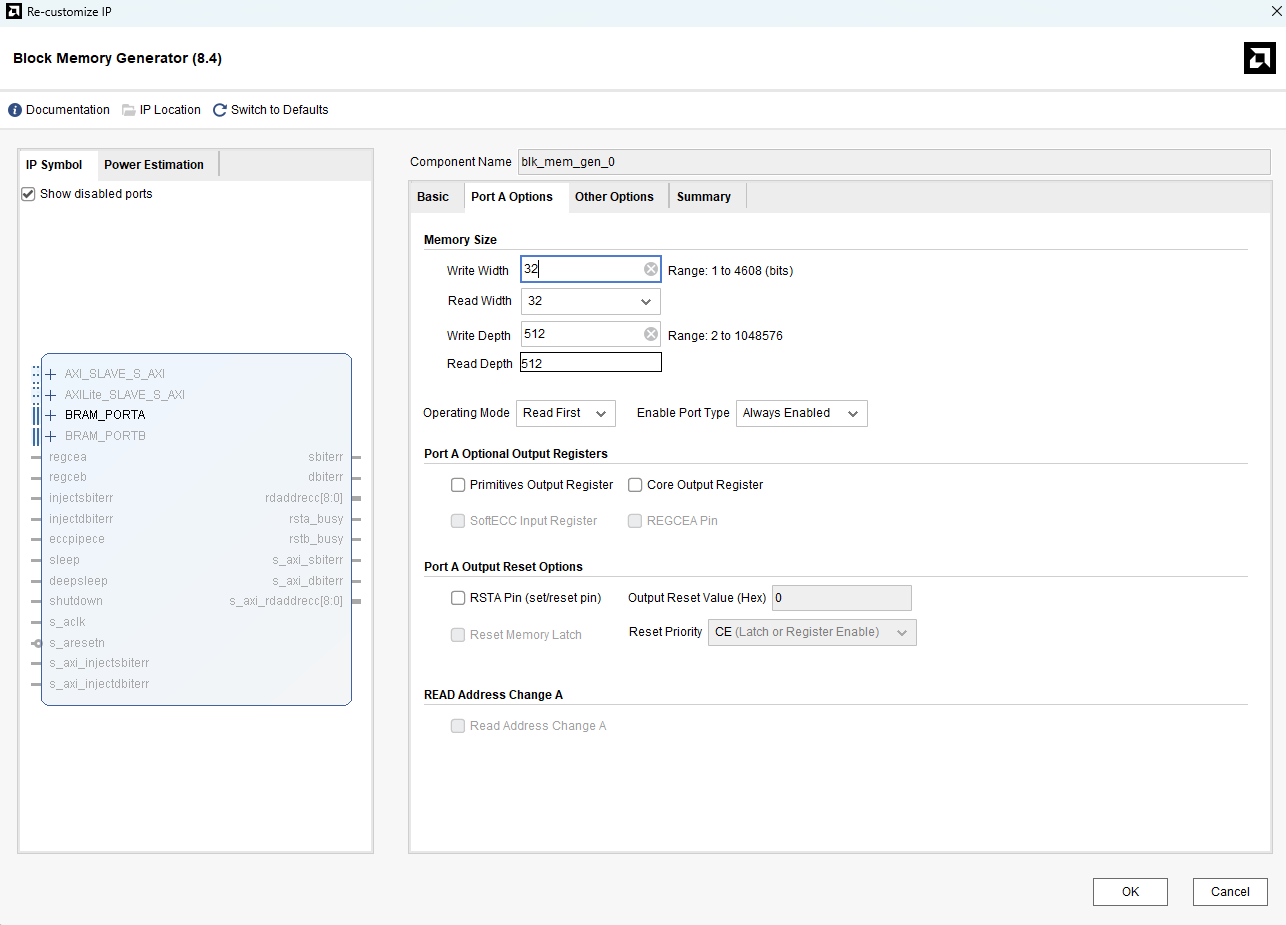
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 31 22 | 21 10 | 9 5 | 4 0 | |  |
| ADDI.W | 0000001010 | si12 | rj | rd | | GR[rd]=GR[rj] + SignExtend(si12, 32) |
| ANDI | 0000001101 | ui12 | rj | rd | | GR[rd] = GR[rj] & ZeroExtend(ui12, 32) |
| ORI | 0000001110 | ui12 | rj | rd | | GR[rd] = GR[rj] | ZeroExtend(ui12, 32) |
| XORI | 0000001111 | ui12 | rj | rd | | GR[rd] = GR[rj] ^ ZeroExtend(ui12, 32) |
| SLTI | 0000001000 | si12 | rj | rd | | tmp = SignExtend(si12, 32)  GR[rd] = (signed(GR[rj]) < signed(tmp)) ? 1 : 0 |
| SLTUI | 0000001001 | si12 | rj | rd | | tmp = SignExtend(si12, 32)  GR[rd] = (unsigned(GR[rj]) < unsigned(tmp)) ? 1 : 0 |
|  | 31 26 | 25 10 | 9 0 | | |  |
| B | 010100 | offs[15:0] | offs[25:16] | | | PC = PC + SignExtend({offs26, 2'b0}, 32) |
| BL | 010101 | offs[15:0] | offs[25:16] | | | GR[1] = PC + 4  PC = PC + SignExtend({offs26, 2'b0}, 32) |
|  | 31 26 | 25 10 | 9 5 | | 4 0 |  |
| BEQ | 010110 | offs[15:0] | rj | rd | | if GR[rj]==GR[rd] :  PC = PC + SignExtend({offs16, 2'b0}, 32) |
| BNE | 010111 | offs[15:0] | rj | rd | | if GR[rj]!=GR[rd] :  PC = PC + SignExtend({offs16, 2'b0}, 32) |
| BLT | 011000 | offs[15:0] | rj | rd | | if signed(GR[rj]) < signed(GR[rd]) :  PC = PC + SignExtend({offs16, 2'b0}, 32) |
| BGE | 011001 | offs[15:0] | rj | rd | | if signed(GR[rj]) >= signed(GR[rd]) :  PC = PC + SignExtend({offs16, 2'b0}, 32) |
| BLTU | 011010 | offs[15:0] | rj | rd | | if unsigned(GR[rj]) < unsigned(GR[rd]) :  PC = PC + SignExtend({offs16, 2'b0}, 32) |
| BGEU | 011011 | offs[15:0] | rj | rd | | if unsigned(GR[rj]) >= unsigned(GR[rd]) :  PC = PC + SignExtend({offs16, 2'b0}, 32) |
| JIRL | 010011 | offs[15:0] | rj | rd | | GR[rd] = PC + 4  PC = GR[rj] + SignExtend({offs16, 2'b0}, 32) |
|  | 31 22 | 21 10 | 9 5 | 4 0 | |  |
| LD.W | 0010100010 | si12 | rj | rd | | vaddr = GR[rj] + SignExtend(si12, 32)  AddressComplianceCheck(vaddr)  paddr = AddressTranslation(vaddr)  word = MemoryLoad(paddr, WORD)  GR[rd] = word |
| ST.W | 0010100110 | si12 | rj | rd | | vaddr = GR[rj] + SignExtend(si12, 32)  AddressComplianceCheck(vaddr)  paddr = AddressTranslation(vaddr)  MemoryStore(GR[rd][31:0], paddr, WORD) |
|  | 31 25 | 24 5 | | 4 0 | |  |
| LU12I.W | 0001010 | si20 | | rd | | GR[rd] = {si20, 12'b0} |
| PCADDI12I | 0001110 | si20 | | rd | | GR[rd] = PC + SignExtend({si20, 12'b0}, 32) |
|  | 31 0 | | | | |  |
| ERTN | 00000110010010000011100000000000 | | | | | 异常返回指令 |

2.处理器架构

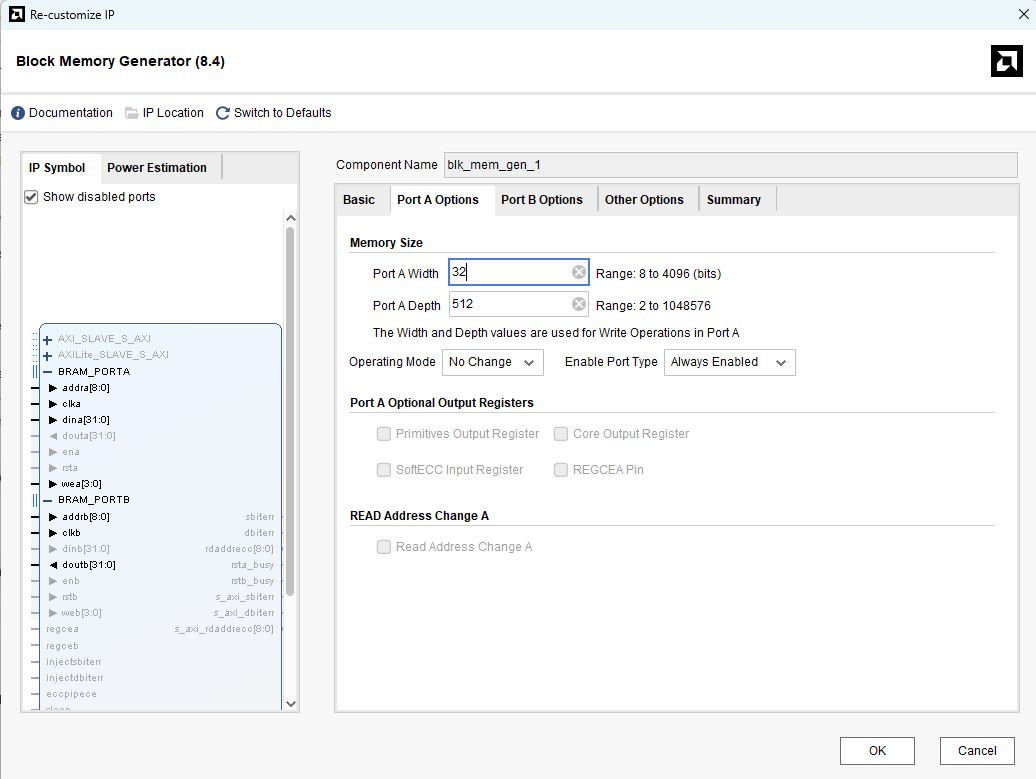


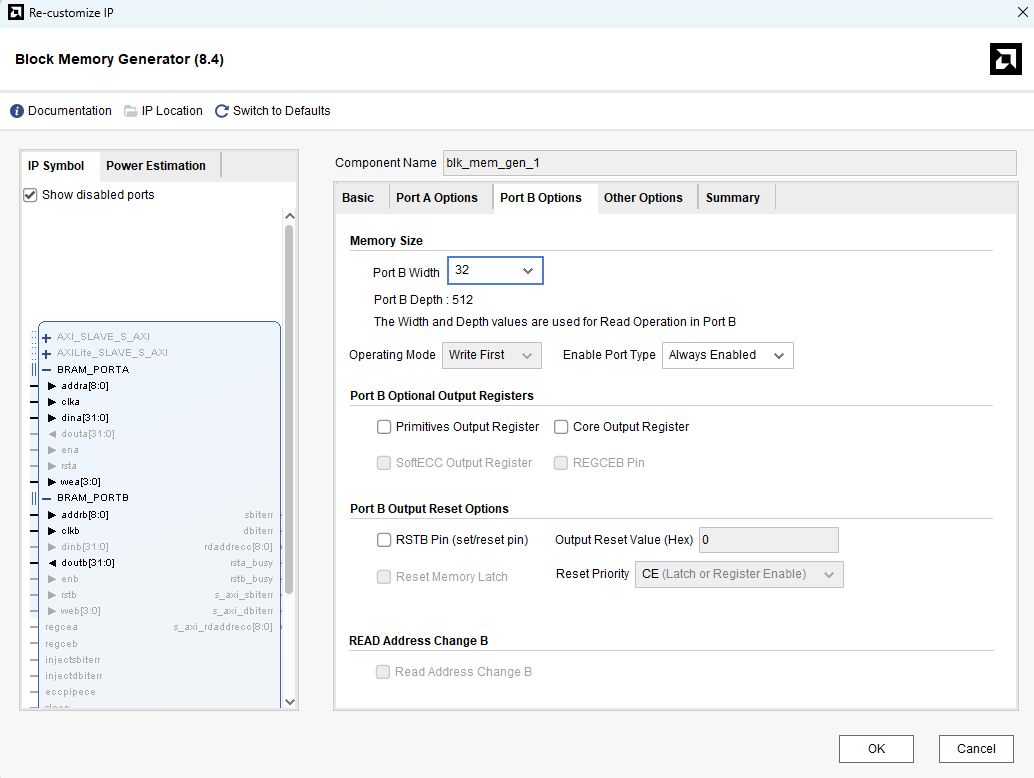
3.模块设计

1. 存储器类
2. 指令存储器采用vivado ip核提供的Bram实现，指令储存器的配置如下图：



1. 数据存储器采用vivado ip核提供的Bram实现，数据储存器的配置如下图：

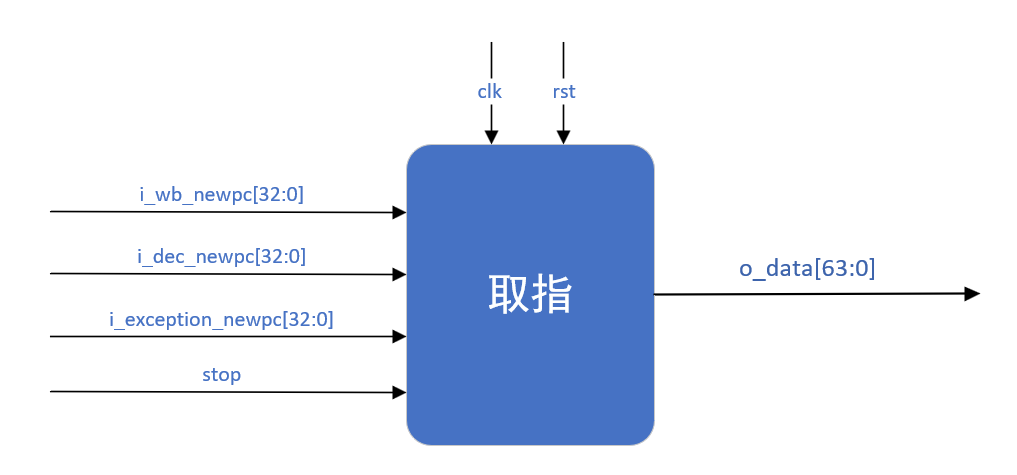




1. 取指模块

取指模块连接多个模块，接收跳转数据。每个时钟周期到来时从指令存储器中取出一条指令并预译码，识别到指令B时直接计算跳转地址跳转，识别到BL、JIRL指令时启动定时器生成阻塞信号，直到BL、JIRL指令完成跳转，其余情况直接向下级流水发送数据。

取指模块如图所示：

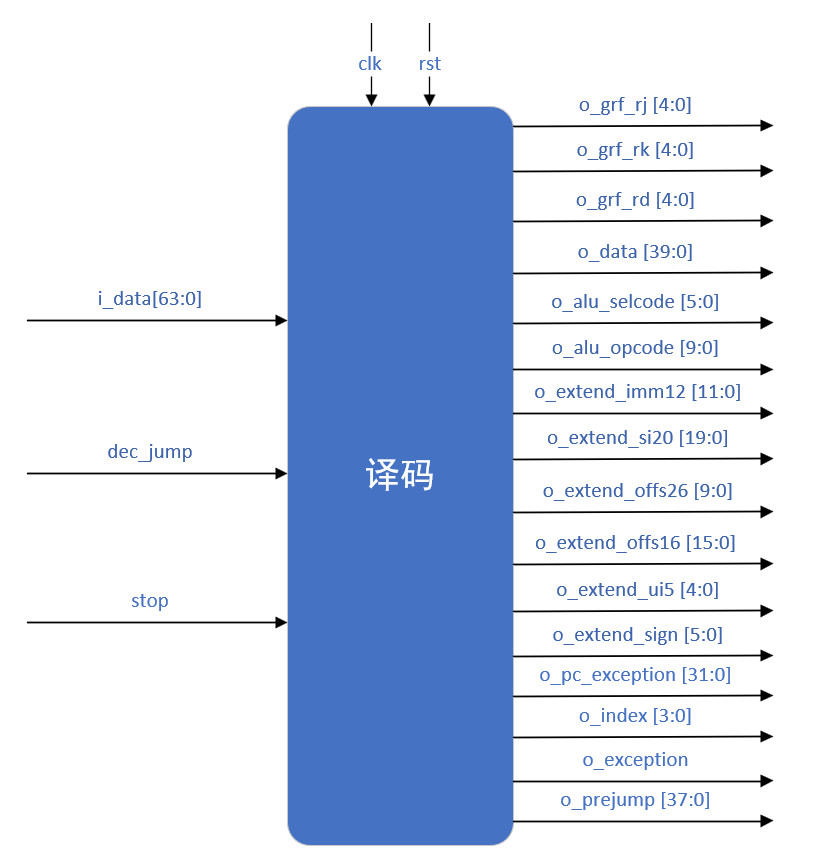


|  |  |  |  |
| --- | --- | --- | --- |
| input | clk | 1 | 时钟信号 |
| input | rst | 1 | 复位信号 |
| input | i\_wb\_newpc | 33 | 写回跳转数据 |
| input | i\_dec\_newpc | 33 | 译码跳转数据 |
| input | i\_exception | 33 | 异常处理跳转数据 |
| input | stop | 1 | 阻塞信号 |
| output | o\_data | 64 | 输出数据 |

1. 译码模块

译码模块模块接收到取指模块发送的数据后识别指令，如果识别到非法指令或者异常指令则发送信号给异常处理模块，跳转到对应的异常处理程序进行处理；如果识别为条件跳转指令，则是发送数据给预跳转模块，由预跳转模块判断是否跳转，条件跳转指令在译码流水阶段即可完成指令生命周期；其他合法指令则正常按照流水级执行。

译码模块如图所示：

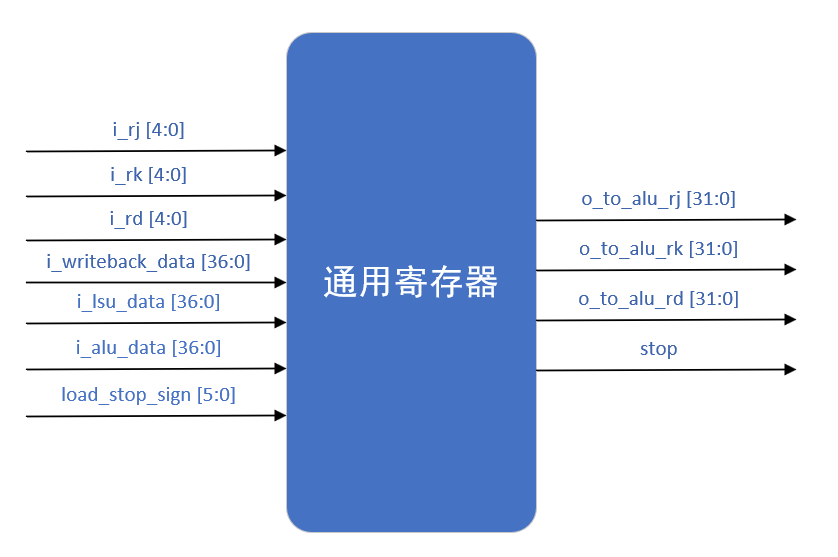


|  |  |  |  |
| --- | --- | --- | --- |
| input | clk | 1 | 时钟信号 |
| input | rst | 1 | 复位信号 |
| input | stop | 1 | 暂停信号 |
| input | dec\_jump | 1 | 译码跳转信号 |
| input | i\_data | 64 | 上级流水线数据包 |
| output | o\_grf\_rj | 5 | 寄存器编号 |
| output | o\_grf\_rk | 5 | 寄存器编号 |
| output | o\_grf\_rd | 5 | 寄存器编号 |
| output | o\_alu\_opcode | 10 | ALU操作码 |
| output | o\_alu\_selcode | 6 | ALU数据选择码 |
| output | o\_extend\_imm12 | 12 | 12位立即数 |
| output | o\_extend\_sign | 6 | 扩展单元数据选择信号 |
| output | o\_extend\_si20 | 20 | 20位立即数 |
| output | o\_extend\_offs16 | 16 | 16位偏移量 |
| output | o\_extend\_offs26 | 10 | 26位偏移量的高10位 |
| output | o\_data | 40 | 输出流水级数据包 |
| output | o\_prejump | 38 | 预跳转数据 |

1. 通用寄存器组（GRF）

处理器中按照指令集架构规定实现了32个通用寄存器，其中0号寄存器的值恒为0。GRF连接ALU和LSU的数据旁路，以解决“写后读”的数据冒险；同时为解决Load指令后一条需要用到Load指令取出数据的情况，当此时ALU中指令为Load型指令时，ALU会发出信号和该条Load指令的目的寄存器号给GRF，由GRF判断此时的译码流水级中指令与该条Load指令是否存在相关性，如果存在则产生stop信号阻塞前两个流水级一个周期，不存在则正常执行。

通用寄存器组如图所示：

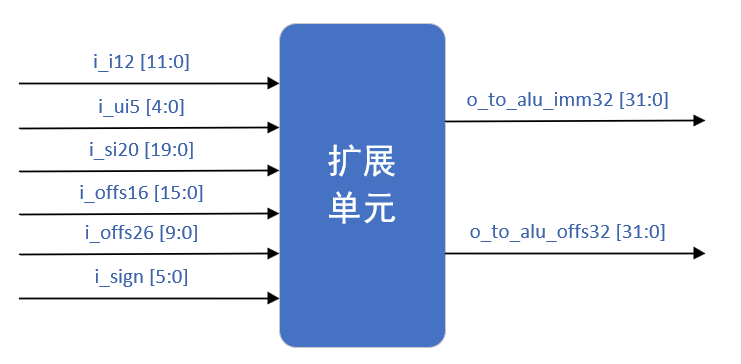


|  |  |  |  |
| --- | --- | --- | --- |
| input | clk | 1 | 时钟信号 |
| input | rst | 1 | 复位信号 |
| input | i\_rj | 5 | 寄存器编号 |
| input | i\_rk | 5 | 寄存器编号 |
| input | i\_rd | 5 | 寄存器编号 |
| input | i\_writeback\_data | 37 | 写回寄存器更改数据 |
| input | i\_lsu\_data | 37 | LSU旁路数据 |
| input | i\_alu\_data | 37 | ALU旁路数据 |
| input | load\_stop\_sign | 6 | load指令特殊情况信号 |
| output | o\_to\_alu\_rj | 32 | 寄存器数据 |
| output | o\_to\_alu\_rk | 32 | 寄存器数据 |
| output | o\_to\_alu\_rd | 32 | 寄存器数据 |
| output | stop | 1 | 特殊情况阻塞信号 |

1. 扩展单元（extend）

扩展单元将立即数和偏移量扩展为32位，根据译码模块的选择信号选择出对应的扩展数据。

扩展单元如图所示：

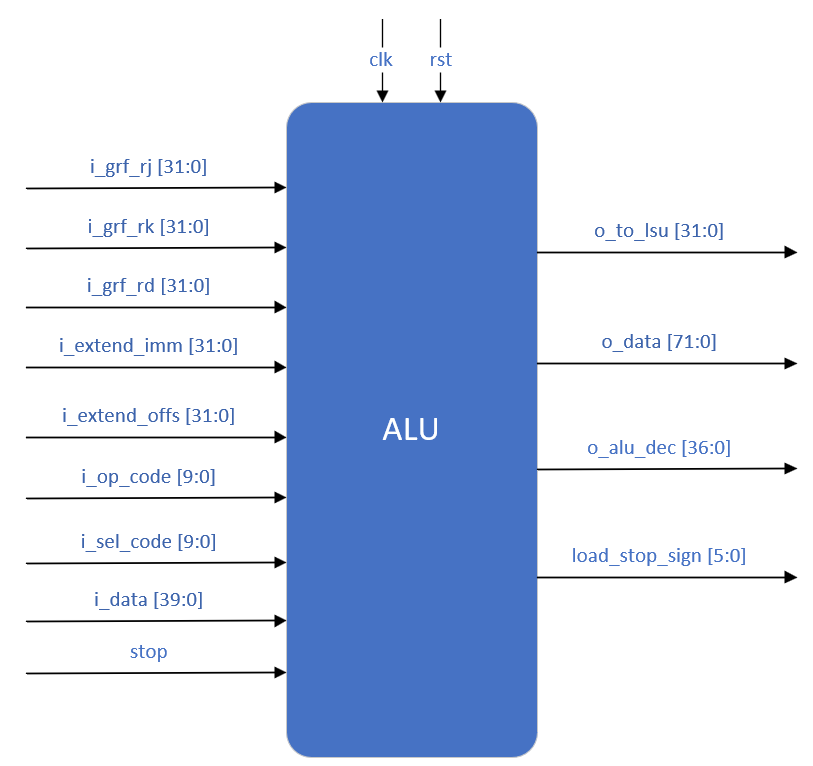


|  |  |  |  |
| --- | --- | --- | --- |
| input | i\_i12 | 12 | 12位立即数 |
| input | i\_ui5 | 5 | 5位无符号立即数 |
| input | i\_si20 | 20 | 20位立即数 |
| input | i\_offs16 | 16 | 16位偏移量 |
| input | i\_offs26 | 10 | 26位偏移量高10位 |
| input | i\_sign | 6 | 数据选择信号 |
| output | o\_to\_alu\_imm32 | 32 | 32位立即数输出 |
| output | o\_to\_alu\_offs32 | 32 | 32位偏移量输出 |

1. ALU模块

ALU模块根据译码模块发送的数据选择信号和操作信号输出对应的结果。

ALU模块如图所示：

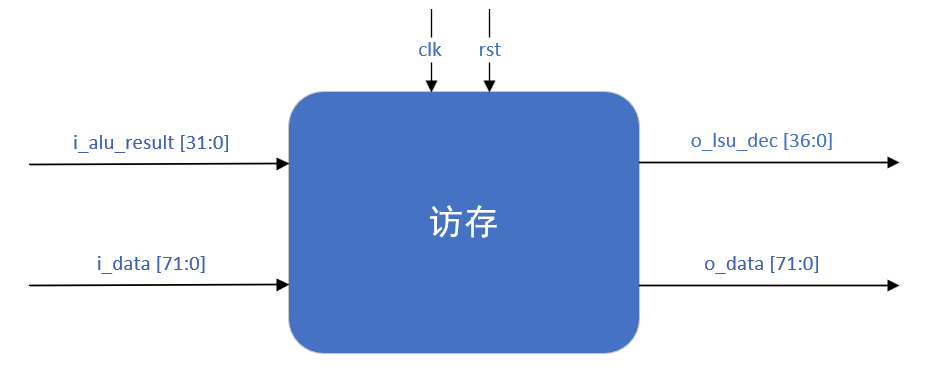


|  |  |  |  |
| --- | --- | --- | --- |
| input | clk | 1 | 时钟信号 |
| input | rst | 1 | 复位信号 |
| input | i\_grf\_rj | 32 | 寄存器数据 |
| input | i\_grf\_rk | 32 | 寄存器数据 |
| input | i\_grf\_rd | 32 | 寄存器数据 |
| input | i\_extend\_imm | 32 | 32位立即数 |
| input | i\_extend\_offs | 32 | 32位偏移量 |
| input | i\_op\_code | 10 | 操作码 |
| input | i\_sel\_code | 6 | 数据选择码 |
| input | i\_data | 40 | 级间寄存器数据包 |
| output | o\_to\_lsu | 32 | ALU计算结果 |
| output | o\_data | 72 | 级间寄存器数据包 |
| output | o\_alu\_dec | 37 | ALU数据旁路 |
| output | load\_stop\_sign | 6 | load指令特殊情况信号 |

1. LSU模块

LSU访存单元通过级间寄存器中的标识信号判断是否为Load指令和Store指令，是则执行对应的访存操作，此时ALU计算结果为访存地址；不是则空操作，此时ALU计算结果为算数指令的寄存器结果。

LSU单元如图所示：

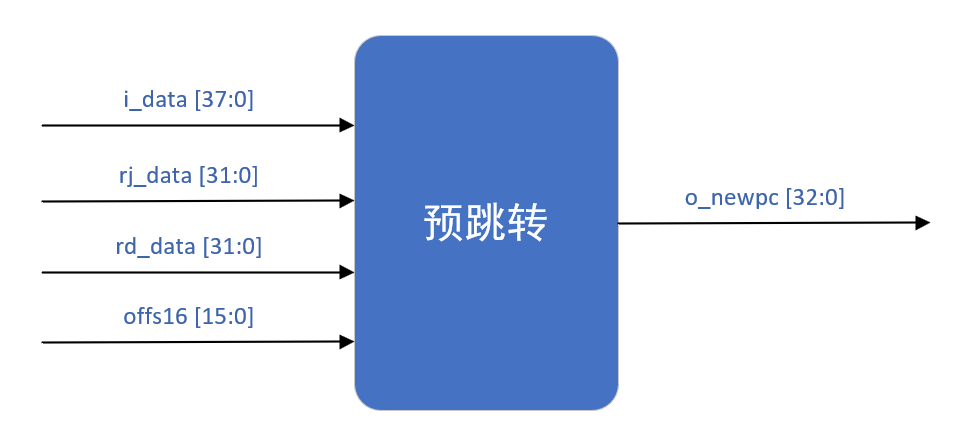


|  |  |  |  |
| --- | --- | --- | --- |
| input | clk | 1 | 时钟信号 |
| input | rst | 1 | 复位信号 |
| input | i\_alu\_result | 32 | ALU计算结果 |
| input | i\_data | 72 | 级间寄存器数据包 |
| output | o\_data | 72 | 级间寄存器数据包 |
| output | o\_lsu\_dec | 37 | LSU数据旁路 |

1. 预跳转单元（prejump）

预跳转单元判断是否进行跳转，同时计算跳转地址，发送给取指模块。

预跳转单元如图所示：

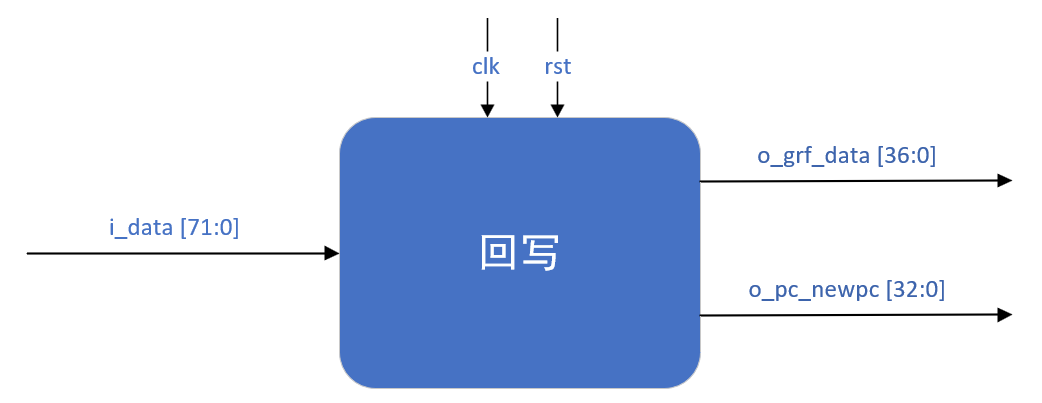


|  |  |  |  |
| --- | --- | --- | --- |
| input | i\_data | 38 | 预跳转数据 |
| input | rj\_data | 32 | 寄存器数据 |
| input | rd\_data | 32 | 寄存器数据 |
| input | offs16 | 16 | 16位偏移量 |
| output | o\_newpc | 33 | 跳转数据 |

1. 回写模块

回写模块，将结果写回到寄存器，并更新PC。

回写模块如图所示：

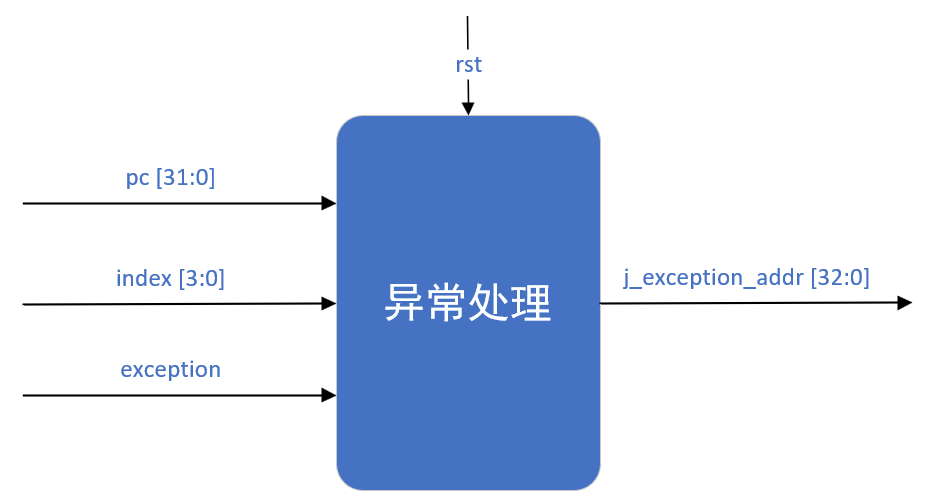


|  |  |  |  |
| --- | --- | --- | --- |
| input | clk | 1 | 时钟信号 |
| input | rst | 1 | 复位信号 |
| input | i\_data | 72 | 级间寄存器数据包 |
| output | o\_grf\_data | 37 | 寄存器更改数据 |
| output | o\_pc\_newpc | 33 | 跳转地址数据 |

1. 异常处理模块（CSR）

异常处理模块根据译码模块发送的异常类型，发送对应的异常跳转地址，跳转到内存中对应的异常处理程序执行，执行结束后恢复地址。

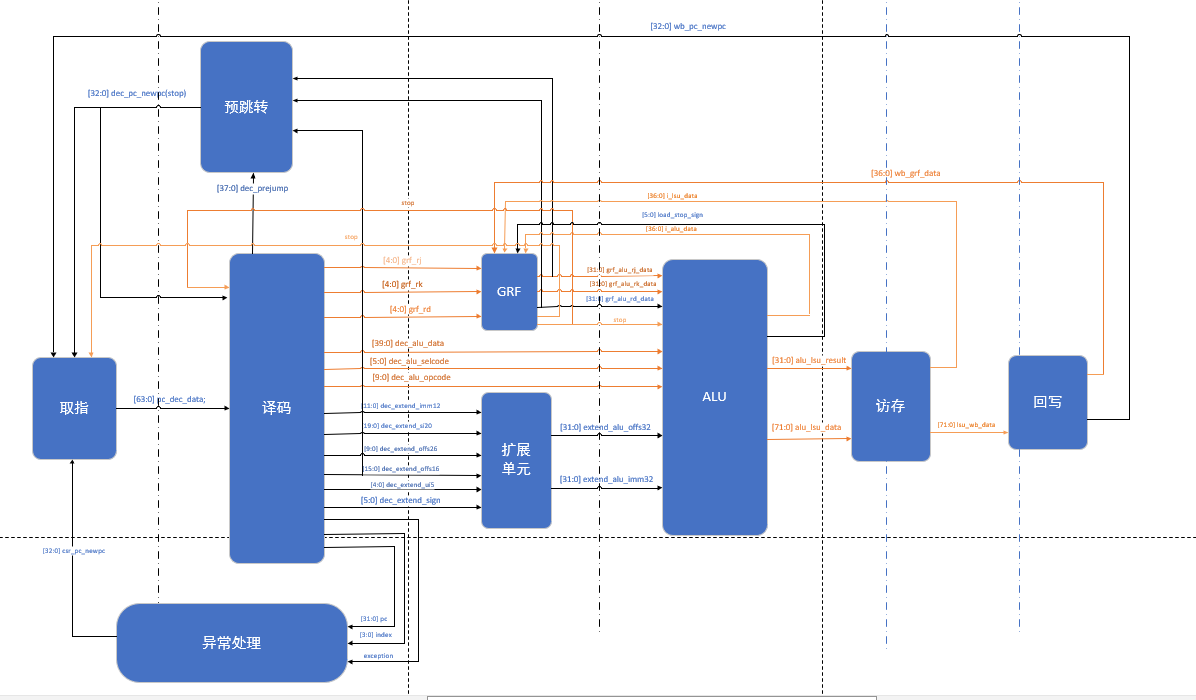
异常处理模块如图所示：



|  |  |  |  |
| --- | --- | --- | --- |
| input | rst | 1 | 复位信号 |
| input | exception | 1 | 异常信号 |
| input | index | 4 | 异常类型 |
| input | pc | 32 | 触发异常的指令地址 |
| output | j\_exception\_addr | 33 | 异常跳转地址 |

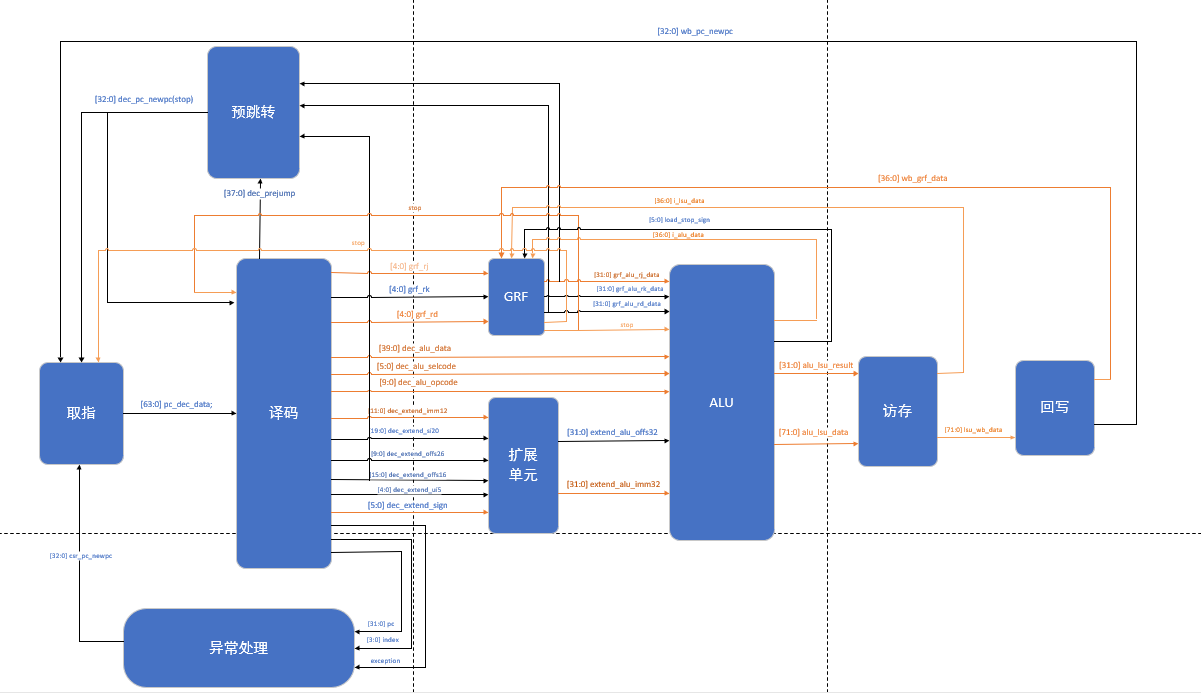
4. 数据通路

（1）R型数据通路

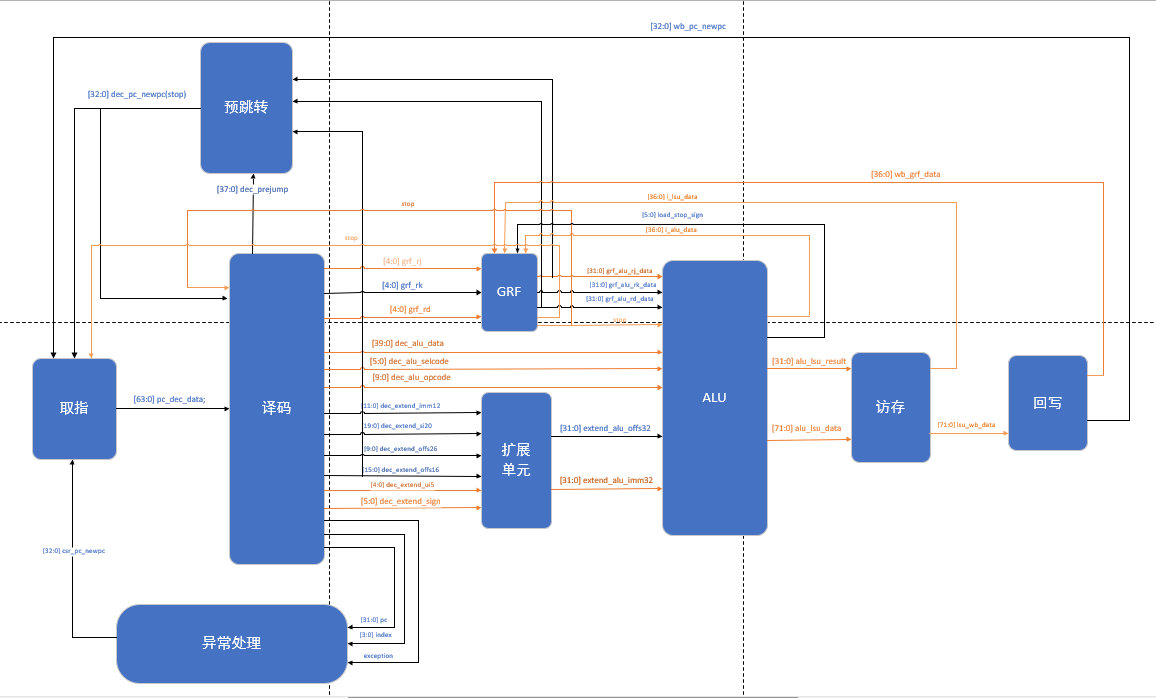


1. I型数据通路

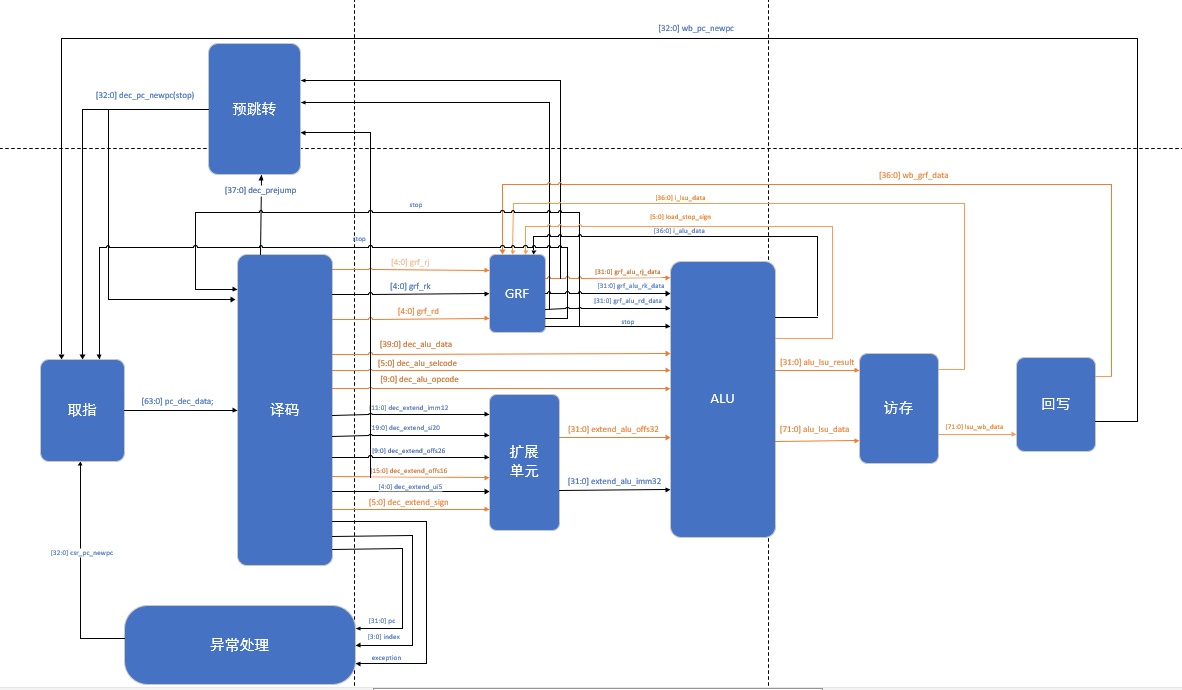
12位立即数



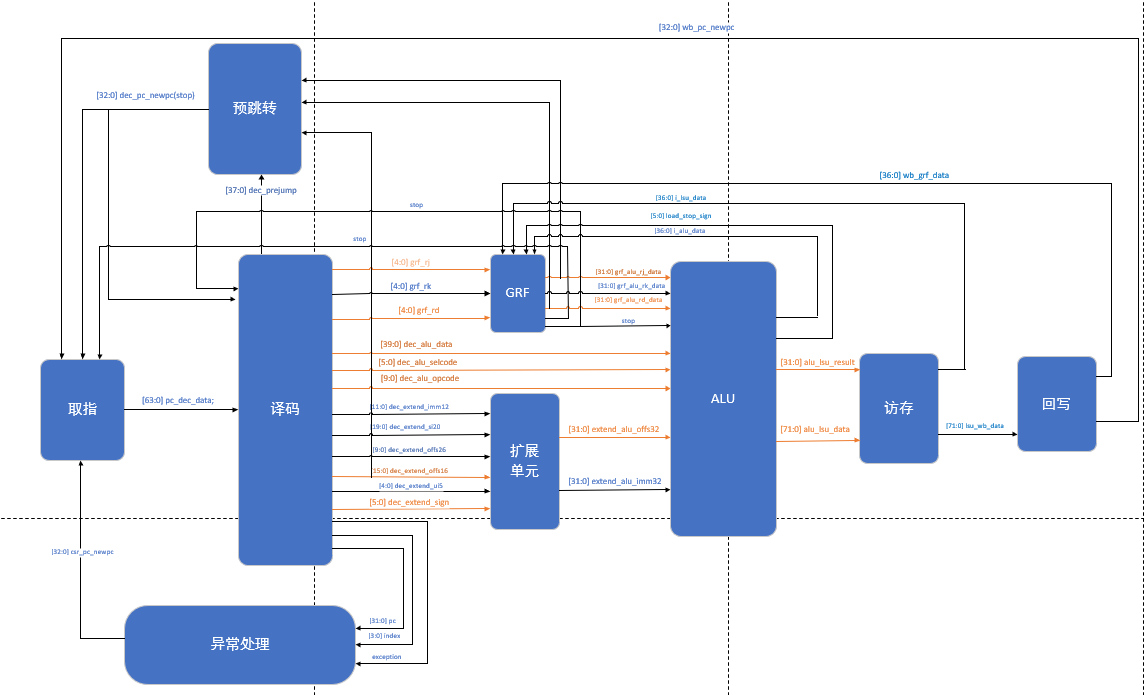
5位立即数



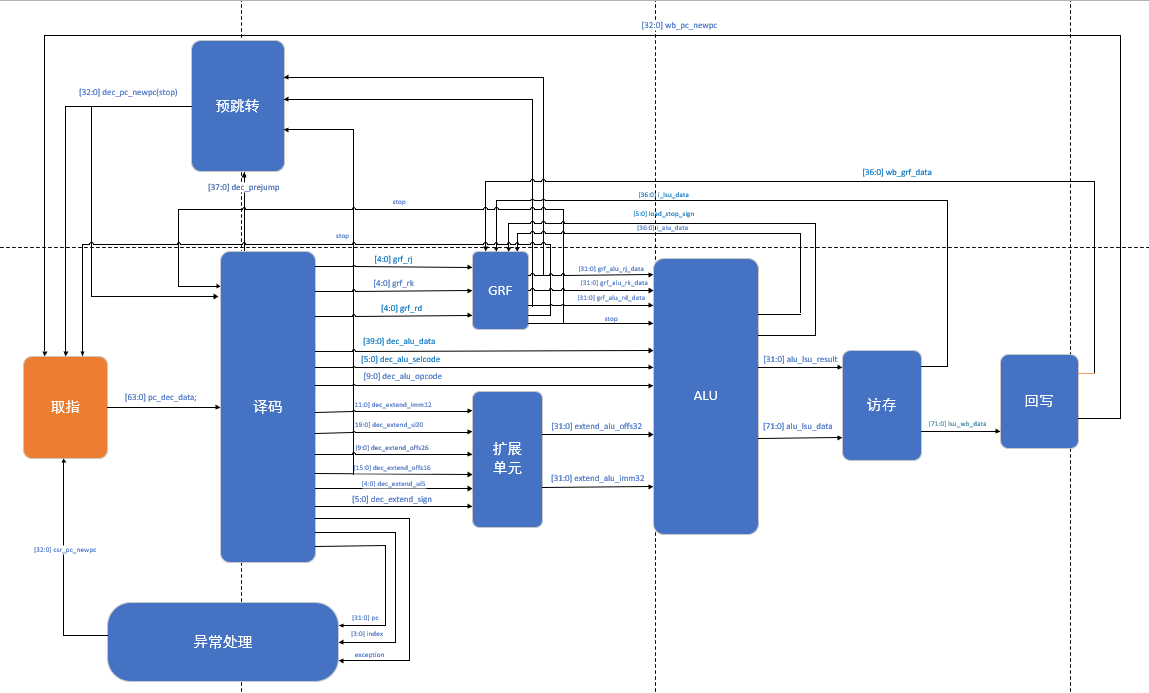
（3）LD指令数据通路



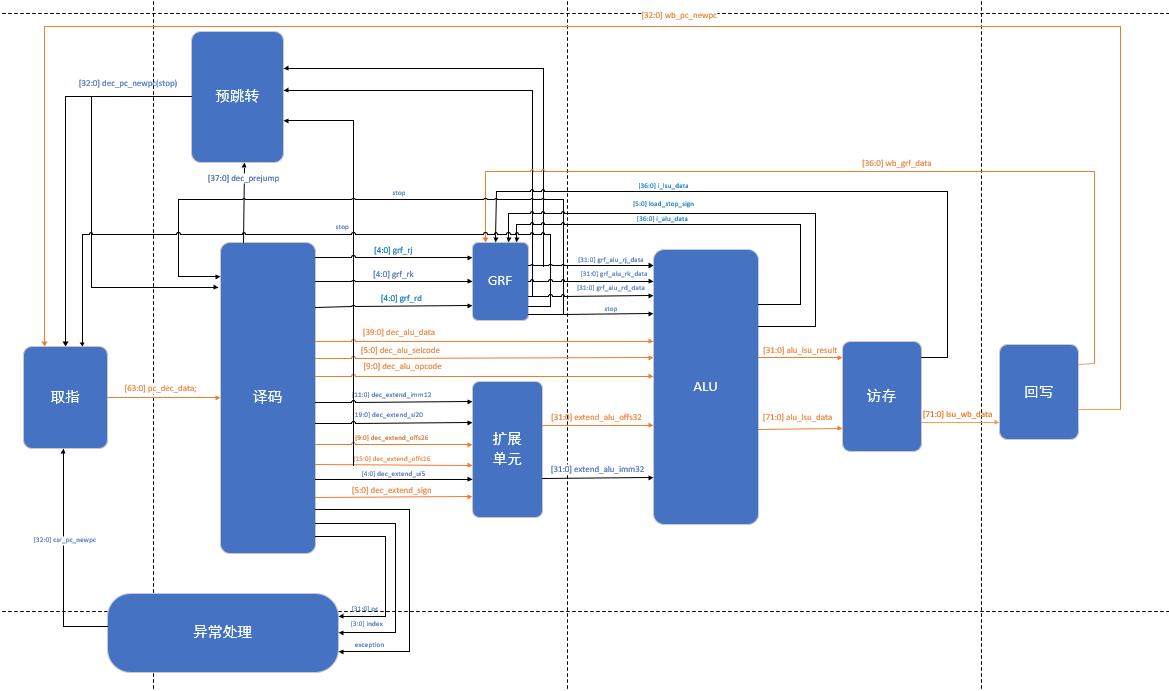
（4）ST指令数据通路



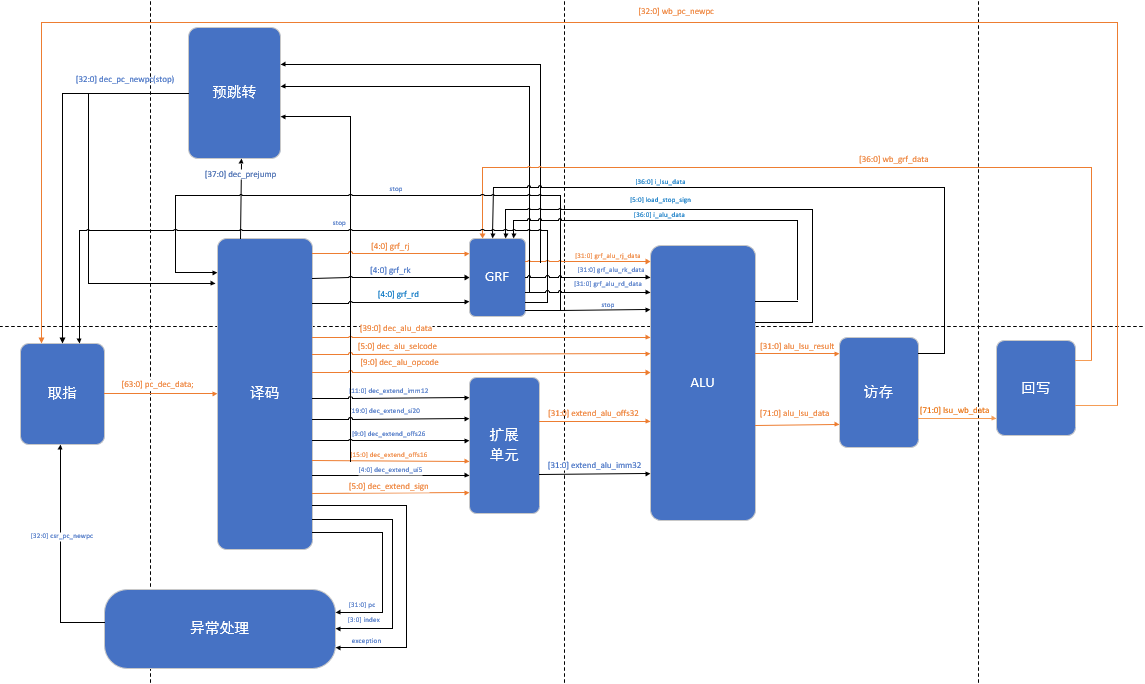
1. B指令数据通路



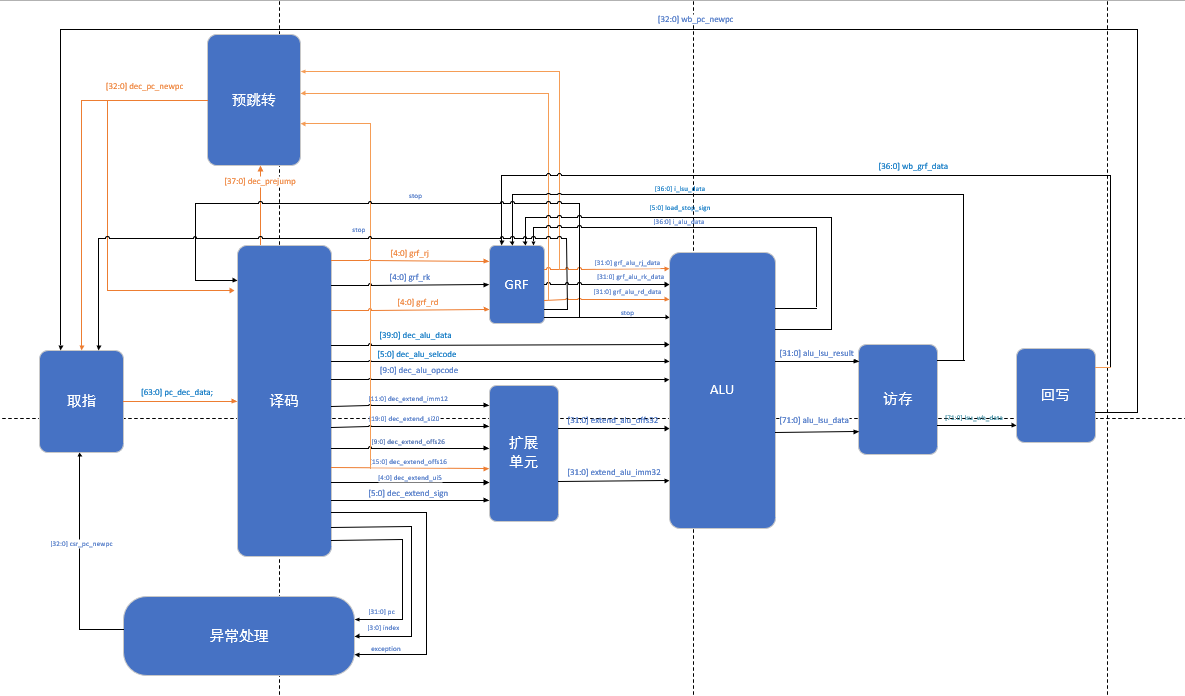
1. BL指令数据通路



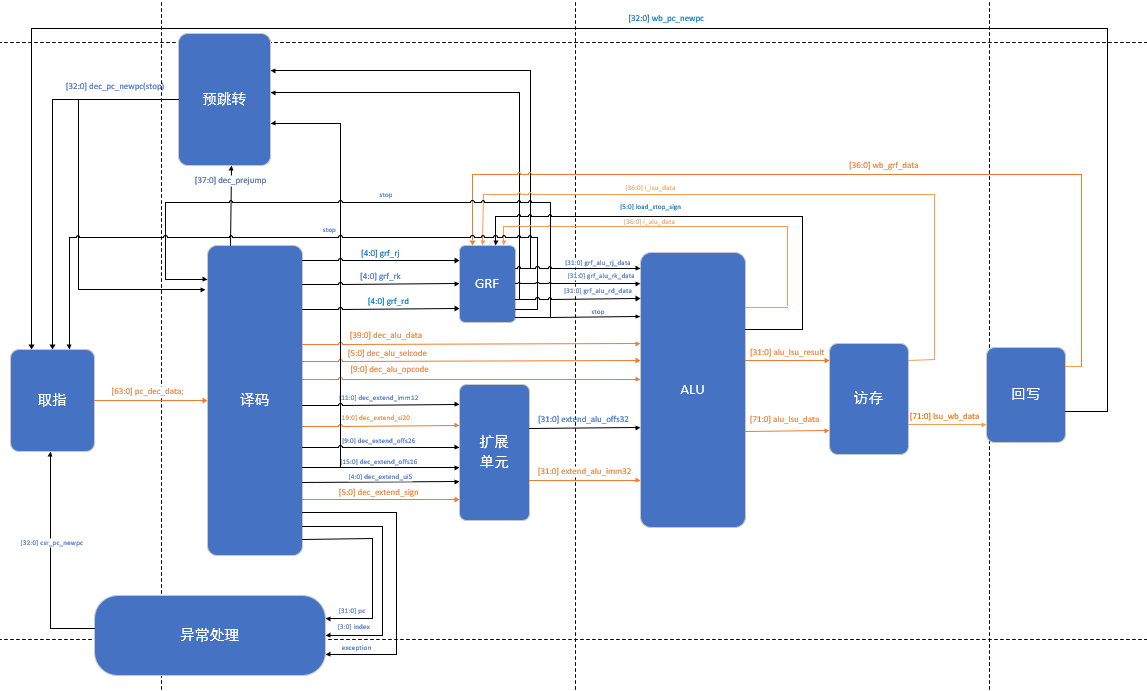
1. JIRL指令数据通路



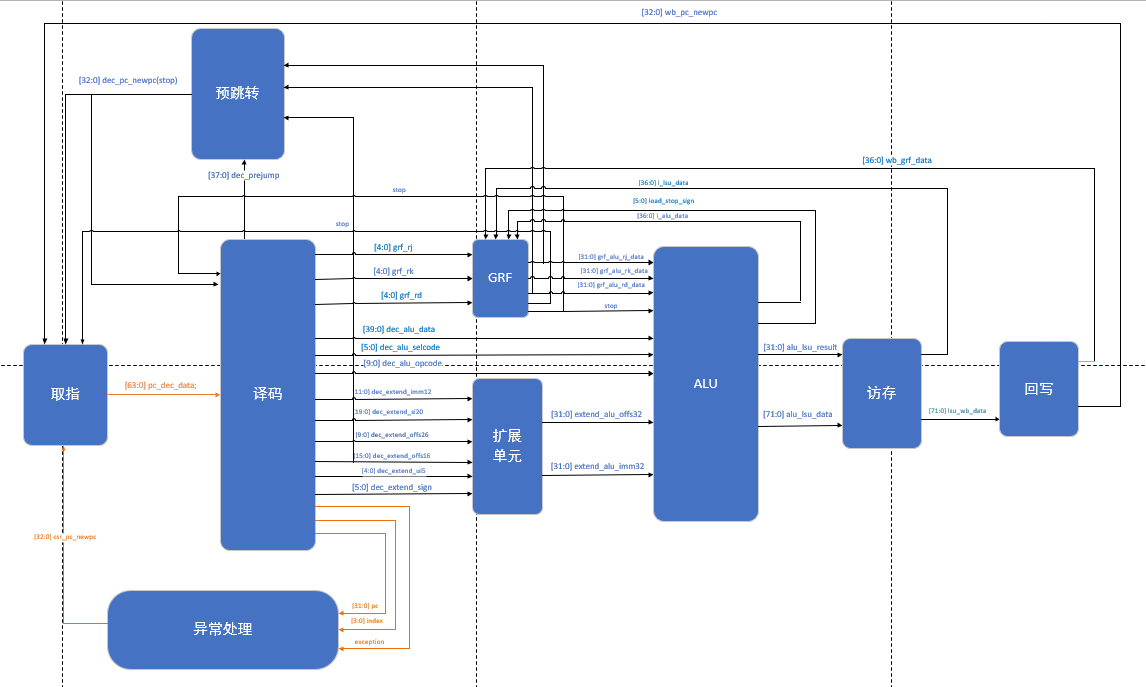
（8）BNE、BEQ、BLT、BGE、BLTU、BEGU令数据通路



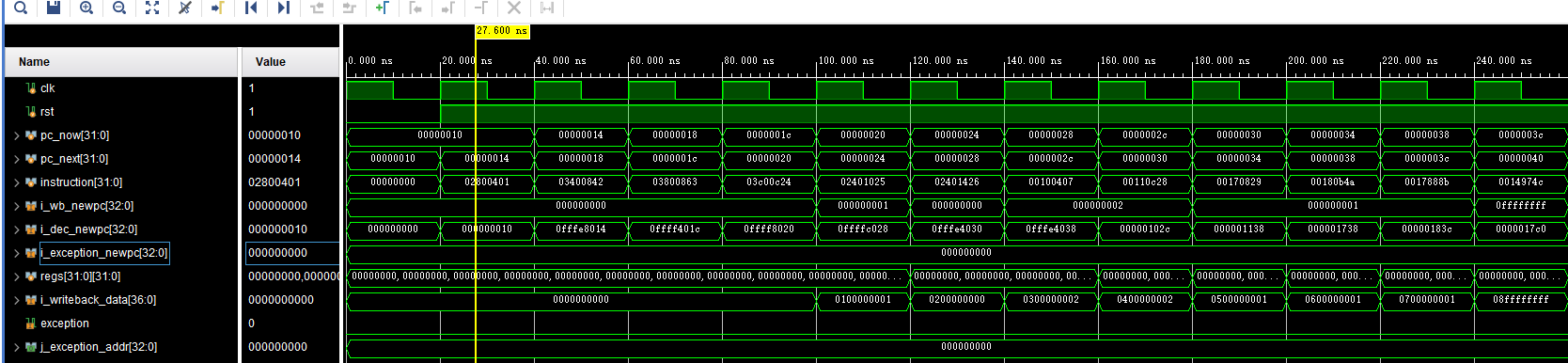
（9）LU12I、PCADDU12I指令数据通路

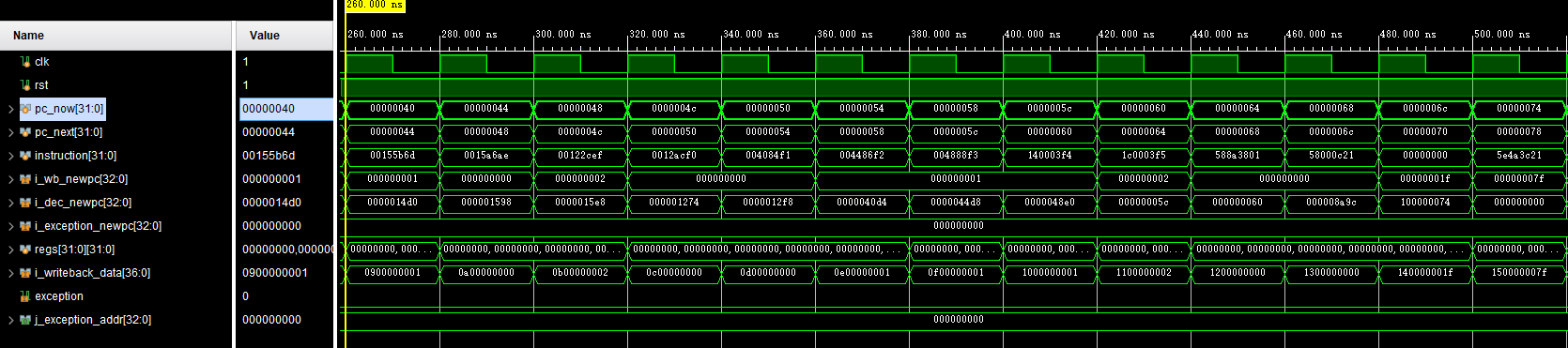


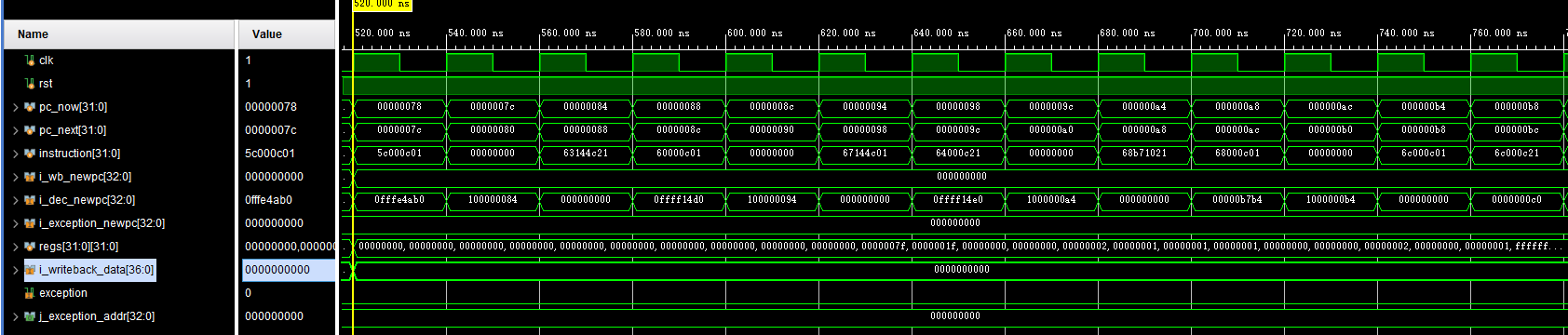
（10）异常指令（SYSCALL、BREAK、ERTN）和非法指令数据通路

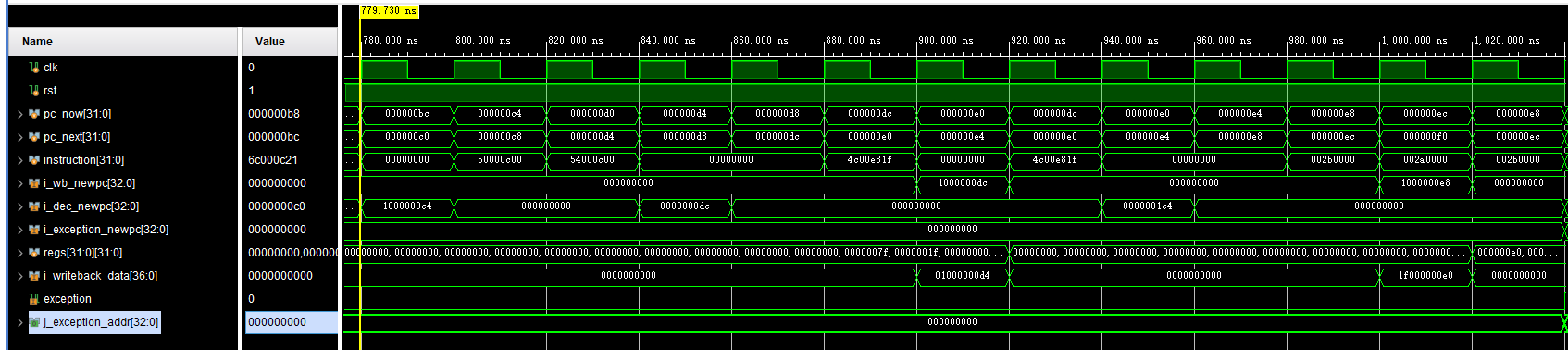


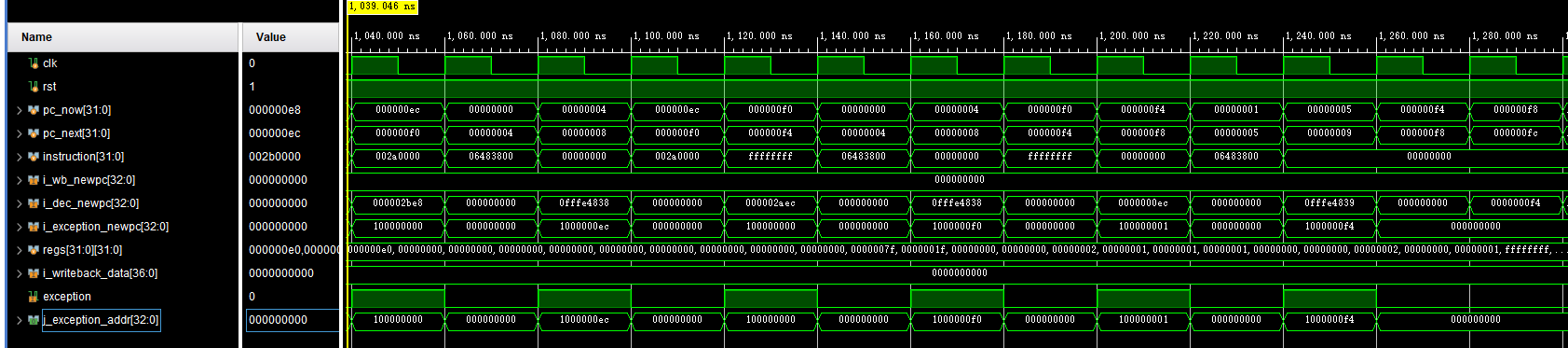
5. 功能测试









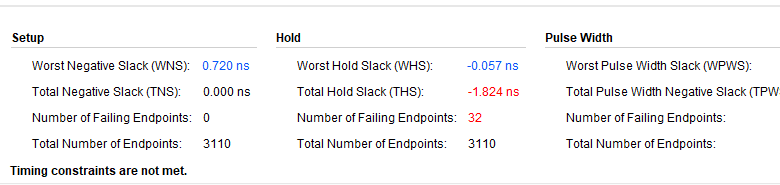


所有指令均能正常执行，功能正确，手工注入异常（非法指令）和异常指令均能成功识别并跳转到异常处理地址和返回。

6. 性能指标

支持35条指令

频率：50MHz

资源使用情况：

