OPEN SOC

小胖达 **RISC-V** 规格书

2024.09.27

目录

二 修订

版本	日期	编辑人	内容
1.00	2024.10.07	陈家耀、孙浩楠	创建了第一个正式版本

三 简介和特性

小胖达 RISC-V 是一个支持中断和异常的 RV32IM 处理器，采用 3 级流水线，具有独立的指令和数据总线。小胖达 RISC-V 主要由**取指单元**、**译码/派遣器**、**数据/资源相关性监测器**、**执行/写回单元**、**总线控制单元**和**通用寄存器堆**组成，其微构架如图 3-1 所示。

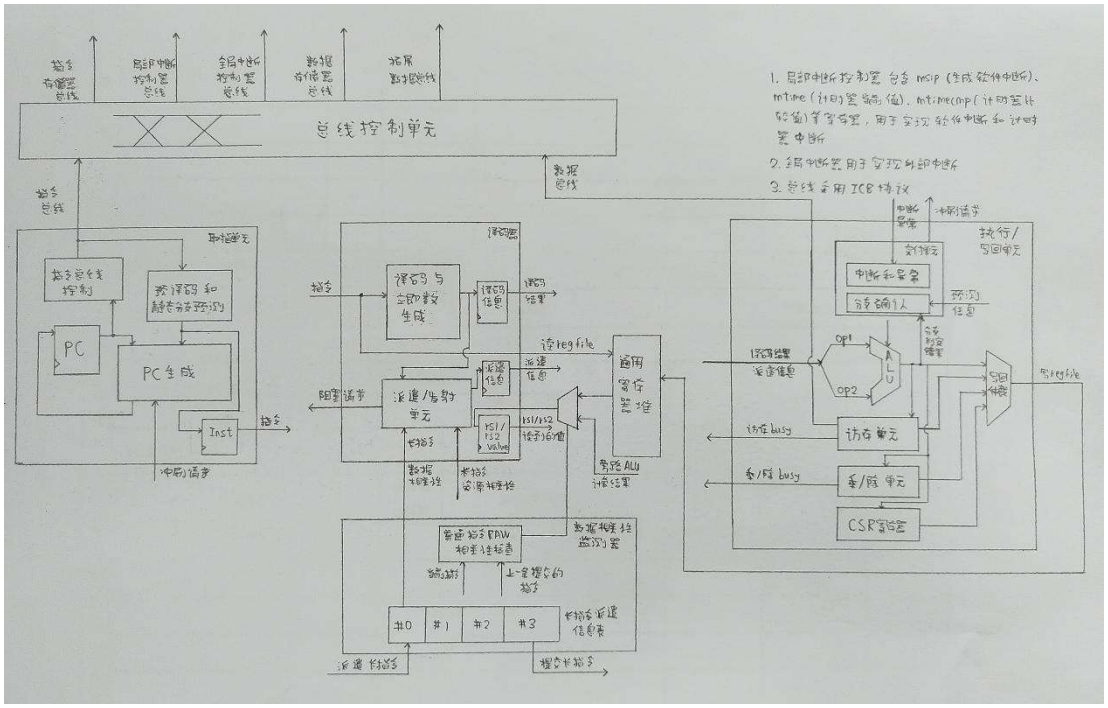

图 3-1 小胖达 RISC-V 微构架

小胖达 RISC-V 具有以下特性：

- **RV32IM**
- **3 级流水线**
- **顺序发射、乱序执行、乱序写回**
- **独立的指令和数据 ICB 总线**
- **支持中断和异常**
- **采用简单的静态分支预测**
- **采用多周期乘除法器**

四 取指单元

取指单元根据当前 PC 从指令总线取出指令，对指令进行预译码和静态分支预测（真正的静态分支预测仅发生在 B 指令上），然后生成新的 PC，将**指令、预译码结果**和该指令对应的 **PC** 值和 **PC+4** 传递给下一级。

*问题：由于指令总线读延迟的存在，建议提前取出 PC+4 的指令？

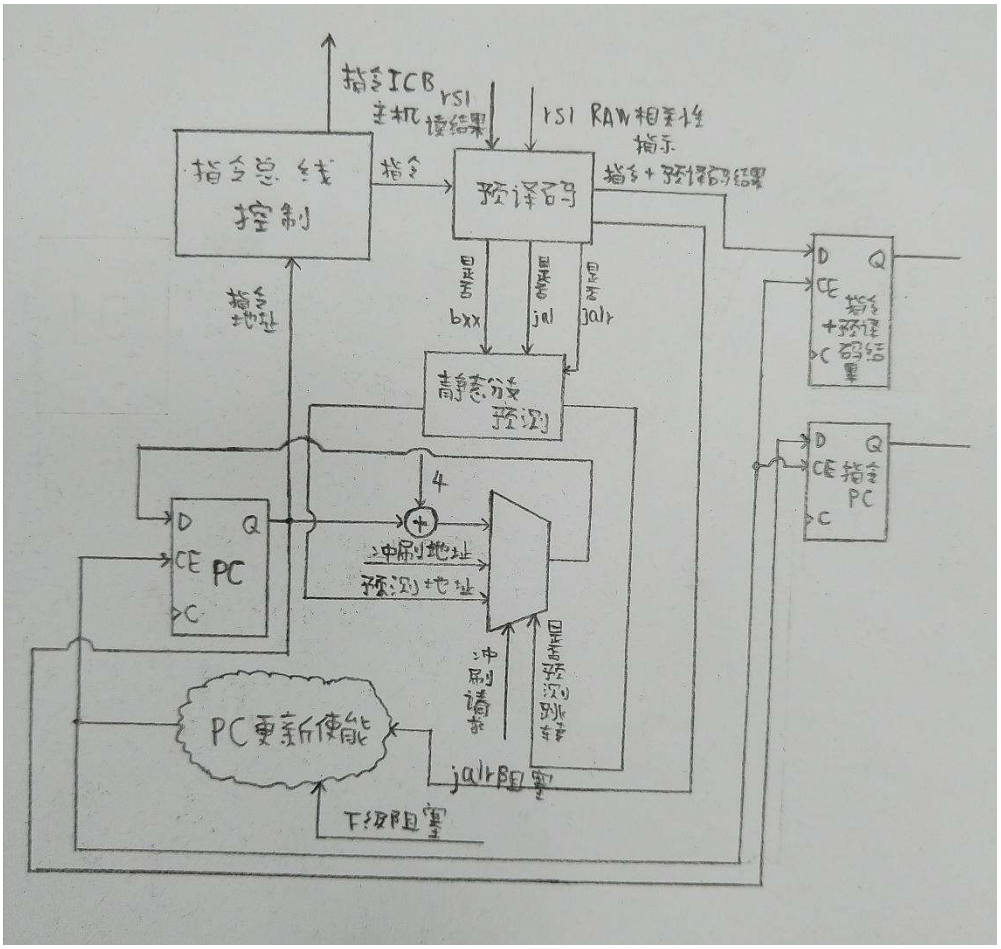

图 4-1 取指单元构架

4.1 指令总线控制

指令总线控制逻辑接收指令存储器访问请求，驱动指令 ICB 主机，返回取到的指令和错误信息。访问指令存储器可能会发生**指令地址非对齐**、**指令总线访问错误**这两种异常。

指令总线的访问可能会被冲刷请求打断。即使 PC 更新被阻塞，已提交的指令总线访问请求仍被继续处理，依然返回取到的指令和错误信息。为了避免取指长时间得不到响应，应当设置一个超时周期数，若超时则返回错误。

4.2 预译码

预译码判断取到的指令是否分支指令，为静态分支预测做准备。预译码应当对**无条件间接跳转指令**（jalr）作特殊处理，将基址寄存器（rs1）的值读回，注意其 RAW 相关性，在必要时拉高阻塞标志来等待 rs1 可读。

预译码可能会发生**非法指令**异常。

预译码应当生成以下信息：

- 是否 B 指令、是否 jal 指令、是否 jalr 指令
- 是否 CSR 读写指令、是否 L/S 指令、是否乘法/除法指令
- 跳转偏移量立即数
- 是否需要读 rs1、是否需要读 rs2、是否需要写 rd
- CSR 寄存器地址
- jalr 指令下的基址寄存器（rs1）读结果
- 异常标志（指令地址非对齐、指令总线访问错误、非法指令）
- 指令本身

4.3 静态分支预测

静态分支预测对 B 指令的预测方法是：向后跳预测为跳，向前跳预测为不跳。静态分支预测对 J 指令总是确定为跳。

静态分支预测生成一个预测地址：对 B 指令或 jal 指令来说，预测地址通过 PC+立即数偏移量生成；对 jalr 指令来说，预测地址通过基址寄存器（rs1）读结果+立即数偏移量生成。

对 B 指令的静态分支预测结果和预测地址应当传递给下级。

4.4 PC 生成

PC 生成逻辑根据**冲刷请求**和**是否预测跳转**来选择下面其中一个作为新的 PC：

- PC+4
- 冲刷地址
- 预测地址

PC 的更新可能会被 jalr 读基址寄存器（rs1）或者下级所阻塞。

五 译码/派遣器

译码/派遣器对取到的指令及附加信息进行翻译处理，生成源/目标寄存器索引、指令类型、指令操作信息等内容，对立即数进行符号位拓展，读通用寄存器堆，考察数据/资源相关性和指令类型并派遣/发射指令。

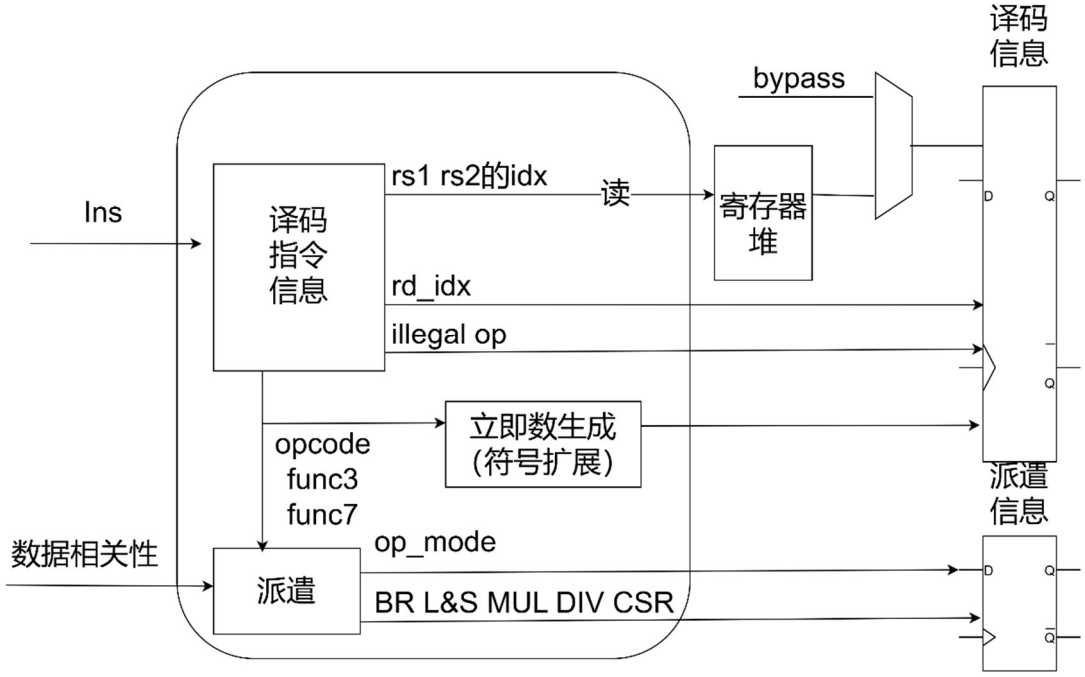

图 5-1 译码/派遣器构架

小胖达 RISC-V 实现基本的整数 I 指令（ECALL、EBREAK 除外，若出现则视为 NOP 指令）和整数乘除法拓展 M 指令，指令表参见 RISC-V 非特权指令集文档 v2.2 的第 104、105 页。

*问题：实现 ECALL、EBREAK 指令的难度如何，有没有必要实现？

5.1 译码与立即数生成

译码与立即数生成单元的主要功能是将从取指单元获取的 32 位指令进行解析，提取出指令的各个字段，包括操作码（opcode）、功能码（funct3、funct7）、源寄存器索引（rs1、rs2）、目标寄存器索引（rd）以及立即数（imm）等。该单元需要根据指令格式对立即数进行正确的符号位拓展，为后续的指令执行做好准备。

从指令寄存器中读取 32 位指令，提取指令的低 7 位操作码（opcode）并确定指令的格式类型（R I S B U J），之后根据不同的指令格式类型提取不同的部分，如下所示。

- **R型指令**：提取`rd`、`funct3`、`rs1`、`rs2`、`funct7`。
- **I型指令**：提取`rd`、`funct3`、`rs1`、`imm[11:0]`。
- **S型指令**：提取`funct3`、`rs1`、`rs2`、`imm[11:5]`和`imm[4:0]`。
- **B型指令**：提取`funct3`、`rs1`、`rs2`、`imm[12]`、`imm[10:5]`、`imm[4:1]`、`imm[11]`。
- **U型指令**：提取`rd`、`imm[31:12]`。
- **J型指令**：提取`rd`、`imm[20]`、`imm[10:1]`、`imm[11]`、`imm[19:12]`。

根据指令格式，将立即数的各个位段正确地组合起来，并根据立即数的最高有效位（符号位），对立即数进行符号位拓展到 32 位。具体组合扩展方式如下图所示。

- **I型指令立即数（12位）**：
  - 最高位为`imm[11]`，如果`imm[11]=1`，则在高位拓展20个`1`；否则拓展20个`0`。
- **S型指令立即数（12位）**：
  - 组合`imm[11:5]`和`imm[4:0]`形成12位立即数，同样根据`imm[11]`进行符号位拓展。
- **B型指令立即数（13位，最低位为0）**：
  - 组合`imm[12]`、`imm[10:5]`、`imm[4:1]`、`imm[11]`，左移一位（最低位为0），根据`imm[12]`进行符号位拓展。
- **U型指令立即数（32位）**：
  - 立即数位于高20位，低12位补`0`。
- **J型指令立即数（21位，最低位为0）**：
  - 组合`imm[20]`、`imm[10:1]`、`imm[11]`、`imm[19:12]`，左移一位（最低位为0），根据`imm[20]`进行符号位拓展。

将解析得到的指令类型、操作信息、源/目标寄存器索引和立即数等信息传递给派遣/发射单元和寄存器堆。

此外还需对非法指令、是否写寄存器堆等信息进行译码

5.2 派遣/发射单元

派遣/发射单元负责根据译码单元提供的指令信息，处理指令之间的相关性，确定指令的执行顺序，并将准备就绪的指令发送到执行单元（如 ALU、乘法器、除法器、加载/存储单元等）。该单元需要确保指令在满足数据和资源条件的情况下被正确地发射。

派遣单元从译码单元接收指令及其相关信息，同时检查执行当前指令所需的功能单元是否空闲。只有当数据相关性不存在且资源可用时，指令才能被派遣。

5.3 读通用寄存器堆

读通用寄存器堆单元负责根据译码单元提供的源寄存器索引，从通用寄存器堆中读取操作数，以供执行单元进行运算。该单元需要确保读取的数据是最新的，考虑到可能存在的写

后读（RAW）相关性。

寄存器堆接收来自译码单元的源寄存器索引 rs1 和 rs2，并通过两个读端口，从寄存器堆中读取 rs1 和 rs2 索引对应的寄存器值。若源寄存器的数据尚未写回寄存器堆，但执行单元已产生结果，则通过 bypass 旁路网络直接获取最新数据。

最后将读到数据写到 DE/EXE 流水线寄存器中。

六 执行/写回单元

执行/写回单元包括具体的**执行机构**（如 ALU、访存单元、CSR 读写控制逻辑）用于实现各种指令的计算、访存和 CSR 读写操作，包含**交付单元**用于处理分支指令和中断/异常，包含**写回控制逻辑**用于仲裁、控制普通指令和长指令的写回操作。

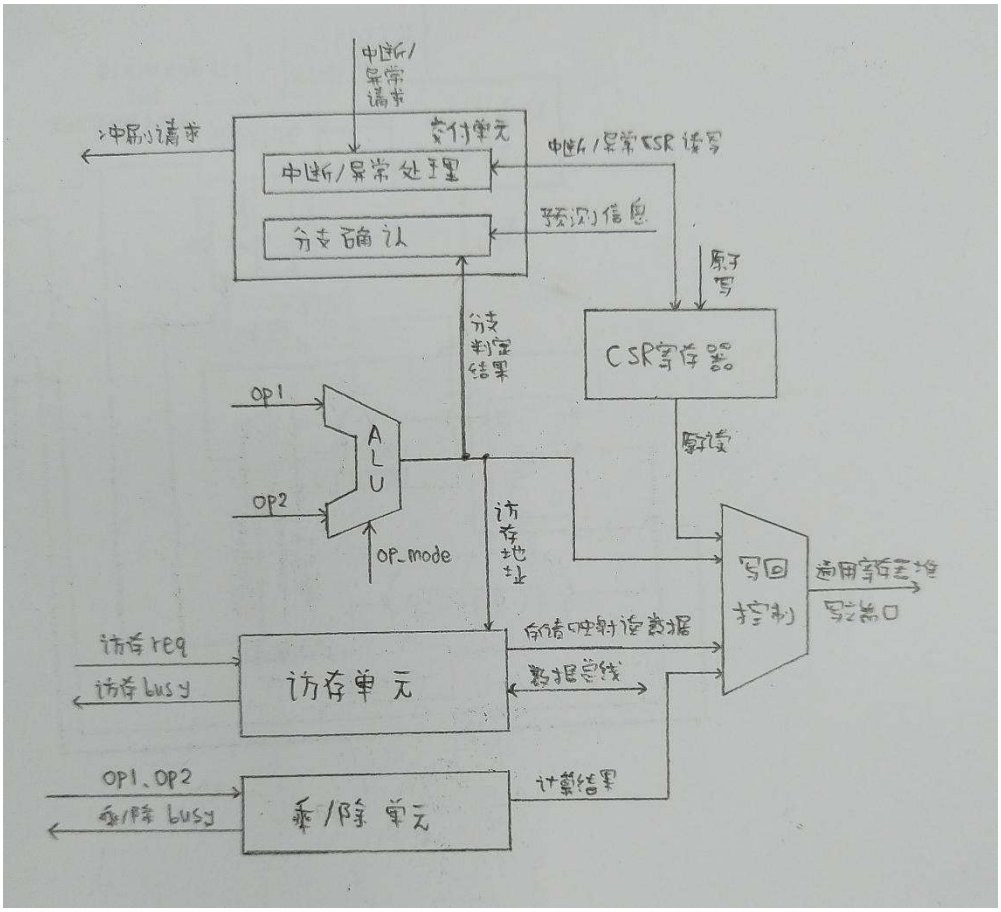

图 6-1 执行/写回单元构架

6.1 ALU

除了 CSR 读写指令以外，任何指令的执行都**必须先经过 ALU**，然后才被传送给访存单元、乘/除单元、分支确认模块做进一步处理，或者被直接写回。ALU 将**前级给出的 op1 和 op2** 作为**操作数 1** 和**操作数 2**。ALU 的计算结果可用于**写目标寄存器、访存地址、分支确认**。

*问题：除 CSR 读写以外的指令都要经过 ALU，是否有必要，会不会导致较长的时序路径？

ALU 支持以下运算类型：

- 加法
- 有符号比较（=，≠，<，≥），无符号比较（<，≥）
- 位逻辑运算（^，|，&）
- 移位（<<，>>，>>>）

6.2 乘/除单元

乘法单元使用多周期的**基 4-Booth 乘法器**来实现，除法单元使用**基于不恢复余数法的除法器**来实现。

6.2.1 基 4-Booth 乘法器

基 4-Booth 乘法器的结构如图 6-2 所示。为了实现有符号和无符号的 32 位乘法，需要对乘数进行 **2 位的符号位拓展**，部分积取 34 位。乘积缓存区需要额外的 1 位作为 **Booth 编码扩充位**。

每次迭代时，取乘积缓存区的低 3 位，根据 **Booth 编码表**生成加数加到乘积缓存区中部分积的所在域，然后对乘积缓存区算术右移 2 位。对于有符号和无符号的 32 位乘法来说，刚好需要 17 次迭代来得到乘积。

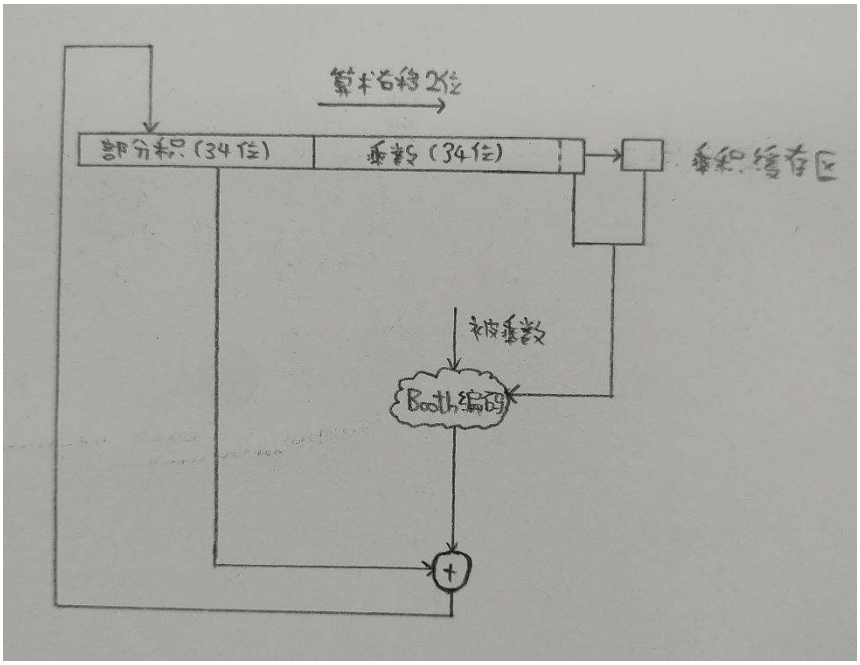

图 6-2 基 4-Booth 乘法器结构图

表 6-1 基 4-Booth 编码表

当前取到的乘积低 3 位	对部分积的操作
000	+0
001	+A
010	+2A
100	-2A
101	-A
110	-A
111	+0

注：A 表示被乘数。

考虑无符号 4 位乘法，乘数为 14，被乘数为 15。乘数表示为 **0011100**，LSB 是 Booth 编码扩充位，乘法计算流程如表 6-2 所示。

表 6-2 14×15 计算流程

迭代轮次	当前 Booth 编码	当前乘数	部分积操作
0	100	15	-2A（-30）
1	111	60	+0（0）
2	001	240	+A（240）
结果			210

考虑有符号 4 位乘法，乘数为-6，被乘数为 15。乘数表示为 **1110100**，LSB 是 Booth 编码扩充位，乘法计算流程如表 6-3 所示。

表 6-3 -6×15 计算流程

迭代轮次	当前 Booth 编码	当前乘数	部分积操作
0	100	15	-2A（-30）
1	101	60	-A（-60）
2	111	240	+0（0）
结果			-90

6.2.2 基于不恢复余数法的除法器

补码形式的不恢复余数法的计算过程如代码段 6-1 所示，具体讲解详见 https://zhuanlan.zhihu.com/p/417008434。图 6-3 展示了补码形式不恢复余数法的 4 个计算示例。

代码段 6-1 补码形式的不恢复余数法

```
X 是被除数
Y 是除数
n 是除法的位宽（含 1 位符号位）
R 是余数寄存器
Q 是商寄存器

{R, Q} = X（需要进行符号位拓展）
若 X 与 Y 异号，则 R1=R0+Y，否则 R1=R0-Y

对于 i 从 1 到 n：
    如果 Ri 与 Y 异号，上商 0，Ri+1=2Ri+Y；否则，上商 1，Ri+1=2Ri-Y

如果 Rn+1 与 Y 异号，上商 0，否则上商 1
仅左移 Q

如果 X 与 Y 异号，修正 Q：Q=Q+1。如果 Rn+1 与 X 异号，修正 Rn+1：若 X 与 Y 异号，Rn+1=Rn+1-Y，否则 Rn+1=Rn+1+Y
可能要对整除的情况作特别修正
```

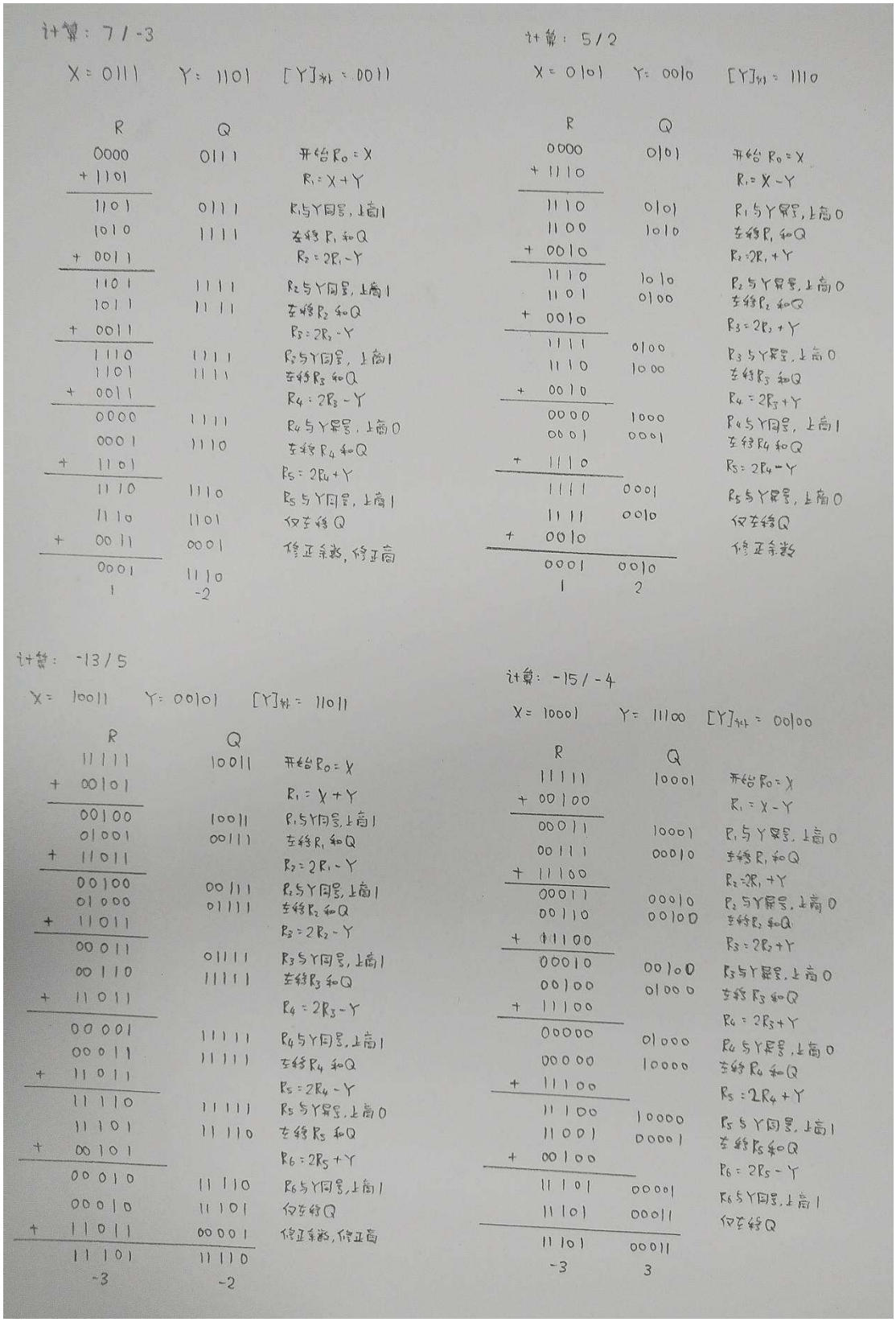

图 6-3 补码形式不恢复余数法计算示例

为了实现有符号和无符号的 32 位除法，需要对被除数/除数进行 **1 位的符号位拓展**，商和余数取 33 位。需要迭代 34 次得到初步的商和余数，另外可能需要额外的 2 个周期来修正商和余数。

RISC-V 非特权指令集文档 v2.2 规定了除 0 和溢出时的商和余数，如表 6-4 所示。

表 6-4 对除 0 和溢出的特殊处理

异常情况	被除数	除数	无符号除法下的商	无符号除法下的余数	有符号除法下的商	有符号除法下的余数
除 0	x	0	$2^{32}-1$	x	-1	x
溢出	-2^{32-1}	-1	---	---	-2^{32-1}	0

注：溢出仅在有符号除法时可能发生。

基于不恢复余数法的除法器的结构如图 6-4 所示。设置**商寄存器 Q**、**余数寄存器 R**、**被除数寄存器 X**、**除数寄存器 Y**，在初始时将符号位拓展后的 X 载入 R 和 Q，每次迭代时根据 R 的符号和 Y 的符号进行上商、更新 R（+Y 或-Y）、左移 R 和 Q 的操作，注意在最后 1 次迭代时仅左移 Q，最后可能需要修正 Q（Q 增加 1）和 R（+Y 或-Y）。

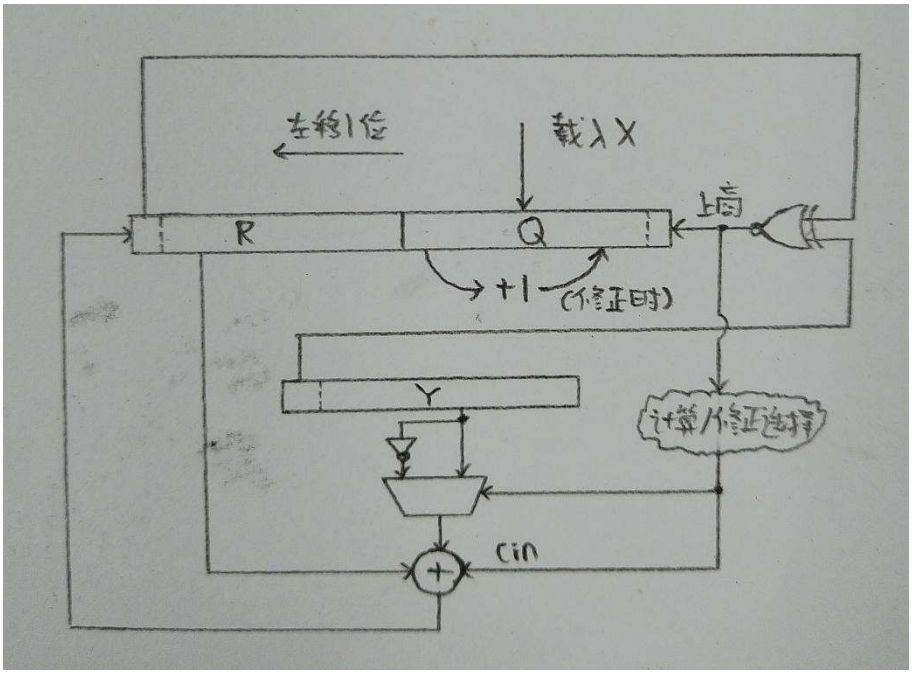

图 6-4 基于不恢复余数法的除法器结构图

6.3 访存单元

访存单元接收**访存请求**，根据 ALU 所产生的**访存地址**，驱动数据 ICB 主机来**读写存储映射**。访存单元可能会产生**写存储映射地址非对齐**、**写存储映射总线错误**、**读存储映射地址非对齐**、**读存储映射总线错误**这 4 种异常。

访存单元应当对每次传输返回以下内容：

- 读数据、目标寄存器索引（若为读传输）
- 错误信息
- 传输 ID 号

为了避免访存长时间得不到响应，访存单元应当设置一个超时周期数，若超

时则返回错误。

6.4 交付单元

交付单元接收中断和异常请求，对 B 指令进行分支确认，判断每条指令应当**提交**还是**取消**。在小胖达 RISC-V 中，影响指令交付的情形主要有**分支预测指令**、**中断和异常**这 2 种。

需要注意的是，对于 FENCH.I 指令而言，总是产生**冲刷请求**，冲刷地址为 FENCH.I 指令所对应的 PC+4，使得 FENCH.I 指令之后的指令被重新取指。

6.4.1 分支确认

对于 1 条 B 指令，需要将 ALU 算得的分支跳转条件和预测的跳转结果进行比较，在两者不符时产生**冲刷请求**：

1、预测为跳但实际上不跳。冲刷地址为 B 指令所对应的 **PC+4**。

2、预测为不跳但实际上跳。冲刷地址为 B 指令所对应的**预测地址**。

无论分支指令是否预测成功，B 指令本身都是交付成功的（即被提交）。

6.4.2 中断和异常

小胖达 RISC-V 实现的中断、异常类型如表 6-5 所示，注意各个中断/异常所记录的返回地址是不同的。遇到中断/异常的指令本身是交付成功的（即被提交）。

表 6-5 小胖达 RISC-V 实现的中断/异常类型

中断/异常类型	记录的返回地址
软件中断	当前正在交付指令所对应的 **PC+4**
计时器中断	
外部中断	
指令地址非对齐	当前正在交付指令所对应的 **PC**
指令总线访问错误	
非法指令	
读存储映射地址非对齐	
写存储映射地址非对齐	
读存储映射总线错误	L/S 指令所对应的 **PC**
写存储映射总线错误	

*问题：读/写存储映射总线错误异常的返回地址应当是 L/S 指令所对应的 PC 还是当前正在交付指令所对应的 PC？

当**进入中断/异常**时，应当执行以下操作：

1、产生**冲刷请求**，冲刷地址为 CSR 寄存器 **mtvec** 中 BASE 域指示的 PC 地址。小胖达 RISC-V 仅支持中断/异常的直接链接模式（Direct）。

2、更新 CSR 寄存器 **mcause**（中断原因）、**mepc**（返回地址）、**mtval**（异常信息）和 **mstatus**（状态寄存器）。

当**退出中断/异常**时，应当执行以下操作：

1、产生**冲刷请求**，冲刷地址为 CSR 寄存器 **mepc** 定义的返回地址。

2、更新 CSR 寄存器 **mstatus**（状态寄存器）。

各种中断/异常的优先级如下（从高到低）：

1、异常。

2、外部中断。

3、软件中断。

4、计时器中断。

6.4.3 长指令的交付

长指令包括了**乘/除指令**和**访存指令**。虽然访存指令可能会在写回时产生**读/写存储映射总线错误**异常，但所有长指令的交付仍然在执行阶段完成，而它的写回则在若干周期后进行。

6.5 CSR 寄存器

CSR 读写控制逻辑实现了对 CSR 的**原子读写操作**（读-修改-写），包含独立的位运算器来对读出的 CSR 值进行位与或位或操作。

小胖达 RISC-V 实现的 CSR 寄存器如表 6-6 所示。

表 6-6 小胖达 RISC-V 实现的 CSR 寄存器

寄存器名	地址	含义
mstatus	0x300	机器模式状态寄存器
misa	0x301	机器模式指令集架构寄存器
mie	0x304	机器模式中断使能寄存器
mtvec	0x305	机器模式异常入口基地址寄存器
mepc	0x341	机器模式异常 PC 寄存器
mcause	0x342	机器模式异常原因寄存器
mtval	0x343	机器模式异常值寄存器
mip	0x344	机器模式中断等待寄存器
mvendorid	0xF11	机器模式供应商编号寄存器
marchid	0xF12	机器模式架构编号寄存器
mimpid	0xF13	机器模式硬件实现编号寄存器
mhartid	0xF14	Hart 编号寄存器
mtime	---	机器模式计时器寄存器
mtimecmp	---	机器模式计时器比较寄存器
msip	---	机器模式软件中断等待寄存器

需要注意的是，**mtime**（系统计时器计数值）、**mtimecmp**（系统计时器比较值）、**msip**（软件中断控制）不属于 CPU 内部的 CSR 寄存器，它们被定义在存储映射上。

6.6 写回控制

只有交付成功（即被提交）且需要写目标寄存器的指令要进行写回操作。写回控制逻辑应在被占用时阻塞普通指令（不是长指令）的交付。长指令写回的同时应当将**长指令派遣信息表**的相应项退出。长指令应当按照**长指令派遣信息表**中 id 号的顺序写回，仲裁模块依据 id 号来授权长指令的写回。

在小胖达 RISC-V 中，普通指令和长指令分别都是顺序写回的，但后面的普通指令却可以比前面的长指令先写回（如果没有数据相关性的话）。

*问题：长指令的写回要严格按照派遣顺序吗？乱序写回是否过于复杂？

写回控制逻辑按照以下优先级（从高到低）对写回操作进行仲裁：

1、授权写回的长指令。

2、普通指令。

七 通用寄存器堆

小胖达 RISC-V 使用 32 个 32 位寄存器作为通用寄存器堆。通用寄存器堆仅支持**1个写端口**和**2个读端口**，应使用仲裁逻辑来避免这些读写端口的访问冲突。

通用寄存器的写端口将目标寄存器（rd）的索引分别与每个通用寄存器的索引进行比较来得到相应的写使能，被选中的寄存器即将写数据写入。通用寄存器的读端口是纯粹的并行多路选择器，多路选择器的选择信号就是 2 个源寄存器（rs1 和 rs2）的索引。

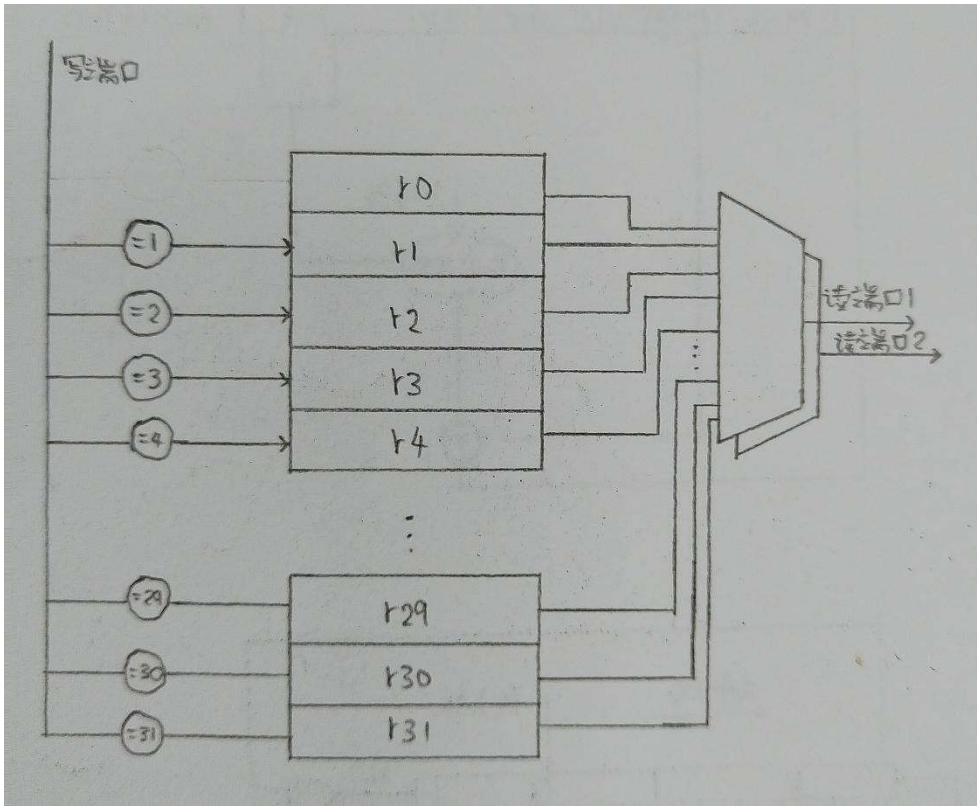

图 7-1 通用寄存器堆结构图

八 数据/资源相关性监测器

监测器对数据相关性的处理策略是：

1、如果**当前指令和上一条普通指令产生 RAW 冲突**，则**将 ALU 计算结果或 CSR 读结果旁路回去**。

2、如果**当前指令和正在执行的长指令产生 RAW 或 WAW 冲突**，则**阻塞流水线**。

监测器使用一个深度为 4 的**长指令派遣信息表**来记录正在执行的长指令。当长指令被派遣时，将它的信息记录到表中。当长指令被写回时，将它的信息从表中移除。长指令派遣信息应当包括：

1、传输 ID 号（如果是 L/S 指令）。

2、目标寄存器（rd）索引。

3、是否需要写回目标寄存器（rd）。

4、指令所对应的 PC 值。

5、长指令类型（乘法、除法、访存）。

FENCH 指令和 FENCH.I 指令需要等待**长指令派遣信息表**为空后才能被派遣。

监测器对资源相关性的处理策略是：

如果 1 条长指令需要用到的执行单元（乘/除单元或访存单元）处于忙碌状态，则**阻塞流水线**。普通指令只用到 ALU 或 CSR 读写控制模块，这些执行单元都是单周期的，不存在资源冲突的问题。

九 总线控制单元

总线控制单元将系统总线（指令和数据总线）与外部总线（指令存储器总线、局部中断控制器总线、全局中断控制器总线、数据存储器总线和拓展数据总线）进行连接，实现取指和读写存储映射。

小胖达 RISC-V 使用 **ICB 总线**作为系统总线。关于 **ICB 总线**协议，可参见《手把手教你设计 CPU：RISC-V 处理器篇》第 206~212 页。

小胖达 RISC-V 的存储映射分配情况如表 8-1 所示。

表 8-1 存储映射分配情况

系统总线类型	内容	偏移量	区间长度
指令总线	指令存储器	0x00000000	4GB
数据总线	局部中断控制器	0x00000000	4KB
	全局中断控制器	0x00001000	4KB
	数据存储器	0x10000000	512MB
	拓展	0x30000000	256MB

总线控制单元将指令总线直接传递到指令存储器总线，将数据总线通过一个“ICB 分发”来连接局部中断控制器总线、全局中断控制器总线、数据存储器总线和拓展数据总线，如图 8-1 所示。总线控制单元不应该在系统总线和外部总线之间引入额外的时延。

“ICB 分发”在数据总线的命令通道进行地址译码，选中 1 个外部数据总线的命令通道进行传输，在其握手成功后将分发信息压入 fifo，使用分发信息 fifo 中的数据或旁路的当前分发信息（如果当前要访问的外部数据总线能立即接受命令且返回响应的话）来选择 1 个外部数据总线的响应通道。

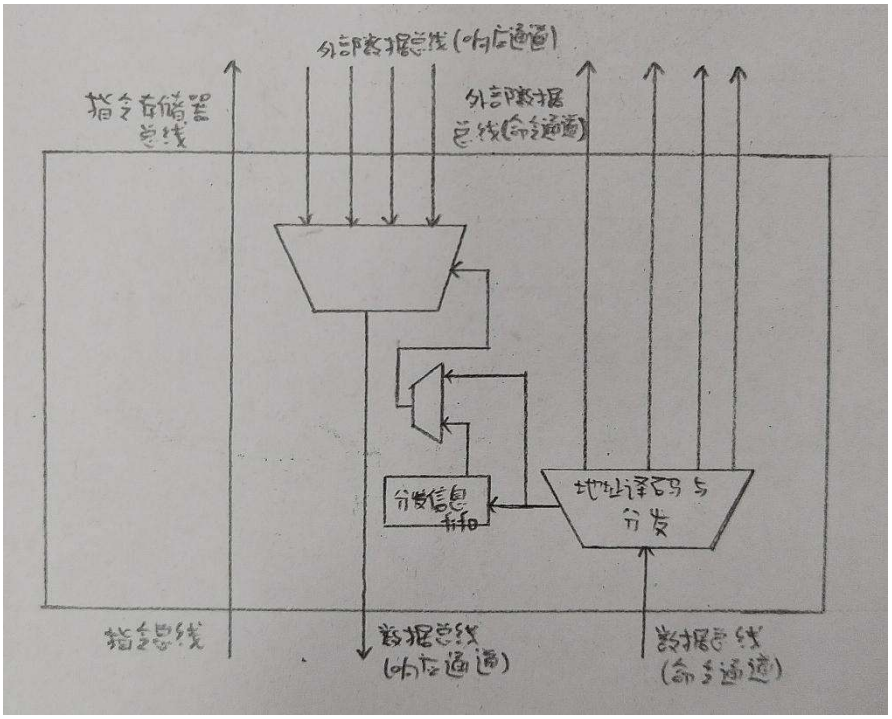

图 8-1 总线控制单元结构图