**实验20**

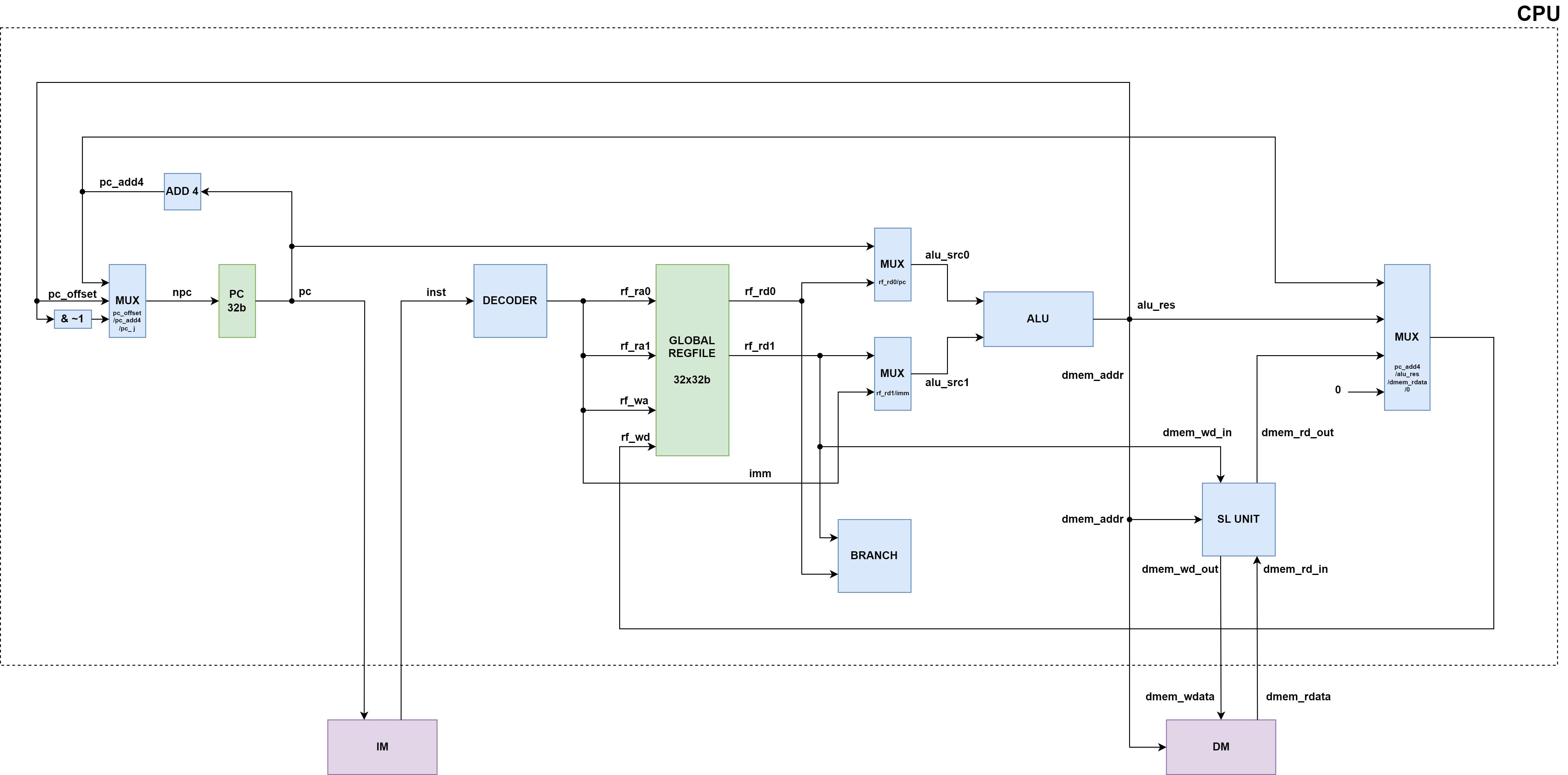
**完整单周期处理器**

# **实验介绍**

在本次实验中，我们将进一步完善上一次实验设计的 CPU，为其增加分支指令和访存指令的相关功能。

# **实验内容**

## 1 整体框架

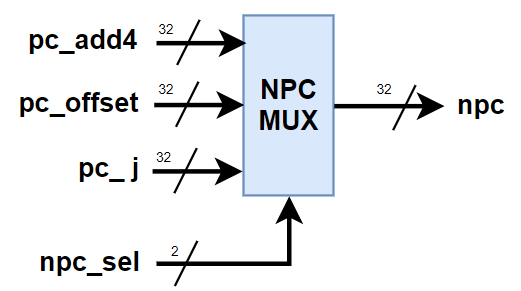
[](https://soc.ustc.edu.cn/COD/lab4/img/CPU_fig/datapath.png)

上图展示了本次实验中 CPU 的数据通路。该数据通路可以兼容 RV32I 版本的 CPU 和 LA32R 版本的 CPU，二者仅在译码器 Decoder 的内部结构上不同。此外，通路中省略了控制信号，例如 alu\_op、rf\_we 等。

## 2 模块设计

### ****2.1 NPC选择器****

与简单单周期不同的是，单周期 CPU 需要能够执行分支指令，所以下一个周期的 PC 值就不一定是 PC + 4 了。为此，我们需要一个数据选择器选择合适的值来传递给 PC 寄存器，以下是其示意图。

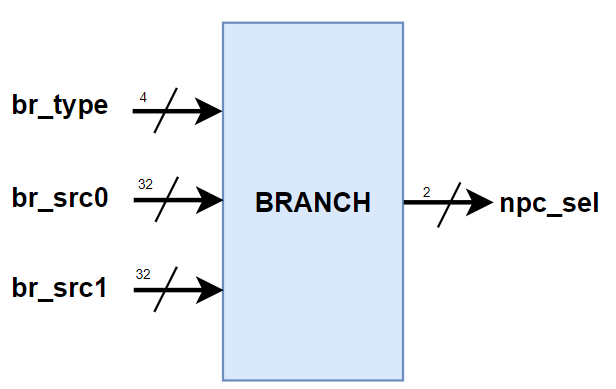
[](https://soc.ustc.edu.cn/COD/lab4/img/CPU_fig/npc_mux_struc.png)

其中，pc\_j 端口对应 jalr 指令的结果，该输入为将 alu\_res 的结果最低位置为 0。

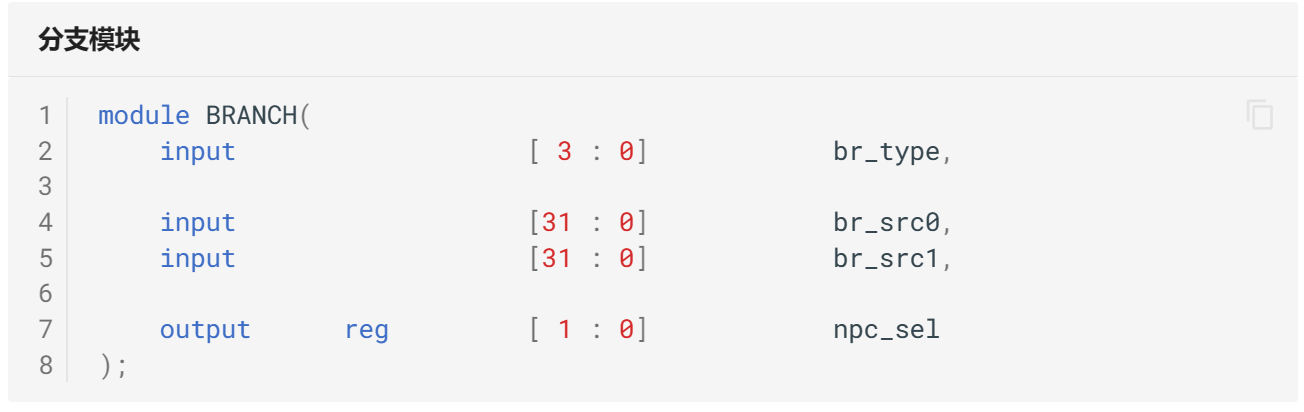
在 LA32R 指令集中，与 jalr 指令对应的是 jirl 指令，但该指令不需要将最低位置位 0。因此，对于选择 LA32R 指令集的同学，可以自主选择是否需要这个 pc\_j 端口。

### ****2.2 分支模块****

与教材不同，我们把条件跳转指令的逻辑判断从 ALU 中移出，封装成了独立的模块。 Branch 模块是专门用来处理分支指令的模块。它接收来自 Decoder 的控制信号，以及来自寄存器堆的两个数据。根据这两个数据之间的大小关系以及控制信号， Branch 模块最终产生相应的跳转信号。以下是本次实验中的分支模块结构示意图。

[](https://soc.ustc.edu.cn/COD/lab4/img/CPU_fig/branch_struc.png)

其端口代码为：

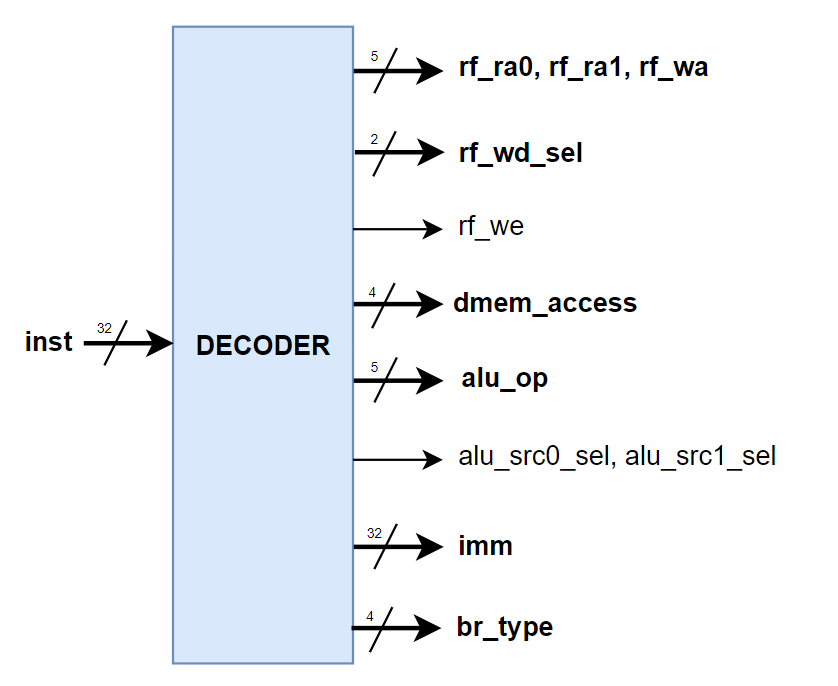


各个端口的作用是：

* br\_type：分支跳转的类型；
* br\_src0、br\_src1：来自寄存器堆的两个数据；
* npc\_sel：npc 选择器的控制信号；

### ****2.3 译码器****

为了实现完整的单周期 CPU ，你需要修改你在 lab3 中的 Decoder 设计，处理简单单周期中没有实现的指令，以下是本次实验中的译码器的结构示意图。

[](https://soc.ustc.edu.cn/COD/lab4/img/CPU_fig/decode_struc.png)

其端口代码为：



相较于简单单周期，新增的各个输出端口的作用是：

* dmem\_access：访存类型；
* rf\_wd\_sel：寄存器堆写回数据选择器的选择信号；
* br\_type：分支跳转的类型；

这些信号的生成和先前描述的一样，都需要根据 inst 的数值，结合对应指令集的译码规则得出。

### ****2.4 访存控制单元****

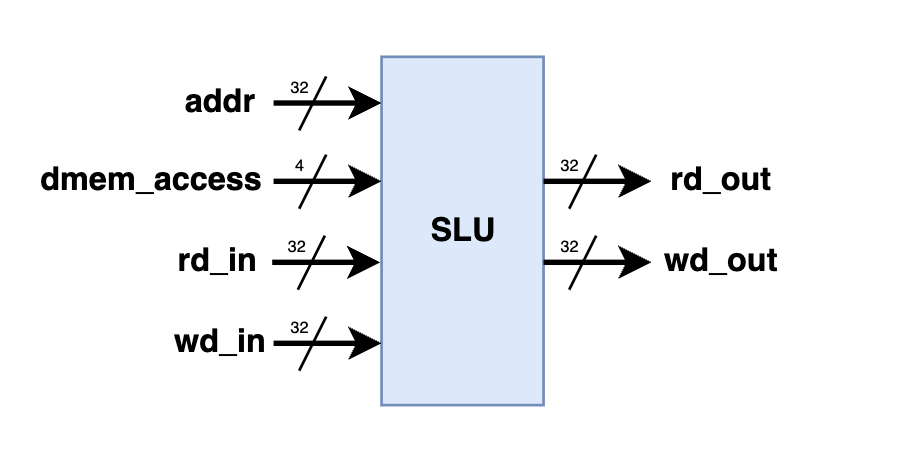
访存控制单元主要是用于实现非整字访存指令的。例如：如果我们想要带符号地读取 0x5 地址上的字节的值，我们就需要先读取 0x4 开始的四个字节上的数据，再取出其中的第 8~15 位，右移 8 位并符号扩展之后写入目标寄存器，总结起来就是：

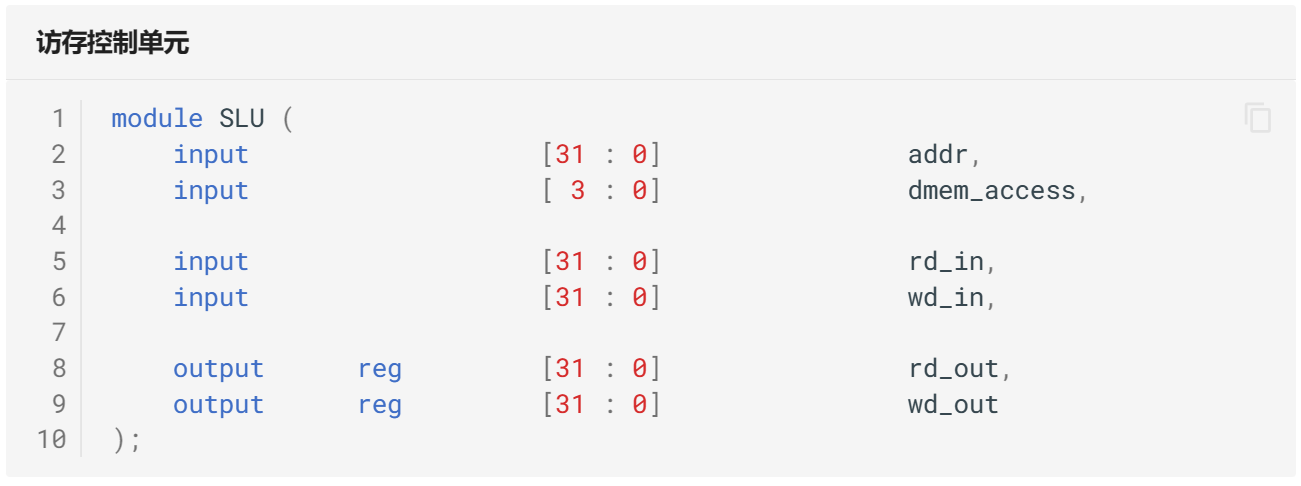
x[rd] = {{24{(M[0x4])[15]}}, (M[0x4])[15:8]}

如果我们想要向 0x5 地址上写入一个字节的数据 0x12，为了不影响其余字节的数据，就需要先将它们读出，与待写入的数据拼接之后再写回。因此，实际执行的操作就是：

M[0x4] = {(M[0x4])[31:16], (x[rd])[7:0], (M[0x4])[7:0]}

我们的 SL\_UNIT 就需要根据指令的具体类型实现上述的访存操作。

[](https://soc.ustc.edu.cn/COD/lab4/img/CPU_fig/slu_struc.png)



各个端口的作用是：

* addr：访存的地址；
* dmem\_access：访存类型；
* rd\_in：从存储器读到的原始数据；
* wd\_in：要写入存储器的原始数据；
* rd\_out：根据 dmem\_access 处理过的读到的数据；
* wd\_out：根据 dmem\_access 处理过的要写入的数据；

在正常情况下，CPU 要求半字访问指令 ld.h（lh）、ld.hu（lhu）、st.h（sh）仅会访问一个字中的低半字（0、1 字节）或高半字（2、3 字节），不会出现跨字访问或中间访问（1、2 字节）。因此，SL\_UNIT 需要根据输入的 dmem\_access 信号，结合 ALU 计算出的访问地址判断此时是否是合法的访问。对于真实的处理器来说，此时会抛出一个「非对齐访问」的异常信号，而我们的单周期 CPU 目前还没有这个功能，因此遇到非正常的半字访问，SL不做处理即可。

### ****2.5 寄存器堆写回选择器****

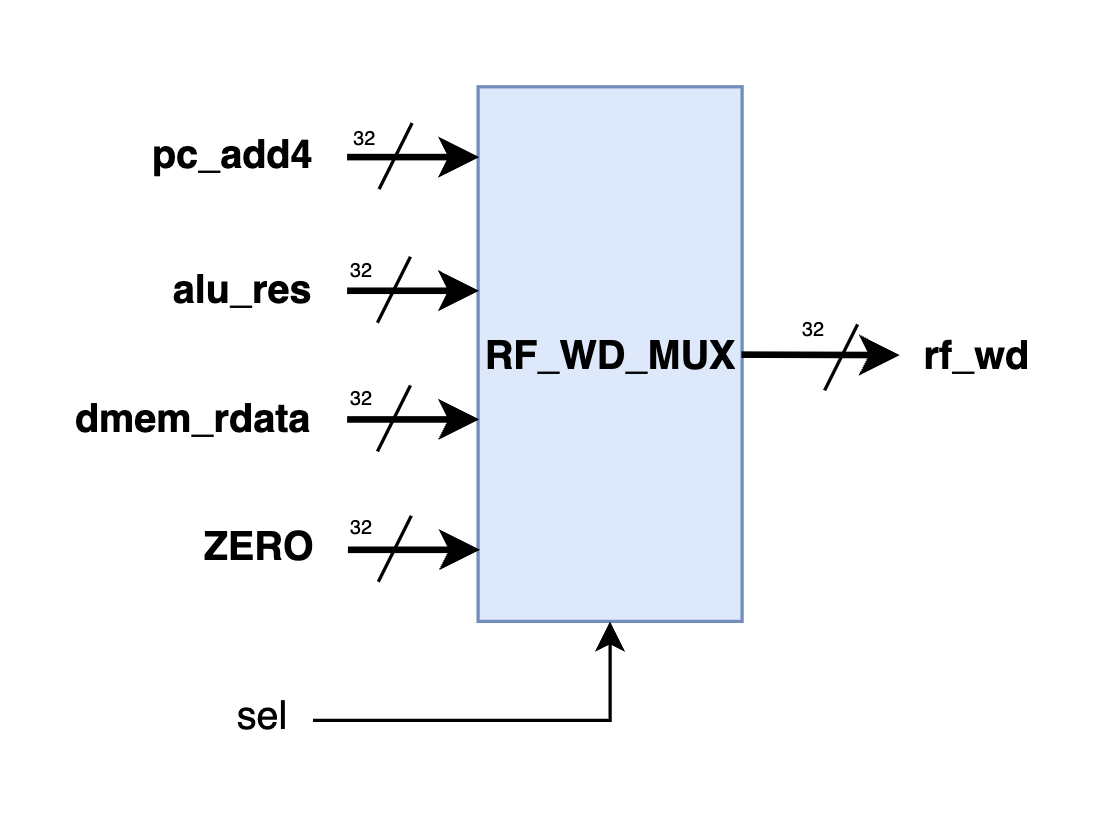
要写回寄存器堆的数据来源由你的 CPU 具体实现决定，在助教的实现中，要写回的数据有四个来源，PC 加四，ALU 的结果、从存储器读到的值和 0。实际上这里的 0 是没有实际作用的，但三选一和四选一选择器的资源开销基本一致，所以我们额外引入了一个输入端口，用一个四选一数据选择器用于控制写回的数据，一个简单的参数化选择器实现如下：



四选一选择器的四个来源分别是：

* pc\_add4
* alu\_res
* dmem\_rdata
* ZERO

以下是结构示意图：

[](https://soc.ustc.edu.cn/COD/lab4/img/CPU_fig/rf_wd_mux_struc.png)

# 思考与练习

1. 根据上面的内容，实现RV32I或LA32R指令集下的完整单周期CPU。