# Lab 4 实验报告



## 一、实验目标

1. 实现指令：CSRRW CSRRS CSRRC CSRRWI CSRRSI CSRRCI

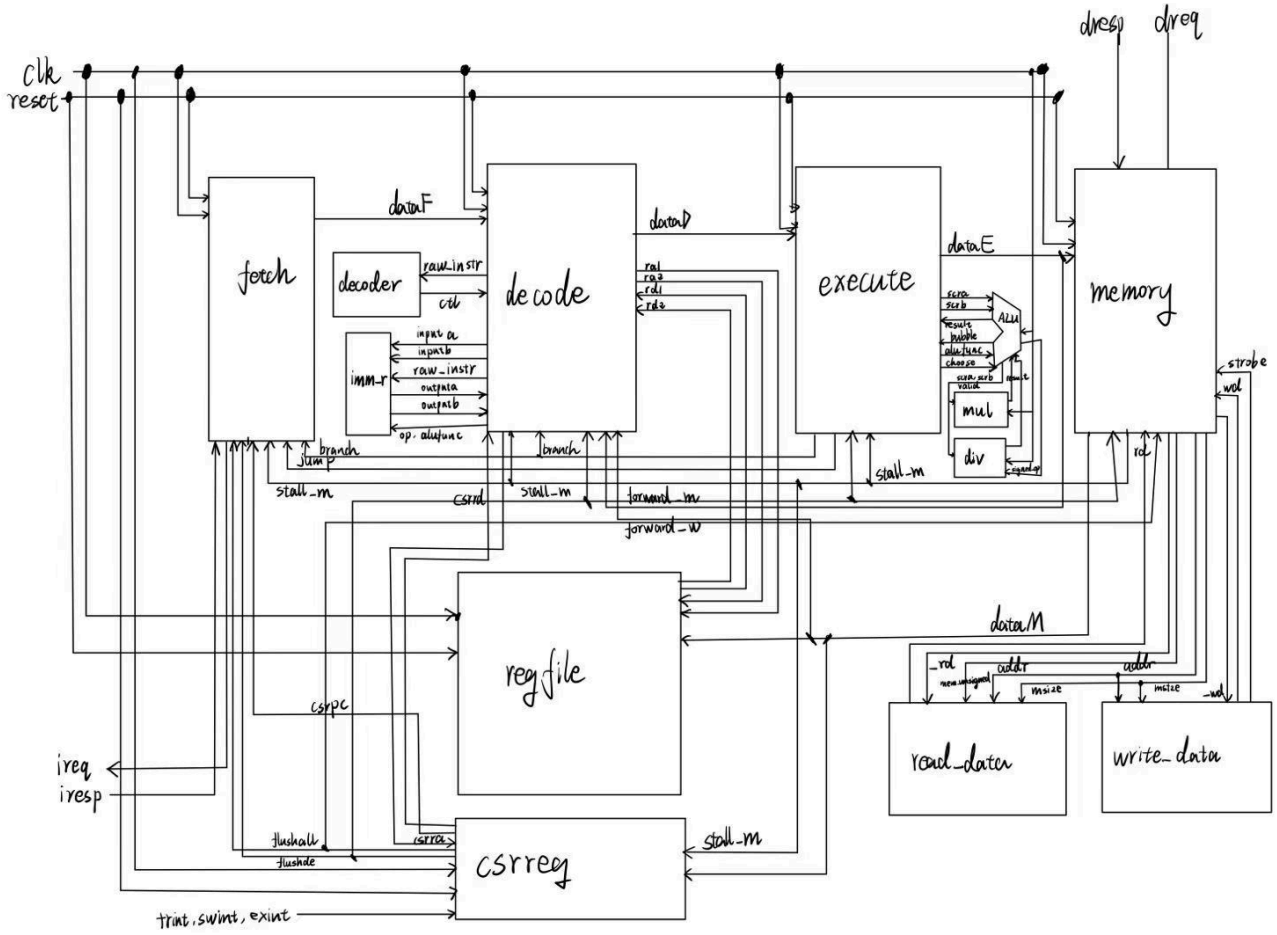
   实现寄存器：mstatus mtvec mip mie mscratch mcause mtval mepc mcycle mhartid satp ，并且需要将这些寄存器对应连接到 `DifftestCSRState`。

2. 完成Vivado仿真与上板

## 二、模块层次

```
core
├── fetch          // 取指阶段
├── decode         // 译码阶段
    ├── decoder
    ├── imm_r
├── execute        // 执行阶段
    ├── alu
        ├── mul
        ├── div
            ├── divu
        ├── divu
├── memory         // 访存阶段
    ├── read_data
    ├── write_data
├── csr          // CSR寄存器文件
    ├── csrreg
└── regfile        // 寄存器文件
```

手绘电路图：




# 三、与Difftest连接方案

需要将mstatus mtvec mip mie mscratch mcause mtval mepc mcycle mhartid satp这些寄存器对应连接到`DifftestCSRState`。在`DifftestCSRState`中没有的寄存器不需要连接。

```
DifftestCSRState DifftestCSRState(
        .clock              (clk),
        .coreid             (csrreg.regs_next.mhartid[7:0]),
        .priviledgeMode     (3),//Machine mode!
        .mstatus            (csrreg.regs_next.mstatus),
        .sstatus            (csrreg.regs_next.mstatus & 64'h800000030001e000),
        .mepc               (csrreg.regs_next.mepc),
        .sepc               (0),
        .mtval              (csrreg.regs_next.mtval),
        .stval              (0),
        .mtvec              (csrreg.regs_next.mtvec),
        .stvec              (0),
        .mcause             (csrreg.regs_next.mcause),
        .scause             (0),
        .satp               (csrreg.regs_next.satp),
        .mip                (csrreg.regs_next.mip),
        .mie                (csrreg.regs_next.mie),
        .mscratch           (csrreg.regs_next.mscratch),
        .sscratch           (0),
        .mideleg            (0),
        .medeleg            (0)
```

```
    );
```

其中，`mhartid`保存了当前CPU核心的编号。将`core.sv`中`DifftestInstrCommit`、`DifftestArchIntRegState`、`DifftestTrapEvent`和`DifftestCSRState`的`coreid`都连接为`mhartid[7:0]`。

```
.coreid             (csrreg.regs_next.mhartid[7:0]),
```

# 四、CSR指令实现

本次实验主要内容实现在`csrreg`模块，实现了控制和状态寄存器（CSR，Control and Status Register）的功能，用于管理处理器的各类状态、中断和异常。代码中该模块支持对CSR寄存器的读取和写入操作，并能够处理错误、中断和CSR操作。

1. **CSR寄存器初始化：**

   ```
   csr_regs_t regs_next;
   csr_regs_t regs = '{
       mhartid: 0,
       mie: 64'h80000000,
      .......
   };
   ```

   其中，csr_regs_t在`csr.sv`中的`csr_pkg`实现。

2. **读操作：** 读操作根据输入的`csrra`（CSR地址），选择相应的CSR寄存器值，并将其存储在`csrrd`寄存器中。

   ```
   always_comb begin
      csrrd = '0;
      unique case(csrra)
         CSR_MSTATUS: csrrd = regs.mstatus;
         CSR_MTVEC: csrrd = regs.mtvec;
         CSR_MIP: csrrd = regs.mip;
         CSR_MIE: csrrd = regs.mie;
         ...
      endcase
   end
   ```

3. **写操作：** 写操作根据传入的控制信号更新CSR寄存器的值。如果请求进行CSR操作，模块会执行相应的ALU函数来修改寄存器的值。每个CSR寄存器都有预定义的掩码（如`MIP_MASK`、`MTVEC_MASK`等，在`csr.sv`中），以确保只修改有效的位。

```
always_comb begin
    regs_next=regs;
    regs_next.mcycle =regs.mcycle+1;
    if(~stall_m)begin
        if (handle_csr_op) begin
            begin
                unique case(dataM.csr_dst)
                    CSR_MIE: regs_next.mie = csr_result;
                    CSR_MIP:  regs_next.mip = csr_result & MIP_MASK;
                    CSR_MTVEC: regs_next.mtvec = csr_result & MTVEC_MASK;
                    CSR_MSTATUS: regs_next.mstatus = csr_result & MSTATUS_MASK;
                    CSR_MSCRATCH: regs_next.mscratch = csr_result;
                    ......
```

4. **刷新逻辑：** csr 作为寄存器，也会有数据冲突。但是，csr 不应该转发。csr 的每次改变，都应刷新流水线（普通的写入，刷新流水线后，从 pc + 4 开始继续执行）。这个由`flushde`和`flushall`信号传出。

```
always_comb begin
   flushde = 0;
   flushall = 0;
   if (~stall_m && handle_csr_op) begin
      flushde = 1;
      flushall = 1;
   end
end
```

# 五、思考题

## 1、参考英文指令集手册，简述一下此次lab中各个csr寄存器的作用

| csr寄存器 | 作用 |
| --- | --- |
| mstatus | 保存全局中断使能，以及许多其他的状态，它主要涉及处理器当前的中断状态和机器模式下的运行配置。 |
| mtvec | 保存发生异常时处理器需要跳转到的地址，它保存了机器模式下异常处理程序的入口地址。当发生异常时，处理器会跳转到该地址执行异常处理程序。 |
| mip | 列出目前正准备处理的中断，存储当前挂起的中断状态。若某个中断源已触发，它的相应位会被置为1。 |
| mie | 指出处理器目前能处理和必须忽略的中断，它控制各个中断源的使能状态。如果某个中断源的相应位被设置为1，则该中断源会被允许触发中断。 |
| mscratch | 暂存一个字大小的数据，用于临时保存一些处理器的状态信息，通常在发生异常或中断时保存现场数据，以便恢复或继续执行。 |
| mcause | 指示发生异常的种类。异常和中断处理程序可以根据该寄存器的内容来判断具体的错误或中断类型。 |

| csr寄存器 | 作用 |
| --- | --- |
| mtval | 保存了陷入（trap）的附加信息：访问地址出错时的地址信息、或者执行非法指令时的指令本身，对于其他异常，它的值为 0。 |
| mepc | 指向发生异常的指令，即发生异常时处理器应返回的位置。处理器恢复执行时会跳转到此地址继续执行。 |
| mcycle | 保存了 CPU 已经运行的时钟周期数，因此将其设置为每周期加一。如果溢出，直接从0重新开始；如果写入，用写入的值覆盖。 |
| mhartid | 保存了当前CPU核心的编号。如果系统中有多个处理器核，每个处理器核会有唯一的`mhartid`值。我们目前只有一个核心，因此一直设为0即可。 |
| satp | 用于管理虚拟地址到物理地址的转换，通常在支持虚拟内存的系统中使用。它保存了当前的地址转换信息。 |

## 2、思考为什么一定要刷新流水线?

刷新流水线是处理器中为了处理特定情况（如异常、错误或中断）而进行的。刷新流水线意味着清除流水线中正在执行的指令，重新开始执行新的指令，目的是确保在发生异常、错误或中断时，流水线中的指令不会影响到程序的正确性和稳定性。通过清空流水线，处理器可以重新从正确的地方开始执行指令，避免继续执行错误的指令或不一致的数据。

以下是为什么需要刷新流水线的几个原因：

1. 异常或中断发生时的恢复

   当发生异常或中断时，处理器需要跳转到异常或中断处理程序来进行处理。这时，流水线中可能还在执行与异常无关的指令。如果不刷新流水线，处理器将继续执行错误的指令或未准备好的指令。因此，必须刷新流水线，丢弃当前正在执行的指令，并跳转到正确的异常处理代码。

2. 指令序列被破坏

   由于流水线的特性，一条指令在不同的阶段可能还没有完全执行完。当发生异常或中断时，流水线中的指令可能已经开始执行，但结果不再是正确的，或者接下来的指令已经不是想要执行的指令。因此，必须刷新流水线，确保从正确的位置开始执行。

3. 处理数据依赖问题

   在多级流水线中，前一级指令的结果可能会影响后续指令。如果发生异常或中断，处理器需要保证当前指令的结果不会影响后续指令，或者重新加载数据。因此，刷新流水线可以确保中断后重新取指和执行时数据一致性不会受到影响。

# 六、实验结果

- 实现指令：CSRRW CSRRS CSRRC CSRRWI CSRRSI CSRRCI

  实现寄存器：mstatus mtvec mip mie mscratch mcause mtval mepc mcycle mhartid satp

- 通过Lab4测试

```
ubuntu20@pc: ~/my_try/2025Spring_lab4/arch-2025
The image is ./ready-to-run/lab4/lab4-test.bin
Using simulated 256MB RAM
Using /home/ubuntu20/my_try/2025Spring_lab4/arch-2025/ready-to-run/riscv64-nemu-
interpreter-so for difftest
[src/device/io/mmio.c:19,add_mmio_map] Add mmio map 'clint' at [0x38000000, 0x38
00ffff]
[src/device/io/mmio.c:19,add_mmio_map] Add mmio map 'uartlite' at [0x40600000, 0
x4060000c]
[src/device/io/mmio.c:19,add_mmio_map] Add mmio map 'uartlite1' at [0x23333000,
0x2333300f]
The first instruction of core 0 has commited. Difftest enabled.
[WARNING] difftest store queue overflow
[src/cpu/cpu-exec.c:393,cpu_exec] nemu: HIT GOOD TRAP at pc = 0x000000008001fff8
[src/cpu/cpu-exec.c:394,cpu_exec] trap code:0
[src/cpu/cpu-exec.c:74,monitor_statistic] host time spent = 28,938 us
[src/cpu/cpu-exec.c:76,monitor_statistic] total guest instructions = 32,766
[src/cpu/cpu-exec.c:77,monitor_statistic] simulation frequency = 1,132,282 instr
/s
Program execution has ended. To restart the program, exit NEMU and run again.
sh: 1: spike-dasm: not found
```

- Vivado仿真与上板