Lab 2 实验报告

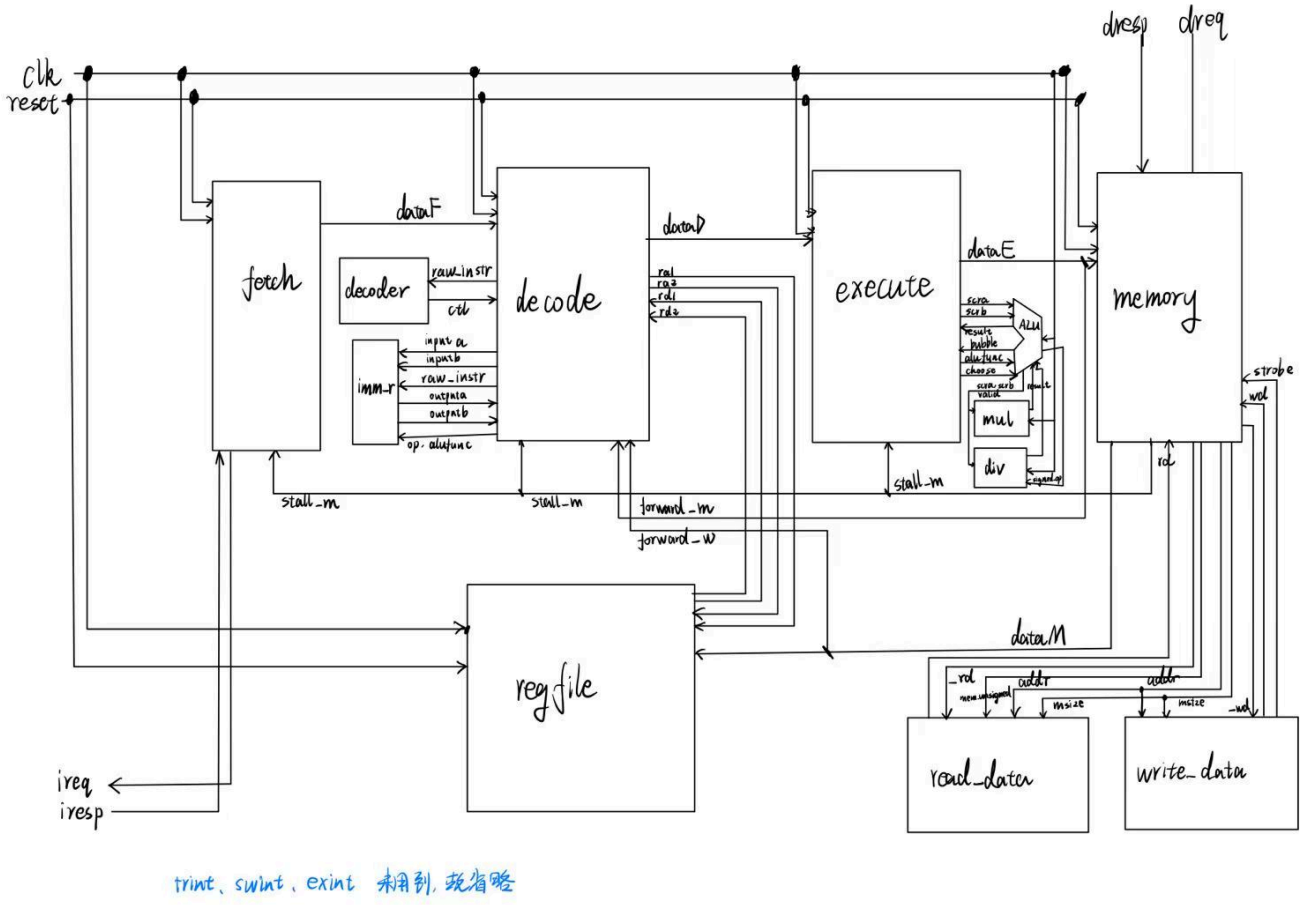
一、设计目标

1. **支持指令集**:
   - ld sd lb lh lw lbu lhu lwu sb sh sw lui
2. **测试要求**:
   - lab2中，内存访问延迟的周期数将会变为随机值，需要根据`data_ok`信号来判断指令/数据访存请求是否完成
   - 运行`make test-lab2`，在输出中能看到 HIT GOOD TRAP 即为测试通过

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
└── regfile       // 寄存器文件
```

手绘电路图:



trint、swint、exint 未用到，现省略

三、加载/存储指令实现方案

由于新增了加载/存储指令，所以lab2除了基础的decode阶段，最主要就是要修改memory阶段。

通过`msize_t`和`mem_unsigned`枚举内存操作类型（B/H/W/D及无符号变体），从而配置访问位宽。

指令类型	代表指令	msize_t	访问粒度	mem_unsigned
字节加载	LB/LBU	MSIZE1	8位	LB=0, LBU=1
半字加载	LH/LHU	MSIZE2	16位	LH=0, LHU=1
字加载	LW/LWU	MSIZE4	32位	LW=0, LWU=1
双字加载	LD	MSIZE8	64位	0
字节存储	SB	MSIZE1	8位	—
半字存储	SH	MSIZE2	16位	—
字存储	SW	MSIZE4	32位	—
双字存储	SD	MSIZE8	64位	—
默认处理	非内存操作指令	MSIZE8	-	—

后面生成总线请求信号`dreq`，具体参考下表。然后将`msize_t`和`mem_unsigned`这两个信息传递给数据对齐处理单元，通过地址偏移量实现非对齐访问的校正。其中，数据对齐单元在2023年的课程网站中已经给好了，就是`readdata.sv`和`writedata.sv`这两个文件，所以直接复制以后稍加修改就可以用了。

信号/操作	字节读（LB/LBU等）	字节写（SB等）	非读写操作
dreq.valid	dataE.valid	dataE.valid	0
dreq.addr	dataE.result（原始地址）	dataE.result（原始地址）	0
dreq.size	MSIZE1/MSIZE2等（根据指令类型）	MSIZE1/MSIZE2等（根据指令类型）	0
dreq.strobe	0	strobe(write_data中传入)	0
dreq.data	0	write_rslt	0
rslt	read_rslt	dataE.result	dataE.result（原始地址）

`readdata.sv`和`writedata.sv`解释如下（从2023课程网站中复制）：

字节读

写入的数据需要做一定的处理，将读取指定的若干字节置于低位。比如，`lbu r1, 0x13(r0)`，dbus 返回的数据是 data = mem[0x10]，写入 r1 的值是`{56'b0, data[31:24]}`。

```
module readdata(
 ......
);
 logic sign_bit;
 always_comb begin
     rd = 'x;
     sign_bit = 'x;
     unique case(msize)
         MSIZE1: begin // LB, LBU
             unique case(addr)
                 3'b000: begin
                     sign_bit = mem_unsigned ? 1'b0 : _rd[7];
                     rd = {{56{sign_bit}}, _rd[7-:8]};
                 end
                 3'b001: begin
                     sign_bit = mem_unsigned ? 1'b0 : _rd[15];
                     rd = {{56{sign_bit}}, _rd[15-:8]};
                 end
             endcase
     // ...
     endcase
 end
```

字节写

写入的数据和 strobe 需要做一定的处理。比如，`sb r1, 0x13(r0)`，strobe 置为 `8'b00001000`，data 置为 `{32'bx, r1[7:0], 24'bx}`。

```
module writedata
 import common::*;
 import decode_pkg::*; (
 ......
);
 always_comb begin
     strobe = '0;
     wd = '0;
     unique case(msize)
         MSIZE1: begin
             unique case(addr)
                 3'b000: begin
                     wd[7-:8] = _wd[7:0];
                     strobe = 8'h01;
                 end
                 3'b001: begin
                     wd[15-:8] = _wd[7:0];
                     strobe = 8'h02;
                 end
             endcase
             // ...
     endcase
 end
```

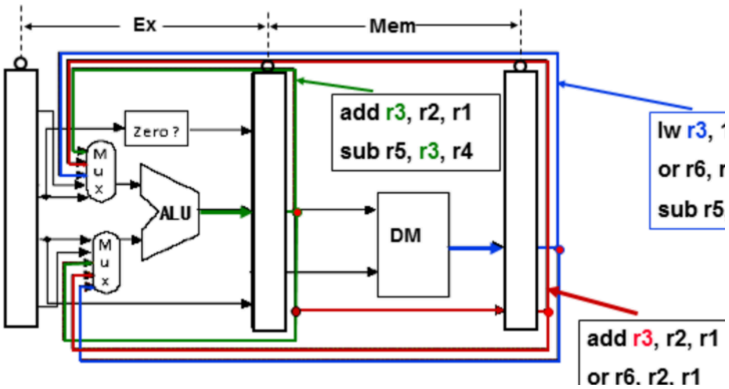
四、数据冒险处理

本次实验因为新增了加载/存储指令，所以需要处理数据冒险，这里通过数据前递（Forward）与流水线阻塞（Stall）协同处理。其中，转发信号只需要从memory阶段和write back阶段前推到decode阶段，阻塞则是从memory阶段传递到前面所有阶段（也就是全部阻塞）。

信号	来源阶段	作用范围
forward_m	EXE---MEM	前推到Decode
forward_w	MEM---WRB	前推到Decode

信号	来源阶段	来源内容	作用范围
stall_m	MEM	(加载/存储指令时~dresp.data_ok) & (dataE.valid)	Fetch,Decode,Execute全阶段阻塞

其中，转发机制可以解决大部分的数据冒险，尤其是lab1只有基础的运算指令的时候，如下图（PPT上给的）：



- C1(a): EX/MEM.RegWr
  and EX/MEM. RegisterRd ≠ 0
  and EX/MEM. RegisterRd=ID/EX. RegisterRs
- C1(b): EX/MEM.RegWr
  and EX/MEM. RegisterRd ≠ 0
  and EX/MEM. RegisterRd=ID/EX. RegisterRt

C2(a)=MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRs) and (MEM/WB.RegisterRd=ID/EX.RegisterRs)

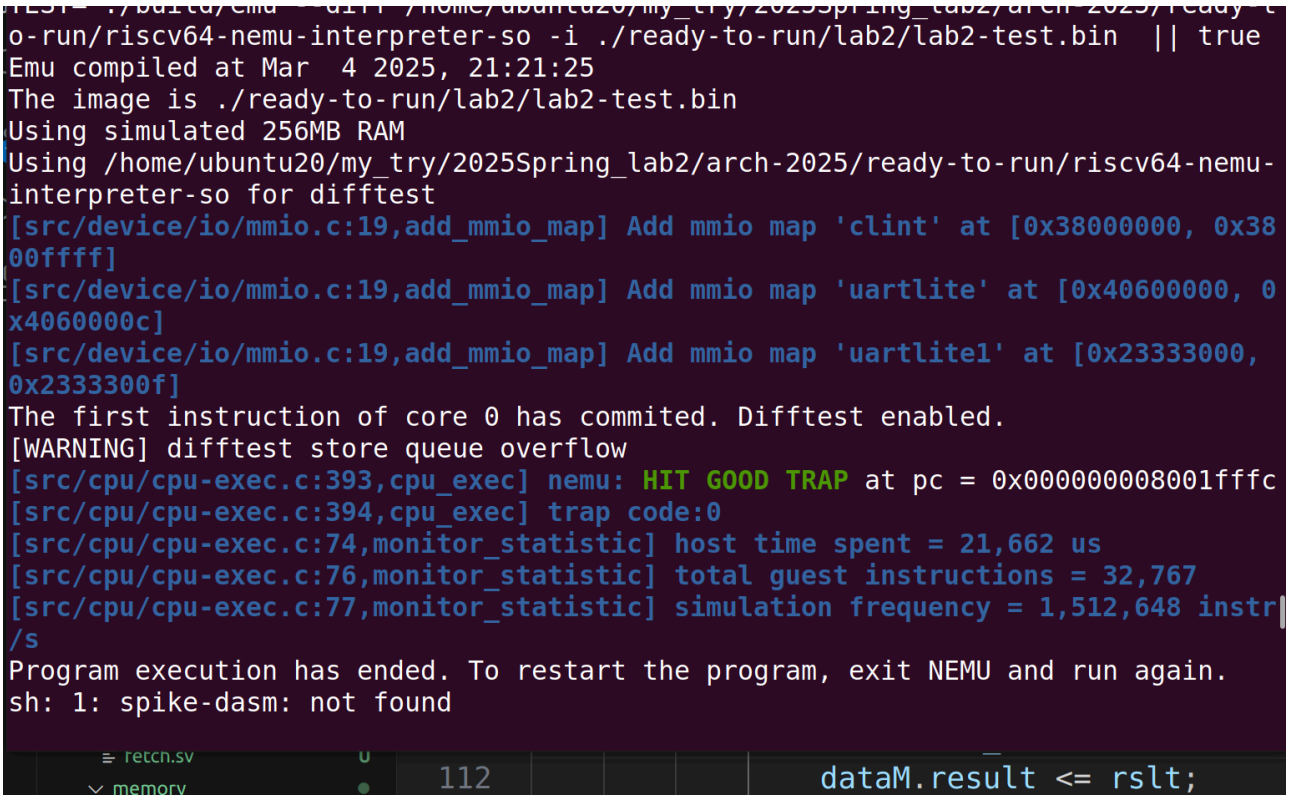
C2(b)=MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0) and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRt) and (MEM/WB.RegisterRd=ID/EX.RegisterRt)

所以一开始只用了转发。但是在更新了测试脚本以后发现过不了了（其实这样看的话上一个lab1在没更新测试脚本时候应该也是没有转发就能做的，但是之前没加转发的时候过不了，看来后面是修改的其他地方让lab1通过的，而不是转发），这是因为lab2中要求LD、SD等加载/存储指令，会导致load-use数据冒险。于是这里必须要通过阻塞来解决。如下网图（来源知乎：处理器设计Data Hazard的含义与解决方法 - 知乎）：

当遇到`load-use`类顺序的指令时，由于存储器数据在MEM阶段结束才能获得，而后续指令的EXE阶段在周期初即需操作数，此时前递无法解决，必须阻塞一个周期的流水线来确保数据有效。

五、实验结果

- 实现RV64I指令：ld sd lb lh lw lbu lhu lwu sb sh sw lui
- 通过Lab2测试


o-run/riscv64-nemu-interpreter-so -i ./ready-to-run/lab2/lab2-test.bin || true
Emu compiled at Mar 4 2025, 21:21:25
The image is ./ready-to-run/lab2/lab2-test.bin
Using simulated 256MB RAM
Using /home/ubuntu20/my_try/2025Spring_lab2/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, 0x3800ffff]
[src/device/io/mmio.c:19,add_mmio_map] Add mmio map 'uartlite' at [0x40600000, 0x4060000c]
[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 = 0x000000008001fffc
[src/cpu/cpu-exec.c:394,cpu_exec] trap code:0
[src/cpu/cpu-exec.c:74,monitor_statistic] host time spent = 21,662 us
[src/cpu/cpu-exec.c:76,monitor_statistic] total guest instructions = 32,767
[src/cpu/cpu-exec.c:77,monitor_statistic] simulation frequency = 1,512,648 instr/s
Program execution has ended. To restart the program, exit NEMU and run again.
sh: 1: spike-dasm: not found
retcn.sv
memory
112
dataM.result <= rslt;