Lab 3 实验报告

一、设计目标

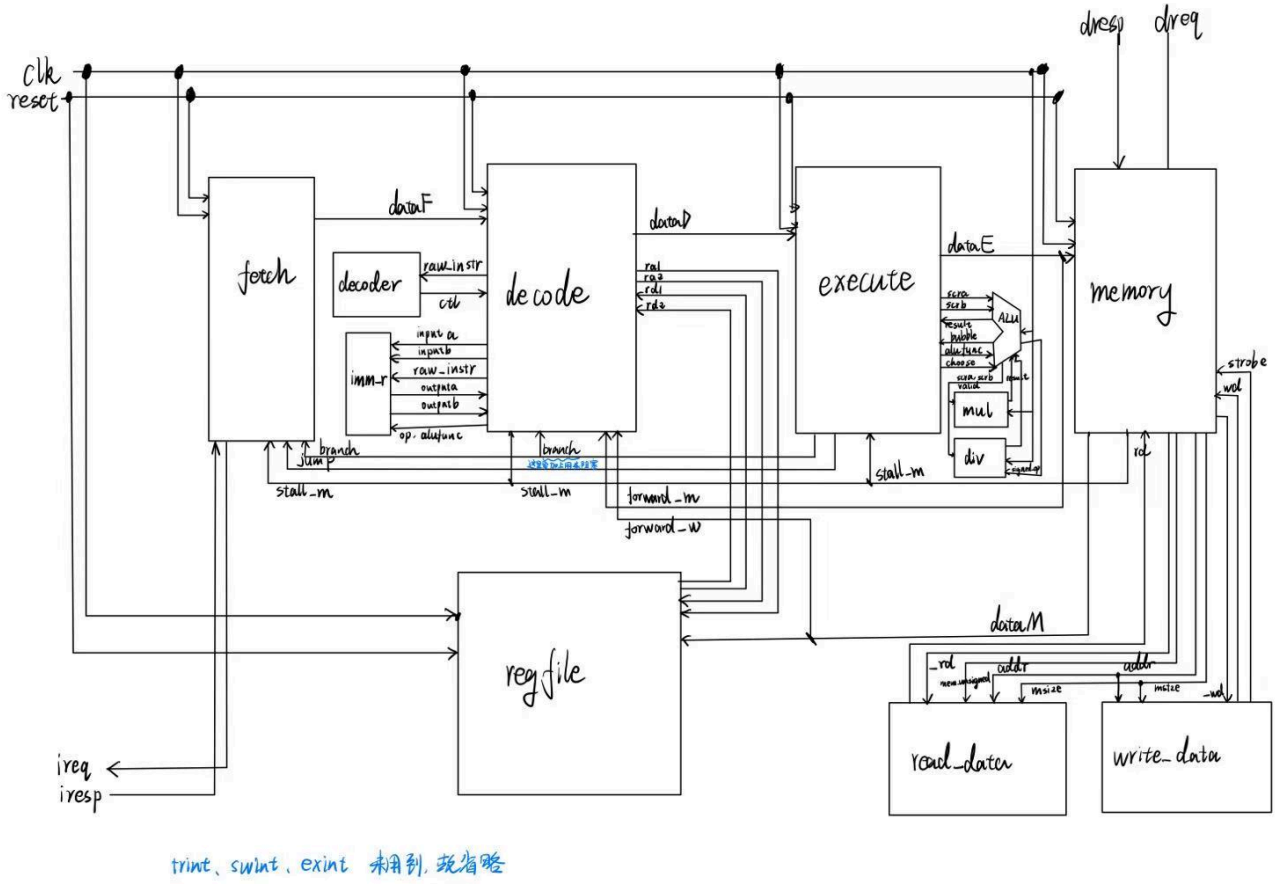
支持指令集:

- beq bne blt bge bltu bgeu slti sltiu slli srli srai sll slt sltu srl sra slliw srliw sraiw sllw srlw sraw auipc jalr jal

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，故省略

三、新增指令实现方案

首先要对 `DifftestInstrCommit` 进行一处修改：

```
.skip    (skip),
```

其中，

```
logic skip;
assign skip=(   dataM.ctl.op==LD||
                dataM.ctl.op==SD||
                dataM.ctl.op==LB||
                dataM.ctl.op==LH||
                dataM.ctl.op==LW||
                dataM.ctl.op==LBU||
                dataM.ctl.op==LHU||
                dataM.ctl.op==LWU||
                dataM.ctl.op==SB||
                dataM.ctl.op==SH||
                dataM.ctl.op==SW    ) && (dataM.addr[31]==0);
```

这里的含义是：如果当前指令是内存读写指令，并且读写的内存地址[31]位为0，就让 Difftest 跳过对这条指令的正确性判断。

移位指令（slli, srli, srai, sll, srl, sra, slliw, srliw, sraiw, sllw, srlw, sraw）

属于较为基础的运算指令，更改decoder之后在alu中按照指令要求加上操作就好了。

```
ALU_SLL: c = a<<b[5:0];
ALU_SRL: c = a>>b[5:0];
ALU_SRA: c = $signed(a)>>>b[5:0];
```

比较指令（slt, sltu,slti, sltiu）

同样属于较为基础的运算指令，alu中加入两种操作

```
ALU_SLT: c= {63'b0,( $signed(a) < $signed(b) )};
ALU_SLTU: c={63'b0,( a < b )};
```

由于后面分支指令要用，所以alu操作再加一个EQUAL

```
ALU_EQUAL: c ={63'b0, (a==b)};
```

分支指令（beq, bne, blt, bge, bltu, bgeu）

这里为了实现跳转需要在execute阶段加jump和branch传出来，jump传给fetch，branch传给fetch和decode。jump是将计算得到的跳转目标传给fetch，branch则是告知fetch要跳转并且对execute之前的这两个阶段进行阻塞。

branch是只要设计分支或者跳转指令并且dataD有效那么就置为一，如果分支条件不成立的话那么就把jump正常设置为pc+4即可。

由于都是条件跳转，所以要通过上面实现好了的alu的比较操作进行条件判断

控制信号 `ctl.op`	**ALU 指令** `ctl.alufunc`
BEQ (Branch if Equal)	ALU_EQUAL
BNE (Branch if Not Equal)	ALU_EQUAL
BLT (Branch if Less Than, signed)	ALU_SLT
BGE (Branch if Greater or Equal, signed)	ALU_SLT
BLTU (Branch if Less Than, unsigned)	ALU_SLTU
BGEU (Branch if Greater or Equal, unsigned)	ALU_SLTU

BLT和BGE都是用的ALU_SLT，区别是在execute的时候若alu结果为1则BLT执行跳转，即`jump = dataD.pc + {{51{r_ins[31]}}, {r_ins[31]}, {r_ins[7]}, {r_ins[30:25]}, {r_ins[11:8]},{1'b0}};`，否则正常`jump = pc + 4`;若alu结果为0则BGE执行跳转，否则正常`jump = pc + 4`。BEQ和BNE同理。

跳转指令（jal, jalr）

都是用于无条件跳转，并且它们都会将返回地址存入寄存器。

功能大致如下：

指令	计算方式
JAL （Jump And Link）	`rd = PC + 4`； `jump = pc + offset;`
JALR （Jump And Link Register）	`rd = PC + 4`； `jump = rs + offset;`

所以两者的alu操作都是加法，并且两个的操作数都是PC和4，至于要传入的rd，就是正常的raw_instr[11:7]，置regwrite为1即可。因为这里需要对PC做加法了，所以需要修改decode阶段的imm_r，把当前PC也传进去。

对于jump的计算则在execute阶段直接执行：

```
JAL:
    jump = dataD.pc + {{43{r_ins[31]}}, {r_ins[31]}, {r_ins[19:12]} ,{ r_ins[20]} ,
{r_ins[30:21]},{1'b0}};
JALR:
    jump = dataD.rd1 + {{52{r_ins[31]}}, r_ins[31:20]};
```

计算地址（auipc）

用于生成PC的相对地址存到寄存器里，`rd = PC + sext(imm << 12)`，所以alu操作为加法，操作数为PC和sext(imm << 12)，然后置regwrite为1即可。

四、数据冒险处理

本次实验数据冒险的处理与lab2相同。

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

五、实验结果

- 实现RV64I指令：beq bne blt bge bltu bgeu slti sltiu slli srli srai sll slt sltu srl sra slliw srliw sraiw sllw srlw sraw auipc jalr jal

- 通过Lab3测试

```
----------------------------------------------------------------
Solution Validates: avg error less than 1.000000e-13 on all three arrays
----------------------------------------------------------------
Run conwaygame
Play Conway's life game for 200 rounds.
seed=6052392









            *          **
            *         * *
            *         **
[src/cpu/cpu-exec.c:393,cpu_exec] nemu: HIT GOOD TRAP at pc = 0x00000000800152c0
[src/cpu/cpu-exec.c:394,cpu_exec] trap code:0
[src/cpu/cpu-exec.c:74,monitor_statistic] host time spent = 29,947,511 us
[src/cpu/cpu-exec.c:76,monitor_statistic] total guest instructions = 59,159,331
```