Lab 3 实验报告

潘文峥（520030910232）

一、 实验目的

1. 掌握不同类型指令在数据通路中的执行路径；
2. 掌握 Vivado 仿真方式。

二、 实验平台

1. 计算机 1 台(尽可能达到 8G 及以上内存)；
2. Xilinx Vivado 开发套件(2020.2 版本)。

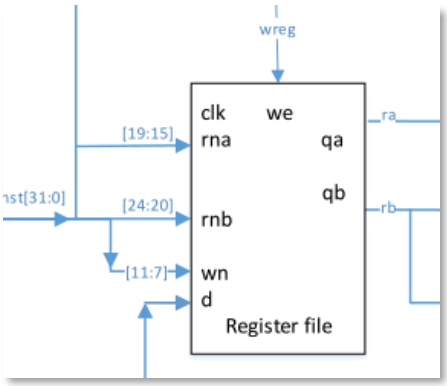
三、 任务步骤

1. 从前序实验中导入已完成的各个模块本次实验中所需完成的模块，并完成指令 rom 和数据 ram 的例化。最终形成的文件树如图 1 所示。

图 1：文件树

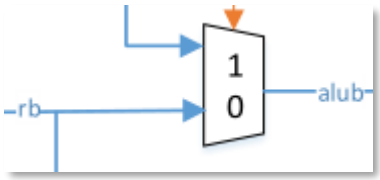
2. 完成 sc_cpu.v 文件的代码实现。这一部分是本实验的关键，需要根据不同指令类型构建数据通路，将前序实验中已经完成的立即数产生模块、alu 模块、控制指令生成模块等子模块连接起来，使 cpu 可以正常连续工作。

以下展示各部分数据传递关系和相应的代码实现：（已提供的部分略去）



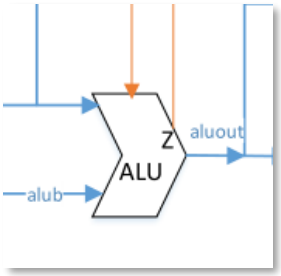

```
regfile rf(
    .clk(clock),
    .clrn(resetn),
    .rna(inst[19:15]),
    .rnb(inst[24:20]),
    .qa(ra),
    .qb(data),
    .wn(inst[11:7]),
    .we(wreg),
    .d(regf_din)
);
```

图 2：寄存器堆的实现



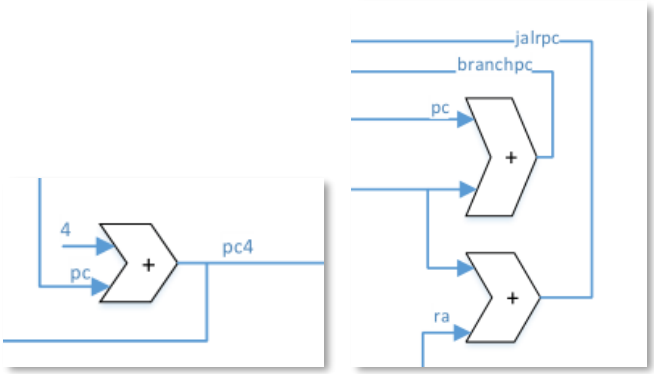

```
mux2x32 alu_b(
    .s(aluimm),
    .a1(immediate),
    .a0(data),
    .y(alub)
);
```

图 3：根据 aluimm 指令决定 alub 的取值



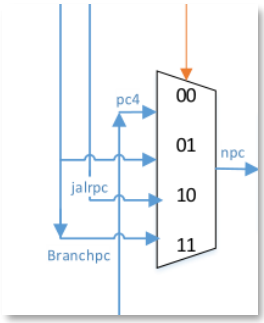

```
alu al_unit(
    .a(ra),
    .b(alub),
    .aluc(aluc),
    .s(aluout),
    .z(zero)
);
```

图 4：ALU 模块的实现



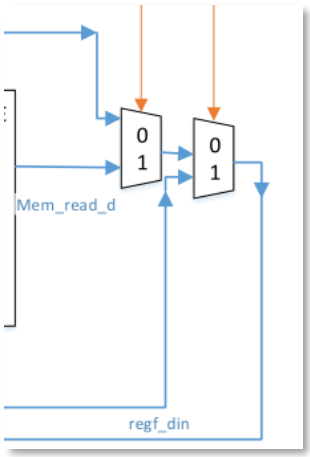

```
cla32 pcplus4(
    .pc(pc),
    .x1(32'h4),
    .p4(p4)
);

cla32 branch_adr(
    .pc(pc),
    .x1(immediate),
    .p4(branchpc)
);

cla32 genjalrpc(
    .pc(ra),
    .x1(immediate),
    .p4(jalrpc)
);
```

图 5：三个加法器：分别计算分支/跳转/顺序情况下的下一条指令地址




```
mux4x32 nextpc(
    .s(pcsource),
    .a0(p4),
    .a1(branchpc),
    .a2(jalrpc),
    .a3(branchpc),
    .y(npc)
);
```

图 6：四路选择器：根据 pcsource 给出下一条指令地址




```
wire [31:0]tmp;
//write back register file,   mux2x32
mux2x32 result(
    .s(m2reg),
    .a1(mem),
    .a0(aluout),
    .y(tmp)
);

mux2x32 link(
    .s(pcsource[1]),
    .a0(tmp),
    .a1(p4),
    .y(regf_din)
);
```

图 7：两个二路选择器：根据 m2reg 和 pcsource[1]决定写回寄存器的内容

四、 出现的问题与解决方案

根据仿真结果中出现的不符合预期的情况，反复调试，找到实验 1、2 中部分问题代码加以修改。

（1）

```
// 修改前： assign branchpc_offset = {12'b0,inst[7],inst[30:25],inst[11:8],1'b0};
assign branchpc_offset = {{19{e}},inst[31], inst[7],inst[30:25],inst[11:8],1'b0};
```

图 8：immext.v 中，分支指令地址偏移应作符号拓展

（2）

```
 (aluc == 4'b1101)? ( (a[31] == 1'b0) ? (a>>b): ({32{1'b1}}<<(32-b))+(a>>b)):
//  原代码：  (aluc == 4'b1101)? ($signed(a))>>>b:
```

图 9：alu 模块中算数右移按原有方式会出错，故手动拼接实现

（3）

```
assign aluimm = i_addi | i_andi | i_ori | i_xori | i_slli |
                i_srli | i_srai | i_lui | i_lw | i_sw;
```

图 10：原 sc_cu 模块中的 aluimm 控制信号漏掉了 lw 和 sw

（4）应特别注意拼写问题：vivado 平台对变量名拼写错误导致的问题不会显式报错，只能通过仿真中出现的问题进行排查分析找到错误，耗费大量时间。

（5）例化 ram 过程中读写位宽选择错误会导致 sw 和 lw 的执行不能得到正确结果，经检查更正为 32bits 即可。

五、 仿真结果演示

图 11：仿真结果（1）

从仿真结果（1）可以看出，1515ns 后，指令正确执行完毕。最后的指令地址为 0000007c；最后一条指令为 0000006f；此时 x18 寄存器的值为 0001ffff。均与实验 2.1 中仿真表给出的结果一致，达到了实验目的。

图 12：仿真结果（2）

从仿真结果（2）可以看到，790ns 后，寄存器 x19 的值为 0000000a，说明处理器正确执行了 sw、lw 指令，并在四次循环过程中正确得到了 1+2+3+4 的结果。