实验 6 报告

学号：2017K8009922027、2017K8009929011
姓名：张磊、郭豪
箱子号：39

一、实验任务（10%）

任务：在实验 5 的 CPU 代码基础上，加入算数逻辑运算指令，乘除运算指令，以及乘除法配套的数据搬运指令，运行 func_lab6，要求成功通过仿真和上板验证；

二、实验设计（40%）

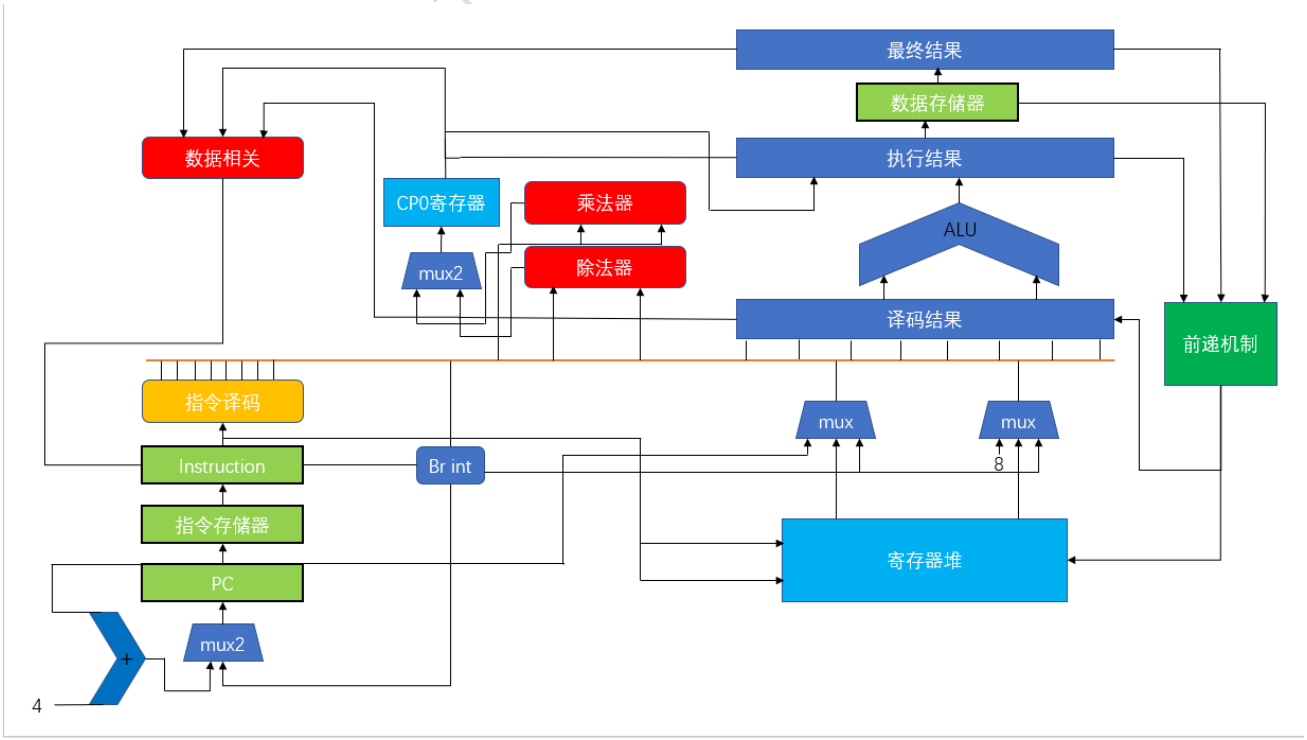
（一）总体设计思路

1. 在译码阶段，通过 op，func，sa 的译码结果完成对新添加指令的译码，并尽可能的复用之前的 CPU 设计控制通路和数据通路；

2. 对于乘法，直接采用“*”完成有符号乘法和无符号乘法，对于除法，由于采用“/”和“%”综合器调用的 IP 会占用很多 LUT 资源，所以，我们采用自制 IP 的方法；

3. 在 EXE 阶段定义 reg 类型信号 CP0_HI，CP0_LO 作为乘除法所需的 HI 和 LO 寄存器，mfhi，mflo，mthi，mtlo 指令对 CPO_HI，CP0_LO 的读写也都在执行阶段。

（二）总体设计思路



（三）重要模块 1 设计：指令译码模块；

```
// lab6 append
assign inst_add    = op_d[6'h00] & func_d[6'h20] & sa_d[5'h00];
assign inst_addi   = op_d[6'h08];
assign inst_sub    = op_d[6'h00] & func_d[6'h22] & sa_d[5'h00];
assign inst_slti   = op_d[6'h0a];
assign inst_sltiu  = op_d[6'h0b];
assign inst_andi   = op_d[6'h0c];
assign inst_ori    = op_d[6'h0d];
assign inst_xori   = op_d[6'h0e];
assign inst_sllv   = op_d[6'h00] & func_d[6'h04] & sa_d[5'h00];
assign inst_srlv   = op_d[6'h00] & func_d[6'h06] & sa_d[5'h00];
assign inst_srav   = op_d[6'h00] & func_d[6'h07] & sa_d[5'h00];
assign inst_mult   = op_d[6'h00] & func_d[6'h18] & (mudi == 10'b0);
assign inst_multu  = op_d[6'h00] & func_d[6'h19] & (mudi == 10'b0);
assign inst_div    = op_d[6'h00] & func_d[6'h1a] & (mudi == 10'b0);
assign inst_divu   = op_d[6'h00] & func_d[6'h1b] & (mudi == 10'b0);
assign inst_mfhi   = op_d[6'h00] & func_d[6'h10] & sa_d[5'h00];
assign inst_mflo   = op_d[6'h00] & func_d[6'h12] & sa_d[5'h00];
assign inst_mthi   = op_d[6'h00] & func_d[6'h11] & (mvto == 15'b0);
assign inst_mtlo   = op_d[6'h00] & func_d[6'h13] & (mvto == 15'b0);
```

1、工作原理

利用译码器对 op，func，sa 的译码结果，对相应的位作出判断，完成相应指令的译码结果；

```
decoder_6_64 u_dec0(.in(op  ), .out(op_d  ));
decoder_6_64 u_dec1(.in(func), .out(func_d));
decoder_5_32 u_dec2(.in(rs  ), .out(rs_d  ));
decoder_5_32 u_dec3(.in(rt  ), .out(rt_d  ));
decoder_5_32 u_dec4(.in(rd  ), .out(rd_d  ));
decoder_5_32 u_dec5(.in(sa  ), .out(sa_d  ));
```

2、功能描述

完成 lab6 新添加指令的译码；

（四）重要模块 2 设计：除法器 IP 及 HI、LO 寄存器模块

自制 IP 完成有符号数，无符号数的除法（这里仅以 DIV 指令举例）；

1、工作原理

调用自制的除法器 IP：

```
my_div u_my_div(
      .aclk (clk),
      .s_axis_divisor_tvalid (div_valid),
      .s_axis_divisor_tready  (div_divisor_ready),
      .s_axis_divisor_tdata  (mudi_src2),
      .s_axis_dividend_tvalid (div_valid),
      .s_axis_dividend_tready (div_dividend_ready),
      .s_axis_dividend_tdata (mudi_src1),
      .m_axis_dout_tvalid (div_done),
      .m_axis_dout_tdata (div_result)
    );
```

完成除法器握手信号的编写：

```
always @(posedge clk)
begin
    if(reset) begin
        div_valid <= 1'b0;
    end
    else if(div_valid & div_ready) begin
        div_valid <= 1'b0;
    end
    else if(ds_to_es_valid && es_allowin) begin
        div_valid <= ds_to_es_bus[142:142];
    end
end
```

完成执行阶段乘法器和除法器，以及 mtohi，mtlo 修改 cp0_hi，cp0_lo 寄存器的时序逻辑；

```
always @(posedge clk)
begin
    if(reset) begin
        cp0_hi <= 32'b0;
        cp0_lo <= 32'b0;
    end
    else if(es_inst_mult) begin
        cp0_hi <= signed_prod[63:32];
        cp0_lo <= signed_prod[31:0];
    end
    else if(es_inst_multu) begin
        cp0_hi <= unsigned_prod[63:32];
        cp0_lo <= unsigned_prod[31:0];
    end
    else if(es_inst_div & div_done) begin
        cp0_lo <= div_result[63:32];
        cp0_hi <= div_result[31:0];
    end
    else if(es_inst_divu & divu_done) begin
        cp0_lo <= divu_result[63:32];
        cp0_hi <= divu_result[31:0];
    end
    else if(es_inst_mthi) begin
        cp0_hi <= es_rs_value;
    end
    else if(es_inst_mtlo) begin
        cp0_lo <= es_rs_value;
    end
```

修改执行阶段的 ready_go 信号，确保除法未完成的过程被阻塞；

```
assign es_ready_go = es_inst_div ? div_done :
                     es_inst_divu ? divu_done :
                                    1'b1;
```

直接使用 * 实现有符号乘法和无符号乘法：

```
assign mudi_src1 = es_rs_value;
assign mudi_src2 = es_rt_value;
assign unsigned_prod = mudi_src1 * mudi_src2;
assign signed_prod = $signed(mudi_src1) * $signed(mudi_src2);
```

2、功能描述

对于乘法，我们直接用“*”，即调用乘法器的 IP 实现，并分别产生有符号乘法和无符号乘法两种结果，然后用二选一选择器对结果进行选取；

对于除法，我们采用自制 IP 的方式，创建了两个除法器 IP，其中一个执行无符号除法，另一个执行有符号除法；

对于 CP0_HI，CP0_LO 寄存器的修改，我们统一到执行阶段，这样既避免了寄存器写后读相关，有方便了乘除法结果的保存；

三、实验过程（50%）

（一）实验流水账

1. 2019 年 10 月 13 日 9:30-10:20 完成指令译码；
2. 2019 年 10 月 13 日 10:30-13:20 完成 rtl 代码编写（包含除法器的实现）；
3. 2019 年 10 月 13 日 14:30-16:30 完成 debug；
4. 2019 年 10 月 13 日 16:50-17:10 完成上板验证；

（二）错误记录

1、错误 1：除法结果的商和余数位置弄反

（1）错误现象

Mfhi 指令读取 CP0_HI 寄存器时结果错误；

```
[ 845507 ns] Error!!!
    reference: PC = 0xbfc46f7c, wb_rf_wnum = 0x15, wb_rf_wdata = 0x00000002
    mycpu    : PC = 0xbfc46f7c, wb_rf_wnum = 0x15, wb_rf_wdata = 0x15b8b7a4
--------------------------------------------------------------
```

（2）分析定位过程

查阅 test.S 代码，发现对应 PC 为 mfhi 指令，且之前执行了 DIV 指令，因此检查执行阶段的 CP0_HI 的数值，发现 CP0_HI 与 CP0_LO 指令的数据刚好相反，所以修改除法之后对除法结果的寄存器的赋值；

（3）错误原因

除法结果存放反了；

```
else if(es_inst_div & div_done) begin
    cp0_hi <= div_result[63:32];
    cp0_lo <= div_result[31:0];
end
else if(es_inst_divu & divu_done) begin
    cp0_hi <= divu_result[63:32];
    cp0_lo <= divu_result[31:0];
end
```

（4）修正效果

修改存放的位置，成功；

```
else if(es_inst_div & div_done) begin
    cp0_lo <= div_result[63:32];
    cp0_hi <= div_result[31:0];
end
else if(es_inst_divu & divu_done) begin
    cp0_lo <= divu_result[63:32];
    cp0_hi <= divu_result[31:0];
end
```

2、**错误 2**：有符号乘法和无符号乘法结果弄反

（1）**错误现象**

乘法之后 mfhi 指令读取的结果不正确；

```
[1013197 ns] Error!!!
    reference: PC = 0xbfc4b120, wb_rf_wnum = 0x16, wb_rf_wdata = 0xf4d903bb
    mycpu    : PC = 0xbfc4b120, wb_rf_wnum = 0x16, wb_rf_wdata = 0x3a920af3
-----------------------------------------------------------------
```

（2）**分析定位过程**

直接来到执行阶段，查看乘法结果，发现 reference 的数值与有符号的结果相同；

（3）**错误原因**

乘法结果的有符号数和无符号数选择弄反了；

```
else if(es_inst_mult) begin
    cp0_hi <= unsigned_prod[63:32];
    cp0_lo <= unsigned_prod[31:0];
end
else if(es_inst_multu) begin
    cp0_hi <= signed_prod[63:32];
    cp0_lo <= signed_prod[31:0];
end
```

（4）**修正效果**

修改后成功；

3、**错误 3**：除法器握手信号错误

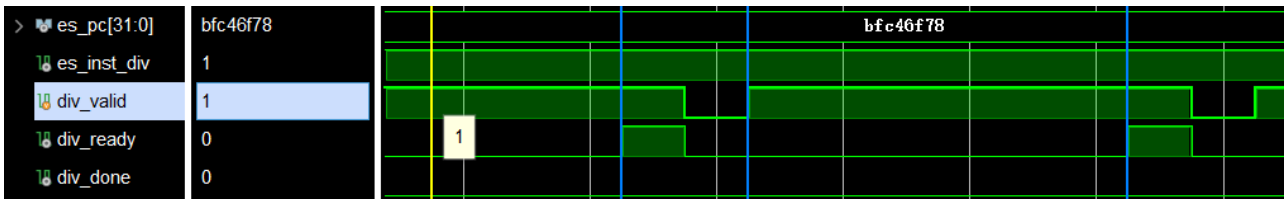
（1）**错误现象**

除法器只执行了一次除法后就不能继续执行第二次除法；

```
[ 845667 ns] Error!!!
    reference: PC = 0xbfc46fb4, wb_rf_wnum = 0x15, wb_rf_wdata = 0x00000000
    mycpu    : PC = 0xbfc46fb4, wb_rf_wnum = 0x15, wb_rf_wdata = 0x00000002
-----------------------------------------------------------------
```

（2）**分析定位过程**

查看波形，发现执行阶段为除法指令时，除法的 valid 信号还没有和除法器的 ready 信号成功握手，就直接去执行下一条指令了，导致这次取出的除法结果还是上一次的结果；



可以看到 div_valid 信号在握手成功被置为 0 后又被置为 1；

（3）错误原因

除法器握手信号错误；

```
    else if(es_valid & es_inst_div) begin
        div_valid <= 1'b1;
    end
```

（4）修正效果

采用执行级的指令进入信号拉高 valid 信号，成功通过；

```
always @(posedge clk)
begin
    if(reset) begin
        divu_valid <= 1'b0;
    end
    else if(divu_valid & divu_ready) begin
        divu_valid <= 1'b0;
    end
    else if(ds_to_es_valid && es_allowin) begin
        divu_valid <= ds_to_es_bus[141:141];
    end
end
```

4、错误 4：除法握手信号错误

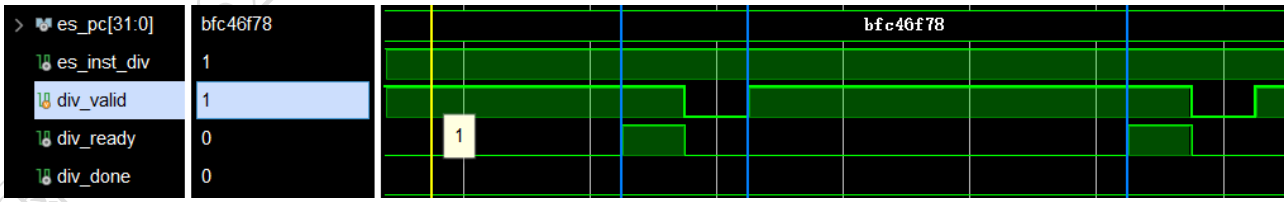
（1）错误现象

除法执行报出 wb_rf_wdata 错误

```
[ 845667 ns] Error!!!
    reference: PC = 0xbfc46fb4, wb_rf_wnum = 0x15, wb_rf_wdata = 0x00000000
    mycpu    : PC = 0xbfc46fb4, wb_rf_wnum = 0x15, wb_rf_wdata = 0x00000002
------------------------------------------------------------------
```

（2）分析定位过程

查看波形，发现执行阶段为除法指令时，除法的 valid 信号还没有和除法器的 ready 信号成功握手，就直接去执行下一条指令了，导致这次取出的除法结果还是上一次的结果；



可以看到 div_valid 信号在握手成功被置为 0 后又被置为 1；

（3）错误原因

除法器握手信号错误；

```
    else if(es_valid & es_inst_div) begin
        div_valid <= 1'b1;
    end
```

（4）修正效果

采用执行级的指令进入信号拉高 valid 信号，成功通过；

```
always @(posedge clk)
begin
    if(reset) begin
        divu_valid <= 1'b0;
    end
    else if(divu_valid & divu_ready) begin
        divu_valid <= 1'b0;
    end
    else if(ds_to_es_valid && es_allowin) begin
        divu_valid <= ds_to_es_bus[141:141];
    end
end
```

四、实验总结（可选）

1. 除法调用过程的主要难点在于握手信号 valid 和 ready 的处理上，当满足除法指令且 ds_to_es_valid 和 es_allowin 握手成功时，将 valid 信号拉高，当 IP 返回的 ready 信号为 1 时，需要在下一拍将 valid 信号清空为 0。为了保证被除数和除数的 valid 信号需要同时变化,可以直接对除数和被除数使用同一个 valid 信号。

2. 乘除法结束后，乘法结果的高 32 位写入 HI 寄存器，低 32 位写入 LO 寄存器。除法的商写入 LO 寄存器，余数写入 HI 寄存器。该部分通过在 EXE 阶段设置两个 reg 类型的 HI 和 LO 变量当作寄存器，通过时序逻辑的方式实现写入的操作。