实验报告成绩：	成绩评定日期：

2021 ~ 2022 学年秋季学期

A3705060050《计算机系统》必修课

课程实验报告

班级：人工智能 1901

组长：李语竹 20195281

组员：孔繁恒 20195250

郭晶晶 20195271

报告日期：2021.12.18

目录

一. 项目简介

1.1 任务分工（含实现指令）

任务概况：

最终过了 pass poin 64, 添加了自制乘法器

1. 逻辑操作指令（实现 or、xor、xori、nor、and、andi 指令）
2. 移位操作指令（实现 sll、sllv、srav、srl、srlv 指令）
3. 流水线数据相关问题（实现 EX、MEM 段到 ID 段的定向 forwarding 路径）
4. 算术操作指令 I
   加减：（实现 add、addi、addiu、addu、sub、subu 指令）
   比较：（实现 slt、slti、sltiu、sltu 指令）
5. 转移指令
   跳转指令（实现 jr、jalr、j、jal）
   分支指令（实现 beq、bgez、bgezal、bgtz、blez、bltz、bltzal、bne）
6. 加载存储指令 I（实现 lw、sw）
7. 插入暂停气泡（解决 forwarding 技术无法解决的问题——load 相关）
8. 算术操作指令 II
   乘除：（实现 div、divu、mult、multu 指令）
   自制移位乘法器（包括 32 个周期暂停的实现）
9. 移动操作指令（实现 mfhi、mthi、mflo、mtlo 指令）
10. 加载存储指令 II（实现 lb、lbu、lh、lhu、sb、sh）

表 1 任务分工

姓名	完成内容
李语竹	EX、MEM 段到 ID 的定向路径、WB 到 regfile 通路、 添加 lw, sw, sltu, bne, slt, slti, sltiu, sltu, bne, slt, slti, sltiu, sltu, bne, slt, slti, sltiu, mthi, mtlo 指令、 插入暂停气泡（乘除法器暂停）、乘除法器接入以及带来的数据相关。
孔繁恒	添加 addu, or, sll, j, add, sub, and, andi, nor, xori, sllv, sra, srav, srl, srlv, sb, sh（区别于 sw 的不同处理）指令、 自制乘除法器（尝试整合在一个文件）。
郭晶晶	添加了 subu, jal, jr, xor, bgez, bgtz, blez, bltz, bltzal, bgezal, jalr, lb, lbu, lh, lhu 指令、 stall 停止控制（load 暂停和数据相关）。

1.2 总体设计

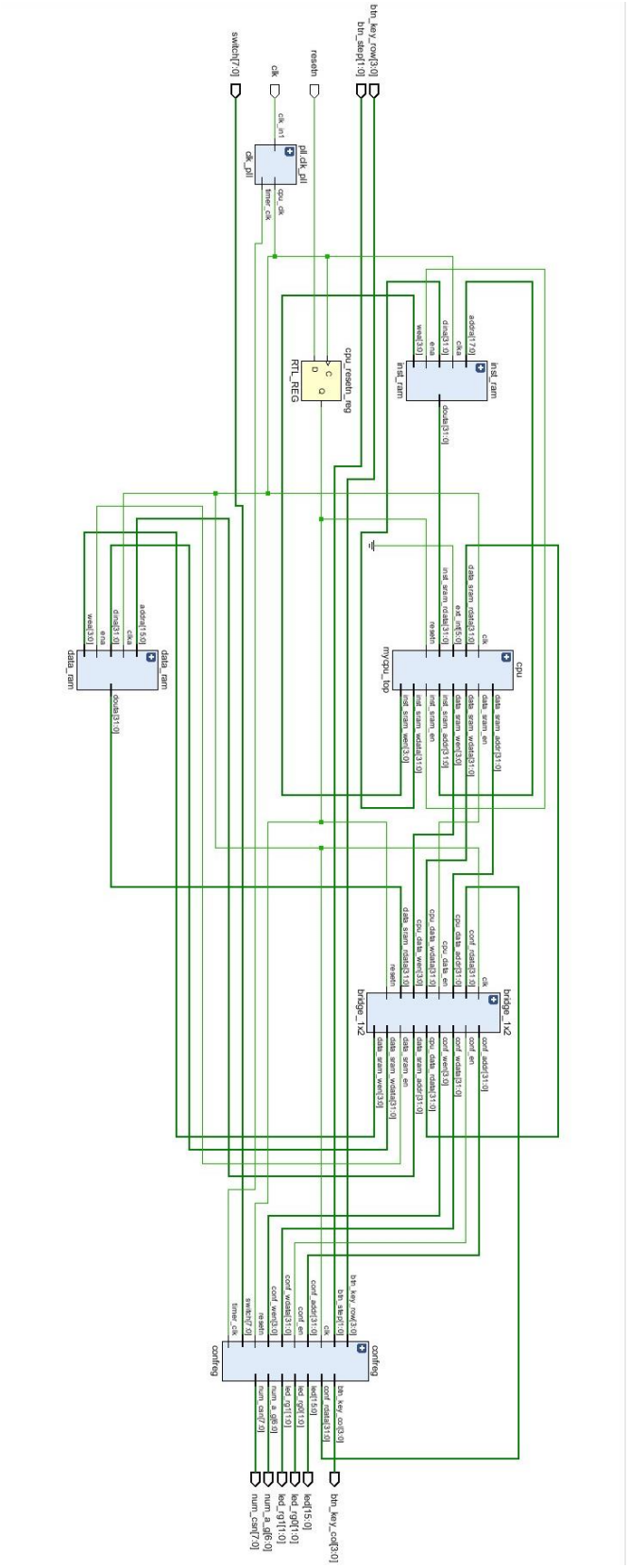


图 1 总线路图

该 CPU 包括输出时钟周期信号的 pll.clk_pll 部分、指令存储器 inst_ram 部分、数据存储器 data_ram 部分、输出复位信号的 cpu_resetn_reg 部分、cpu 部分、内存虚地址映射 bridge_1x2 部分、confreg 部分，通过各部分的输入输出端口、使能信号实现交互。

1.3 不同流水段之间连线图

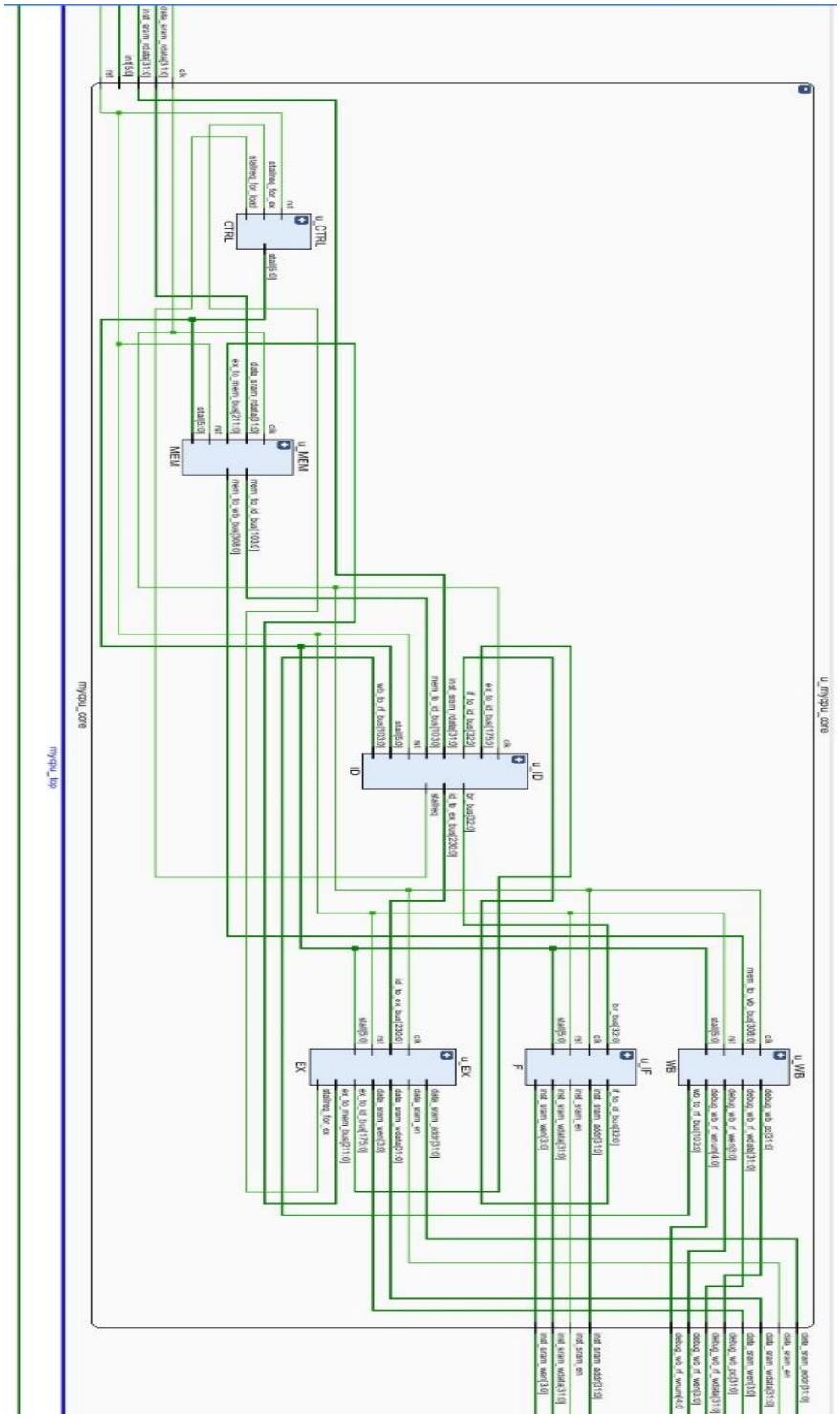


图 2 流水段之间的连线图

五段流水线 IF（取指）、ID（译码）、EX（执行）、MEM（访存）、WB（写回）由时钟信号 clk、复位信号 rst 以及 6 位暂停信号 stall[5:0]共同控制。

首先，IF 段根据当前 PC 值取指，给指令存储器传入 inst_sram_en（指令存储器使能）、inst_sram_addr（要读取的指令地址），指令寄存器再以输出信号 inst_sram_rdata 传入 ID 段；同时通过 ID 段传入的分支 br_bus[32:0]判断是否成功，判断 next_pc 的增值（加 4 或者跳转地址 br_addr）。

其次，ID 段接收：读取的指令 inst_sram_rdata，IF 段传入值 if_to_id_bus、EX 段（ex_to_id_bus）和 MEM 段（mem_to_id_bus）定向 forwarding 路径传入值以及 WB 段写回寄存器中值（wb_to_rf_bus）。

根据三种指令格式 I/J/R，从 inst_sram_rdata 取出 opcode、rs、rt、rd、sa、func、imm、instr_index、code、base、offset、sel；

与寄存器 regfile 进行数据交互（包括当前指令的读和之前指令的写，也包括普通寄存器和 hilo 寄存器），在读数据的时候，需要处理定向路径能够解决的数据相关；以及不能解决的 load 相关，需要输出 stallreq 用于插入暂停气泡，并在暂停时设置寄存器暂存下一条指令；

根据读取的寄存器值，提前判断分支是否成功 br_e 以及计算跳转地址 br_addr，用于输出端口 br_bus，输出给 IF 段用以 next_pc 赋值。

接着，EX 段执行运算操作，将 ID 段读取的源操作数传入 alu 或者乘除法器模块（乘除法器需要通过 stallreq_for_ex 输出信号实现暂停）；同时，计算访存地址，处理写内存的一系列指令，通过 data_sram_en（内存使能）、data_sram_addr（访存地址）、data_sram_wen（内存写使能）、data_sram_wdata（内存写入数据）等输出信号，实现与数据存储器的交互；

作为 ex_to_id_bus 定向路径的输入端，将计算得到的结果和相关使能信号直接回传。

然后，MEM 段接收输入 data_sram_rdata 值以及 EX 段传来的端口 ex_to_mem_bus，实现对内存的读取；

作为 mem_to_id_bus 定向路径的输入端，将从内存加载得到的数据和相关使能信号直接回传给 ID 段。

最后，WB 段接收 MEM 段传来的与写寄存器相关的一系列信号 mem_to_wb_bus，通过 wb_to_rf 输出，传递给 ID 段用于与其中的 regfile 模块交互

1.4 运行环境与使用工具

1.运行环境：

Windows10

2.使用工具：

Vivado 软件 + Verilog 语言

二．单个流水段说明

2.1 取指（IF）阶段

2.1.1 整体功能说明

1.根据当前 PC 值从内存中取指令：

inst_sram_rdata（ID 段的输入） ← inst_sram [pc_reg]

2.PC 增值：（a）顺序执行：next_pc ← pc_reg＋4

（b）分支成功：next_pc ← br_addr

2.1.2 端口与信号介绍

表 2 IF 段接口

序号	接口名	宽度(bit)	输入/输出	作用
1	clk	1	输入	时钟信号
2	rst	1	输入	复位信号
3	stall	6	输入	暂停信号
4	br_bus	33	输入	分支端口，从 ID 段传出，IF 段接收
	={br_e,	1		分支使能信号，成功为 1，失败为 0
	br_addr}	32		分支成功跳转地址
5	if_to_id_bus	33	输出	IF 到 ID 段端口
	{ce_reg,	1		指令存储器使能信号
	pc_reg}	32		IF 阶段的指令对应的地址
6	inst_sram_en	1	输出	指令存储器使能信号
7	inst_sram_wen	4	输出	指令存储器写使能信号
8	inst_sram_addr	32	输出	要读取的指令地址
9	inst_sram_wdata	32	输出	指令存储器要写入的数据

其中，信号 stall 含义如下：

stall[0]表示取指地址 PC 是否保持不变，为 1 表示保持不变；

stall[1]表示流水线取指阶段是否暂停，为 1 表示暂停。

stall[3]表示流水线执行阶段是否暂停，为 1 表示暂停。

stall[4]表示流水线访存阶段是否暂停，为 1 表示暂停。

stall[5]表示流水线回写阶段是否暂停，为 1 表示暂停。

2.1.3 功能模块说明

对应 2.1.1：

1. 根据两根与指令存储器交互的输出信号 inst_sram_en, inst_sram_addr，将指令读入 inst_sram_rdata，将其在 ID 段作为输入信号；

2. 根据分支使能信号 br_e，判断是否分支成功（通过多路选择器实现）：

```
assign next_pc = br_e ? br_addr
               : pc_reg + 32'h4;
```

若成功（使能为 1），则直接转移到分支成功地址，即将 next_pc 赋值为 br_addr；

否则，顺序执行下一条指令，即将下一条读取指令的地址 next_pc+4.

3. 在 IF 段开始的时候，通过时序逻辑判断是否被复位或者暂停，代码如下：

（之后的 ID、EX、MEM、WB 段，时序逻辑判断类似）

```
always @ (posedge clk) begin
    if (rst) begin
        pc_reg <= 32'hbfbf_fffc;
    end
    else if (stall[0]==`NoStop) begin
        pc_reg <= next_pc;
    end
end

always @ (posedge clk) begin
    if (rst) begin
        ce_reg <= 1'b0;
    end
    else if (stall[0]==`NoStop) begin
        ce_reg <= 1'b1;
    end
end
```

代码 1 IF 段时序逻辑

上述代码表示，在时钟信号的控制下，如果被复位，那么 PC 值被置为上述值，指令存储器不使能（置为 0）；否则，如果取指地址 PC 变化，那么取 next_pc，指令存储器使能（置为 1）。

2.1.4 结构示意图

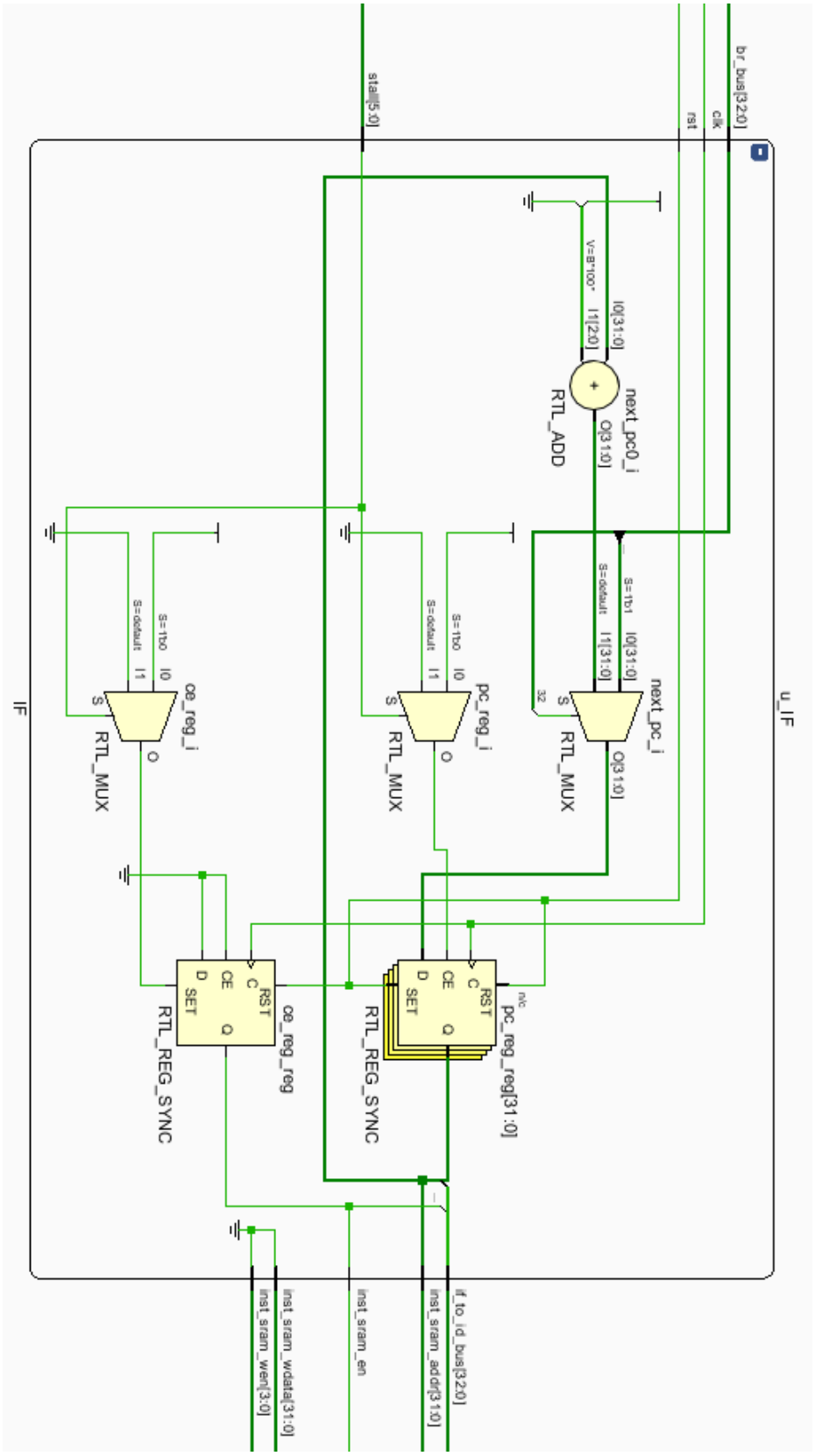


图 3 IF 段结构图

2.2 译码（ID）阶段

2.2.1 整体功能说明

1. 指令译码：

（a）从指令中按以下三种指令格式，分别取出 opcode、rs、rt、rd、sa、func、imm、instr_index、code、base、offset、sel：

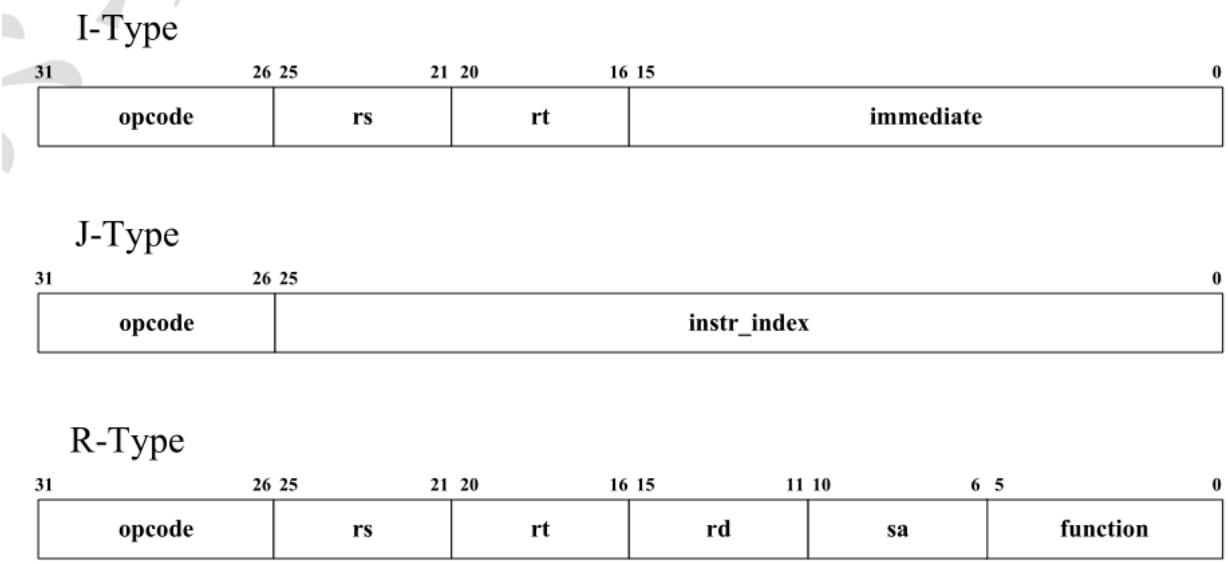


图 4 三类指令格式

（b）通过译码器将除了 div、divu、mult、multu、mthi、mtlo（需要写 hilo 寄存器的指令）之外的所有指令，转换成独热（one-hot）码；

（c）由（b）中得到的独热码，进行使能信号的赋值

2.定向技术：接收 EX、WB 段传过来的 forwarding 通路。

3.暂停气泡：处理 forwarding 技术无法解决的 load 数据相关（ID 段插入暂停气泡）、乘除法 32 个周期导致的 EX 段暂停。

4.与寄存器进行数据交互：包括普通寄存器和 hilo 寄存器。

5.分支判断和跳转地址计算：提前判断分支成功与否，并计算分支成功跳转地址，将其传给 IF 段，用于 next_pc 的赋值。

2.2.2 端口与信号介绍

表 3 ID 段接口

序号	接口名	宽度(bit)	输入/输出	作用
1	clk	1	输入	时钟信号

2	rst	1	输入	复位信号
3	stall	6	输入	暂停信号
4	stallreq	1	输出	用于控制暂停的信号
5	if_to_id_bus	33	输入	取指到译码段端口
	{ce_reg,	1		指令存储器使能信号
	pc_reg}	32		取指阶段的指令对应的地址
6	inst_sram_rdata	32	输入	ID 段读入指令
7	wb_to_rf_bus	104	输入	WB 段传来的端口，与寄存器交互（以下信号均是在 WB 段产生）
	={hi_wb_we,	1		hi 寄存器写使能信号
	hi_wb_wdata,	32		写入 hi 寄存器的数据
	lo_wb_we,	1		lo 寄存器写使能信号
	lo_wb_wdata,	32		写入 lo 寄存器的数据
	wb_rf_we,	1		普通寄存器写使能信号
	wb_rf_waddr,	5		写入普通寄存器的地址
	wb_rf_wdata}	32		写入普通寄存器的数据
8	ex_to_id_bus	176	输入	EX 段到 ID 段的定向路径（以下信号均是在 EX 段产生）
	={ex_r_lo,	1		lo 寄存器读使能信号
	ex_r_lo_data,	32		读到 lo 寄存器的数据
	ex_r_hi,	1		hi 寄存器读使能信号
	ex_r_hi_data,	32		读到 hi 寄存器的数据
	hi_ex_to_id_we,	1		hi 寄存器写使能信号
	hi_ex_wdata,	32		写入 hi 寄存器的数据
	lo_ex_to_id_we,	1		lo 寄存器写使能信号
	lo_ex_wdata,	32		写入 lo 寄存器的数据
	ex_rf_we,	1		普通寄存器写使能信号
	ex_rf_waddr,	5		写入普通寄存器的地址
	ex_result,	32		EX 计算后写入普通寄存器的数据
	ex_op_i}	6		用于记录上一条指令（判断是否是 lw 指令，处理 load 相关暂停）
	mem_to_id_bus	104		MEM 段到 ID 段的定向路径（以下信号均是在 MEM 段产生）
	={hi_mem_we,	1		hi 寄存器写使能信号

9	hi_mem_wdata,	32	输入	写入 hi 寄存器的数据
	lo_mem_we,	1		lo 寄存器写使能信号
	lo_mem_wdata,	32		写入 lo 寄存器的数据
	mem_rf_we,	1		普通寄存器写使能信号
	mem_rf_waddr,	5		写入普通寄存器的地址
	mem_rf_wdata	32		写入普通寄存器的数据
10	id_to_ex_bus	231	输出	ID 到 EX 段端口
	={inst_sh_e,	1		sh 指令使能
	inst_sb_e,	1		sb 指令使能
	inst_h,	1		lh 指令使能
	inst_hu,	1		lhu 指令使能
	inst_b,	1		lb 指令使能
	inst_bu,	1		lbu 指令使能
	r_lo,	1		lo 寄存器读使能信号
	r_lo_data,	32		读到 lo 寄存器的数据
	r_hi,	1		hi 寄存器读使能信号
	r_hi_data,	32		读到 hi 寄存器的数据
	id_pc,	32		ID 阶段的指令对应的地址
	inst,	32		ID 段指令
	alu_op,	12		运算符号
	sel_alu_src1,	3		操作数 1 选择信号
	sel_alu_src2,	4		操作数 2 选择信号
	data_ram_en,	1		内存使能信号
	data_ram_wen,	4		内存写使能
	rf_we,	1		普通寄存器写使能
	rf_waddr,	5		写入普通寄存器的地址
	sel_rf_res,	1		寄存器写入数据选择信号
	rdata1,	32		操作数 1
	rdata2}	32		操作数 2
11	br_bus	33	输入	分支端口，从 ID 段传出，IF 段接收
	={br_e,	1		分支使能信号，成功为 1，失败为 0
	br_addr}	32		分支成功跳转地址

2.2.3 功能模块说明

对应 2.2.1：

1. 指令译码：

（a） opcode ← $inst_{26..31}$

rs ← $inst_{21..25}$

rt ← $inst_{16..20}$

rd ← $inst_{11..15}$

sa ← $inst_{6..10}$

func ← $inst_{0..5}$

imm ← $inst_{0..15}$

instr_index ← $inst_{0..25}$

code ← $inst_{6..25}$

base ← $inst_{21..25}$

offset ← $inst_{0..15}$

sel ← $inst_{0..2}$

（b）通过 u0_decoder_6_64、u1_decoder_6_64、u2_decoder_5_32、u0_decoder_5_32、u1_decoder_5_32 分别将 opcode、func、sa、rs、rt，转换成独热码；

（c）利用（b）中独热码，

（1）指令使能：inst_xxxx 是 1 位信号，对应指令激活对应信号，

如：

```
assign inst_ori     = op_d[6'b00_1101];
```

代码 2 指令使能

表示 opcode 为 001101 时，inst_ori 被激活；

（2）读/写等使能：包括 r_hi（hi 寄存器读使能），r_lo（lo 寄存器读使能），data_ram_en（内存使能），data_ram_wen（内存写使能），rf_we（寄存器写使能），

如：以下实现了内存使能

```
assign data_ram_en = inst_lw | inst_sw | inst_lb | inst_lbu | inst_lh | inst_lhu | inst_sb | inst_sh;
```

代码 3 内存使能

（3）选择信号：包括 sel_alu_src1（源操作数 1 选择信号），sel_alu_src2（源操作数 2 选择信号），alu_op（运算符号选择信号），sel_rf_dst（目的寄存器选择信号），如：以下实现了寄存器写地址的选择

```
// store in [rd]
assign sel_rf_dst[0] = inst_subu | inst_addu | inst_or | inst_sll | inst_xor |inst_sltu |
                       inst_slt | inst_add | inst_sub | inst_and | inst_nor | inst_sllv |
                       inst_sra | inst_srav | inst_srl | inst_srlv | inst_jalr | inst_mfhi |
                       inst_mflo;
// store in [rt]
assign sel_rf_dst[1] = inst_ori | inst_lui | inst_addiu |inst_lw | inst_slti |
                       inst_sltiu | inst_addi | inst_andi | inst_xori | inst_lb | inst_lbu|
                       inst_lh | inst_lhu ;
// store in [31]
assign sel_rf_dst[2] = inst_jal | inst_bltzal | inst_bgezal;

// sel for regfile address
assign rf_waddr = {5{sel_rf_dst[0]}} & rd
                | {5{sel_rf_dst[1]}} & rt
                | {5{sel_rf_dst[2]}} & 32'd31;
```

代码 4 选择信号实现

如上代码，寄存器写地址为 3 位的信号 sel_rf_dst，其中从低位到高位被置为 1 分别对应：存储在 rd 寄存器中、存储在 rt 寄存器中、存储在 31 号寄存器中，因此根据 1 得到的指令一位使能信号 inst_xx 控制其选择以上三种；

接着，寄存器写地址 rf_waddr 根据选择信号的三位数值进行扩展之后分别与 rd、rt、31 按位与，得到最终结果。

2.定向技术：从 ex_to_id_bus 和 mem_to_id_bus 端口接收定向路径，传给 regfile 处理 forwarding，以 rdata1 为例：

```
// read out 1
assign rdata1 = (raddr1 == 5'b0) ? 32'b0 :
                ((ex_to_id_we == 1'b1)&&(ex_to_id_waddr == raddr1)) ? ex_wdata :
                ((mem_to_id_we == 1'b1)&&(mem_to_id_waddr == raddr1)) ? mem_wdata :
                ((wb_to_id_we == 1'b1)&&(wb_to_id_waddr == raddr1)) ? wb_wdata :
                 reg_array[raddr1];
```

代码 5 定向技术处理数据相关

3.暂停气泡：

首先判断：

load 数据相关导致的 ID 段暂停：

若上一条指令

（此时上条指令在 EX 段，因此在 EX 段首先判断是否进行了访存，若访存，则记录 opcode：

```
assign ex_op_i = data_sram_en ? inst[31:26]:6'b000000;
```

代码 6 暂停条件判断//EX.v）

是 lw 指令且写入寄存器地址与当前 rs 或 rt 相同，证明存在 load 数据相关，输出暂停信号：

```
assign stallreq = ((ex_op_i == 6'b100011)&&(ex_rf_we == 1'b1)&&((ex_rf_waddr == rs) | (ex_rf_waddr == rt))) ? 1'b1: 1'b0;
```

代码 7 输出 ID 段的暂停信号

乘除法 32 个周期导致的 EX 段暂停：（见 EX 段）

为了防止下一条指令将未处理的暂停指令覆盖，用 if_to_id_inst_r 存储下一条指令，直到暂停结束，再将存储的下一条指令读入 inst 中：

```
always @ (posedge clk) begin
    if (stall[2]==`Stop && stall[3]==`NoStop) begin//ID段暂停
        if_to_id_inst_r <= inst_sram_rdata;
        inst_flag <= 1'b1;
    end
    else if (stall[2]==`NoStop) begin
        if_to_id_inst_r <= 32'b0;
        inst_flag <= 1'b0;
    end
end

always @ (posedge clk) begin
    if (stall[3]==`Stop && stall[4]==`NoStop && hilo_inst_r==32'b0) begin//EX段暂停
        hilo_inst_r <= inst_sram_rdata;
        inst_hilo_flag <= 1'b1;
    end
    else if (stall[3]==`NoStop) begin
        hilo_inst_r <= 32'b0;
        inst_hilo_flag <= 1'b0;
    end
end


assign inst = inst_flag ? if_to_id_inst_r :
              inst_hilo_flag ? hilo_inst_r :
              inst_sram_rdata;
```

代码 8 暂停期间暂存下一条指令

4.与寄存器进行数据交互：

rdata1 ← Regs[rs]

rdata2 ← Regs[rt]

r_hi_data ← Regs_hi

r_lo_data ← Regs_lo

5.分支判断：

利用译码得到的独热码以及读寄存器得到的数据判断分支是否成功：

（此部分由于将普通寄存器与 hilo 寄存器分开存储，因而也需要处理数据相关，以 rdata1 为例：

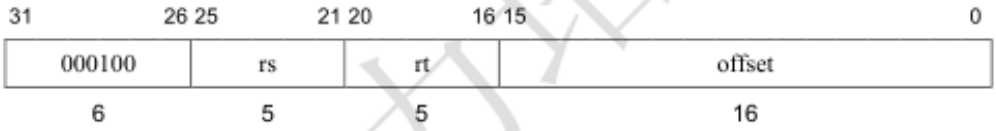
```
assign rdata1_end = ex_r_lo ? ex_r_lo_data :
                    ex_r_hi ? ex_r_hi_data :
                    rdata1;
```

代码 9 处理 hilo 寄存器的数据相关

）

以 beq 指令为例：

3.6.1 BEQ



汇编格式： BEQ rs, rt, offset

功能描述： 如果寄存器 rs 的值等于寄存器 rt 的值则转移，否则顺序执行。转移目标由立即数 offset 左移 2 位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到。

判断条件
```
assign rs_eq_rt = (rdata1_end == rdata2_end);
```

代码 10 beq 指令判断条件

是否成功
```
assign br_e     = (inst_beq & rs_eq_rt)
```

代码 11 beq 指令是否跳转成功

跳转地址
```
assign br_addr = inst_beq ? (pc_plus_4 + {{14{inst[15]}},inst[15:0],2'b0}) :
```

代码 12 beq 指令的跳转地址计算

其中，跳转地址也是使用多路选择器实现对不同分支指令的不同跳转。

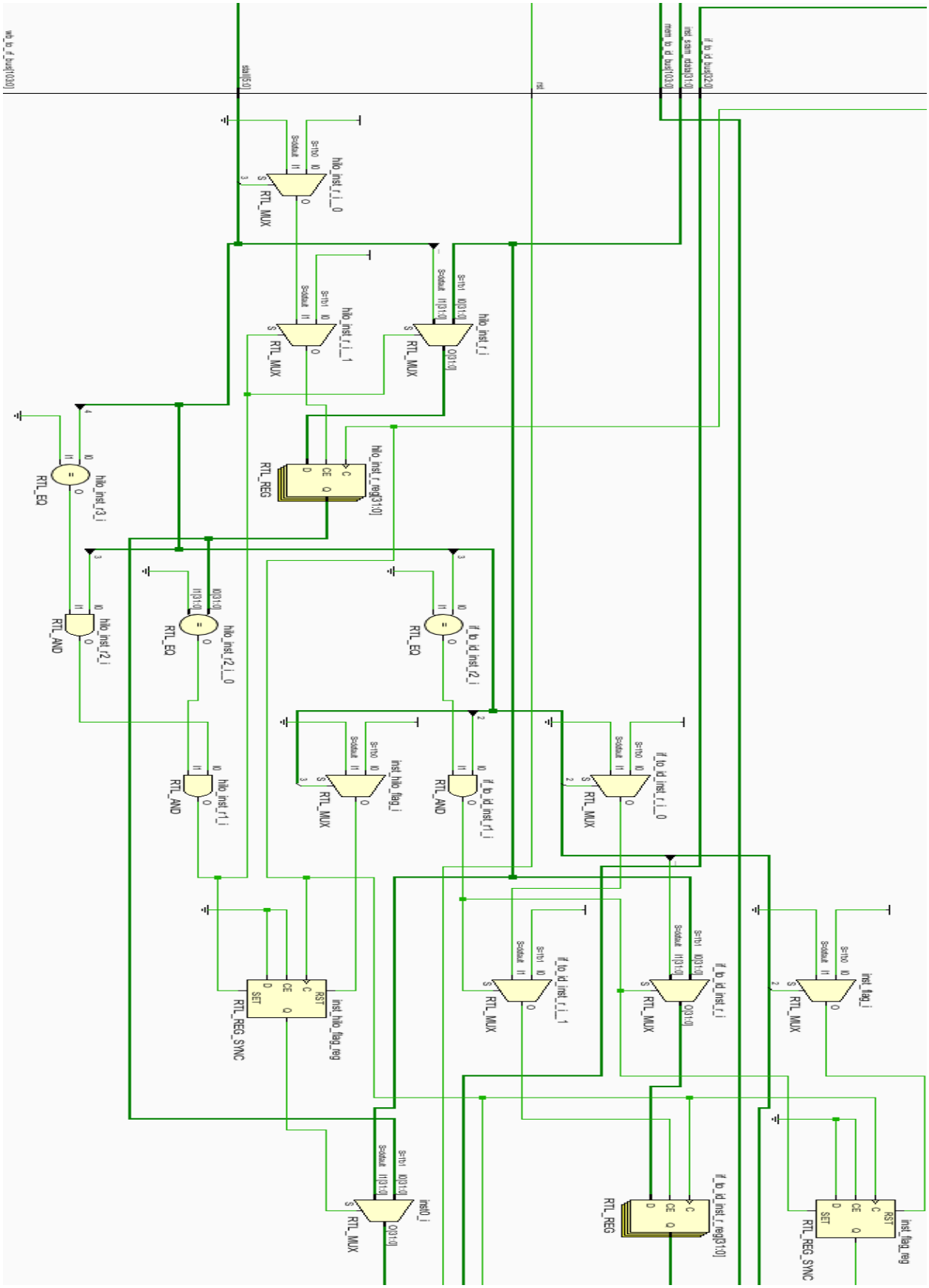
在判断寄存器值是否＞/=/＜0 时，不能将其值与零单纯进行比较，因为寄存器中的值存的是有符号数，所以应该通过最高位判断其符号：

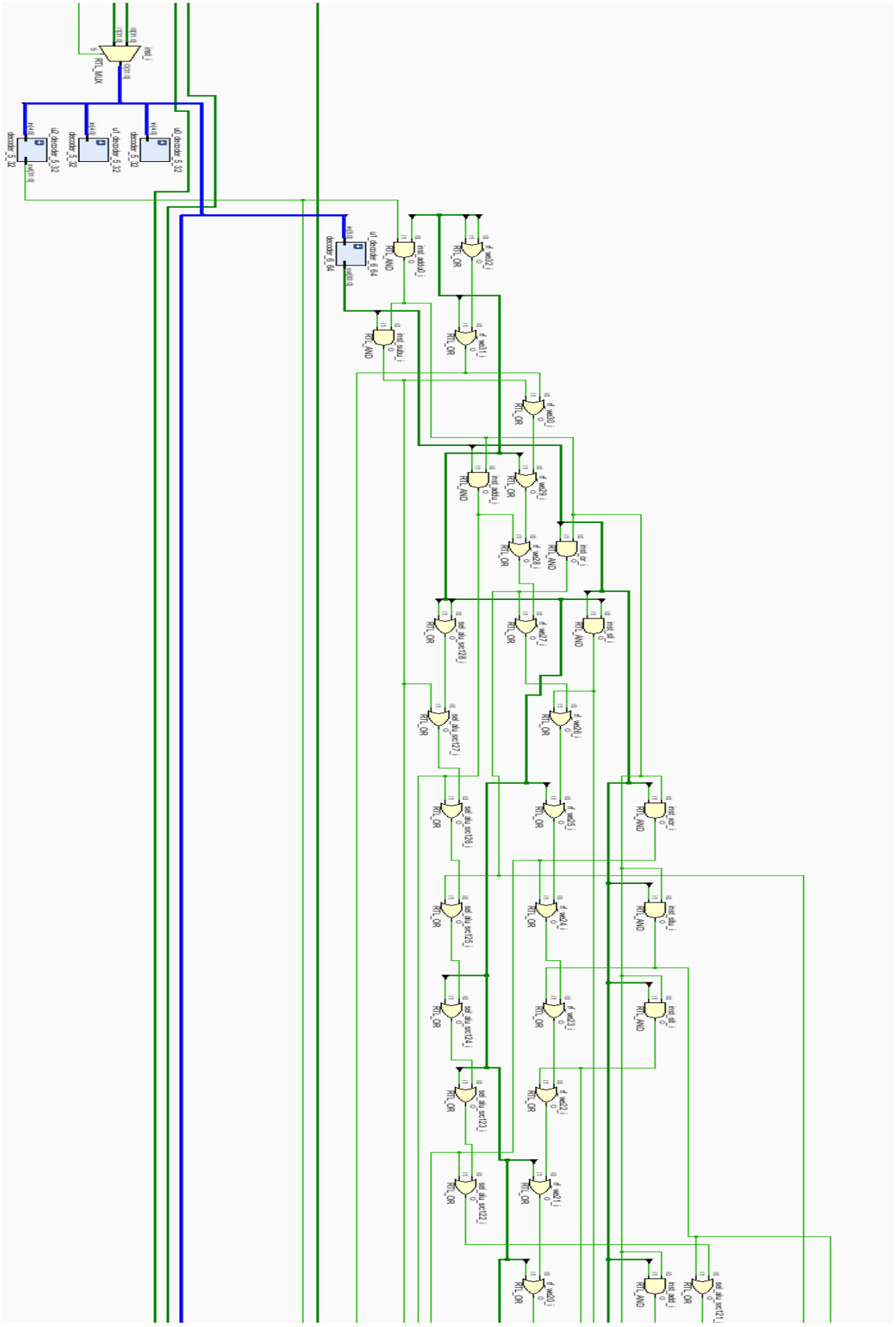
```
assign jump_bgez = (rdata1_end[31]==1'b0);
assign jump_bgtz=((rdata1_end[31]==1'b0 )&&(rdata1_end!=32'b0));
assign jump_blez = ((rdata1_end[31]==1'b1 )|(rdata1_end==32'b0));
assign jump_bltz = (rdata1_end[31]==1'b1);
```

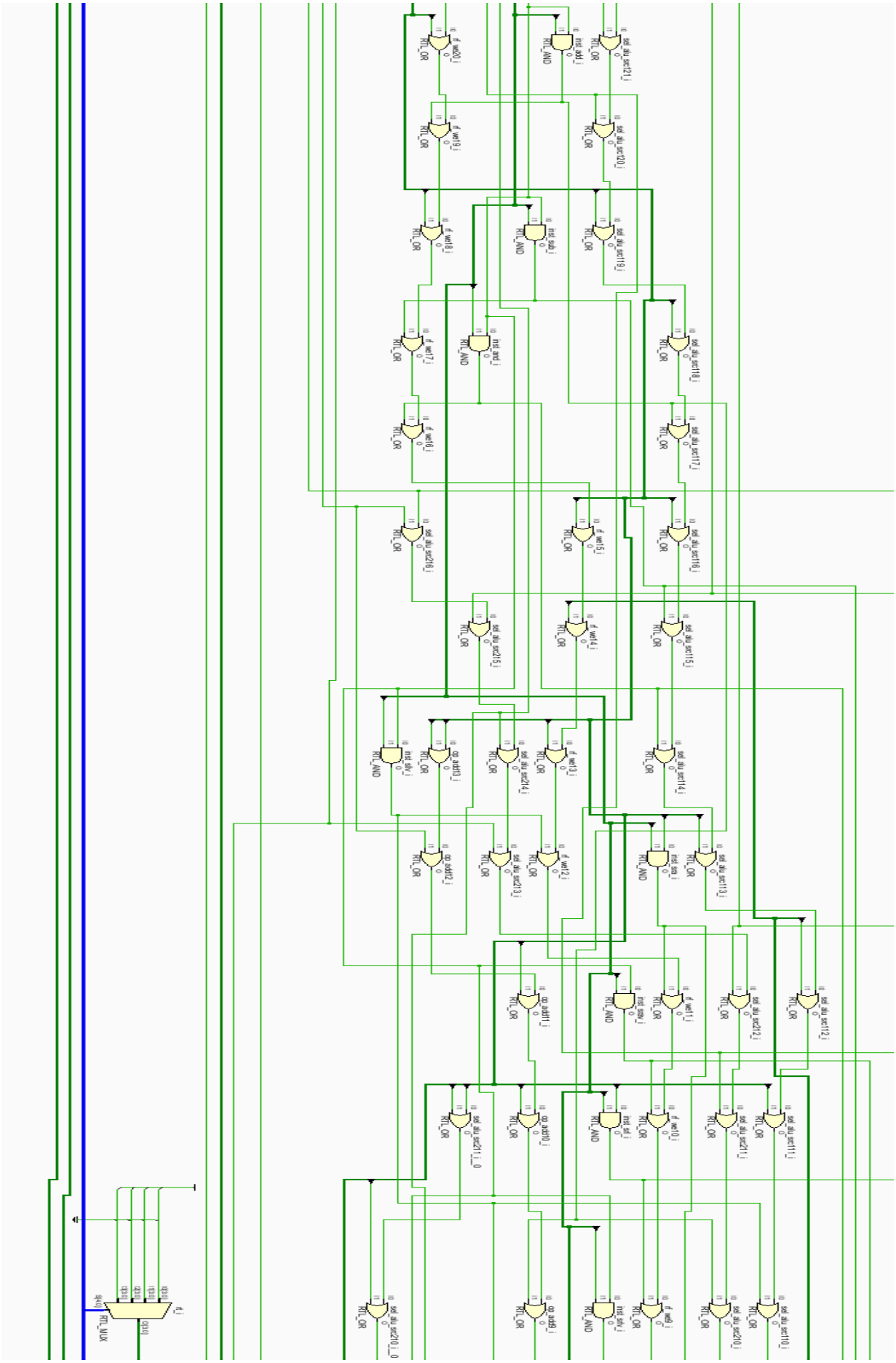
代码 12 其他跳转指令的判断条件

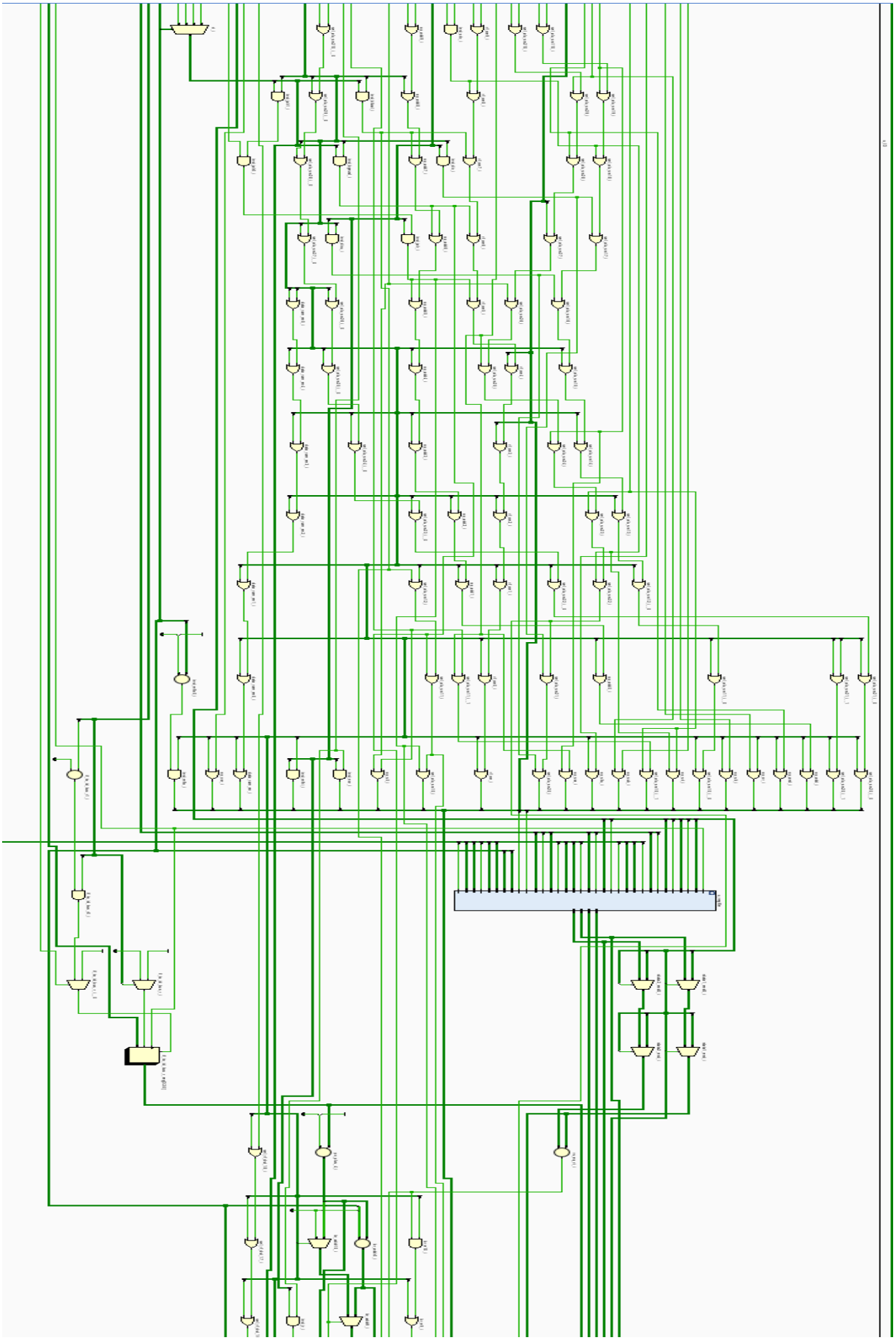
（其中，bgez、bgtz、blez、bltz 指令跳转条件分别为 rs 中值≥0、＞0、≤0、＜0）

2.2.4 结构示意图


wb_to_id_bus[103:0]
stall[5:0]
rst
mem_to_id_bus[103:0]
inst_sram_rdata[31:0]
id_to_id_bus[32:0]
hilo_inst_r_i_0
RTL_MUX
hilo_inst_r_i_1
hilo_inst_r_i
RTL_EQ
hilo_inst_r3_i
hilo_inst_r_reg[31:0]
RTL_REG
RTL_AND
hilo_inst_r2_i
hilo_inst_r2_i_0
id_to_id_inst_r_i_0
id_to_id_inst_r_i_1
id_to_id_inst_r_i
id_to_id_inst_r1_i
hilo_inst_r1_i
inst_hilo_flag_i
inst_hilo_flag_reg
RTL_REG_SYNC
inst_flag_i
inst_flag_reg
inst0_i







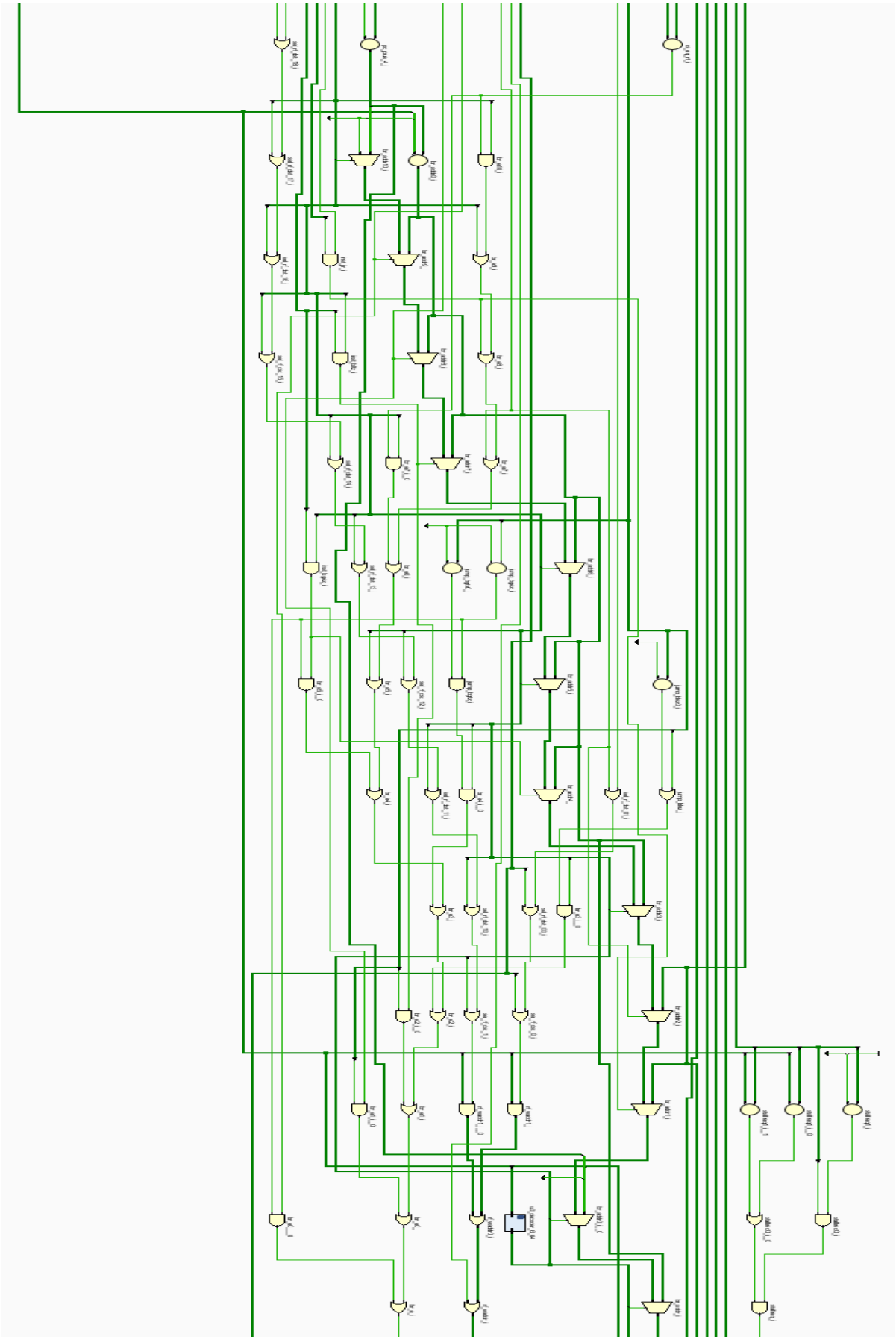


图 4 ID 段结构图

2.3 执行（EX）阶段

2.3.1 整体功能说明

1.算术操作：

（a）确定源操作数；

（b）计算：alu 基本算术操作+乘除法器

- 寄存器－寄存器

  alu_result ← rf_rdata1 op rf_rdata2

- 寄存器－立即值[有无符号扩展]

  alu_result ← rf_rdata1 op [$(imm_{15})^{16}$ ##]imm

2.计算访存地址

alu_result ← ex_pc＋$(imm_{15})^{16}$ ## imm

3.存储指令的分类处理：

包括 sw, sb 和 sh 指令

2.3.2 端口与信号介绍

表 4 EX 段接口

序号	接口名	宽度(bit)	输入/输出	作用
1	clk	1	输入	时钟信号
2	rst	1	输入	复位信号
3	stall	6	输入	暂停信号
4	stallreq_for_ex	1	输出	EX 段用于控制暂停的信号
5	id_to_ex_bus	231	输入	ID 到 EX 段端口
	={inst_sh_e,	1		sh 指令使能
	inst_sb_e,	1		sb 指令使能
	inst_h,	1		lh 指令使能
	inst_hu,	1		lhu 指令使能
	inst_b,	1		lb 指令使能
	inst_bu,	1		lbu 指令使能
	r_lo,	1		lo 寄存器读使能信号
5	r_lo_data,	32	输入	读到 lo 寄存器的数据
	r_hi,	1		hi 寄存器读使能信号
	r_hi_data,	32		读到 hi 寄存器的数据

	ex_pc,	32		EX 阶段的指令对应的地址
	inst,	32		ID 段指令
	alu_op,	12		运算符号
	sel_alu_src1,	3		操作数 1 选择信号
	sel_alu_src2,	4		操作数 2 选择信号
	data_ram_en,	1		内存使能信号
	data_ram_wen,	4		内存写使能
	rf_we,	1		普通寄存器写使能
	rf_waddr,	5		写入普通寄存器的地址
	sel_rf_res,	1		寄存器写入数据选择信号
	rf_rdata1,	32		操作数 1
	rf_rdata2}	32		操作数 2
6	ex_to_mem_bus	212	输出	EX 到 MEM 段端口
	={inst_h,	1		lh 指令使能
	inst_hu,	1		lhu 指令使能
	inst_b,	1		lb 指令使能
	inst_bu,	1		lbu 指令使能
	hi_ex_we,	1		EX 段产生的 hi 寄存器写使能信号
	hi_ex_wdata,	32		EX 段写入 hi 寄存器的数据
	lo_ex_we,	1		EX 段产生的 lo 寄存器写使能信号
	lo_ex_wdata,	32		EX 段写入 lo 寄存器的数据
	r_lo,	1		lo 寄存器读使能信号
	r_lo_data,	32		读到 lo 寄存器的数据
	r_hi,	1		hi 寄存器读使能信号
	r_hi_data,	32		读到 hi 寄存器的数据
	ex_pc,	32		EX 阶段的指令对应的地址
	data_ram_en,	1		内存使能信号
	data_ram_wen,	4		内存写使能
	rf_we,	1		普通寄存器写使能
	rf_waddr,	5		写入普通寄存器的地址
	sel_rf_res,	1		寄存器写入数据选择信号
	ex_result}	32		EX 段计算结果
	ex_to_id_bus	176		EX 段到 ID 段的定向路径 （以下信号均是在 EX 段产生）
	={ex_r_lo,	1		lo 寄存器读使能信号

7	ex_r_lo_data,	32	输出	读到 lo 寄存器的数据
	ex_r_hi,	1		hi 寄存器读使能信号
	ex_r_hi_data,	32		读到 hi 寄存器的数据
	hi_ex_to_id_we,	1		hi 寄存器写使能信号
	hi_ex_wdata,	32		写入 hi 寄存器的数据
	lo_ex_to_id_we,	1		lo 寄存器写使能信号
	lo_ex_wdata,	32		写入 lo 寄存器的数据
	ex_rf_we,	1		普通寄存器写使能信号
	ex_rf_waddr,	5		写入普通寄存器的地址
	ex_result,	32		EX 计算后写入普通寄存器的数据
	ex_op_i}	6		用于记录上一条指令（判断是否是 lw 指令，处理 load 相关暂停）
8	data_sram_en	1	输出	内存使能信号
9	data_sram_wen	4	输出	内存写使能
10	data_sram_addr	32	输出	访存地址
11	data_sram_wdata	32	输出	写入内存的数据

2.3.3 功能模块说明

对应 2.3.1：

1.算术操作：

（a）利用 ID 段产生的 sel_alu_src1/2 信号选择源操作数：

```
assign imm_sign_extend = {{16{inst[15]}},inst[15:0]};
assign imm_zero_extend = {16'b0, inst[15:0]};
assign sa_zero_extend = {27'b0,inst[10:6]};

assign alu_src1 = sel_alu_src1[1] ? ex_pc :
                  sel_alu_src1[2] ? sa_zero_extend : rf_rdata1;

assign alu_src2 = sel_alu_src2[1] ? imm_sign_extend :
                  sel_alu_src2[2] ? 32'd8 :
                  sel_alu_src2[3] ? imm_zero_extend : rf_rdata2;
```

代码 13 选择源操作数

（b）传入 alu 模块进行运算：

```
alu u_alu(
    .alu_control (alu_op ),
    .alu_src1    (alu_src1    ),
    .alu_src2    (alu_src2    ),
    .alu_result  (alu_result  )
);
```

代码 14 传入 alu 模块

或者传入乘除法器进行运算：

首先将以下写 hilo 寄存器的指令的信号在 EX 段激活（方法类似于 ID 段译码）：

```
wire inst_div, inst_divu, inst_mult, inst_multu, inst_mthi, inst_mtlo;
```

通过时序逻辑判断乘/除法（需要 32 个周期）是否完成，进而给 start，ready 端口赋值；同时判断该运算是有/无符号运算（通过判断四个信号 inst_mult，inst_div[有符号]，inst_multu，inst_divu[无符号]的激活情况，进而给 e_signed 端口赋值）：

在此以有符号为例，即 mult/div 指令：

```
case ({(inst_mult | inst_div),(inst_multu | inst_divu)})
    2'b10:begin
        if (md_ready_i == `ResultNotReady) begin
            md_opdata1_o = rf_rdata1;
            md_opdata2_o = rf_rdata2;
            md_start_o = `Start;
            signed_md_o = 1'b1;
            stallreq_for_md = `Stop;
        end
        else if (md_ready_i == `ResultReady) begin
            md_opdata1_o = rf_rdata1;
            md_opdata2_o = rf_rdata2;
            md_start_o = `MDStop;
            signed_md_o = 1'b1;
            stallreq_for_md = `NoStop;
        end
        else begin
            md_opdata1_o = `ZeroWord;
            md_opdata2_o = `ZeroWord;
            md_start_o = `MDStop;
            signed_md_o = 1'b0;
            stallreq_for_md = `NoStop;
        end
    end
```

代码 15 乘除法器的时序逻辑

乘除法器实现：

乘法原理：

乘法采用的是移位相加，需要 32 个时钟周期，流程如下：

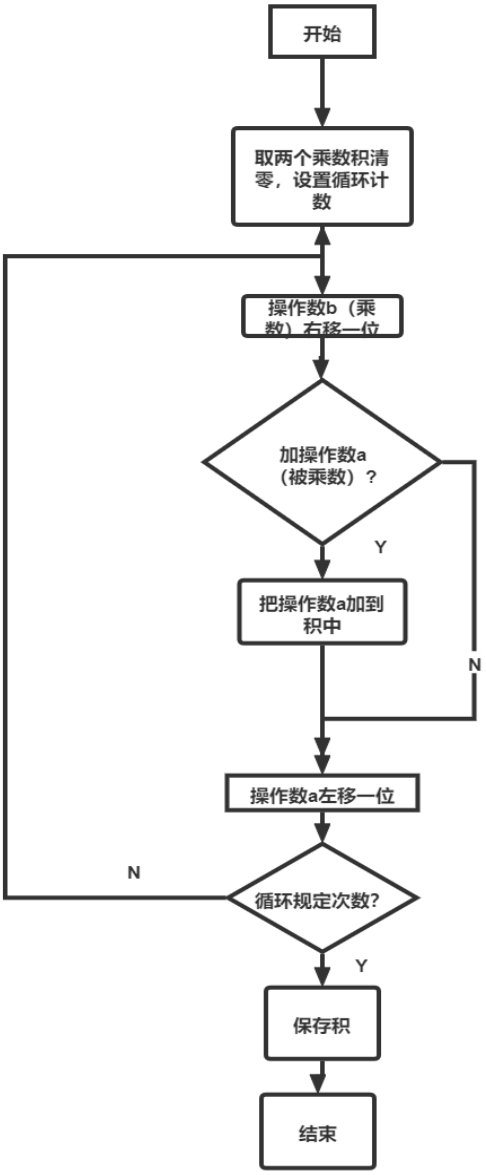


图 5 移位乘法流程图

状态转移图如下：

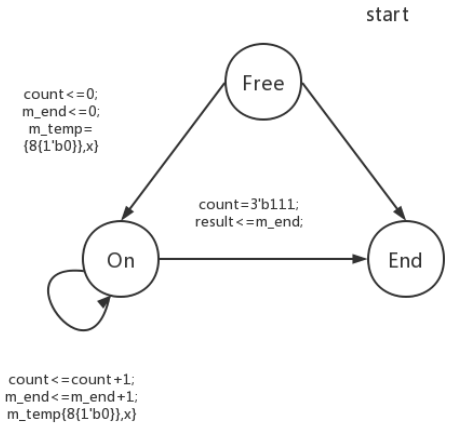


图 6 状态转移图

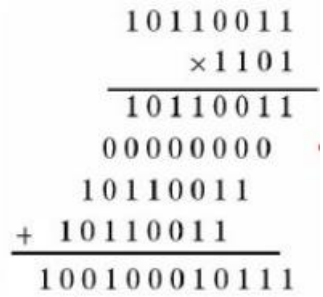


图 7 例子

以上图中运算为例，初始状态设置乘积为被乘数：

```
m_temp <= {{32{1'b0}}, temp_op1};
```

观察乘数最低位，若为 0，则不把左移之后的被乘数加到乘积中；

```
  //乘法
  if(y_reg[0] == 1'b1)//y_reg最低位是1的话，就会按照顺序向左进位进行求解
      m_end <= m_end + m_temp;
  else
      m_end <= m_end;
  y_reg <= y_reg >> 1;
  m_temp <= m_temp << 1;
end

count <= count + 1;
state <= `On;
```

代码 16 乘法运算的移位

如此往复至乘数的最高位，最后累加和就是乘积；

```
result <= m_end;
```

代码 17 乘法结果

但若是有符号乘法，需要多设置一个时钟周期，用于判断乘积是否是负数（即有符号运算使能被置为 1，且乘数与被乘数的最高位异号）

```
`On: begin//运算状态
    if(count == 6'b100000) begin//运算结束取反


            else m_end[63:0] <= (~m_end[63:0] + 1);//乘法

        state <= `End;
        count <= 6'b000000;
```

代码 18 有无符号运算的判断

除法原理：

有两条除法指令需要加，分别为 div 和 divu,区别在于有无符号，作用是将地址为 rs 的通用寄存器的值，与地址为 rt 的通用寄存器的值，做有（无）符号进行除法运算，将商保存在寄存器 lo，余数保存在寄存器 hi 上。

我们沿用了雷思磊中的除法方法:试商法（32 周期）。

设被除数是 m,除数是 n,商保存在 s 中，被除数的位数是 k.

首先，取出被除数的最高位，使用这个数减去除数 n,如果结果大于等于 0，则商的最低位为 1，否则为 0。为 0，则代表当前的被减数小于了减数，除法不成功，此时，取出被除数剩下的值的最高位，与当前被减数组合为新一轮的被减数；如果得到的结果是 1，则代表被减数大于减数，则利用上一步计算的结果与被减数剩下的值的最高位，组合成新的被减数，并执行位数减一。新的被减数减去除数，结果要是大于等于 0，则商的当前位为 1，否则置为 0，进行循环此操作 32 次。

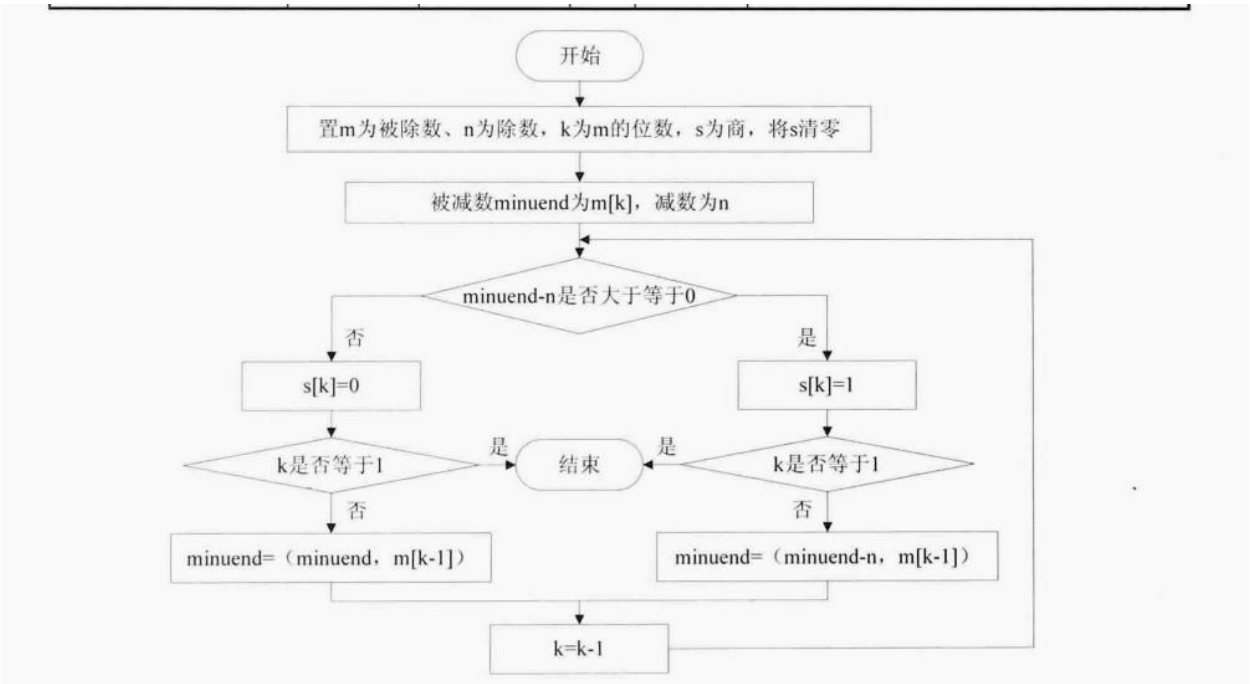


图 8 试商法流程图

实现思路:新建一个模块 DIV，在其中实现采用试商法的 32 位除法运算。当流水线执行阶段的 EX 模块发现当前指令是除法指令的时候，首先暂停流水线，然后将被除数、除数等信息送到 DIV 模块，开始除法运算。DIV 模块在除法运算结束后，通知 EX 模块，并将除法结果送到 EX 模块，后者依据除法结果设置寄存器的写信息，同时取消暂停流水线。

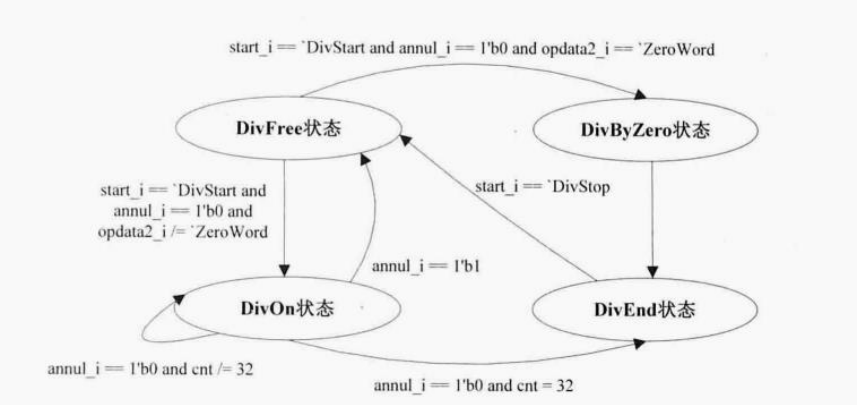


图 9 状态转移图

EX 段的修改可以分成三段修改：

第一段：如果是 div 指令，并且此模块没有声明除法结束，那么输出的被除数、除数、除法的开始信号，有符号的除法信息到 div 块，设置气泡停止，表示由于除法运算请求流水线暂停。反之，如果 DIV 声明除法结束，那么设置除法停止，同时表示不是由于除法运算请求流水线暂停。

第二段：给出流水线请求信号的 stallreq，目前已实现的惩罚累加、乘法累减，都会请求流水线暂停。

第三段：除法指令的结果最终要写入 HI,LO 寄存器，所以在第三段给出对寄存器信息。

在做乘除法运算时，需要将流水线暂停：

```
assign stallreq_for_ex = stallreq_for_md;
```

代码 19 乘除法器的流水线暂停信号请求

2. 地址计算：

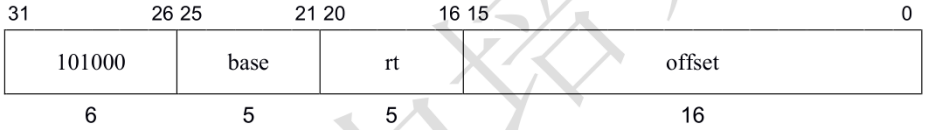
从 alu 模块拿到 alu_result，传给输出信号访存地址：

```
assign data_sram_addr = ex_result;
```

代码 20 访存地址计算

3. 存储指令：

3.9.6 SB



汇编格式： SB rt, offset(base)

功能描述： 将 base 寄存器的值加上符号扩展后的立即数 offset 得到访存的虚地址，据此虚地址将 rt 寄存器的最低字节存入存储器中。

操作定义： vAddr ← GPR[base] + sign_extend(offset)
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, STORE)
databyte ← $GPR[rt]_{7..0}$
StoreMemory(CCA, BYTE, databyte, pAddr, vAddr, DATA)

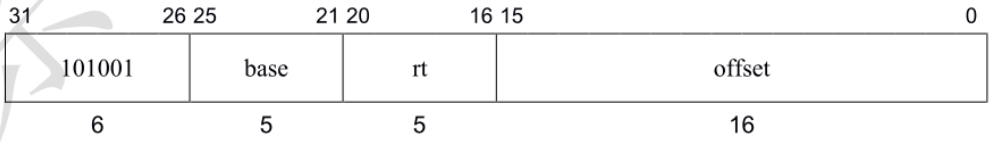
sb 指令一次只写入一个字节，所以可能是 4'b0001 4'b0010 4'b0100 4'b1000 这四种情况，具体选择那种，根据写地址的最低两位（addr[1:0]）判断。00 对应低位字节，11 对应高位字节。

```
assign data_sram_wen = (inst_sb_e && (ex_result[1:0]==2'b11) && data_ram_en) ? 4'b1000:
                       (inst_sb_e && (ex_result[1:0]==2'b10) && data_ram_en) ? 4'b0100:
                       (inst_sb_e && (ex_result[1:0]==2'b01) && data_ram_en) ? 4'b0010:
                       (inst_sb_e && (ex_result[1:0]==2'b00) && data_ram_en) ? 4'b0001:

assign data_sram_wdata = (inst_sb_e && data_ram_en && (data_sram_wen == 4'b1000)) ? {4{rf_rdata2[7:0]}}:
                         (inst_sb_e && data_ram_en && (data_sram_wen == 4'b0100)) ? {4{rf_rdata2[7:0]}}:
                         (inst_sb_e && data_ram_en && (data_sram_wen == 4'b0010)) ? {4{rf_rdata2[7:0]}}:
                         (inst_sb_e && data_ram_en && (data_sram_wen == 4'b0001)) ? {4{rf_rdata2[7:0]}}:
```

代码 21 sb 指令的写内存处理

3.9.7 SH



汇编格式： SH rt, offset(base)

功能描述： 将 base 寄存器的值加上符号扩展后的立即数 offset 得到访存的虚地址，如果地址不是 2 的整数倍则触发地址错例外，否则据此虚地址将 rt 寄存器的低半字存入存储器中。

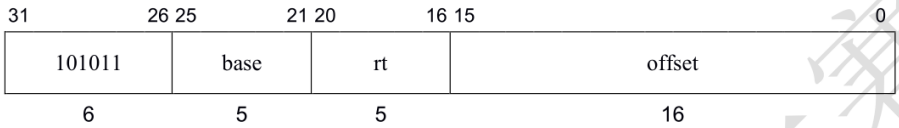
操作定义：
vAddr ← GPR[base] + sign_extend(offset)
if $vAddr_0 \neq 0$ then
    SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, STORE)
datahalf ← $GPR[rt]_{15..0}$
StoreMemory(CCA, HALFWORD, datahalf, pAddr, vAddr, DATA)

sh 指令类似于 sb 指令，但其只有两种情况，地址最低两位为 00 时，4'b0011，即写低位两个字节；地址最低两位为 10 时，4'b1100，即写高位两个字节。

```
                          (inst_sh_e && (ex_result[1:0]==2'b10) && data_ram_en) ? 4'b1100:
assign data_sram_wen =    (inst_sh_e && (ex_result[1:0]==2'b00) && data_ram_en) ? 4'b0011:
                           (inst_sh_e && data_ram_en && (data_sram_wen == 4'b1100)) ? {2{rf_rdata2[15:0]}}:
assign data_sram_wdata = (inst_sh_e && data_ram_en && (data_sram_wen == 4'b0011)) ? {2{rf_rdata2[15:0]}}:
```

代码 22 sh 指令的写内存处理

3.9.8 SW



汇编格式： SW rt, offset(base)

功能描述： 将 base 寄存器的值加上符号扩展后的立即数 offset 得到访存的虚地址，如果地址不是 4 的整数倍则触发地址错例外，否则据此虚地址将 rt 寄存器存入存储器中。

操作定义：
vAddr ← GPR[base] + sign_extend(offset)
if $vAddr_{1..0} \neq 0$ then
    SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, STORE)
dataword ← $GPR[rt]_{31..0}$
StoreMemory(CCA, WORD, dataword, pAddr, vAddr, DATA)

例　外： 地址最低 2 位不为 0，触发地址错例外

sw 指令不需要考虑写入高低位：

```
assign data_sram_wen =     data_ram_wen;
assign data_sram_wdata =  rf_rdata2 ;
```

代码 23 sw 指令的写内存处理

2.3.4 结构示意图

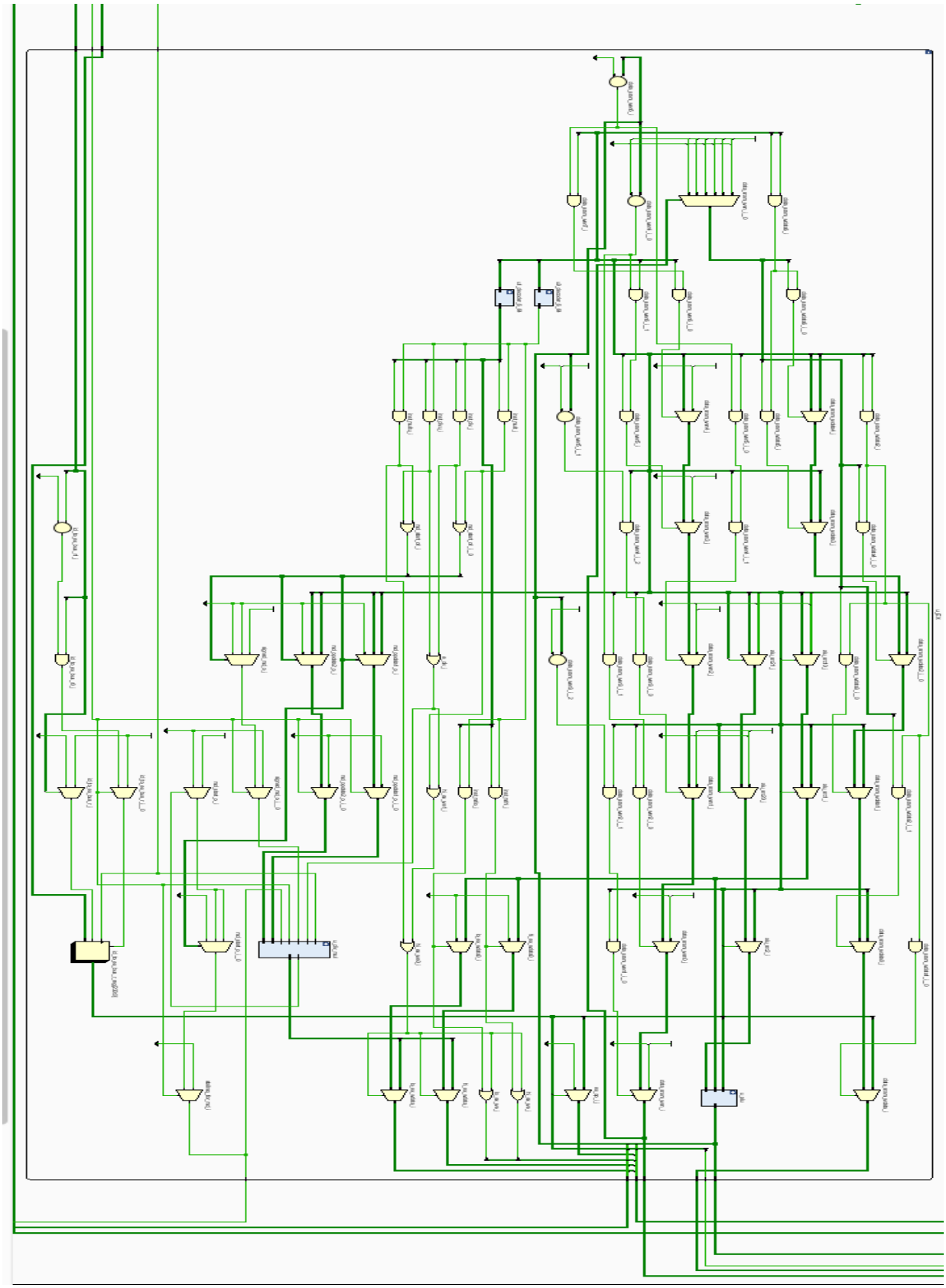


图 10　EX 段结构图

2.4 访存（MEM）阶段

2.4.1 整体功能说明

◆ 存储器访问

rf_wdata ← data_sram_rdata

其中，lb，lbu，lh，lhu 指令需要分类讨论。

2.4.2 端口与信号介绍

表 5 MEM 段接口

序号	接口名	宽度(bit)	输入/输出	作用
1	clk	1	输入	时钟信号
2	rst	1	输入	复位信号
3	stall	6	输入	暂停信号
4	ex_to_mem_bus	212	输出	EX 到 MEM 段端口
	={inst_h,	1		lh 指令使能
	inst_hu,	1		lhu 指令使能
	inst_b,	1		lb 指令使能
	inst_bu,	1		lbu 指令使能
	hi_ex_we,	1		EX 段产生的 hi 寄存器写使能信号
	hi_ex_wdata,	32		EX 段写入 hi 寄存器的数据
	lo_ex_we,	1		EX 段产生的 lo 寄存器写使能信号
	lo_ex_wdata,	32		EX 段写入 lo 寄存器的数据
	r_lo,	1		lo 寄存器读使能信号
	r_lo_data,	32		读到 lo 寄存器的数据
	r_hi,	1		hi 寄存器读使能信号
	r_hi_data,	32		读到 hi 寄存器的数据
	mem_pc,	32		MEM 阶段的指令对应的地址
	data_ram_en,	1		内存使能信号
	data_ram_wen,	4		内存写使能
	rf_we,	1		普通寄存器写使能
	rf_waddr,	5		写入普通寄存器的地址
	sel_rf_res,	1		寄存器写入数据选择信号

	ex_result}	32		EX 段计算结果
5	data_sram_rdata	32	输入	
6	mem_to_wb_bus	309	输出	MEM 到 WB 段端口
	={hi_ex_we,	1		hi 寄存器写使能信号
	hi_ex_wdata,	32		写入 hi 寄存器的数据
	lo_ex_we,	1		lo 寄存器写使能信号
	lo_ex_wdata,	32		写入 lo 寄存器的数据
	mem_pc,	32		EX 阶段的指令对应的地址
	rf_we,	1		普通寄存器写使能
	rf_waddr,	5		写入普通寄存器的地址
	rf_wdata}	32		写入普通寄存器的数据
7	mem_to_id_bus	309	输出	MEM 段到 ID 段的定向路径 （以下信号均是在 MEM 段产生）
	={hi_ex_we,	1		hi 寄存器写使能信号
	hi_ex_wdata,	32		写入 hi 寄存器的数据
	lo_ex_we,	1		lo 寄存器写使能信号
	lo_ex_wdata,	32		写入 lo 寄存器的数据
	mem_pc,	32		EX 阶段的指令对应的地址
	rf_we,	1		普通寄存器写使能
	rf_waddr,	5		写入普通寄存器的地址
	rf_wdata}	32		写入普通寄存器的数据

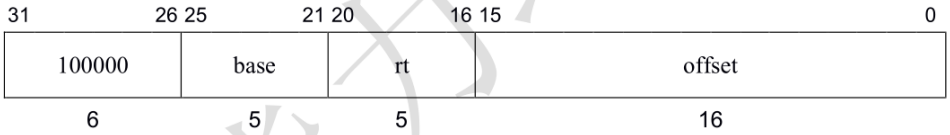
2.4.3 功能模块说明

存储器访问：

load 类指令与 store 类指令略有不同，由于这个存储器只配置了片选使能和字节写使能，所以读取的时候一律是先读回 CPU（此时不区分是哪个 load 指令，一律使用 4'b0000），再进行更细分的操作。那么就需要把细分的指令操作（即 ID 段译码激活的 inst_b, inst_bu, inst_h, inst_hu 使能信号）随流水线传到 MEM 段，使 MEM 段能知道应该根据哪种方式处理数据。

load 类指令的字节选择和 store 类相同，也是根据 EX 段计算得到的访存地址判断：

3.9.1 LB

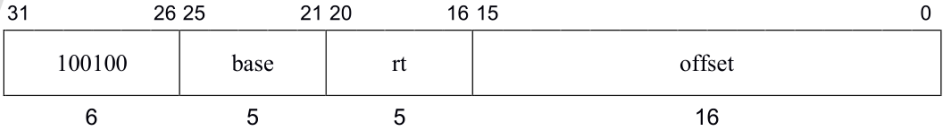


汇编格式：LB rt, offset(base)

功能描述：将 base 寄存器的值加上符号扩展后的立即数 offset 得到访存的虚地址，据此虚地址从存储器中读取 1 个字节的值并进行符号扩展，写入到 rt 寄存器中。

操作定义：vAddr ← GPR[base] + sign_extend(offset)
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, LOAD)
membyte ← LoadMemory(CCA, BYTE, pAddr, vAddr, DATA)
GPR[rt] ← sign_extend($membyte_{7..0}$)

3.9.2 LBU

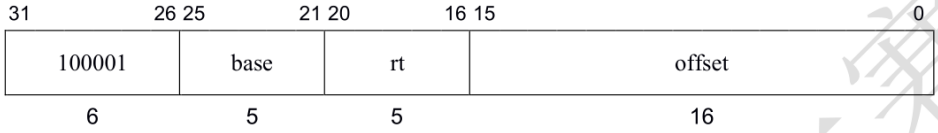


汇编格式：LBU rt, offset(base)

功能描述：将 base 寄存器的值加上符号扩展后的立即数 offset 得到访存的虚地址，据此虚地址从存储器中读取 1 个字节的值并进行 0 扩展，写入到 rt 寄存器中。

操作定义：vAddr ← GPR[base] + sign_extend(offset)
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, LOAD)
membyte ← LoadMemory(CCA, BYTE, pAddr, vAddr, DATA)
GPR[rt] ← zero_extend($membyte_{7..0}$)

3.9.3 LH



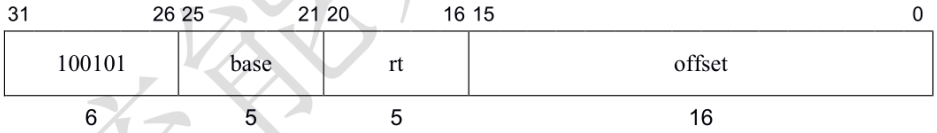
汇编格式：LH rt, offset(base)

功能描述：将 base 寄存器的值加上符号扩展后的立即数 offset 得到访存的虚地址，如果地址不是 2 的整数倍则触发地址错例外，否则据此虚地址从存储器中读取连续 2 个字节的值并进行符号扩展，写入到 rt 寄存器中。

操作定义：vAddr ← GPR[base] + sign_extend(offset)
if $vAddr_0 \neq 0$ then
    SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, LOAD)
memhalf ← LoadMemory(CCA, HALFWORD, pAddr, vAddr, DATA)
GPR[rt] ← sign_extend($memhalf_{15..0}$)

例　外：地址最低 1 位不为 0，触发地址错例外

3.9.4 LHU



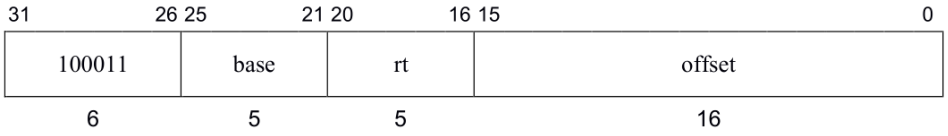
汇编格式： LHU rt, offset(base)

功能描述： 将 base 寄存器的值加上符号扩展后的立即数 offset 得到访存的虚地址，如果地址不是 2 的整数倍则触发地址错例外，否则据此虚地址从存储器中读取连续 2 个字节的值并进行 0 扩展，写入到 rt 寄存器中。

操作定义：
vAddr ← GPR[base] + sign_extend(offset)
if $vAddr_0 \neq 0$ then
    SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, LOAD)
memhalf ← LoadMemory(CCA, HALFWORD, pAddr, vAddr, DATA)
$GPR[rt] \leftarrow zero\_extend(memhalf_{15..0})$

例　外： 地址最低 1 位不为 0，触发地址错例外

3.9.5 LW



汇编格式： LW rt, offset(base)

功能描述： 将 base 寄存器的值加上符号扩展后的立即数 offset 得到访存的虚地址，如果地址不是 4 的整数倍则触发地址错例外，否则据此虚地址从存储器中读取连续 4 个字节的值，写入到 rt 寄存器中。

操作定义：
vAddr ← GPR[base] + sign_extend(offset)
if $vAddr_{1..0} \neq 0$ then
    SignalException(AddressError)
endif
(pAddr, CCA) ← AddressTranslation(vAddr, DATA, LOAD)
memword ← LoadMemory(CCA, WORD, pAddr, vAddr, DATA)
$GPR[rt] \leftarrow sign\_extend(memword_{31..0})$

例　外： 地址最低 2 位不为 0，触发地址错例外

```
assign rf_wdata = ((ex_result[1:0]==2'b11) && data_ram_en && inst_b) ? {{24{data_sram_rdata[31]}},data_sram_rdata[31:24]}:
                  ((ex_result[1:0]==2'b10) && data_ram_en && inst_b) ? {{24{data_sram_rdata[23]}},data_sram_rdata[23:16]}:
                  ((ex_result[1:0]==2'b01) && data_ram_en && inst_b) ? {{24{data_sram_rdata[15]}},data_sram_rdata[15:8]}:
                  ((ex_result[1:0]==2'b00) && data_ram_en && inst_b) ? {{24{data_sram_rdata[7]}},data_sram_rdata[7:0]}:

                  ((ex_result[1:0]==2'b11) && data_ram_en && inst_bu) ? {24'b0,data_sram_rdata[31:24]}:
                  ((ex_result[1:0]==2'b10) && data_ram_en && inst_bu) ? {24'b0,data_sram_rdata[23:16]}:
                  ((ex_result[1:0]==2'b01) && data_ram_en && inst_bu) ? {24'b0,data_sram_rdata[15:8]}:
                  ((ex_result[1:0]==2'b00) && data_ram_en && inst_bu) ? {24'b0,data_sram_rdata[7:0]}:

                  ((ex_result[1:0]==2'b10) && data_ram_en && inst_h) ? {{16{data_sram_rdata[31]}},data_sram_rdata[31:16]}:
                  ((ex_result[1:0]==2'b00) && data_ram_en && inst_h) ? {{16{data_sram_rdata[15]}},data_sram_rdata[15:0]}:

                  ((ex_result[1:0]==2'b10) && data_ram_en && inst_hu) ? {16'b0,data_sram_rdata[31:16]}:
                  ((ex_result[1:0]==2'b00) && data_ram_en && inst_hu) ? {16'b0,data_sram_rdata[15:0]}:

                 (data_ram_en & (data_ram_wen == 4'b0000)) ? data_sram_rdata:

                 sel_rf_res ? mem_result :
                 r_hi ? r_hi_data :
                 r_lo ? r_lo_data :
                 ex_result;
```

2.4.4 结构示意图

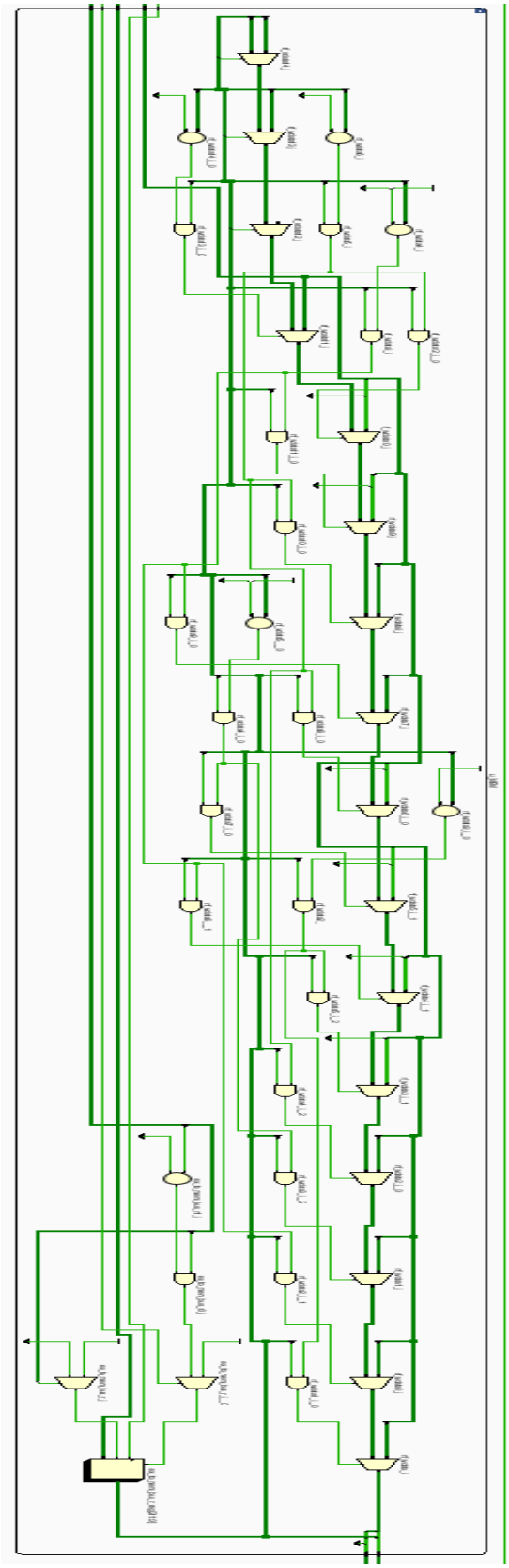


图 11 MEM 段结构图

2.5 写回（WB）阶段

2.5.1 整体功能说明

1.接收写普通寄存器、hilo 寄存器的使能、（地址、）数据，传给 regfile；
2.控制流水线是否复位或暂停。

2.5.2 端口与信号介绍

表 6 WB 段接口

序号	接口名	宽度(bit)	输入/输出	作用
1	clk	1	输入	时钟信号
2	rst	1	输入	复位信号
3	stall	6	输入	暂停信号
4	mem_to_wb_bus	309	输出	MEM 到 WB 段端口
	={hi_ex_we,	1		hi 寄存器写使能信号
	hi_ex_wdata,	32		写入 hi 寄存器的数据
	lo_ex_we,	1		lo 寄存器写使能信号
	lo_ex_wdata,	32		写入 lo 寄存器的数据
	wb_pc,	32		WB 阶段的指令对应的地址
	rf_we,	1		普通寄存器写使能
	rf_waddr,	5		写入普通寄存器的地址
	rf_wdata}	32		写入普通寄存器的数据
5	wb_to_rf_bus	104	输入	WB 段传来的端口，与寄存器交互（以下信号均是在 WB 段产生）
	={hi_wb_we,	1		hi 寄存器写使能信号
	hi_wb_wdata,	32		写入 hi 寄存器的数据
	lo_wb_we,	1		lo 寄存器写使能信号
	lo_wb_wdata,	32		写入 lo 寄存器的数据
	wb_rf_we,	1		普通寄存器写使能信号
	wb_rf_waddr,	5		写入普通寄存器的地址
	wb_rf_wdata}	32		写入普通寄存器的数据
6	debug_wb_pc	32	输出	用于检查 PC 值、寄存器写信号、写入数据等是否正确。
7	debug_wb_rf_wen	4		

8	debug_wb_rf_wnum	5		
9	debug_wb_rf_wdata	32		

2.5.3 功能模块说明

对应 2.5.1：

1. 传递写寄存器相关信号：

```
assign wb_to_rf_bus = {
    hi_ex_we,
    hi_ex_wdata,
    lo_ex_we,
    lo_ex_wdata,
    rf_we,
    rf_waddr,
    rf_wdata
};
```

2. 控制复位或暂停：（类似于 IF 段）

通过时序逻辑，判断复位信号 rst 是否被置为 1，是则清零；

判断 WB 段是否被暂停，暂停则清零，否则直接接收 MEM 段传来的信号和数据。

控制暂停的信号设置，在 CTRL.v 中，

（1）当处于 EX 段的指令请求暂停时，要求 IF、ID、EX 段暂停，而 MEM、WB 段继续；

（2）当处于 ID 段的指令请求暂停时，要求 IF、ID 段暂停，而 EX、MEM、WB 段继续；

因此通过以下时序逻辑实现：

```
always @ (*) begin
    if (rst) begin
        stall = `StallBus'b0;
    end else if(stallreq_for_ex == 1'b1) begin
        stall = `StallBus'b001111;
    end else if(stallreq_for_load == 1'b1) begin
        stall = `StallBus'b000111;
    end
    else begin
        stall = `StallBus'b0;
    end
end
```

2.5.4 结构示意图

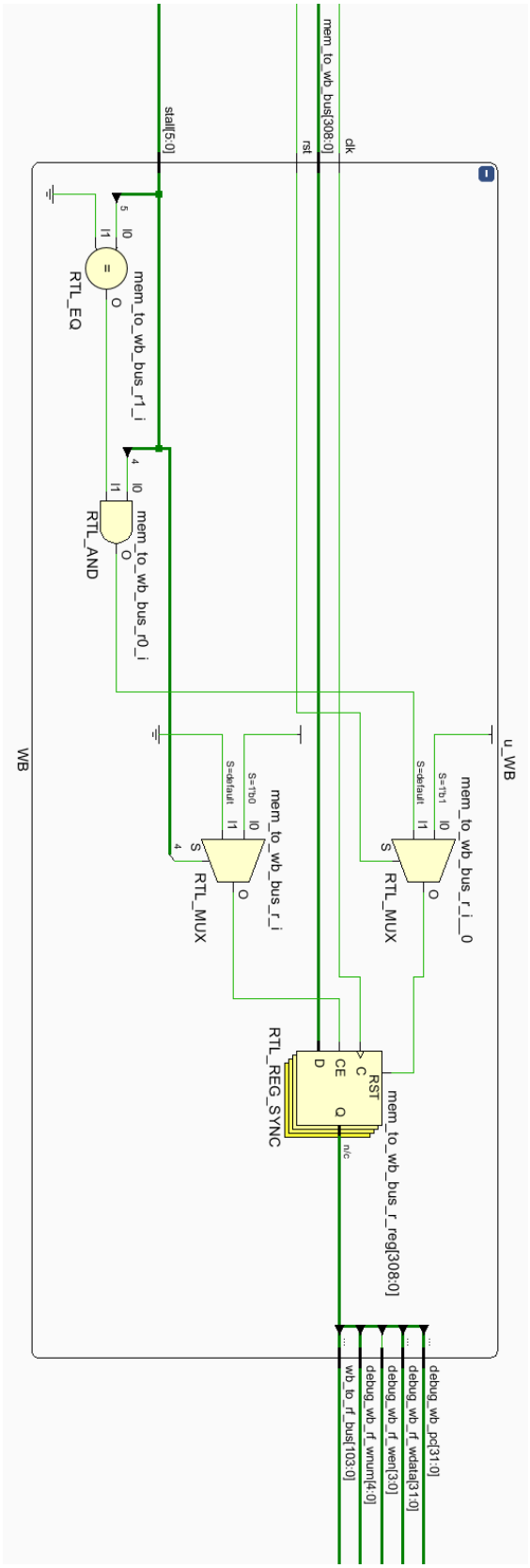


图 12 WB 段结构图

三．总结与体会

李语竹：收获和体会：一开始拿到这个实验的时候，不知道何从下手。在老师和学长的点拨和指导，以及自己从书中慢慢摸索后，逐渐从盲目困惑一步步走向正轨。因此整个过程中，我觉得不仅仅是对计算机系统学习的一次复习巩固以及应用，更加是对自己自学能力的一种挑战。在调试的过程中，就会遇到各种各样的 bug，一开始不熟悉 Vivado 这个软件的调试方法，不会看波形图，在学长和同学们的帮助下，学会了自己添加需要检查的信号 debug，比如在处理 load 数据相关时，发现下一条指令将当前暂停未处理完的指令覆盖，因此需要设置寄存器暂存下一条指令。通过实验的学习之后，对指令和流水段有了更实际地理解，将理论进一步运用到实践中。看着自己的代码越来越完善自己的心里也充满了成就感。

改进与建议：希望分配给实验的学时更长一点，或者专门开设一门课程设计用于此实验，这样可以投入更多时间和精力在这方面。

孔繁恒：收获和体会：在近半个学期的实验中我学到了很多，确实如老师所说，光是学计算机体系结构的理论课，对于这门课的理解是不够的，只有自己动手做实验才会深刻理解这门课的知识。在实验中，流水线技术、分支跳转指令、暂停气泡等内容得以复现，理论与实践相结合使我对课上的理论内容加深了印象，理论课和实验相辅相成。在添加指令和乘除法器中要考虑多种因素，如乘除法器中如果两个数是一正一负应该怎么算、被除数是 0 应该怎么处理、什么时候表示计算结束等等。CPU 作为计算机系统的运算和控制核心，而我国目前缺少自主知识产权的 CPU 技术，信息产业受制于人，因此 CPU 应该作为我们未来的重点研究方向之一，这个实验一定程度上为我奠定了一些基础。

改进与建议：因为这次实验除了上课讲的内容外没有任何前置知识，起手很难，且花费时间很长，建议本次实验可以单独拎出来作为一门课程设计，让我们有充足的时间和精力放到这上面来。

郭晶晶：收获和体会：经过了近几星期的学习实践，我明白了硬件的魅力并不只存在于理论知识，更多的在于实践。我对它的制作有了更多的认识与了解，学习了很多种操作指令和他们相关的添加方法等等，并且明白了只有通过上机实践操作才能更好地掌握这门课程，我也因此明白了自己在理论学习中存在的不足。首先在第一个点位的气泡的处理上比较繁琐，通过不断地尝试和向同学和助教请教才有所改变，然后在一些具体的指令添加上，有时候忘记声明接口，有时候忘记更改线宽。之前对数据相关一直没能深刻理解，经过这次锻炼也有所提高。我认识到，操作比理论知识更重要，只有实践才能更好的掌握知识，才能写出更好的程序，收获更多的知识。

改进与建议：希望实验时间再稍微延长一点，我们都很喜欢这个实验，前期上手挺难的，但是做到后面就慢慢有乐趣了，可以学到很多知识，很喜欢这个实验。

四．参考文献

[1] 雷思磊. 自己动手写 CPU[M]. 第 1 版. 电子工业出版社, 2014-9-1.

[2] “系统能力培养大赛”. MIPS 指令系统规范[M]. v1.01. .

[3] lucky lures. Verilog 乘法器[EB/OL]. [2021-11]. https://blog.csdn.net/weixin_43862765/article/details/98886486.

[4] qq1327798176. 移位相加乘法汇编原理[EB/OL]. [2021-11]. https://blog.csdn.net/qq1327798176/article/details/72776513.