重庆大学课程设计报告

课程设计题目:	Gemini CPU
学　　院:	计算机学院
专 业 班 级:	计算机科学与技术　卓越 1 班　卓越 2 班
年　　级:	2019
学　　生:	陈诗冲　钟祥新
学　　号:	20194190　20194207
完 成 时 间:	2022 年　1 月　5 日
成　　绩:	
指 导 教 师:	钟将

重庆大学教务处制

项目	分值	优秀 $100 > x \geq 90$	良好 $90 > x \geq 70$	中等 $80 > x \geq 70$	及格 $70 > x \geq 60$	不及格 $x < 60$	评分
		参考标准					
学习态度	15	学习态度认真，科学作风严谨，严格保证设计时间并按任务书中规定的进度开展各项工作	学习态度比较认真，科学作风良好，能按期圆满完成任务书规定的任务	学习态度尚好，遵守组织纪律，基本保证设计时间，按期完成各项工作	学习态度尚可，能遵守组织纪律，能按期完成任务	学习马虎，纪律涣散，工作作风不严谨，不能保证设计时间和进度	
技术水平与实际能力	25	设计合理、理论分析与计算正确，实验数据准确，有很强的实际动手能力、经济分析能力和计算机应用能力，文献查阅能力强、引用合理、调查调研非常合理、可信	设计合理、理论分析与计算正确，实验数据比较准确，有较强的实际动手能力、经济分析能力和计算机应用能力，文献引用、调查调研比较合理、可信	设计合理，理论分析与计算基本正确，实验数据比较准确，有一定的实际动手能力，主要文献引用、调查调研比较可信	设计基本合理，理论分析与计算无大错，实验数据无大错	设计不合理，理论分析与计算有原则错误，实验数据不可靠，实际动手能力差，文献引用、调查调研有较大的问题	
创新	10	有重大改进或独特见解，有一定实用价值	有较大改进或新颖的见解，实用性尚可	有一定改进或新的见解	有一定见解	观念陈旧	
论文(计算书、图纸)撰写质量	50	结构严谨，逻辑性强，层次清晰，语言准确，文字流畅，完全符合规范化要求，书写工整或用计算机打印成文；图纸非常工整、清晰	结构合理，符合逻辑，文章层次分明，语言准确，文字流畅，符合规范化要求，书写工整或用计算机打印成文；图纸工整、清晰	结构合理，层次较为分明，文理通顺，基本达到规范化要求，书写比较工整；图纸比较工整、清晰	结构基本合理，逻辑基本清楚，文字尚通顺，勉强达到规范化要求；图纸比较工整	内容空泛，结构混乱，文字表达不清，错别字较多，达不到规范化要求；图纸不工整或不清晰	

指导教师评定成绩：

指导教师签名：

Gemini CPU

陈诗冲、钟祥新

1 设计简介

我们小组设计的处理器名为 **Gemini CPU**，采用双发射五级流水，可以执行 MIP32 的指令子集，包含移位指令、逻辑运算指令、算术运算指令、数据移动指令、分支跳转指令、访存指令。

我们小组采用取指阶段发射两条指令，主流水线 (master) 和辅流水线 (slave) 并行执行的方法实现双发射，通过数据前推减少流水线阻塞，在最大程度上提高 IPC。

通过运行计算机组成原理 lab4 的 testbench，我们在一定程度上验证了该处理器的正确性。

1.1 小组分工说明

- 陈诗冲
  1. 参与 Sirius 参考代码的阅读和标注
  2. 负责 datapath、dual_engine、pc_crtl 等代码编写
  3. 负责主体代码的代码调试
  4. 负责数据通路图等绘图工作以及少量报告写作
- 钟祥新
  1. 参与 Sirius 参考代码的阅读和标注
  2. 负责 hazard、forwarding、decode_field 等代码编写；
  3. 负责部分代码调试
  4. 负责主体报告的写作

2 设计方案

2.1 总体设计思路

本次硬件综合设计的目的是设计一个支持 MIP32 精简汇编指令子集的双发射五级流水线 CPU，旨在通过超标量技术最大程度地提高处理器 IPC。

我们小组的设计基于计算机组成原理实验 4 的单发射五级流水线，通过添加第 2 条流水

线实现了主流水线 (master) 和辅流水线 (slave) 的并行执行。为了解决数据冒险,对于可以前推的数据,进行 master-master, master-slave, slave-slave 的必要前推。对于无法前推的数据,进行必要的阻塞,等到操作数就位,再执行指令,使得两条流水线中的指令可以正确执行。Gemini CPU 的 datapath 见图1。

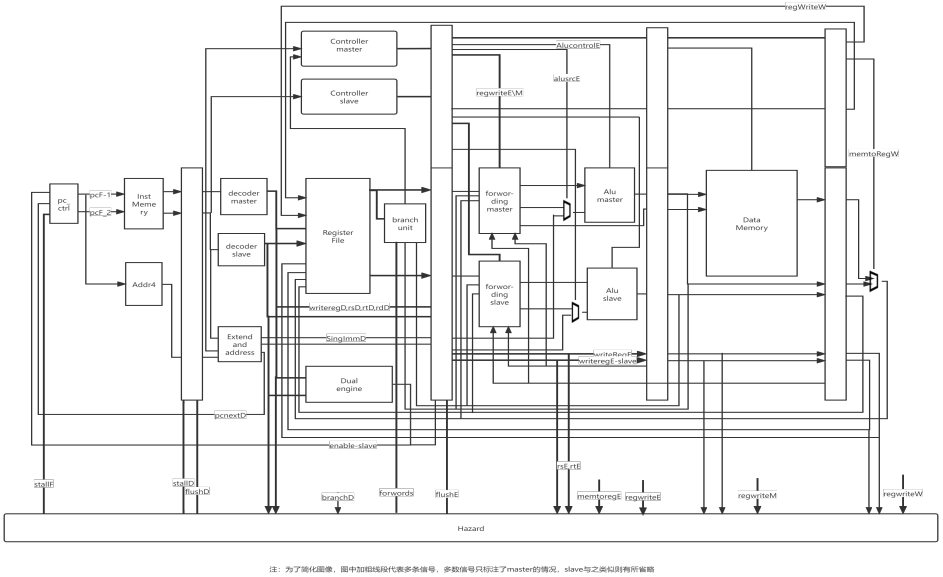

图 1: 数据通路图

在**取指阶段**,PC 的值被送往指令存储器 inst_mem_dual,指令存储器的前两条指令会被取出。在**译码阶段**,双发射判断模块 dual_engine 根据双发射判断逻辑决定是否发射第 2 条指令和如何更新 pc_next。若发射第 2 条指令,则 pc_next 被赋值为 pc+8;否则,pc_next 被赋值为 pc+4。同时在译码阶段,分支跳转指令也在此执行,一旦需要跳转,则跳转指令被送往 PC,PC 在下一周期更新到目的地址。在**执行阶段**,最多两条指令会并行地执行 ALU 操作,执行结果被送往访存阶段。在**访存阶段**,主流水线正常地执行访存指令,辅流水线不执行访存指令。访存结果被送往写回阶段,运算/访存结果被送往目的寄存器。

2.2 模块设计

2.2.1 对外接口

mips 模块对外的接口及含义如表1所示,datapath、controller、memory 连接示意图如图2所示。

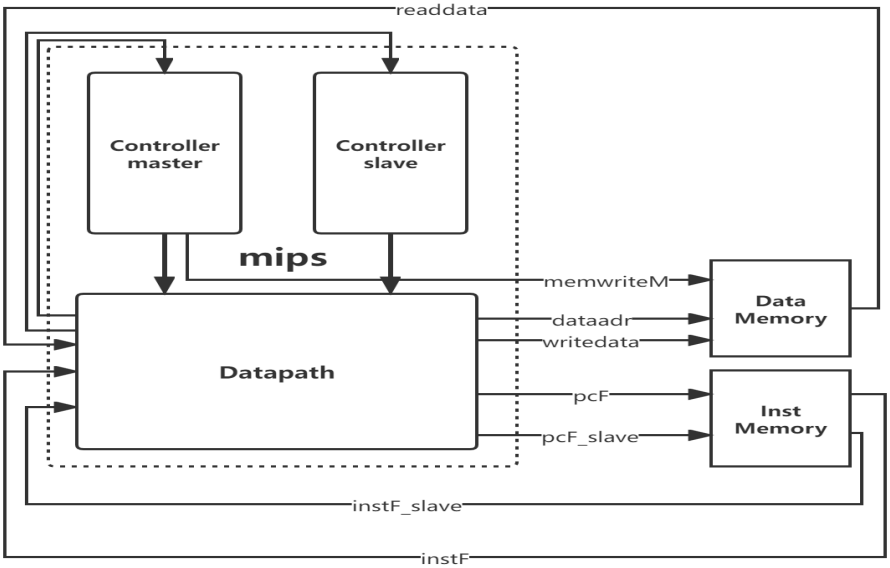

图 2: 顶层模块结构图

表 1: mips 模块接口

信号名	方向	位宽	功能描述
clk	Input	1 bit	时钟信号
rst	Input	1 bit	复位信号
pcF	Output	32 bits	master 需要取的指令地址
instrF	Input	32 bits	master IF 阶段取的指令
memwriteM	Output	1 bit	master MEM 阶段存储器写使能
aluoutM	Output	32 bits	master MEM 阶段的 ALU 结果
writedataM	Output	32 bits	master MEM 阶段要写入寄存器堆的数据
readdataM	Input	32 bits	master MEM 阶段从存储器中读取的数据
pcF_slave	Output	32 bits	slave 需要取的指令地址
instrF_slave	Input	32 bits	slave IF 阶段取的指令

2.2.2 取指阶段

取指阶段的主要模块是 pc_ctrl.sv,接口信号及其含义见2。

表 2: pc_ctrl 模块接口

信号名	方向	位宽	功能描述
clk	Input	1 bit	时钟信号
rst	Input	1 bit	复位信号
en	Input	1 bit	分支跳转使能信号
inst_ok_1	Input	1 bit	第 1 条指令有效位
inst_ok_2	Input	1 bit	第 2 条指令有效位
branch_taken	Input	1 bit	发生跳转信号
branch_address	Input	1 bit	分支跳转地址
pc_address_1	Output	32 bits	输出指令地址 1
pc_address_2	Output	32 bits	输出指令地址 2
flushD_dual	Output	1 bit	master 刷新 ID 阶段流水线
flushD_dual_slave	Output	1 bit	slave 刷新 ID 阶段流水线

取指阶段,首先从指令存储器取 2 条连续指令。PC 寄存器内放置下一条被取指的指令地址,pc_ctrl 模块在译码阶段根据上一周期取出的两条指令的 inst_ok 信号判断下一个时钟周期主流水线和辅流水线的 PC 值,判断原则有以下 2 条:

1. 主流水线和辅流水线在 decode 时的 inst_ok 都是 1,则 PC 加 8
2. 只有主流水线的 inst_ok 是 1,PC 更新为此时取出的第一条指令地址减 4,即 $pc\_next = pc\_address\_one - 4$,并刷新译码到执行阶段的寄存器

```

always @(*) begin
    if(rst) begin
        pc_address_next <= 32'h0000_0000;
        pc_address_one <= 32'h0000_0000;
        pc_address_two <= 32'h0000_0000;
    end
    else if(branch_taken)
            pc_address_next <= branch_address;
    else if(en) begin
        flushD_dual <= 0;
        flushD_dual_slave <= 0;
        if(inst_ok_1 && inst_ok_2) begin
            pc_address_next <= pc_address_one + 32'd8;
        end
        else if(inst_ok_1) begin
                pc_address_next <= pc_address_one - 32'd4;
                flushD_dual <= 1;
                flushD_dual_slave <= 1;
            end
        else begin
            pc_address_next <= pc_address_next;
            pc_address_one <= pc_address_one;
            pc_address_two <= pc_address_two;
        end

    end
    else begin
        pc_address_next <= pc_address_next;
    end
end
```

2.2.3 译码阶段

译码阶段需要完成指令译码、读取寄存器堆、双发射判断、分支跳转工作，除此以外，旁路机制也在译码阶段完成。

2.2.3.1 译码模块 译码模块由主译码模块 maindec 和解码模块 decode_field 组成，二者均由组合逻辑实现。解码模块负责解码指令，根据输入指令的类型生成不同字段。解码模块将操作码传入主译码模块，主译码模块根据操作码生成控制信号，控制后续流水线的执行。上述两个模块的信号及定义如下表所示。

表 3: maindec 模块接口

信号名	方向	位宽	功能描述
op	Input	6 bits	操作码
memtoreg	Output	1 bit	写回寄存器堆的选择信号
memwrite	Output	1 bit	存储器的写使能
branch	Output	1 bit	分支跳转信号
alusrc	Output	1 bit	ALU 操作数选择信号
regdst	Output	1 bit	寄存器读端口选择信号
regwrite	Output	1 bit	寄存器堆写使能
jump	Output	1 bit	无条件跳转信号
aluop	Output	2 bits	ALU 操作信号

表 4: decode_field 模块接口

信号名	方向	位宽	功能描述
instruction	Input	32 bits	指令
opcode	Output	6 bits	操作码
rs	Output	5 bits	R 型指令第 1 个操作数寄存器地址
rt	Output	5 bits	R 型指令第 2 个操作数寄存器地址
rd	Output	5 bits	R 型指令目的寄存器地址
funct	Output	5 bits	功能码
mem_op_type	Output	2 bits	访存指令类型

2.2.3.2 寄存器堆模块 我们小组修改了计算机组成原理实验 4 的 regfile 文件，将端口改成了 4 个读端口和 2 个写端口，主流水线和辅流水线共享一个寄存器堆。寄存器堆由 32 个 32 位寄存器组成，在译码阶段和写回阶段被使用。寄存器堆的实现有 3 点值得注意：

1. $0 寄存器返回值永远为 0
2. 当一个寄存器号同时进行读写时，读的值等于写入的值
3. 辅流水线的写优先级高于主流水线

寄存器堆的接口信号及含义如下所示：

表 5: 寄存器堆模块接口

信号名	方向	位宽	功能描述
ra1_a	Input	5 bits	master 1 号读端口
ra2_a	Input	5 bits	master 2 号读端口
rd1_a	Output	32 bits	master 1 号端口读出的数据
rd2_a	Output	32 bits	master 2 号端口读出的数据
we_a	Input	1 bit	master 寄存器堆的写使能
wa_a	Input	5 bits	master 寄存器堆的写地址
wd_a	Input	32 bits	master 寄存器堆的写数据
ra1_b	Input	5 bits	slave 1 号读端口
ra2_b	Input	5 bits	slave 2 号读端口
rd1_b	Output	32 bits	slave 1 号端口读出的数据
rd2_b	Output	32 bits	slave 2 号端口读出的数据
we_b	Input	1 bit	slave 寄存器堆的写使能
wa_b	Input	5 bits	slave 寄存器堆的写地址
wd_b	Input	32 bits	slave 寄存器堆的写数据

寄存器堆关键读写代码如下所示:

```
always_comb begin: read_data2_b
    if (ra2_b == 5'b00000)
        rd2_b = 32'h0000_0000;
    else if (we_a && wa_a == ra2_b)
        rd2_b = wd_a;
    else if (we_b && wa_b == ra2_a)
        rd2_b = wd_b;
    else
        rd2_b = tmp_register[ra2_b];
end


always_ff @( posedge clk ) begin : write_data
    if (rst) begin
        for (int i = 0; i < 31; i++)
            tmp_register[i] <= 32'h0000_0000;
    end
    else begin
        if (we_a) begin
            tmp_register[wa_a] <= wd_a;
        end
        if (we_b) begin
            tmp_register[wa_b] <= wd_b;
        end
    end
end
```

2.2.3.3 双发射控制模块 双发射控制模块 dual_engine 的作用是根据当前的情况判断是否进行双发射。辅流水线是否发射由以下 4 条原则确定:

1. 主流水线不发射,辅流水线一定不发射
2. 辅流水线是分支跳转指令,一定不发射
3. 辅流水线是访存指令,一定不发射
4. 主流水线译码阶段写入的目标寄存器与辅流水线读取的寄存器相同时,辅流水线一定不发射

5. 主流水线和辅流水线间存在 load-use 阻塞时，辅流水线一定不发射

双发射判断逻辑的核心代码如下：

```
always @(*) begin
    if( (!enable_master) ||
        (branchD_slave) ||
        (memtypeD_slave != `MEM_NOT)
    ) begin
        _enable_slave = 1'b0;
    end
    else begin
        if(regwriteD && (writeregD != 5'd0)) begin
            if(opD_slave == 6'd0) begin
                _enable_slave = (|((writeregD ^ rsD_slave) & (writeregD ^ rtD_slave)));
            end
            else begin
                _enable_slave = (|((writeregD ^ rsD_slave)));
            end
        end
        else begin
            _enable_slave = 1'b1;
        end
    end
end
always @(*) begin
    if(memtypeE == `MEM_LOAD && ((writeregE == rsD_slave) ||
            (writeregE == rtD_slave)))
        load_use_stall_slave = 1'd1;
    else
        load_use_stall_slave = 1'd0;
end
```

2.2.3.4 旁路模块 我们改变了计算机组成原理实验 4 的旁路实现机制，旁路模块输出的是前推的数据而不是多选器的选择信号。因为双发射处理器有两条流水线，每条流水线的执行和访存阶段都可能前推数据到 ALU 输入端口，因此旁路模块需要传入主流水线和辅流水线的执行和访存阶段写入寄存器堆的数据。由于两条流水线的执行阶段 ALU 两个端口都需要传入数据，因此需要实例化 4 个旁路模块。

实现旁路模块时需要注意主/辅流水线前推的优先级，我们按照图3定义数据前推的优先级。旁路模块根据写寄存器号和读寄存器号是否相同，如果相同则将写寄存器的数据直接作为读寄存器的结果送到下一流水级。旁路模块的接口及含义如表6所示。

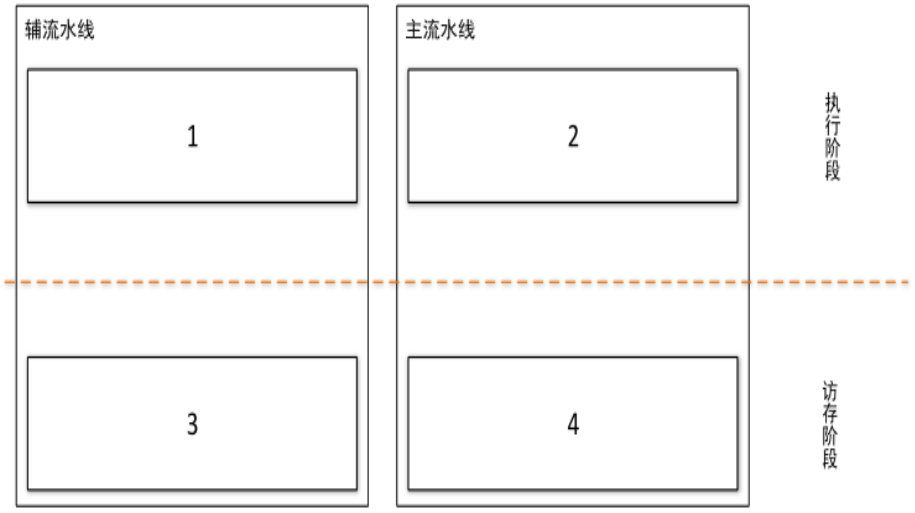

图 3: 前推优先级示意图

表 6: 旁路模块接口

信号名	方向	位宽	功能描述
regwriteM_slave	Input	1 bit	slave E 阶段的指令是否写寄存器
writeregM_slave	Input	5 bits	slave E 阶段的指令写入寄存器目标地址
wdataM_slave	Input	32 bits	slave E 阶段的指令写入寄存器的数据
regwriteM_master	Input	1 bit	master E 阶段的指令是否写寄存器
writeregM_master	Input	5 bits	master E 阶段的指令写入寄存器目标地址
wdataM_master	Input	32 bits	master E 阶段的指令写入寄存器的数据
regwriteW_slave	Input	1 bit	slave M 阶段的指令是否写寄存器
writeregW_slave	Input	5 bits	slave M 阶段的指令写入寄存器目标地址
wdataW_slave	Input	32 bits	slave M 阶段的指令写入寄存器的数据
regwriteW_master	Input	1 bit	master M 阶段的指令是否写寄存器
writeregW_master	Input	5 bits	master M 阶段的指令写入寄存器目标地址
wdataW_master	Input	32 bits	master M 阶段的指令写入寄存器的数据
reg_addr	Input	5 bits	寄存器读地址
reg_data	Input	32 bits	寄存器读出来的值
result_data	Output	32 bits	前推的数据

2.2.3.5 分支模块 参考计算机组成原理实验 4 的分支逻辑,我们在 decode 阶段判断当前指令是否是分支指令以及是否执行分支跳转操作。

分支跳转逻辑由 hazard 模块实现,除此以外,在 hazard 模块中我们处理了 load-use stall 和 branch stall,hazard 模块的接口如表7所示:

表 7: hazard 模块接口

信号名	方向	位宽	功能描述
stallF	Output	1 bit	master IF 阶段阻塞信号
stallF_slave	Output	1 bit	slave IF 阶段阻塞信号
rsD	Input	5 bits	master 读寄存器号
rtD	Input	5 bits	master 读寄存器号
branchD	Input	1 bit	master branch 信号
forwardaD	Output	1 bit	master 数据前推选择信号
forwardbD	Output	1 bit	master 数据前推选择信号
stallD	Output	1 bit	master ID 阶段阻塞信号
stallD_slave	Output	1 bit	slave ID 阶段阻塞信号
rsD_slave	Input	5 bits	slave 读寄存器号
rtD_slave	Input	5 bits	slave 读寄存器号
rtE	Input	5 bits	master 存储器要写回的寄存器
writeregE	Input	5 bits	master EX 阶段要写回的寄存器
regwriteE	Input	1 bit	master EX 阶段寄存器堆写使能
memtoregE	Input	1 bit	master WB 阶段写回寄存器的数据选择信号
flushE	Output	1 bit	master 刷新 EX-MEM 寄存器
flushE_slave	Output	1 bit	slave 刷新 EX-MEM 寄存器
flushD	Output	1 bit	master 刷新 ID-EX 寄存器
flushD_slave	Output	5 bits	slave 刷新 ID-EX 寄存器
rtE_slave	Input	5 bits	slave 存储器要写回的寄存器
writeregE_slave	Input	5 bits	slave EX 阶段要写回的寄存器
regwriteE_slave	Input	1 bit	slave EX 阶段寄存器堆写使能
memtoregE_slave	Input	1 bit	slave WB 阶段写回寄存器的数据选择信号
writeregM	Input	5 bits	master MEM 阶段要写回的寄存器号
regwriteM	Input	1 bit	master MEM 阶段寄存器写使能
memtoregM	Input	1 bit	master MEM 存储器读出数据写入寄存器的使能信号
writeregM_slave	Input	5 bits	slave MEM 阶段要写回的寄存器号
regwriteM_slave	Input	1 bit	slave MEM 阶段寄存器写使能
memtoregM_slave	Input	1 bit	slave MEM 存储器读出数据写入寄存器的使能信号
writeregW	Input	5 bits	master WB 阶段要写回的寄存器号
regwriteW	Input	1 bit	master WB 阶段寄存器写使能
branch_taken	Input	1 bit	master 分支跳转信号

首先介绍分支跳转的逻辑。我们将分支指令的判断提前到 decode 阶段，减少两条指令的执行。由于双发射加入了辅流水线，所以分支指令的数据前推需要同时考虑主流水线和辅流水线在访存阶段要写入的目的寄存器是否与分支指令 decode 阶段的读寄存器是否相同，如果相同就需要执行数据前推。与前文的旁路模块相似，我们令辅流水线前推的优先级大于主流水线前推的优先级，即如果主/辅流水线在访存阶段写回寄存器相同，则前推辅流水线要写回的数据。数据前推代码如下所示：

```
always_comb begin : branchD_a
  if (rsD != 0 & rsD == writeregM & regwriteM)
    forwardaD = 2'b00;
  if (rsD != 0 & rsD == writeregM_slave & regwriteM_slave)
    forwardaD = 2'b01;
  else
    forwardaD = 2'b10;
end
```

```
always_comb begin : branchD_b
  if (rtD != 0 & rtD == writeregM & regwriteM)
    forwardbD = 2'b00;
  if (rtD != 0 & rtD == writeregM_slave & regwriteM_slave)
    forwardbD = 2'b01;
  else
    forwardbD = 2'b10;
end
```

下面考虑分支阻塞,由于只前推了访存阶段的数据,所以存在以下情况会发生分支阻塞:

1. master 执行阶段要写回的寄存器号与 master 分支指令 decode 阶段读寄存器号相同
2. slave 执行阶段要写回的寄存器号与 master 分支指令 decode 阶段读寄存器号相同
3. master 要前推的指令是 load 型指令

分支阻塞关键代码如下:

```
  assign  branchstallD = branchD &
        (regwriteE &
        (writeregE == rsD | writeregE == rtD) |
        memtoregM &
        (writeregM == rsD | writeregM == rtD));
  assign  branchstallD_slave = branchD &
        (regwriteE_slave & (writeregE_slave == rsD | writeregE_slave == rtD));
```

2.2.4 执行阶段

执行阶段主要完成算数逻辑运算和生成访存地址等工作。主/辅流水线各有一个 ALU,主流水线的 ALU 完成算数逻辑运算和访存地址计算,辅流水线仅用来执行算数逻辑运算。由于计算机组成原理实验 4 的 testbench 不涉及乘除法,在朴素双发射 CPU 的设计中不考虑乘除法。在后续完善指令中,拟调用 Xilinx 提供的 IP 核完成乘除法指令,并辅以计数器以及运算前/后的求补器完成各种有符号/无符号乘法运算。

2.2.5 访存阶段

访存阶段要完成的功能主要是访问存储器。我们调用 Xilinx 提供的 Block Memory 分别实现指令存储器和数据存储器。由于取指阶段每次需要取出两条指令,所以指令存储器由双端口 ROM 实现,见图4

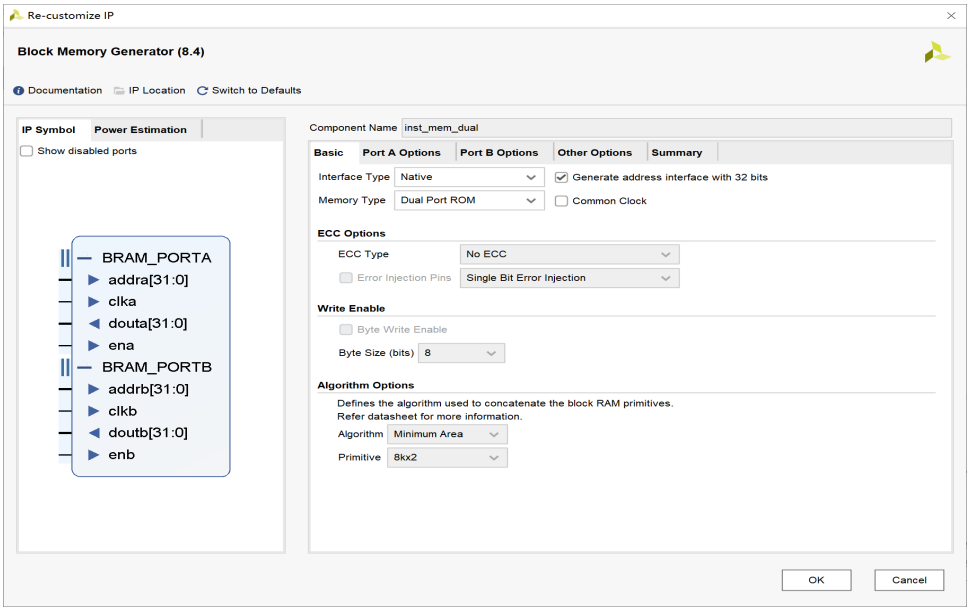

图 4: 指令存储器配置图

数据存储器由单端口 RAM 实现，见图5

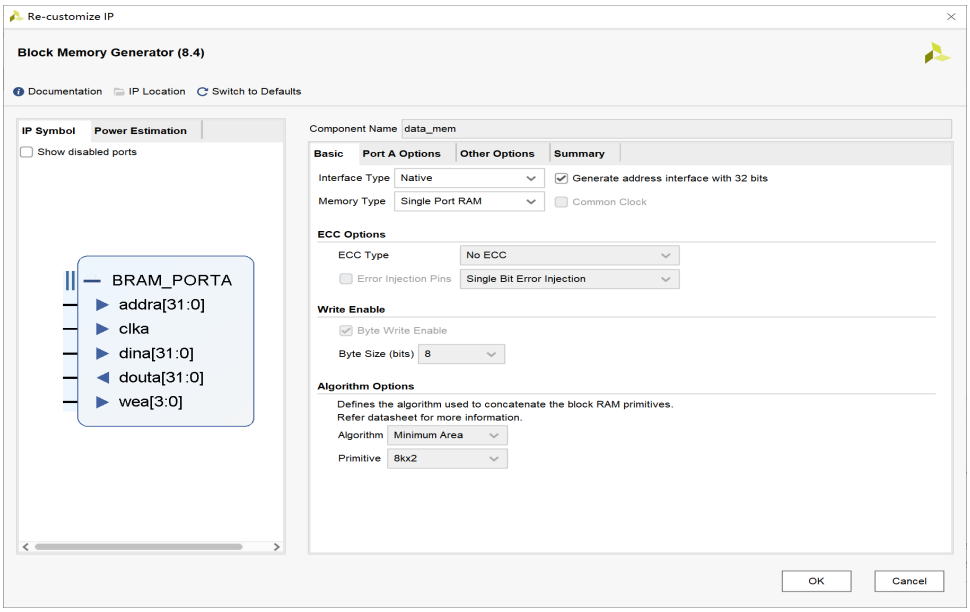

图 5: 数据存储器配置图

2.2.6 写回阶段

写回阶段主要完成存储器读出的数据或者 ALU 运算结果写回寄存器堆的操作，设计与实现思路与计算机组成原理实验 4 相似，此处不再赘述。

3 设计过程

3.1 设计流水账

1 月 2 号下午 (交完作业) 两个人再次把硬综的所有资料和 cache,axi 总线部分的代码再回顾了一下,开始学习北邮的代码。

1 月 3 号两个人一起学习了一整天北邮的 Sirius 双发射流水线 cpu(没注释太难读了,我们添加了详尽的注释)。

1 月 4 号开始写代码,尝试将实验四改成双发射流水线:一个改人主要写 datapath,另一个写功能器件。

1 月 5 号大体框架已经成型了,debug 到凌晨 3 点 (果然还是时序问题最难调)。

1 月 6 号一个优化代码、画图;一个写报告,讨论了取值判断的逻辑。

3.2 错误记录

3.2.1 错误 1

(1) 错误现象: 双发射 pc 值在两条指令不能同时发射时更新错误;
(2) 分析定位过程: 双发射流水线在检测到当前提取的两条指令不能同时发射时需要修正下次提取指令的位置,在仿真过程中第一次检测到此种情况时,PC 的更新值计算正确,但是在紧接着提取的两条指令也不能同时发射时,PC 的更新值就错误了。
(3) 错误原因: 卡了半天,我们最开始看波形图观察到规律是的当前的不能同时发射的两条指令的上一状态时双发还是单发会影响当前 pc 值的更新计算,就加入了寄存器记录上一次发射的状态,然后继续添加了逻辑修复了 bug。后面再次优化代码的时候我们对比了错误前后的波形图,发现错误的原因是因为之前指令只能单发射时并没有 flush 掉提出的错误指令 (因为在 ID 阶段才能检测出发射逻辑)。所以问题本质是当前上下流水线阶段的未对齐导致的。
(4) 修正效果: 因此当检测到 ID 阶段的两条指令不能双发射时,生成 flushD 控制信号将在下一阶段将此时错误提取的指令进行刷新。此时相当于两条指令的执行需要 6 个时钟周期,也是合理的(毕竟顺序单发两条指令就是 6 个周期)。
(5) 归纳总结(可选): 清楚代码的时序逻辑和组合逻辑,注意时序问题。首先一定要理清整个阶段的更新逻辑再写代码。

3.2.2 错误 2

(1) 错误现象: 当比如 load-use 导致的 master 流水线阻塞时,后序指令的运行结果错误。

(2) 分析定位过程: 观察波形图我们发现是在 master 恢复执行时下一条指令执行阶段获得的前推数据错误。然后我们追踪之前指令的执行情况,分析前推逻辑定位到了问题。

(3) 错误原因: 问题是因为当 master 被 stall 时 slave 流水线正常执行,导致 slave 比 master 的相应指令状态超前一个周期,从而在数据前推时得到了错误的数据。此时流水线已经脱离了正常的顺序执行顺序。

(4) 修正效果: 修正方法即是当 master 流水线 stall 时 slave 也跟着 stall 以保证指令的顺序执行。

(5) 归纳总结(可选): 当前我们的设计还是局限于指令的顺序执行阶段,不能违反这个大前提。因此就算打 slave 没有任何数据冒险产生的 stall 也必须跟 master 维持指令状态的同步。

4 设计结果

本项目最终得到了整个项目的 rtl 代码,其构成了整个数据通路,在计算机组成原理实验 4 的测试文件中通过所有指令测试,验证了双发射处理器的正确性。GitHub 仓库地址为`https://github.com/xuanranxiaoshi/mips-cpu`。

4.1 设计交付物说明

4.1.1 目录层次

所提交的文件包含如下代码文件,其功能都写在注释中。

```
rtl:
    adder.v # 加法器
    alu.v # ALU
    aludec.v # ALU 控制信号生成
    common.vh # 宏定义文件
    controller.v # 控制器
    datapath.v # 数据通路
    decode_field.sv # 指令分段
    dual_engine.v # 控制双发射
    eqcmp.v # 判断两个数是否相等
    fake_top.v # 验证双端口 ROM 的伪 top
    flopenr.v # 带有 enable、reset 的触发器
    flopenrc.v # 带有 enable、reset 与 clear 的触发器
    flopr.v # 带有 reset 的触发器
    floprc.v # 带有 reset 与 clear 的触发器
    forwarding.sv # 旁路单元
    hazard.sv # 阻塞单元
    maindec.v # 主译码器
    mips.v # CPU 顶层文件
    mipstest.coe # 测试指令集
    mux2.v # 二选一多路选择器
    mux3.v # 三选一多路选择器
    pc.v # PC
    pc_ctrl.sv # 选择下一条发射指令
    regfile.v # 老版的寄存器堆,两个读端口,一个写端口
    register.sv # 新版的寄存器堆,四个读端口,两个写端口
```

```
    signext.v  # 有符号扩展
    sl2.v  # 左移两位
    tb_inst_mem_dual.v  # 双端口测试文件
    testbench.v  # 测试文件
    top.v  # 最顶层模块
    tree.txt  # 文件树

  ip
      data_mem
           data_mem.xci  # 数据存储器 IP

      inst_mem
            inst_mem.xci  # 指令存储器 IP
```

4.2 设计演示结果

Gemini CPU 成功通过计算机组成原理实验 4 的 testbench，验证了双发射处理器的正确性，波形图如图6所示，仿真结果如图7所示。

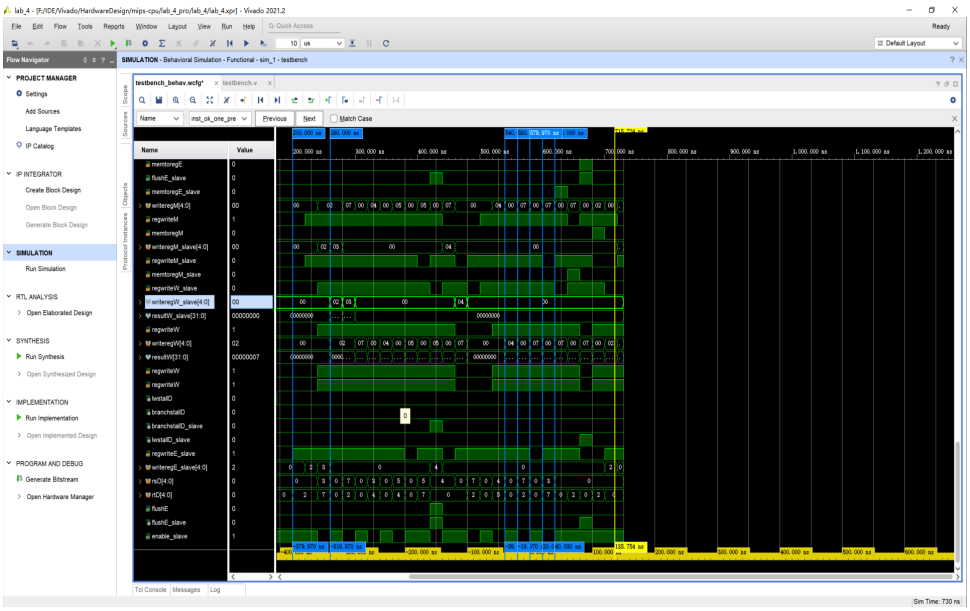

图 6: 仿真波形图

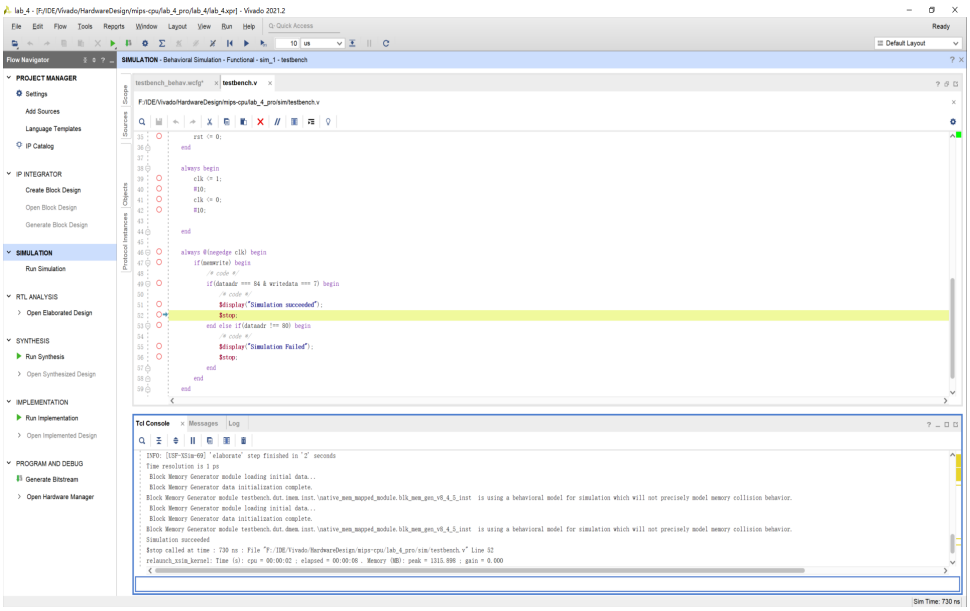

图 7: 仿真通过图

5 参考设计说明

5.1 IP 核

本项目中的 IP 核均由 Xilinx 提供，包括指令存储器、数据存储器。

5.2 旁路模块

本项目的旁路模块在计组实验 4 的基础上改进，以及双发射控制模块的思路来自于`https://github.com/name1e5s/Sirius`。

5.3 单发射五级流水线 CPU

由于小组成员计组实验 4 的代虽然通过 testbench，但个人代码可读性较差，所以实验 4 的 CPU 代码来自吕昱峰学长的代码，仓库地址为`https://github.com/lvyufeng/step_into_mips/tree/lab_4`。

6 总结

本次项目实现了一个顺序双发五级流水线 CPU，通过了计组实验 4 的 testbench。由于时间不够，所以本次实验仅实现了计组实验 4 需要的 MIP32 指令。小组成员在没有任何往届双发射资料的情况下，通过查阅资料、寻找开源实现方法，加上自己对双发射 CPU 的

理解与思考,完成了初定的目标。也为后面的同学设计双发射 CPU 提供了借鉴方法。通过本次实验,小组成员加深了对超标量处理器设计和计算机系统结构的理解,并掌握了使用 System Verilog 设计一个双发射 CPU 的方法,收获颇丰。

7 供同学们吐槽之用。有什么问题都可以直接写在这。

7.1 陈诗冲

1. 硬综和考试时间重合实在是有点多难受,希望以后能想办法改进吧
2. 顺序双发射对于相邻指令之间数据依赖强的指令而言效果有限,就比如的实验四的执行指令,多数指令都使得双发射流水线退化成了单发射,而这一点对于顺序执行的双发射指令而言是不能避免的。
3. 再则就是如果时间足够,确实能尝试一下有趣的事情,就比如乱序执行。但是我们学校这方面资料确实没啥祖传的,确实是待发展。我们做的顺序双发也仅仅是很小很小的一个拓展。

7.2 钟祥新

1. 时间安排的不是很合理,考完期末只剩下不到 10 天的时间做硬综。中间还夹杂着计网作业、project,部分同学还有离散数学期末考试,希望以后时间的安排可以更合理一点。
2. 开源资料太少,对于想做多发射的同学,除了陈泱宇学长提供指导外,网上查找资料也很困难。我们小组只有联系龙芯杯做过多发射的队伍,请教双发射的设计方法,因此希望今后能够提供充足的指导。

8 参考文献

[3] [2] [1]

参考文献

[1] Hennessy and JohnL. *Computer Architecture A Quantitative Approach:* 计算机体系结构量化研究方法. China Machine Press, 2002.

[2] 姚永斌. 超标量处理器设计. 清华大学出版社, 2014.

[3] 雷思磊. 自己动手写 *CPU*. 电子工业出版社, 2014.