MIPS-16E 流水 CPU 实验报告

钱迪晨　计 35　2013011402
叶子鹏　计 35　2013011404
朱俸民　计 35　2012011894

2015 年 12 月 10 日

1　概述

我们实现了一个无延迟槽的带动态分支预测的流水线 CPU。我们实现了一个通用性极佳（支持软硬件中断，支持绘图）的计算机。

我们最初的目的是尽可能不插，少插气泡，由于某些冲突必须暂停流水线，我们最终仅在 3 种情况插 1 个气泡，而绝大多数时候我们都无需插气泡。我们的 CPU 的主频最高可以达到 21M（我们使用了 ISE 自带的模拟器件 IP 核 DCM 来分频）对于老师给的 5 个测例，我们花费的时间都非常少，因为我们几乎不用插气泡，所以花费的时间约等于指令数除以主频。

为了提高运行效率，我们增加了分支预测功能，branch 指令和 jump 指令均在 decode 阶段进行，从而使得因为跳转引入的气泡尽可能的少。分支预测采取的是一个大小为 3 的查询表，每次使用 pc 进行查询，如果出现一次错误则更新，即记录上一次的结果。

除了 CPU 的核心以外，我们做了硬件中断，软件中断，像素映射的 VGA 接口的显示器（由于片内的 RAM 容量不大，不足以存下 RGB，我们的显示是蓝白的）。

由于我们是像素映射，硬件的接口不仅单一而且不利于编程（非常繁琐，由于屏幕非常大，不足以用 16 位表示坐标，我们得传 2 次参数才能确定一个点），我们用软件实现了如下几个画图的接口：（详见第六节）

1. 画一个点
2. 画一条细线段
3. 画一条粗线段
4. 画一个等腰直角三角形
5. 从数据 RAM 中读取一个形状（用于实现字符集，比如 Unicode 或者 ASCII）

2 特色

1. 多条旁路
2. 动态分支预测
3. **消除延迟槽**：我们所有的跳转指令都不需要延迟槽，也不会暂停流水线
4. **内存统一编址**
5. **外设・键盘**：一个支持字母，数字，空格，退格，回车，shift 组合键的键盘
6. **支持硬件中断**：支持键盘触发硬件中断，为其他设备触发硬件中断留下了接口
7. **支持软件中断**：支持软件中断 scanf
8. **外设・屏幕**：像素映射的屏幕，我们提供了字符和图形的库
9. **支持绘图**：比字符映射的屏幕更通用

3 具体实现

3.1 CPU 模块

CPU 模块无可厚非是我们本次实现中最重要的一个模块，这个模块里面包含了非常多的原件，我们使用了如下组件来实现我们整个 CPU。下面会一一列举。

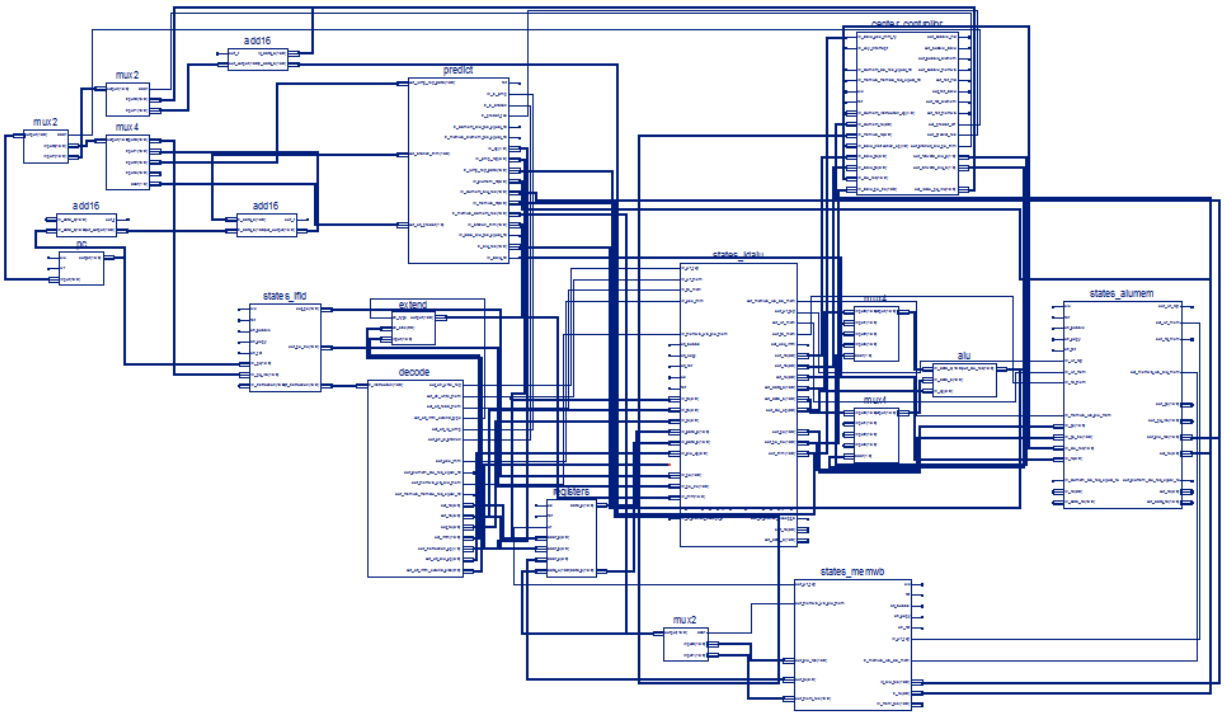

图 1: CPU 结构图

3.1.1 锁存单元

锁存单元包含 if/id 阶段，id/alu 阶段，alu/mem 阶段，mem/wb 阶段四个大的锁存器，在上升沿触发。这几个锁存器的行为都受到中央控制单元的控制，中央控制单元可以命令其进行气泡的插入，以及重置功能。

这四个部件的图如**图 2-图 3**所示，具体信号如**表 1-表 3**所示。

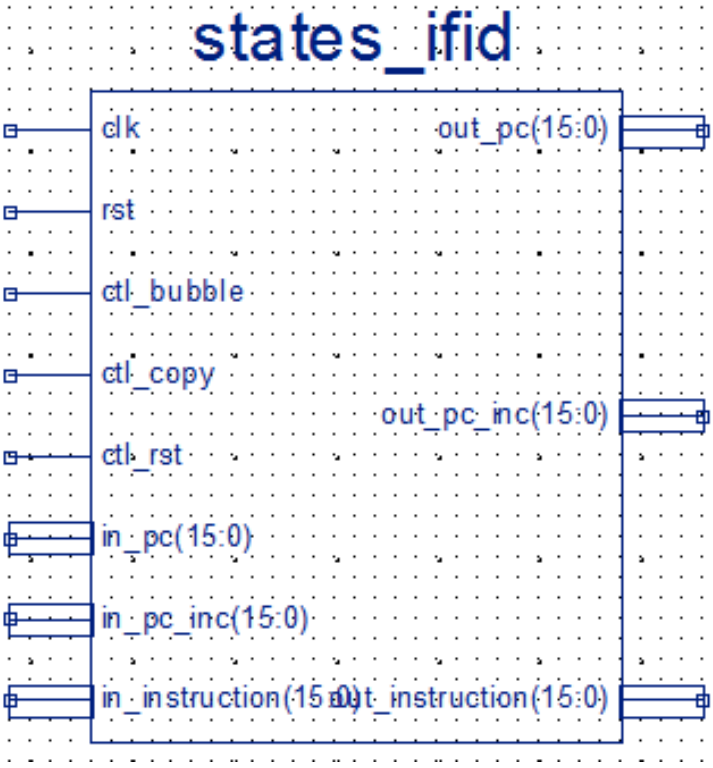

图 2: IF/ID 阶段锁存器设计图

表 1: IF/ID 阶段锁存器信号

信号	信号描述
clk	cpu 的时钟信号，上升沿的时候根据 ctl_bubble 和 ctl_rst 进行控制。如果 ctl_bubble 和 ctl_rst 均为低电平则进行锁存，将 in_pc, in_pc_inc, in_instruction 进行锁存并输出。
rst	异步清空信号，由外部控制开关接入。
ctl_bubble	气泡控制信号，由中央控制单元给出，如果该信号为高电平则表示下一个时钟上升沿，输出数据保持不变，低电平则该控制无效。
ctl_copy	由中央控制单元给出，用来进行数据拷贝。

ctl_rst	重置控制信号，由中央控制单元给出，如果如果该信号为高电平则表示下一个时钟输出清空即为一条 NOP 指令，低电平则该控制无效。
in_pc	表示下一条将要锁存的指令的 pc。
in_pc_inc	表示下一条将要锁存的指令的 pc+1。
in_instruction	表示下一条将要锁存的指令内容
out_pc	表示已经锁存的指令的 pc。
out_pc_inc	表示已经锁存的指令的 pc+1。
out_instruction	表示已经锁存的指令。

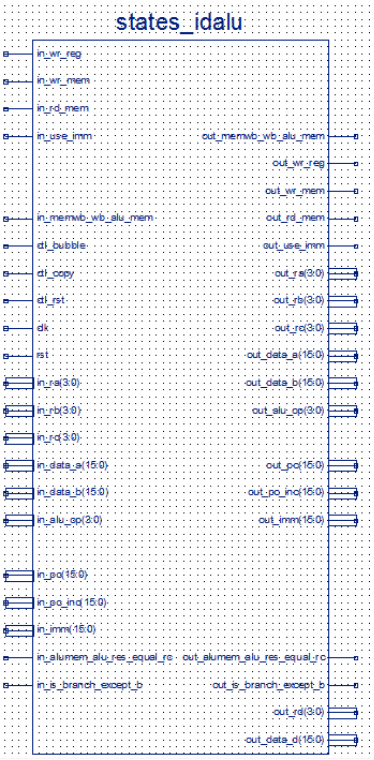

图 3: ID/ALU 阶段锁存器设计图

表 3: ID/ALU 阶段锁存器信号

信号	信号描述
in_ra	这是下一条指令 decode 出来的 alu 操作数 a 的寄存器值，注意不是数值，会传递给中央控制单元进行旁路选择。
in_rb	这是下一条指令 decode 出来的 alu 操作数 b 的寄存器值，注意不是数值，会传递给中央控制单元进行旁路选择。

in_rc	这是下一条指令 decode 出来的寄存器 c 的值，注意不是数值，会传递给中央控制单元进行旁路选择，也会传递给 alumem 锁存器。c 的寄存器表示的是写回的寄存器，非常重要，所以要一直往后传。
in_data_a	这是下一条指令 decode 出来的 alu 操作数 a 的值，用来传输给选择器，（选择器可能会选择旁路），最后送到 alu 进行计算。
in_data_b	这是下一条指令 decode 出来的 alu 操作数 b 的值，用来传输给选择器，（选择器可能会选择旁路），最后送到 alu 进行计算。
in_alu_op	这是下一条指令 alu 的操作码，会传输给三个 alu，具体内容请看 alu 部分。
in_pc	表示下一条将要锁存的指令的 pc。
in_pc_inc	表示下一条将要锁存的指令的 pc+1。
in_imm	表示下一条将要锁存的 decode 出来的立即数。
in_wr_reg	表示下一条指令是否需要在 writeback 阶段写回寄存器。
in_wr_mem	表示下一条指令是否需要在 memory 阶段写内存。
in_rd_mem	表示下一条指令是否需要在 memory 阶段读内存。
in_use_imm	表示下一条指令在 alu 阶段是否需要使用立即数，这个信号会帮助中央控制单元进行 alu_data_b 旁路的控制。
in_alumem_alu_res_equal_rc	表示下一条指令到 memory 阶段的时候，alu 的出的结果是否会在 writeback 阶段写回寄存器。这个信号也是为了帮助中央控制单元进行旁路控制。
in_memwb_wb_alu_mem	表示下一条指令在 writeback 阶段写回的数据是 memory 阶段读出的数据，还是在 alu 阶段算出的结果。
in_is_branch_except_b	表示下一条指令是否是 branch 指令，除了 b 指令以外的 branch 指令都是高电平，这个信号是帮助中央控制单元进行分支预测的检验使用的信号。
ctl_bubble	气泡控制信号，由中央控制单元给出，如果该信号为高电平则表示下一个时钟上升沿，输出数据保持不变，低电平则该控制无效。
ctl_copy	由中央控制单元给出，用来进行数据拷贝。
ctl_rst	重置控制信号，由中央控制单元给出，如果如果该信号为高电平则表示下一个时钟输出清空即为一条 NOP 指令，低电平则该控制无效。
clk	cpu 的时钟信号，上升沿的时候根据 ctl_bubble 和 ctl_rst 进行控制。如果 ctl_bubble 和 ctl_rst 均为低电平则进行锁存，将所有带有 out 前轴的信号对应的 in 信号锁存然后输出。

rst	异步清空信号，由外部控制开关接入。
out_ra	表示 alu 阶段运算的寄存器 a 的值，这个值会送到中央控制单元进行旁路的控制。
out_rb	表示 alu 阶段运算的寄存器 b 的值，这个值会送到中央控制单元进行旁路的控制。
out_rc	表示 writeback 阶段写回的寄存器 c 的值。
out_rd	rd 是一个特殊的输出，只会在 sw 这条指令进行使用，rd 也需要送到中央控制单元进行旁路的控制，他的内容和 rb 完全一致。
out_data_a	表示 alu 阶段运算 a 的值，这个值是从一个四选一的选择器得到的，这个选择器的控制由中央控制单元给出。
out_data_b	表示 alu 阶段运算 b 的值，这个值是从一个四选一的选择器得到的，这个选择器的控制由中央控制单元给出。
out_data_d	表示 memory 阶段 sw 指令可能用到的值，这个值是从一个四选一的选择器得到的，这个选择器的控制由中央控制单元给出。
out_alu_op	表示当前 alu 进行的运算符号。
out_pc	表示已经锁存的指令的 pc。
out_pc_inc	表示已经锁存的指令的 pc+1。
out_imm	表示已经锁存的指令的立即数，这个数字会连接到 alu_data_b 的选择器。
out_alumem_alu_res_equal_rc	表示当前指令到 memory 阶段的时候，alu 的出的结果是否会在 writeback 阶段写回寄存器。这个信号也是为了帮助中央控制单元进行旁路控制。
out_memwb_wb_alu_mem	表示当前指令在 writeback 阶段写回的数据是 memory 阶段读出的数据，还是在 alu 阶段算出的结果。
out_is_branch_except_b	表示当前指令是否是 branch 指令，除了 b 指令以外的 branch 指令都是高电平，这个信号是帮助中央控制单元进行分支预测的检验使用的信号。
out_wr_reg	表示当前指令是否需要在 writeback 阶段写回寄存器。
out_wr_mem	表示当前指令是否需要在 memory 阶段写内存。
out_rd_mem	表示当前指令是否需要在 memory 阶段读内存。
out_use_imm	表示当前指令在 alu 阶段是否需要使用立即数，这个信号会帮助中央控制单元进行 alu_data_b 旁路的控制。

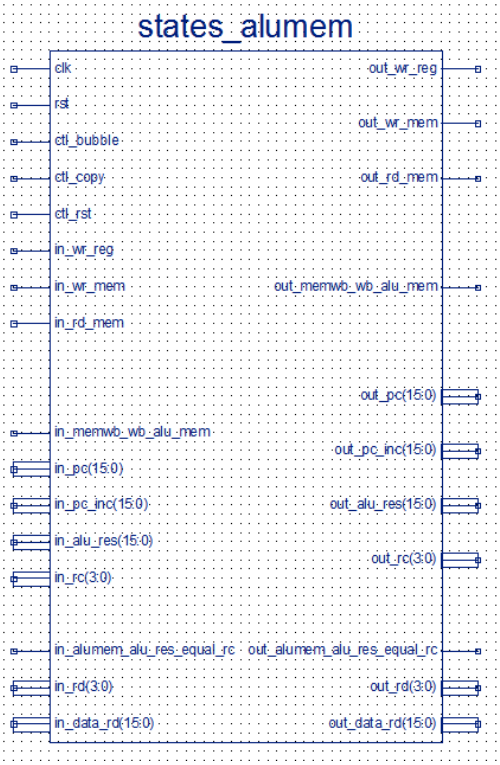

图 4: ALU/MEM 阶段锁存器设计图

表 5: ALU/MEM 阶段锁存器信号

信号	信号描述
in_rc	这是下一条指令 decode 出来的寄存器 c 的值，注意不是数值，会传递给中央控制单元进行旁路选择，也会传递给 memwb 锁存器。c 的寄存器表示的是写回的寄存器，非常重要，所以要一直往后传。
in_rd	这是专门为了 sw 指令设计的寄存器的值。
in_data_rd	这是专门为了 sw 指令设计的寄存器的值，通过一个 4 选 1 选择可以确保数据的正确性。由于我们的设计问题，alu 在进行 sw 指令的时候无法将 sw 的源寄存器值进行旁路计算解决数据冲突，所以使用了单独的一条旁路来解决这个问题。
in_pc	表示下一条将要锁存的指令的 pc。
in_pc_inc	表示下一条将要锁存的指令的 pc+1。
in_wr_reg	表示下一条指令是否需要在 writeback 阶段写回寄存器。
in_wr_mem	表示下一条指令是否需要在 memory 阶段写内存。
in_rd_mem	表示下一条指令是否需要在 memory 阶段读内存。
in_alu_res	表示下一条指令 alu 得出的结果。

in_alumem_alu_res_equal_rc	表示下一条指令 alu 计算出的结果是否会在 writeback 阶段写回寄存器。这个信号也是为了帮助中央控制单元进行旁路控制。
in_memwb_wb_alu_mem	表示下一条指令在 writeback 阶段写回的数据是 memory 阶段读出的数据，还是在 alu 阶段算出的结果。
clk	cpu 的时钟信号，上升沿的时候根据 ctl_bubble 和 ctl_rst 进行控制。如果 ctl_bubble 和 ctl_rst 均为低电平则进行锁存，将所有带有 out 前轴的信号对应的 in 信号锁存然后输出。
rst	异步清空信号，由外部控制开关接入。
ctl_bubble	气泡控制信号，由中央控制单元给出，如果该信号为高电平则表示下一个时钟上升沿，输出数据保持不变，低电平则该控制无效。
ctl_copy	由中央控制单元给出，用来进行数据拷贝。
ctl_rst	重置控制信号，由中央控制单元给出，如果如果该信号为高电平则表示下一个时钟输出清空即为一条 NOP 指令，低电平则该控制无效。
out_pc	表示已经锁存的指令的 pc。
out_pc_inc	表示已经锁存的指令的 pc+1。
out_alu_res	表示当前指令在 alu 阶段通过 alu 计算出来的值。
out_rc	表示当前指令 decode 出来的目的寄存器 c 的值。
out_wr_reg	表示当前指令是否需要在 writeback 阶段写回寄存器。
out_wr_mem	表示当前指令是否需要在 memory 阶段写内存。
out_rd_mem	表示当前指令是否需要在 memory 阶段读内存。
out_memwb_wb_alu_mem	表示当前指令在 writeback 阶段写回的数据是 memory 阶段读出的数据，还是在 alu 阶段算出的结果。

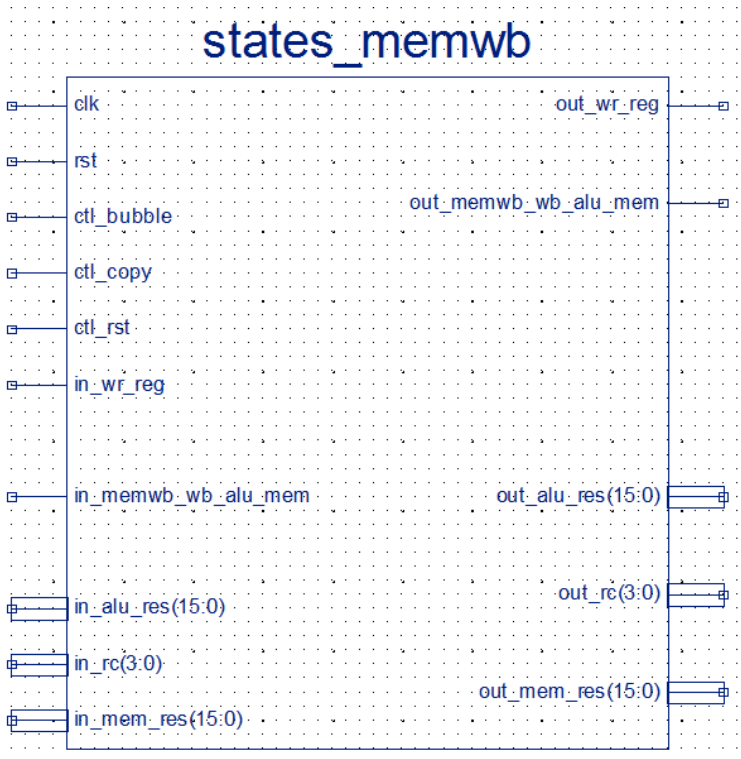

图 5: MEM/WB 阶段锁存器设计图

表 7: MEM/WB 阶段锁存器信号

信号	信号描述
clk	cpu 的时钟信号，上升沿的时候根据 ctl_bubble 和 ctl_rst 进行控制。如果 ctl_bubble 和 ctl_rst 均为低电平则进行锁存，将所有带有 out 前轴的信号对应的 in 信号锁存然后输出。
rst	异步清空信号，由外部控制开关接入。
ctl_bubble	气泡控制信号，由中央控制单元给出，如果该信号为高电平则表示下一个时钟上升沿，输出数据保持不变，低电平则该控制无效。
ctl_copy	由中央控制单元给出，用来进行数据拷贝。
ctl_rst	重置控制信号，由中央控制单元给出，如果如果该信号为高电平则表示下一个时钟输出清空即为一条 NOP 指令，低电平则该控制无效。
in_alu_res	表示下一条指令 alu 阶段计算出来的结果，可能用于写回。
in_rc	表示下一条指令写回的寄存器值。
in_wr_reg	表示下一条指令是否写回寄存器。
in_mem_res	表示下一条指令 memory 阶段得到的结果。
in_memwb_wb_alu_mem	表示下一条指令写回寄存器堆的是 alu 计算的结果还是 memory 访存的结果。

out_wr_reg	当前指令用来控制寄存器堆的写使能信号，使其可以在下降沿的时候更新。
out_memwb_wb_alu_mem	当前指令写回内容 2 选 1 的控制信号，是选择 alu 计算的结果还是 memory 访存的结果。

3.1.2 PC 锁存单元

PC 锁存器，用来锁存 PC，保证 PC 的改写受到中央控制部分的控制。我们规定在下降沿写入，组合逻辑输出。具体信号请看下表。

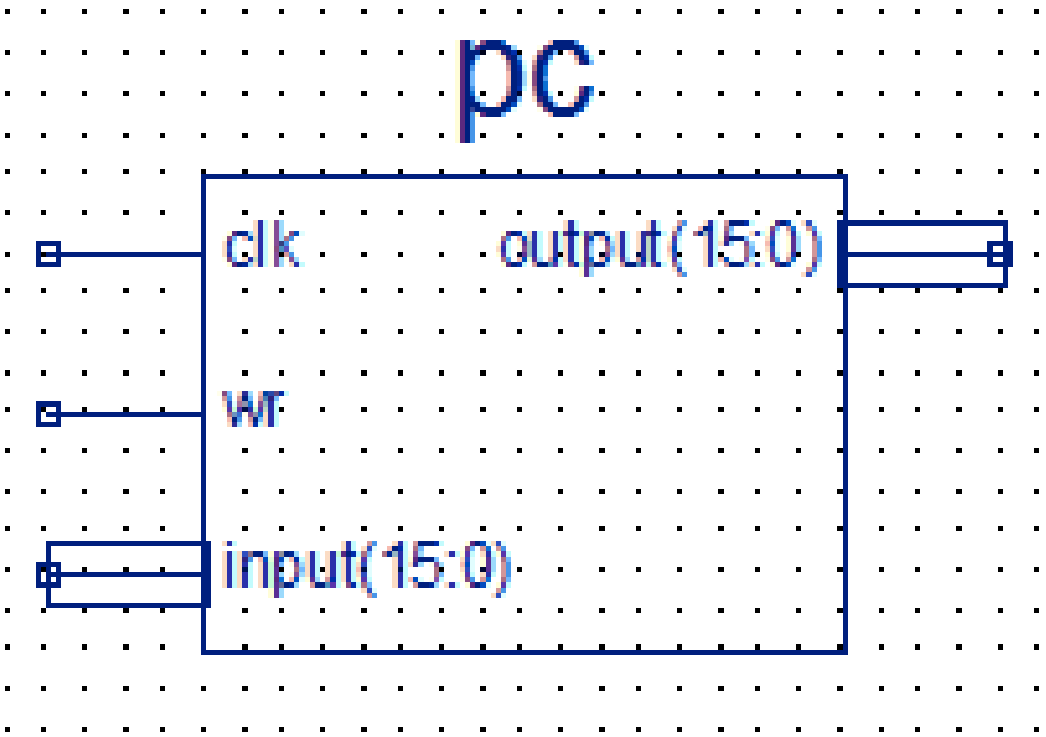

图 6: PC 锁存器

表 9: PC 锁存器

信号	信号描述
input	表示修改信号的输入，即修改的值。
clk	CPU 时钟。
wr	修改使能，如果为高电平则在下降沿修改 pc 的锁存器的值。
rst	异步清空信号，由外部控制开关接入。
output	当前 pc 锁存的值，组合逻辑。

下一条 PC 的选择是一个非常复杂的结构，具体的原理图如下。

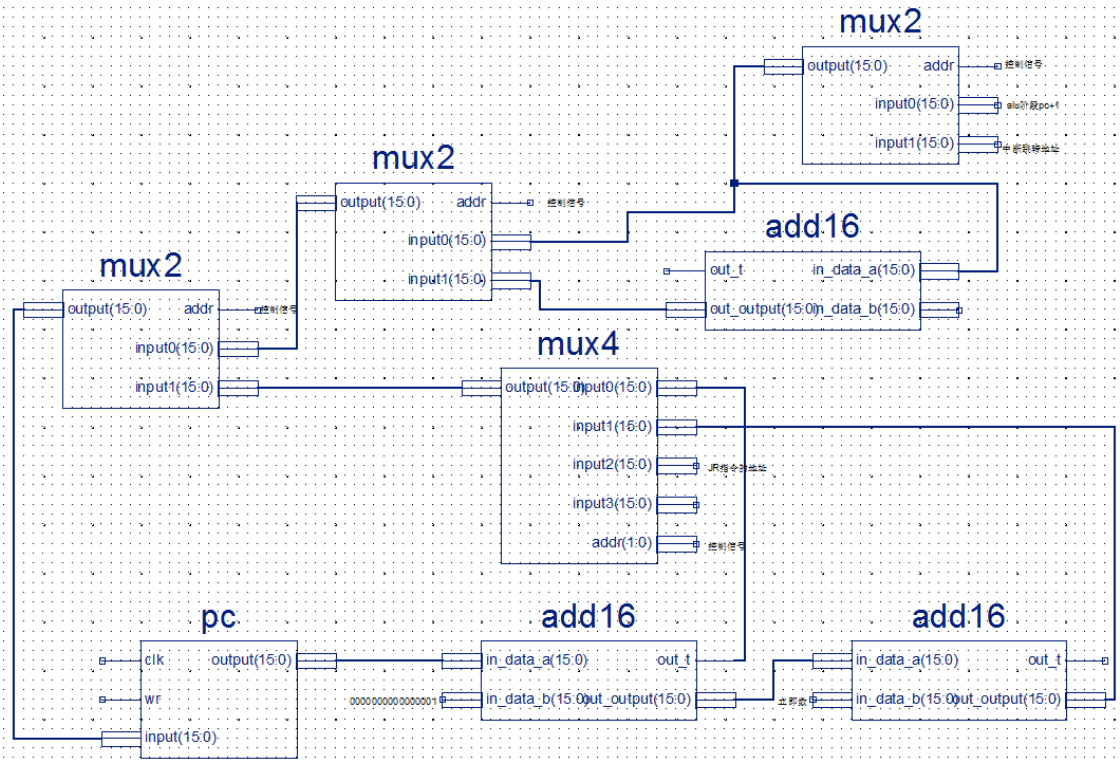

图 7: PC 结构图

具体而言，PC 的选择由两条路线决定。

1. 在 decode 阶段进行的跳转，包含 jr 指令以及分支预测。这里使用了图中下部分的部件，4 选 1 的选择器可以选择 pc+1，pc+1+imm，reg 的值进行跳转。控制信号都是中央控制单元给出的。

2. 在 alu 发现分支预测错误或者进行中断的跳转，则在上部分，最右边是一个 2 选 1，选择进行跳转的基础地址是 alu 阶段的记录下来的 pc 地址，或者是中断的指令的地址，左边的 2 选 1 则是是否加上立即数的选择器，最后连向最左边的 2 选 1 选择器。

综合上面的结构，构成了我们的 pc 整个模块，是的 cpu 的跳转可以正常的工作。由于大部分控制信号都是有中央控制单元给出，控制信号将统一在中央控制器里面描述。

3.1.3 跳转单元

跳转单元，用来在 decode 阶段进行计算和控制跳转的目的地址，这里面有旁路来处理 JR 指令的数据冲突。具体信号请看下表。

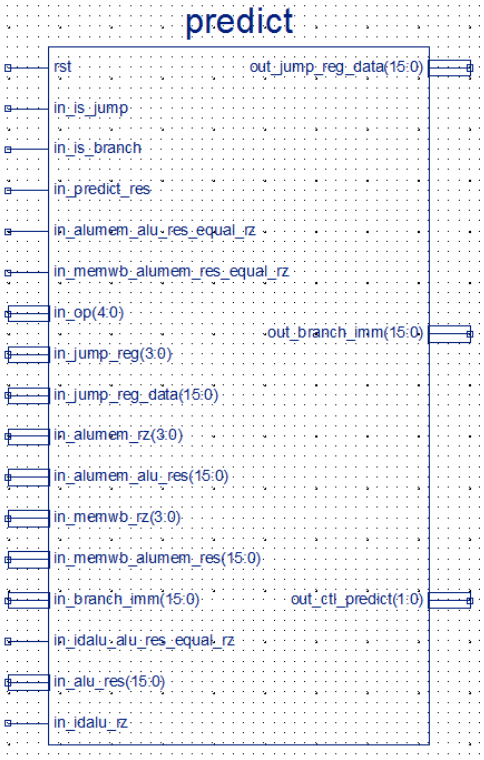

图 8: 跳转单元

表 11: 跳转单元

信号	信号描述
rst	异步清空信号，由外部控制开关接入。
in_is_jump	由 decode 给出的当前指令是不是 jump 指令。
in_is_b	由 decode 给出的当前指令是不是 b 指令。
in_is_branch_except_b	由 decode 给出的当前指令是不是除了 b 的 branch 指令。
in_predict_res	由中央控制单元给出的分支预测的结果。
in_jump_reg	由 decode 给出的 jr 指令的寄存器的值。
in_jump_reg_data	由寄存器堆给出的 jr 指令的寄存器的数据。
in_idalu_alu_res_equal_rc	这是为旁路设计的，用来计算 alu 阶段出现的数据冲突，这个信号表示 alu 阶段 alu 的值会在最后写回 c 寄存器。
in_idalu_rc	当前 alu 阶段 c 寄存器的值。
in_alu_res	当前 alu 阶段 alu 的值。
in_alumem_rc	当前 alu 阶段 c 寄存器的值。

in_alumem_alu_res_equal_rc	这是为旁路设计的，用来计算 mem 阶段出现的数据冲突，这个信号表示 mem 阶段 alu 的值会在最后写回 c 寄存器。
in_alumem_alu_res	当前 memory 阶段 alu 的值。
in_memwb_rc	当前 writeback 写回的寄存器 c 的值。
in_memwb_alume_res_equal_rc	这是为旁路设计的，用来计算 mem 阶段出现的数据冲突，这个信号表示 writeback 阶段 alu 或者 mem 的值会在最后写回 c 寄存器。
in_memwb_alume_res	writeback 阶段写回的值。
in_branch_imm	这个数据表示 branch 指令立即数的值，将会给 pc 模块用来进行加法。
out_jump_reg_data	这个输出用来连接到 pc 模块里面的 4 选 1 选择器，表示 jr 指令的目的地值。
out_branch_imm	这个数据用来输出表示 branch 指令立即数的值，将会给 pc 模块用来进行加法。
out_ctl_predict	这个数据用来输出表示分支预测的结果，连接到选择器上。

3.1.4 寄存器堆

寄存器堆用来输出寄存器的值，以及在时钟下降沿提供修改寄存器值的功能。具体信号请看下表。

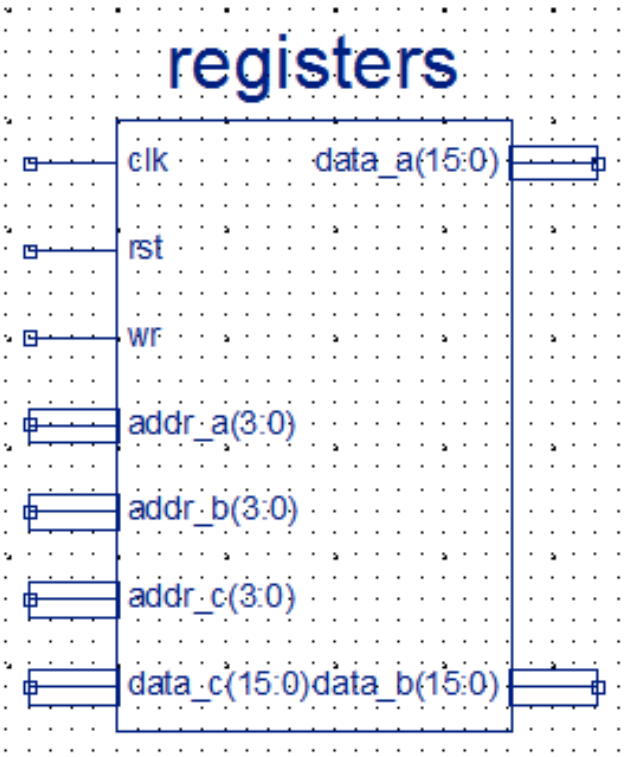

图 9: 寄存器堆

表 13: 寄存器堆

信号	信号描述
clk	CPU 时钟信号。
rst	异步清空信号，由外部控制开关接入。
wr	寄存器堆写使能，如果为高电平则在 clk 下降沿的时候将 data_c 写入对应的 addr_c 里面。
addr_a	接受 decode 出来的寄存器 a。
addr_b	接受 decode 出来的寄存器 b。
addr_c	接受 writeback 将要写回的寄存器 c。
data_a	输出 decode 出来的寄存器 a 的数值。
data_b	输出 decode 出来的寄存器 b 的数值。
data_c	写回阶段修改寄存器的值。